

Análise de Sistemas

Texto Preliminar

Geraldo Xexéo

7 de julho de 2021 18:53

DRAFT

DRAFT

Licença

Este texto é distribuído com uma licença Creative Commons - Atribuição - NãoComercial - Compartilha Igual 4.0 Internacional.



Você tem o direito de:

- **Compartilhar** – copiar e distribuir o material em qualquer suporte ou formato.
- **Adaptar** – remixar, transformar, e criar a partir do material.

De acordo com os termos seguintes:

- **Atribuição** – Você deve dar o crédito apropriado, prover um link para a licença e indicar se mudanças foram feitas. Você deve fazê-lo em qualquer circunstância razoável, mas de nenhuma maneira que sugira que o licenciante apoia você ou o seu uso.
- **NãoComercial** – Você não pode usar o material para fins comerciais.
- **CompartilhaIgual** – Se você remixar, transformar, ou criar a partir do material, tem de distribuir as suas contribuições sob a mesma licença que o original.
- **Sem restrições adicionais** – Você não pode aplicar termos jurídicos ou medidas de caráter tecnológico que restrinjam legalmente outros de fazerem algo que a licença permita.

Mais informações podem ser encontradas em https://creativecommons.org/licenses/by-nc-sa/4.0/deed.pt_BR

DRAFT

Sumário

Lista de Figuras	xiii
Lista de Tabelas	xix
Lista de Programas	xxi
1. Introdução	1
I. Conceitos Iniciais	3
2. Valor	5
2.1. Conceituação Genérica de Valor	6
2.2. Valor na Economia	7
2.2.1. Utilidade	7
2.2.2. Microeconomia Moderna e o Custo de Oportunidade	8
2.3. Valor em Marketing	9
2.3.1. Os Elementos do Valor - B2C	10
2.3.2. Os Elementos do Valor - B2B	12
2.3.3. Outras Técnicas	14
2.4. Valor Financeiro de Projetos	15
2.4.1. Custo Total de Propriedade	17
2.4.2. Retorno do Investimento	18
2.4.3. Valor Presente Líquido	19
2.4.4. Taxa Interna de Retorno	19
2.4.5. Priorização Baseada em Valor com o Cliente	20
2.4.6. Análise de Kano	20
2.5. Valor em Software	21
2.5.1. Triângulo do Valor	22

Sumário

2.5.2.	Valor em Métodos Ágeis	23
2.6.	Conclusão	24
2.7.	Exercícios	24
3. 5W2H		27
3.1.	As Perguntas Certas	28
3.2.	Usos do 5W2H	29
3.2.1.	<i>O Framework de Zachman</i>	30
3.3.	Onde Está o Valor?	30
3.4.	Várias Perguntas para cada Palavra	31
3.4.1.	Perguntas sobre o que	31
3.4.2.	Perguntas sobre quem	32
3.4.3.	Perguntas sobre quando	32
3.4.4.	Perguntas sobre onde	33
3.4.5.	Perguntas sobre como	33
3.4.6.	Perguntas sobre por que	33
3.4.7.	Perguntas sobre quanto	34
3.5.	Exercícios	34
4. Modelos e Abstrações		35
4.1.	Modelos	36
4.2.	Tipos de Abstrações	39
4.2.1.	Ocultação da Informação	39
4.2.2.	Classificação	39
4.2.3.	Generalização	41
4.2.4.	Composição ou Agregação	41
4.2.5.	Identificação	42
4.2.6.	Simplificação pelo Caso Normal	43
4.2.7.	Foco e Inibição	43
4.2.8.	Refinamento Sucessivo	43
4.2.9.	Separação de Interesses	44
4.3.	Trabalhando com as abstrações	44
4.4.	Exercícios	44
5. Dados e Informação		45
5.1.	Usando Significados Diferentes	46
5.2.	Conclusão	48
6. Sistema		49
6.1.	Tipos de Sistemas	51
6.2.	Sistemas Automatizados	52
6.3.	Princípios Gerais de Sistemas	52

7. Organizações	55
7.1. Funções Típicas de uma Organização	57
7.2. Estrutura das Organizações	57
7.3. Estratégia Organizacional	59
7.4. A Necessidade de Sistemas de Informação	59
7.5. Processo de Informatização nas Organizações	60
7.6. A necessidade centralização dos dados	62
7.7. Organograma	62
7.7.1. Relações em um organograma	63
7.7.2. Outros formatos de organograma	64
7.8. Conclusão	68
7.9. Exercícios	68
8. Sistemas de Informação	69
8.1. Características dos Sistemas de Informação	71
8.2. Sistemas de Informação Típicos e a Organização	72
8.3. Tipos de Projetos de Sistemas de Informação	72
8.4. Porque são feitos projetos de SI	73
8.4.1. Benefícios do Sistema	73
8.5. O Poder está com o usuário	74
8.6. Exercícios	75
9. Introdução a Engenharia de Software	77
9.1. O que é Software	78
9.1.1. Como cobrar? A pergunta mais comum nos cursos de ES.	80
9.2. Por que software é importante	81
9.2.1. A Ubiquidade do Software	82
9.3. Quais os riscos que software traz?	83
9.4. Por que é difícil fazer software?	85
9.4.1. O Relatório CHAOS	86
9.5. O que é Engenharia de Software	89
9.5.1. Uma Visão da Engenharia de Software	90
9.6. Por que a Engenharia de Software é necessária?	90
9.6.1. Diferenças para outras engenharias	90
9.7. Qual o Segredo do Sucesso de um Projeto de Software	91
9.8. O Corpo de Conhecimento da ES	92
9.9. As normas da Engenharia de Software	93
9.10. SEMAT	94
9.10.1. O Kernel Essencial	94
9.11. Quem é o Engenheiro de Software	97
9.12. Exercícios	97
10. O Que é um Projeto	99
10.1. Recursos e Entregas	100

Sumário

10.2.	Partes Interessadas	101
10.3.	Sucesso, Fracasso e Riscos	101
10.3.1.	A Incerteza dos Projetos e os Riscos	101
10.3.2.	Benefícios da Gerência de Projeto	102
10.4.	Pontos Críticos ou Pontos Chave de Sucesso	102
10.5.	Conclusão	103
11. Análise de Sistemas		105
II. Partes Interessadas		107
12. Partes Interessadas		109
12.1.	O Que São Partes Interessadas	111
12.1.1.	Usuários	113
12.1.2.	Perspectivas dos Usuários	113
12.2.	Partes Interessadas e Valor	115
12.3.	Gerência das Partes Interessadas	115
12.4.	A abordagem SEMAT para Partes Interessadas	116
12.5.	Caracterização da Parte Interessada	116
12.5.1.	Interesses e Objetivos	117
12.6.	Classificação das Partes Interessadas	119
12.6.1.	Análise das Partes Interessadas	121
12.7.	Representação das Partes Interessadas	123
12.8.	Engajamento das Partes Interessadas	123
12.9.	Matriz RACI	123
12.10.	Conclusão	125
12.11.	Exercícios	125
III. Oportunidades		127
13. Oportunidades e Problemas		129
13.1.	Oportunidades	130
13.2.	Problemas	131
13.3.	Tipos de Problemas	131
13.4.	Identificando Problemas	132
13.4.1.	A Técnica de Pareto	133
13.4.2.	Calculando a Quantidade	133
13.4.3.	Usando o Impacto dos Problemas	135
13.4.4.	Buscando Causas	137
13.4.5.	Organização do Diagrama Espinha de Peixe	140
13.5.	Caracterização do Problema	141

13.6.	Oportunidade e 5W2H	141
13.6.1.	Buscando Oportunidades com as Partes Interessadas	142
13.7.	Propondo Oportunidades ao Cliente	143
13.8.	Oportunidade e Valor	143
13.9.	Exercícios	144
14. Project Model Canvas		145
14.1.	Uso do 5WH	147
14.2.	As Fases da Metodologia	148
14.3.	Requisitos para a Reunião	149
14.4.	Um Exemplo de Projeto	149
14.5.	A Fase de Conceber	149
14.5.1.	O <i>Pitch</i>	150
14.5.2.	Por Que?	151
14.5.3.	Ponto de Decisão	158
14.5.4.	O Que?	159
14.5.5.	Quem?	162
14.5.6.	Como?	166
14.5.7.	Quando e Quanto?	170
14.6.	A Fase de Integração	174
14.7.	A Fase de Resolver	174
14.8.	A Fase de COmpartilhar	175
14.9.	Exercícios	175
15. Modelagem de Processos de Negócio		177
15.1.	Processos de Negócio	178
15.2.	Exemplos de Processos	179
15.3.	Representação de Processos de Negócio	180
15.4.	Por que Entender os Processos de Negócio	180
15.5.	Exercícios	181
16. Introdução ao Aris EPC		183
16.1.	Os softwares de desenho e as notações ARIS	184
16.2.	O Modelo Básico do EPC	185
16.2.1.	Atividades	186
16.2.2.	Eventos	187
16.2.3.	Regras	188
16.2.4.	Interface de Processo	189
16.2.5.	Exemplos Adicionais de EPC	190
16.3.	eEPC	192
16.3.1.	5W2H no EPC	192
16.4.	Passos para construir modelos EPC/EPCe	195
16.4.1.	EPC e Loops/Laços	195
16.5.	Deadlocks	196

Sumário

16.6.	Alguns padrões em EPC	197
16.7.	Revisão das regras básicas do EPC	200
16.8.	Exercícios	200
17.BPMN		201
17.1.	Usos de BPMN	202
17.2.	Participantes	203
17.3.	Símbolos básicos	203
17.4.	Atividades	203
17.5.	Eventos	206
17.5.1.	Eventos Iniciais	207
17.5.2.	Eventos Finais	207
17.5.3.	Eventos Intermediários	208
17.6.	<i>Gateways</i>	209
17.7.	Fluxos	211
17.8.	Alguns Exemplos	212
17.9.	Eventos Anexados a Atividades	212
17.10.	Exercícios	212
18.Regas de Negócio		213
18.1.	Classificações para Regras de Negócio	215
18.2.	Declarações Estruturais	216
18.2.1.	Termos	216
18.2.2.	Fatos	216
18.2.3.	Declarações estruturais e o modelo de dados	217
18.3.	Declarações de Ações (ou Restrições)	218
18.4.	Derivações	219
18.5.	Regras e Comportamento	219
18.6.	Escrevendo Regras de Negócio	219
18.6.1.	Como Definir Termos	219
18.6.2.	Representações Gráficas	222
18.7.	Outros Modelos de Regras de Negócio	223
18.8.	Conclusão	225
IV. Requisitos		227
19.Requisitos		229
19.1.	Definição Formal	230
19.1.1.	Exemplos de Requisitos	231
19.2.	Requisitos x Benefícios	234
19.3.	Tipos de Requisitos	234
19.3.1.	Requisitos Funcionais	234
19.3.2.	Requisitos não funcionais	234

19.3.3.	Restrições	236
19.3.4.	Outros tipos de requisitos	236
19.4.	O Papel dos Requisitos no Projeto	236
19.5.	Engenharia de Requisitos	237
19.6.	Elicitação de Requisitos	237
19.6.1.	Processo de Elicitação de Requisitos	239
19.7.	Níveis de Abstração dos Requisitos	242
19.8.	O Risco de Requisitos Falsos	242
19.9.	Características de um Bom Requisito	243
19.9.1.	Efeitos de Requisitos com Problemas	244
19.10.	Atributos dos Requisitos	245
19.10.1.	Volere e o <i>snowcard</i>	245
19.11.	Priorizando Requisitos	247
19.12.	Requisitos Mudam com o Tempo	248
19.13.	Requisitos e Necessidades	248
19.14.	Requisitos de Acordo com Normas IEEE	249
19.14.1.	Requisitos Funcionais na Norma IEEE	251
19.14.2.	O Documento de Especificação de Requisitos	251
19.15.	A Visão do Novo Sistema	252
19.16.	Descrevendo Requisitos	253
19.16.1.	Exemplos de Requisitos	253
19.17.	Exercício	255
20. Histórias do Usuário		257
20.1.	Conceituação de Histórias do Usuário	259
20.2.	Um <i>Template</i> Padrão para Histórias de Usuário	260
20.2.1.	Outros formatos possíveis	261
20.3.	INVEST	262
20.3.1.	Independente	262
20.3.2.	Negociável	264
20.3.3.	Com Valor	264
20.3.4.	Estimável	264
20.3.5.	Pequena	265
20.3.6.	Testável	266
20.4.	Estados de Uma História do Usuário	266
20.5.	Exercícios	266
21. Casos de Uso		267
21.1.	Conceituação de Caso de Uso	269
21.2.	Conceitos Importantes nos Casos de Uso	271
21.2.1.	Autor	271
21.2.2.	Objetivos	272
21.2.3.	Cenários	273
21.2.4.	As Alternativas	276

Sumário

21.2.5. Casos de Uso em UML	278
21.3. A Narrativa do Caso de Uso	278
21.4. O Nível de Abstração	278
21.5. O Escopo do Caso de Uso	279
21.6. Casos de Uso Especiais	280
21.7. Partes do Caso de Uso	281
21.8. Exemplo de Um Caso de Uso	282
21.9. Levantando Casos de Usos	283
21.10. Diagramas de Caso de Uso	286
21.10.1. Especificando o <i>Subject</i>	287
21.10.2. Generalização entre Atores	287
21.10.3. Relações Entre Casos de Uso	289
21.10.4. Usando Cardinalidades em Casos de Uso	292
21.11. O Que o Diagrama de Caso de Uso Não Informa	293
21.12. Exercícios	293
22. Métodos Simples de Elicitação de Requisitos	295
22.1. A entrevista	295
22.1.1. Entrevista por Questionário	297
22.1.2. Entrevista Aberta	297
22.1.3. Entrevista Estruturada	298
22.1.4. O processo da entrevista	298
22.1.5. Preparando a entrevista	299
22.1.6. Realizando a entrevista	300
22.1.7. O Comportamento do Entrevistador	301
22.1.8. Roteiro Básico	302
22.1.9. Documentando a Entrevista	303
22.1.10. As perguntas de conclusão	304
22.2. A Reunião de JAD	304
V. Modelo Estrutural	307
23. Modelo de Entidades e Relacionamentos	309
23.1. Modelo Conceitual	309
23.1.1. Modelo Lógico	310
23.1.2. Modelo Físico	311
23.2. Modelo de Entidades e Relacionamentos	311
23.3. Diagrama de Entidades e Relacionamentos	314
23.4. Desenvolvendo um Modelo Conceitual	314
23.5. Entidades	315
23.5.1. Onde encontrá-las	317
23.5.2. Descrevendo Entidades	318

23.6.	Relacionamentos	318
23.6.1.	O Relacionamento de Herança	320
23.6.2.	Cardinalidade	321
23.6.3.	Descrevendo Relacionamentos	324
23.7.	Atributos	325
23.7.1.	Descrevendo Atributos	325
23.7.2.	Atributos Identificadores (Chaves Candidatas e Chaves Primárias)	325
23.7.3.	Relacionamentos Identificadores	326
23.8.	Descrição Gráfica do Modelo	326
23.9.	Exemplos de notação da Engenharia de Informação	327
23.10.	Exercícios	328
24. Modelo Orientado a Objeto		329
24.1.	Exercícios	329
VI. Estimativas		331
25. Medidas e Estimativas do Tamanho do Software		333
25.1.	Esforço	335
25.2.	Algumas Medidas Conceitualmente Simples	336
25.2.1.	A Medida Mais Simples: Linhas de Código	336
25.2.2.	Pontos de História	338
25.2.3.	Tamanhos de Camisa	339
25.3.	Visão Geral dos Métodos de Estimativa	340
25.4.	Métodos de Estimativa Baseado em Opinião	341
25.4.1.	O Método de Estimativa do PERT	341
25.4.2.	O Método de Delphi	341
25.4.3.	O Planning Poker	341
26. Análise de Pontos de Função		343
26.1.	Visão Geral de Pontos de Função	344
26.2.	Procedimento de Contagem	344
26.3.	Funções Transacionais	345
26.4.	Funções de Dados	346
26.5.	Identificando Funções de Negócio	346
26.5.1.	Identificando Saídas	347
26.5.2.	Identificando Consultas	347
26.6.	Identificando Entradas	347
26.6.1.	Entendendo as Lógicas de Processamento	348
26.7.	Identificando Arquivos Internos	349
26.7.1.	Identificando Arquivos Externos	349
26.7.2.	Identificando Itens de Dados (DETs)	349
26.8.	As Perguntas	351

Sumário

26.9.	Cálculo dos Pfs Finais	352
26.10.	Conclusão	353
27. Uma Visão Reduzida do Modelo COCOMO II		355
27.1.	Dois Modelos em Um	356
27.2.	Esforço em Função do Tamanho	356
27.3.	Existe uma Relação Entre Esforço e Tempo	357
27.3.1.	Cálculo dos Fatores de Escala	358
A. Agradecimentos		375

DRAFT

Lista de Figuras

2.1.	Elementos do Valor B2C. Fonte:Almquist, Senior e Bloch, 2016	11
2.2.	Hierarquia das Necessidades de Maslow. Fonte: Wikipedia Commons por Felipe Sanchez (CC-BY-SA 3.0) e J. Finkelstein (GFDL)	11
2.3.	Elementos do Valor B2B. Fonte: Almquist, Cleghorn e Sherer, 2018 .	12
2.4.	Cálculo do valor futuro de um investimento de R\$10.000,00 por mês com uma taxa de juros de 1%	16
2.5.	Gráfico do TCO de impressoras em função de da quantidade de páginas impressas. Percebe-se que após aproximadamente 3500 páginas é melhor comprar uma impressora a laser mais cara, caso esse fosse o único fator de análise.	18
2.6.	O triângulo do valor para a maioria dos projetos.	23
3.1.	O Framework de Zachman fornece uma visão da arquitetura empresarial a partir de 6 dimensões definidas pelo 5WH. Reprinted with permission by John A. Zachman, Zachman International.	30
4.1.	Diferentes tipos de mapas, ou seja, modelos, cada um com um nível diferente de abstração e dedicado a uma utilização distinta.	37
4.2.	Exemplo de Diagrama de Casos de Uso.	38
4.3.	Um brinquedo é composto de suas partes. Foto do autor	42
5.1.	Pirâmide da hierarquia do DIKW	47
6.1.	O Sistema Solar é um sistema natural. Imagem por Don Cloud no Pixabay 	51
6.2.	Um <i>smartphone</i> é um sistema artificial. Imagem 	52
7.1.	Um organograma simples, de uma empresa hipotética.	63

Lista de Figuras

7.2.	Um exemplo de um organograma de um pedaço de uma empresa hipotética escrito em ARIS.	64
7.3.	A relação de subordinação, normalmente mantida na vertical, entre o gerente geral e seus gerentes subordinados, que aparecem em um mesmo nível.	64
7.4.	A relação de assessoria é normalmente feita com uma barra horizontal, como a Assessorial Internacional aparece no diagrama acima.	65
7.5.	Um exemplo de organograma radial, com uma espécie de “diagrama de Venn” indicando que os diretores e o presidente compõe a diretoria colegiada.	66
7.6.	Organograma do Poder Judiciário, em forma inversa, criado pelo site Guia de Direitos, deixado em Copyleft.	67
8.1.	Uma tela de um sistema de informações real	70
9.1.	Um software executando na web e parte do seu código.	79
9.2.	Sala de controle de uma usina nuclear, feita originalmente com tecnologia analógica e modernizada com computadores e software	82
9.3.	Esquema da virtualização do hardware.	83
9.4.	Tamanho de código de alguns projetos conhecidos.	87
9.5.	Curva de falha do hardware	88
9.6.	Curva de falha do software	89
9.7.	Pirâmide que explica a construção de conceitos na Engenharia de Software.	90
9.8.	Mind map para as áreas de conhecimento da Engenharia de Software: Fonte:(IEEE Computer Society, 2014a)	93
9.9.	O Kernel Essencial do SEMAT. Fonte:(Jacobson, Ng et al., 2013) . .	95
9.10.	Os Espaços de Atividades no Kernel do SEMAT. Fonte:(Jacobson, Ng et al., 2013)	95
9.11.	As Competências no Kernel do SEMAT. Fonte:(Jacobson, Ng et al., 2013)	96
12.1.	O Kernel Essencial do SEMAT. Fonte:(Jacobson, Ng et al., 2013) . .	110
12.2.	O impacto das partes interessadas.	111
12.3.	3 tipos comuns de usuários dos sistemas. Fonte:próprio autor	114
12.4.	Quadro para o tratamento das partes interessadas. Fonte:(PMI, 2017)	120
12.5.	Classificação das partes interessadas. Fonte: (Mitchell, Agle e Wood, 1997)	122
13.1.	Gráfico com a quantidade de problemas.	135
13.2.	Pareto com custo	138
13.3.	Um diagrama de Espinha de Peixe	139
13.4.	Um diagrama de Espinha de Peixe resumido, construído com uma ferramenta de <i>mind map</i>	139

13.5.	Um diagrama de Espinha de Peixe expandido, construído com uma ferramenta de <i>mind map</i>	140
13.6.	Um Diagrama de Causa Raiz vertical.	141
14.1.	O canvas do Project Model Canvas. Fonte:(Finocchio Jr., 2013)	146
14.2.	PMC mostrando o 5W2H	148
14.3.	Passos PMC em BPMN	148
14.4.	Ordem de trabalho do PMC	150
14.5.	Preenchendo o pitch do PMC.	151
14.6.	Preenchendo a justificativa do PMC.	152
14.7.	Preenchendo os objetivos SMART do PMC.	154
14.8.	Preenchendo os benefícios do PMC.	156
14.9.	Preenchendo o produto do PMC.	160
14.10.	Preenchendo os benefícios do PMC.	161
14.11.	Preenchendo as partes interessadas do PMC.	164
14.12.	Preenchendo a equipe do PMC.	165
14.13.	Preenchendo as premissas do PMC.	167
14.14.	Preenchendo as entregas do PMC.	168
14.15.	Preenchendo as restrições do PMC.	170
14.16.	Preenchendo os riscos do PMC.	171
14.17.	Preenchendo a linha de tempo do PMC.	172
14.18.	Preenchendo os custos do PMC.	173
14.19.	Uma alternativa para calcular inicialmente os custos do PMC.	174
16.1.	Um processo de recepção e atendimento de pedido escrito em ARIS/EPC, desenhado no ARISExpress	184
16.2.	Representação de uma atividade no Aris	186
16.3.	Representação de um evento em Aris	188
16.4.	Exemplos de eventos	188
16.5.	Um conector só deve ser...	189
16.6.	Formas de representação de uma interface	190
16.7.	Indicando que um processo é continuação lógica de outro processo	190
16.8.	Processo simplificado de entrega de pizzas	191
16.9.	Processo simplificado de atendimento de ordem de serviço	191
16.10.	Elementos complementares dos diagramas eEPC.	193
16.11.	Nova lista e imagens dos elementos complementares dos diagramas EPC no ARISExpress.	193
16.12.	Exemplo de eEPC	194
16.13.	Um EPC válido	196
16.14.	O modelo acima, demonstra um deadlock	197
16.15.	Descrição EPC para Se X então Fazer A, Se Y, então Fazer B	198
16.16.	Caminhos paralelos, na notação do ARISExpress	199
16.17.	Descrição EPC para Se Não X então Fazer A	199

Lista de Figuras

17.1.	Um exemplo de processo descrito em BPMN. Fonte: imagem do autor	202
17.2.	Exemplo de uso de um evento paralelo em BPMN	211
17.3.	Exemplo do atendimento da ordem de serviço em BPMN	212
18.1.	Exemplo de duas regras de negócio descritas juntas e utilizando uma notação de DER simplificada.	222
18.2.	Exemplo da descrição das regras de negócio em separado.	222
18.3.	Regras com cardinalidades.	223
19.1.	Um cartão Volere.	232
19.2.	Exemplo de um diagrama de casos de uso	233
19.3.	Exemplo de um mapa mental.	233
19.4.	Pequena taxonomia de requisitos não funcionais. Fonte: (Sommerville, 2015)	235
19.5.	O papel dos requisitos no desenvolvimento do sistema. Fonte:(J. Robertson e S. Robertson, 2006)	237
19.6.	A elicitação de requisitos é um processo interativo onde, por aproximações sucessivas, o desenvolvedor consegue a aprovação de uma especificação de requisitos.	240
19.7.	O processo de elicitação de requisitos, adaptado de Christel e Kand (1992).	240
19.8.	Cada requisito se posicionará em algum ponto desse gráfico, de acordo com as notas de avaliação.	246
20.1.	Exemplo de uma história do usuário em um cartão. Fonte: do autor.	259
20.2.	Partes encontradas em um cartão de história do usuário, segundo um modelo original da Connextra (Patton e Economy, 2014).	260
21.1.	Exemplo de cenário principal de caso de uso.	268
21.2.	Exemplo de um caso de uso.	269
21.3.	Representação de um ator em um diagrama de caso de uso.	272
21.4.	Representação de dois atores identificados.	272
21.5.	Exemplo de cenário principal do caso de uso sacar dinheiro	274
21.6.	Representação abstrata de um cenário principal e dos fluxos alternativos	276
21.7.	Representação abstrata de cada alternativa investigada nesse caso.	276
21.8.	Exemplo de cenário principal do caso de uso atribuir notas	277
21.9.	Exemplo de cenário principal do caso de uso emitir boletim	284
21.10.	Os elementos básicos de um diagrama de caso de uso são: atores, casos de uso e associações. Fonte: do autor	287
21.11.	Indicando o sistema ao qual os casos de uso se referem. Fonte:do autor.	288
21.12.	Nesse diagrama de casos de uso, o gerente é um especialização de funcionário, logo pode fazer tudo que o funcionário faz, mas alguns casos de uso que são exclusivos dele. Fonte: do autor	288

21.13.	Um <i>Professional de Saúde</i> é a generalização de Enfermeira, Médico ou Auxiliar de Enfermagem, mas nenhum ator é um profissional de saúde se não for de um dos outros tipos. Fonte: do Autor.	289
21.14.	Representações das associações entre casos de uso.	290
21.15.	Exemplo da associação de inclusão entre casos de uso	290
21.16.	Um exemplo da associação de extensão entre casos de uso.	291
21.17.	Exemplo da associação de herança entre casos de uso	292
21.18.	Exemplo do uso de cardinalidades em casos de uso. Fonte: do autor .	293
22.1.	Uma configuração ideal para realização de uma sessão de JAD.	306
23.1.	As três camadas de modelo de dados.	310
23.2.	Um diagrama de entidades e relacionamentos simples, sem mostrar atributos	312
23.3.	Um modelo de entidades e relacionamentos escrito na notação da Engenharia de Informação	313
23.4.	Modelo ER só com entidades e relacionamentos, notação da Engenharia da Informação	328
23.5.	Modelo ER com atributos, notação Engenharia da Informação	328
25.1.	Ilustração da sequência do Planning Poker com duas rodadas	342
26.1.	O processo geral de contagem de Pontos de Função. Fonte: do autor .	345
26.2.	Planilha de registro para ajudar a contar os Pontos de Função	353
27.1.	Efeitos das constantes A e B na relação entre tamanho e esforço	357

DRAFT

Lista de Tabelas

2.1. Valores usados para calcular o TCO de uma impressora.	17
2.2. Tabela de cálculo do Valor Presente Líquido, considerando saldos diferentes em cada período. Foi usada uma taxa de 5% a cada período. A fórmula para cada Valor Presente é $\text{Caixa}_i / (1 + \text{Taxa})^{\text{Período}}$	19
2.3. Resolvendo a classe de Kano por meio do cruzamento de duas perguntas (Moorman, 2012)	21
3.1. Perguntas e respostas a partir da notícia.	29
4.1. Sequência de imagens que mostram abstrações crescentes de uma imagem inicial que representa uma mulher. Foto por Michael Jatremski.	36
4.2. Exemplos de Classificação	40
4.3. Exemplos de Generalização	41
4.4. Exemplos de Composição	42
9.1. Resolução de projetos de software por tamanho e por método de desenvolvimento, para mais de 10.000 projetos. Fonte:(The Standish Group, 2015)	87
9.2. Fatores de Sucesso de um Projeto de Software	92
12.1. Exemplo de formulário para registro do interesse e objetivo da parte interessada.	118
12.2. Exemplo de matriz de cruzamento para registro do interesse e objetivo da parte interessada.	119
12.3. Matriz de Engajamento: C significa comportamento corrente, D o comportamento desejado.	120
12.4. Exemplo de Matriz RACI	124
13.1. Uma contagem de tipos de erros para uso da técnica de Pareto	134
13.2. Custo médio relativo dos problemas	136

Lista de Tabelas

13.3. Custo total relativo	137
16.1. Objetos básicos do EPC, em várias versões do Aris.	185
17.1. Símbolos para os eventos que podem aparecer no início de um processo. .	205
17.2. Símbolos para os eventos que podem aparecer no início de um processo. .	207
17.3. Símbolos para os eventos que podem aparecer no fim de um fluxo de processo.	208
17.4. Eventos que podem aparecer no meio de um processo.	209
17.5. Símbolos para os gateway.	210
18.1. Exemplos de Fatos	217
18.2. Classificação e exemplos das regras de negócio do tipo fato(Hay et al., 2000)	218
18.3. Formas de descrever regras de negócio e suas características adaptado de (Von Halle, 2002).	220
18.4. Exemplos de definições bem e mal escritas segundo as orientações de R. G. Ross (2017).	221
18.5. Classificação para regras, definição e templates adaptada de (Von Halle, 2002)	224
19.1. Adequação das formas de elicitação propostas por K. E. Wiegers e Beatty (2013) e alguns tipos de projeto.	238
19.2. Processo de Elicitação de Requisitos segundo Christel e Kand (1992). . .	241
21.1. Verbos comumente usados para nomear casos de uso	273
21.2. Uma forma alternativa de representar os cenários possíveis a partir de um cenário principal e as alternativas associadas	277
21.3. Ícones que representam o nível de abstração dos casos de uso. Fonte:(Cockburn, 2000)	279
21.4. Ícones que representam o nível de escopo dos casos de uso. Fonte:(Cockburn, 2000)	280
21.6. Associações que podem ocorrer entre atores e casos de uso.	287
26.1. Tabela de apoio a determinação do tipo de função transacional. P=possível, N=Não permitida, O=obrigatório, O [*] = obrigatório que pelo menos uma das condições aconteça.	348
26.2. Cálculo da complexidade das saídas externas.	350
26.3. Cálculo da complexidade das entradas externas.	350
26.4. Cálculo da complexidade das consultas externas.	350
26.5. Cálculo da complexidade dos arquivos lógicos internos.	351
26.6. Cálculo da complexidade das interfaces lógicas externas.	351
27.1. Conversão de Pontos de Função não ajustados para linhas de código. Fonte: (B. Boehm, Abts e Brad Clark, 1997)	356
27.2. Valores das constantes no modelo COCOMO II padrão	358

Lista de Programas

25.1. Programa Hello World para Win32(Rösler, 1994)	336
25.2. Programa Hello World em Python 3(Rösler, 1994)	337

DRAFT

DRAFT

1

Introdução

Esse livro foi desenvolvido para os cursos de Análise de Sistemas e Modelagem de Sistemas de Informação.

DRAFT

DRAFT

Parte I.

Conceitos Iniciais

DRAFT

DRAFT

Conteúdo

2.1.	Conceituação Genérica de Valor	6
2.2.	Valor na Economia	7
2.3.	Valor em Marketing	9
2.4.	Valor Financeiro de Projetos	15
2.5.	Valor em Software	21
2.6.	Conclusão	24
2.7.	Exercícios	24

O custo de uma coisa é a quantidade de vida que é necessária para ser trocada por aquilo imediatamente ou a longo prazo.

(Henry David Thoreau)

O objetivo de qualquer sistema de informações é **entregar valor às partes interessadas**. O conceito de valor, porém, tem várias acepções. Mesmo que muitos métodos e autores se baseiem neste objetivo, raramente valor é definido. Assim, fornecedores e clientes se perguntam: o que é valor? Como posso medir valor?

A resposta pode depender da escolha de uma visão comum do significado de valor que depende de todas as partes interessadas do sistema, logo, é influenciada também por todo o contexto onde o projeto está inserido, incluindo o *background* das partes interessadas.

Este capítulo busca mostrar as várias facetas da ideia de valor. Começa com uma

2. Valor

abordagem genérica e depois mostra alguns entendimentos da palavra valor na economia, no marketing e na área de software. Obviamente há alguma similaridade entre as definições, mas os diferentes interesses entre as áreas também fornecem um conhecimento mais amplo do que é valor e de como as pessoas entendem o que é valor.

2.1. Conceituação Genérica de Valor

A palavra **valor** pode ser usada em vários sentidos (Cairncross, 1951, pg. 140):

- o **valor moral**, por exemplo, consideramos um *valor* a liberdade, a honestidade;
- o **valor estético**, por exemplo, damos *valor* a obra de Camões;
- o **valor em uso**, ou seja, ligado a utilidade, a como uma coisa atende as necessidades de alguém, como o valor da água, que é alto para a existência de vida, mesmo ela sendo barata;
- o **valor de troca**, por exemplo, o preço de uma casa a ser vendida, e
- o **valor ideal de troca**, o preço que um comprador imaginário pagaria pela casa.

Procuramos entender a acepção de valor com relação aos assuntos ligados aos negócios, em busca de entender **o que é valor para uma organização que está adquirindo ou desenvolvendo um software**. Isto não significa, porém, que a presença de atributos éticos e estéticos em um software não agregue valor, justamente o contrário. É possível, inclusive, que uma organização procure um produto que faça as mesmas funções que o produto que ela já possui, mas que seja mais agradável de usar, valor estético, ou garanta algumas propriedades ligadas ao valor ético, como privacidade.

Em todo caso, mesmo que a organização busque originalmente outro tipo de valor, no momento do projeto é mais provável que a interpretação dada ao conceito seja, na prática, ligada a fatores econômicos, financeiros ou legais, de acordo com a situação.

Da forma mais geral, é possível entender valor como **uma qualidade atribuída a algo ou alguém**, mas essa definição é muito ampla. Beleza, por exemplo, também é uma qualidade atribuída. Neste quadro em especial, valor é uma qualidade relacionada com a **utilidade que um produto ou serviço tem para os que o consomem ou usufruem**.

A princípio, valor econômico é “a importância que um indivíduo dá a determinado bem ou serviço, seja para uso pessoal ou organizacional, seja para a troca” (Holanda Ferreira, 1986).

Essa definição do Dicionário Aurélio é interessante para sistema de informações , porque já diz que o valor é ligado a importância dada ao bem ou serviço que está sendo recebido para o uso, ou sendo produzido para a troca.

Nas próximas seções veremos definições de valor nas seguintes áreas:

- Economia;
- Marketing;

- Projetos, e
- Software.

2.2. Valor na Economia

As tentativas de definir algo que pode ser chamado de valor na Economia tem, historicamente, também relação com o entendimento do preço dos objetos e serviços.

Na Economia Clássica, Smith, em 1776, primeiro reconhece que valor pode ter dois significados diferentes, um ligado a utilidade de um objeto particular, e outro ao poder de comprar outro bens. O primeiro seria o “valor de uso”, o segundo o “valor de troca”(Smith, 2003, Livro I, Cap. 4, para. 13)

Smith ainda discutiu valor, no sentido de preço, ou o preço relativo entre bens e serviços (Cairncross, 1951), com três abordagens (King e McLure, 2014), sendo que a terceira antecipou a teoria subjetiva de valor que apareceria mais tarde:

- o trabalho incorporado, que seria adequado as sociedades primitivas;
- a soma dos custos de produção, incluindo terra, capital e trabalho, mais adequado ao capitalismo, e
- a quantidade trabalho e o incômodo de adquirir, ou que é pougado e pode ser imposto a outro, que é seu valor de troca (Strathern, 2003).

Podemos dizer que muitas empresas usam abordagens como essa para calcular o preço de seu software: calcular todos os custos de produção e somar um lucro, ou **markup**, sobre esse preço.

Também as organizações ou pessoas que estão pagando por um produto de software podem ser associadas a definição mais evoluída de Adam Smith: elas pagam para evitar algum incômodo.

Já em 1821, Ricardo (1996) propôs que para se saber o valor é necessário saber a utilidade, e que existiria um preço primário e natural, dado pela quantidade comparativa de trabalho necessário para a produção (King e McLure, 2014).

Novamente, a interpretação clássica de valor tem relação com a prática de desenvolvimento de software: utilidade para o usuário, custo de produção para o desenvolvedor.

2.2.1. Utilidade

Utilidade é o grau satisfação que obtemos do consumo de bens e serviços (Krugman e Wells, 2013). Não é possível medir utilidade de forma prática, ou seja, dizer que algo tem 100 ou 2354 de utilidade para alguém. Porém, teoricamente, se discute a **função utilidade**, que é individual. Isso significa que cada pessoa tem uma satisfação diferente consumindo um produto, e não é possível comparar utilidade entre pessoas.

2. Valor

De acordo com o **princípio da diminuição da utilidade marginal**, cada unidade adicional de bens ou serviços adiciona menos a utilidade total do que a unidade prévia. Esse é um princípio genérico. A verdade é que existem produtos que tem utilidade marginal que aumenta, por exemplo, se você precisa de uma quantidade X para atingir um objetivo mínimo, a utilidade marginal não vai diminuir até você atingir X unidades, e pode até aumentar se for uma solução pior ter $X - 1$ unidades (Krugman e Wells, 2013). Um exemplo seria construir um muro de tamanho pré-determinado com tijolos, a utilidade dos tijolos não vai diminuir até atingir a quantidade necessária para o muro. Porém se você quer um muro que garanta privacidade, sem especificar a altura, a utilidade dos tijolos vai começar a diminuir quando certo tamanho é atingido, referente a uma privacidade necessária, até que a privacidade máxima seja alcançada e a utilidade marginal seja zero.

Software é um desses casos. Para um software funcionar, é necessária uma funcionalidade mínima que permita que ele comece a funcionar, porém após essa funcionalidade mínima, cada funcionalidade a mais vai seguir esse princípio, até o ponto que o cliente não mais vai querer pagar por uma funcionalidade, porque não vai valer a pena, já que o investimento vai ser maior que o retorno.

Utilidade é uma coisa importante no valor de produtos de software. Basicamente, em todas metodologias que usam o conceito de valor, o que se pode deduzir das explicações dadas é que **o mais útil em um certo contexto tem mais valor e deve ser priorizado**. Em software, utilidade pode ser um sinônimo de valor em alguns projetos.

2.2.2. Microeconomia Moderna e o Custo de Oportunidade

O **custo de oportunidade** indica “do que alguém deve desistir para obter o que deseja”(Greenlaw, Shapiro e Taylor, 2017), ou seja, indica a oportunidade perdida de se consumir ou usufruir outra coisa quando se consome ou usufrui de algo. É o “custo da próxima melhor alternativa” ao que está sendo consumido (Greenlaw, Shapiro e Taylor, 2017).

Segundo Krugman e Wells (2013), “o verdadeiro custo de algo é o seu custo de oportunidade”. Se alguém deseja um software, qual o verdadeiro custo disso? Isso envolve então o custo de oportunidade de uma segunda opção, por exemplo, não desenvolver o software e arcar com as consequências e investir o dinheiro.

Assim, novamente em software, o custo de oportunidade é um bom significado de valor. Perguntas que podem ser feitas, em caso de discussões de valor, estão ligadas alternativa de não fazer o produto, ou não corrigir um erro. E, no cálculo de um valor financeiro, o preço das outras ofertas que vão ser recebidas pelo cliente.

O conceito de valor na Economia, como tratado aqui, nos permite trabalhar com algumas ideias onde o homem é visto como um ser racional, que faz cálculos e toma a decisão de forma acertada baseada nesses cálculos. Porém o homem também é um animal emocional, e sua visão de valor envolve necessidades e desejos, como veremos na

abordagem que o Marketing dá ao conceito de valor.

2.3. Valor em Marketing

O estudo de valor em marketing se inicia com o entendimento do que são **necessidades**, ou seja, “os requisitos básicos dos seres humanos”: água, comida, recreação, educação, etc. (Kotler e Keller, 2012). Necessidades são estados onde se sente uma privação (Kotler, Armstrong et al., 2017). Em uma empresa, comprar um software que emite nota fiscal segundo a legislação vigente é uma necessidade.

Necessidades se transformam em **desejos** “quando são direcionadas para objetivos específicos que podem satisfazê-las”(Kotler e Keller, 2012). Desejos “são a forma que as necessidades tomam quando moldadas pela cultura e personalidade individual”(Kotler, Armstrong et al., 2017).

Por exemplo, a fome gera uma necessidade de se alimentar, que pode se configurar como o desejo de um hambúrguer para um americano, ou um prato feito de arroz, feijão, bife e batata frita para um brasileiro.

Da mesma forma, dada uma necessidade de um tipo de software, uma organização pode desejar características específicas adicionais, como ser feita de código aberto.

As necessidades podem ser (Kotler e Keller, 2012):

- **declaradas**, que são ditas pelo cliente, mas não são necessariamente verdade;
- **reais**, que são o que o cliente realmente precisa;
- **não declaradas**, que são o que o cliente espera receber;
- **de algo mais (delight)**, que são o que o cliente gostaria, e
- **secretas**, que são como o cliente deseja ser visto pela sociedade.

Kotler e Keller (2012) ainda lembram que responder apenas a necessidade declarada pode não ser o bastante, e isso é verdade especialmente **em projetos de software, onde muitas vezes o cliente nem mesmo sabe suas necessidades reais**.

Uma **demand**a é um “desejo por um determinado produto, suportado por uma capacidade de pagar”(Kotler e Keller, 2012). Muitos querem lagosta no almoço, mas só podem pagar por algo mais barato.

Em software, a questão financeira é importante. É possível cotar um projeto que descrito em um parágrafo como algo entre 30 mil reais e quatro milhões de reais, dependendo da extensão do produto e da capacidade de operação necessária, como servidores e previsão de milhões de usuários.

Uma demanda pode ter oito estados (Kotler e Keller, 2012) em relação a um, ou mais, produtos ou serviços:

1. **demand negativa**, quando os consumidores não gostam do produto e podem até pagar para evitá-lo;
2. **demand não existente**, quando os consumidores não sabem que o produto existe

2. Valor

- ou não tem interesse no mesmo;
3. **demandas latentes**, quando os consumidores têm uma necessidade grande que não é satisfeita por nenhum produto existente;
 4. **demandas declinantes**, quando os consumidores passam a comprar menos ou deixam de comprar um produto;
 5. **demandas irregulares**, quando existe uma variação na compra do produto, em pequena escala de tempo (horas) ou grandes (estações do ano, meses);
 6. **demandas plenas**, onde os consumidores estão comprando todos os produtos postos no mercado;
 7. **demandas excessivas**, onde os consumidores gostariam de comprar mais produtos do que podem ser fornecidos, e
 8. **demandas indesejadas**, onde os consumidores são atraídos por produtos com consequências sociais indesejadas.

Compreendendo essa sequência que começa com a necessidade, passa para o desejo e chega à demanda, é possível então falar de valor.

Para o marketing, o valor é a “a somatória dos benefícios tangíveis e intangíveis proporcionados pelo produto subtraída da somatória dos custos financeiros e emocionais envolvidos na aquisição desse produto”(Kotler e Keller, 2013). O valor é então uma “combinação de qualidade, serviço e preço”(Kotler e Keller, 2013). As percepções de valor “aumentam com a qualidade e o serviço, mas diminuem com o preço”(Kotler e Keller, 2013).

Um dos objetivos do marketing é aumentar esse valor, gerando satisfação no cliente. A **satisfação** é então um “julgamento comparativo de uma pessoa sobre o desempenho percebido de um produto em relação às expectativas”(Kotler e Keller, 2013).

2.3.1. Os Elementos do Valor - B2C

Almquist, Senior e Bloch (2016) identificaram 30 elementos que fornecem valor para **consumidores**, i.e., pessoas, divididos em uma pirâmide¹ de quatro camadas². Essa pirâmide é apresentada a seguir em forma de lista.

- mais alta*
- **Impacto social**: auto-transcendência (ajudar outros ou a sociedade de forma mais ampla).
 - **Mudança de vida**: fornecer esperança, auto-atualização, motivação, herança para as próximas gerações, afiliação/pertencimento.
 - **Emocional**: reduzir ansiedade, recompensa pessoal, nostalgia, design/estética, representação de status ou aspirações, bem estar, valor terapêutico, diversão/entretenimento, aumentar a atração pessoal, fornecer acesso a outros itens de valor.
- mais baixa*
- **Funcional**: poupar tempo, simplificar, fazer dinheiro, reduzir risco, organizar, integrar aspectos diferentes da vida, conectar com outras pessoas, reduzir esforço,

¹A pirâmide é uma construção usada para passar a ideia que os níveis inferiores são essenciais para os níveis superiores poderem ser alcançados

²Uma versão interativa pode ser acessada em <https://media.bain.com/elements-of-value/>, link válido em 9/2/2020

2.3. Valor em Marketing

evitar problemas, reduzir custo, qualidade, variedade de escolha, apelo sensorial, informar.



Figura 2.1.: Elementos do Valor B2C. Fonte:Almquist, Senior e Bloch, 2016

Essa pirâmide pode ser comparada com a pirâmide de Maslow, apresentada na Figura 2.2, que mostra também as necessidades das pessoas em camadas cada vez mais abstratas. Sua aplicação é apropriada ao analisar valor em software que vai ser vendido para pessoas.

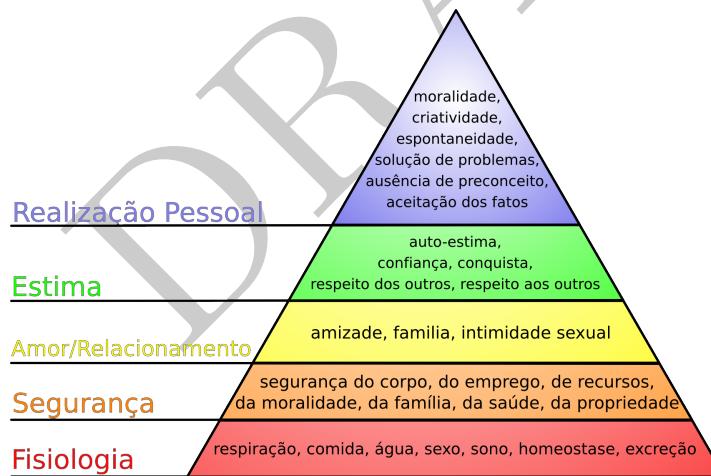


Figura 2.2.: Hierarquia das Necessidades de Maslow. Fonte: Wikipedia Commons por Felipe Sanchez (CC-BY-SA 3.0) e J. Finkelstein (GFDL)

Os elementos podem ter importância diferente por pessoas e por indústria. Por exemplo, os elementos mais importantes para consumidores de *smartphones* são, em ordem decrescente (Almquist, Senior e Bloch, 2016): qualidade, reduzir esforço, variedade

2. Valor

de escolha, organizar e conectar com outras pessoas³.

2.3.2. Os Elementos do Valor - B2B

Mais tarde, Almquist, Cleghorn e Sherer (2018) desenvolveram modelo semelhante para negócios B2B, obtendo uma pirâmide um pouco mais complicada, com 40 elementos, onde há sub-níveis, representada na Figura 2.3 e na lista a seguir⁴. A importância de cada elemento novamente varia com a indústria. Essa segunda pirâmide é aplicável a software que vai ser vendido a organizações, porém leva em conta que as vendas são feitas para pessoas que participam das organizações. Os compromissos básicos são exigências para o vendedor.

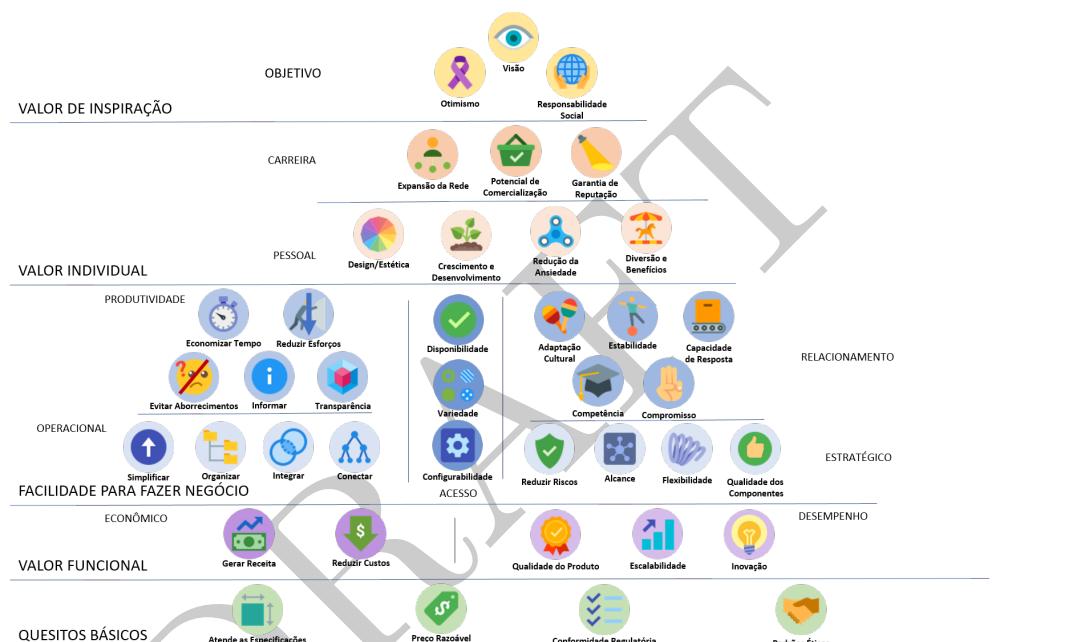


Figura 2.3.: Elementos do Valor B2B. Fonte: Almquist, Cleghorn e Sherer, 2018

mais alta • Valor inspiracional

- objetivo:
 - ◊ visão, ajuda o cliente a antecipar a direção do mercado;
 - ◊ esperança, dá aos compradores e usuários esperanças no futuro da empresa, e
 - ◊ responsabilidade social, ajuda o cliente a ser mais responsável socialmente.

• Valor individual:

- carreira:

³Apesar de ser a missão original do *smartphone*, seu valor como conexão é só o quinto mais valorizado.

⁴Uma versão interativa pode ser acessada em <https://media.bain.com/b2b-eov/index.html>, link válido em 9/2/2020.

- ◊ expansão da rede (*network*), ajuda os usuários e colegas a expandir a rede profissional;
- ◊ *marketability*, faz clientes e colegas mais *marketables* em seu campo, e
- ◊ garantia de reputação, não atrapalha e pode aumentar a reputação do comprador no trabalho.
- pessoal:
 - ◊ design e estética, fornece produtos e serviços que são estéticamente agradáveis;
 - ◊ crescimento e desenvolvimento, ajuda usuários e colegas a crescer pessoalmente;
 - ◊ redução da ansiedade, ajuda compradores e outros na organização a se sentir mais seguros, e
 - ◊ diversão e vantagens, é agradável de interagir com ou dá recompensa de alguma forma.
- **Valor da facilidade de fazer negócios:**
 - produtividade:
 - ◊ poupar tempo;
 - ◊ reduzir esforço;
 - ◊ diminuir problemas;
 - ◊ informação, e
 - ◊ transparência, fornece uma visão clara da organização do comprador.
 - operacional:
 - ◊ organização;
 - ◊ simplificação, reduz complexidade;
 - ◊ conexão, conecta organização e usuários com outros interna e externamente, e
 - ◊ integração, ajuda o comprador a integrar diversas facetas do negócio.
 - acesso:
 - ◊ disponibilidade, garante que o bem ou serviço está disponível quando e onde necessário;
 - ◊ variedade, fornece uma variedade para escolha, e
 - ◊ configurabilidade, permite configurar o bem ou serviço de acordo com as necessidades do comprador.
 - relacionamento:
 - ◊ responsividade, responde rápida e profissionalmente as necessidades da organização;
 - ◊ perícia, fornece *know-how* para o mercado ou indústria relevante;
 - ◊ compromisso, mostra que está compromissado com o sucesso do comprador;
 - ◊ estabilidade, é uma empresa(bem ou produto) estável no futuro previsível, e
 - ◊ ajuste cultural, se encaixa na cultura do comprador.
 - estratégico:
 - ◊ redução de risco, protegendo o comprador;

2. Valor

- ◊ alcance, permite o comprador operar em mais locais ou segmentos do mercado;
- ◊ flexibilidade, vai além dos bens ou padrões comuns para permitir customização, e
- ◊ qualidade de componentes, melhora a qualidade percebida dos produtos e serviços do comprador.

- **Valor funcional:**

- econômico:
 - ◊ redução de custos, e
 - ◊ aumento de receitas.
- desempenho:
 - ◊ qualidade do produto;
 - ◊ escalabilidade, e
 - ◊ inovação.

mais baixa ● **Compromissos básicos (*table stakes*)⁵**

- satisfazer especificações;
- preço aceitável;
- conformidade regulatória, e
- padrões éticos.

Ambas as pirâmides tentam ser exaustivas, mas foram criadas a partir de pesquisas de campo, o que quer dizer que podem existir outros elementos não detectados, porém isso não é esperado em geral.

Além disso, elas devem ser usadas em função dos elementos mais desejados em uma indústria ou cliente específico. Por exemplo, investigando a contribuição proporcional dos elementos para os clientes de infraestrutura de TI, com uma soma de 100%, (Almquist, Cleghorn e Sherer, 2018) encontraram que os três principais elementos são: qualidade do produto (7,8%), perícia (6,1%) e responsividade (5,5%). Poupar tempo, por exemplo, era décimo elemento na ordem e com proporção de apenas 3,0%. Na prática, a importância de cada elemento pode ser levantada com cada cliente.

2.3.3. Outras Técnicas

Tendo em vista que a finalidade do Marketing é engajar e gerenciar relações lucrativas com os compradores, é razoável que parte do trabalho de Marketing seja identificar o que é valor para o comprador, de forma que a organização possa atendê-lo com qualidade. Assim, muitas técnicas de Marketing se dedicam a isso e está além deste livro cobrir todas.

Outras técnicas interessantes provenientes do Marketing são o QFD (Franceschini, 2016), adequada tanto a software sob encomenda quanto a produtos a serem lançados no

⁵No linguajar de negócios em inglês, *table stakes* é o mínimo que você deve fazer para um mercado ou negócio específico.

mercado, e a Estratégia do Oceano Azul (Kim e Mauborgne, 2005), adequada apenas a novos produtos a serem lançados no mercado.

2.4. Valor Financeiro de Projetos

Apesar de não ser o objetivo principal desse capítulo, é interessante observar as formas de calcular o valor financeiro de um projeto, ou subprojeto.

Antes de tratar das estimativas financeiras usadas em projetos, é importante conhecer os seguintes conceitos, como definidos por Dal Zot e Castro (2015):

- **principal** (P) é o capital inicial de um empréstimo ou uma aplicação financeira, também conhecido como **valor presente** (VP), **valor atual** (VA), **valor descon-tado**, ou por seu nome em inglês, ***present value*** (PV);
- **juro** (J) é a **remuneração** do capital emprestado, da parte de quem paga é uma despesa ou custo financeiro, da parte de quem receber é um rendimento ou renda financeira;
- **montante** (S) é o **saldo**, ou **valor futuro** (VF), ou ***future value*** (FV), de um empréstimo ou de uma aplicação financeira. Pode ser também chamado de **valor de resgate**, ver equação 2.1.
- **prazo** (n) é o período de tempo que dura o empréstimo ou a aplicação financeira, podendo ser medido em dias, meses, anos...
- **prestaçāo ou pagamento** (P,PGTO,PMT) se refere ao valor de pagamentos quando feitos em um número maior do que 1.
- **taxa de juros** (i) , ou simplesmente **taxa**, é o quociente entre juros e o principal, ver equação 2.2;

$$S = P + J \quad (2.1)$$

$$i = \frac{J}{P} \quad (2.2)$$

Analizando essas definições podemos detectar uma propriedade importante do dinheiro e que é a base da Matemática Financeira: o dinheiro muda de valor ao longo do tempo. Normalmente, como os cenários de inflação são muito mais comuns que os de deflação, R%100,00 hoje compram mais que comprarão daqui a um ano e menos do que compravam um ano atrás. Por meio de cálculos como o do Valor Presente, como veremos no caso do Valor Presente Líquido, podemos comparar opções de investimentos ou empréstimos.

Calculadores financeiros, como a HP-12C e o Excel permitem calcular facilmente o Valor Presente e o Valor Futuro com pagamentos, ou retiradas, fixas, em um período determinado, ao com uma taxa de juros fixas.

Por exemplo, se você tiver R\$10.000,00 por mês e puder aplicar em um projeto, deve

2. Valor

procurar uma boa alternativa de investimento para poder saber se, financeiramente, vale a pena usar o dinheiro no projeto, ou se é melhor usar esse investimento. Com a fórmula do Valor Futuro (VF em português ou FV em inglês) esse cálculo pode ser facilmente feito no Excel, como mostrado na Figura 2.4.

Investimento Alternativo	
Taxa de Juros	1,00%
Período	12
Pagamento	R\$ 10.000,00
Valor Presente	R\$ 0,00
Valor Futuro	R\$ 126.825,03

(a) Valores e formatação

Investimento Alternativo	
Taxa de Juros	0,01
Períodos	12
Pagamento	10000
Valor Presente	0
Valor Futuro	=FV(Taxa_de_Juros;Períodos;Pagamento;Valor_Presente;0)*-1

(b) Fórmulas

Figura 2.4.: Cálculo do valor futuro de um investimento de R\$10.000,00 por mês com uma taxa de juros de 1%.

Existem várias práticas que são utilizadas para calcular o valor financeiro de um projeto, entre elas (Satpathy et al., 2016):

- Custo Total de Propriedade (TCO - *Total Cost of Ownership*), que calcula todo custo de um serviço ou produto, incluindo a realização e a o uso do mesmo;
- Retorno do Investimento (ROI - *Return on Investment*), o método mais usado que é basicamente o lucro sobre o custo;
- Valor Presente Líquido - VPL (NPV - *Net Peesent Value*), onde se calcula a diferença entre a receita do projeto e seus custo ao longo do tempo, corrigido para o tempo presente, e
- Taxa Interna de Retorno - TIR (IRR - *Internal Rate of Return*) é a taxa de desconto de um investimento que torna o NPV zero para todos os fluxos de caixa de um projeto, não podendo ser calculado analiticamente, quanto maior for, mais desejável é o projeto (Hayes, 2019).

Devemos ter cuidado com projetos com resultados subjetivos, que não podem ser medidos diretamente pelo valor financeiro. Essa questão não será discutida nesse livro.

Tabela 2.1.: Valores usados para calcular o TCO de uma impressora.

Impressora	Laser A	Ink A	Laser B
Preço	R\$ 880,00	R\$ 400,00	R\$ 2.100,00
Preço Toner	R\$ 480,00	R\$ 105,00	R\$ 539,00
Páginas Toner	1000	480	12.000

2.4.1. Custo Total de Propriedade

Quando se analisa o custo de um sistema é normal falar de Custo Total de Propriedade, conhecido pela sigla em inglês TCO (Total Cost of Ownership). O TCO é o valor presente de todos os custos a serem feitos durante a vida de um sistema, produto, serviço ou equipamento(Anklesaria, 2008).

Por exemplo, se decidirmos trocar todo o parque computacional de uma empresa que usa Windows para Linux, mesmo que o custo do Linux seja zero, o TCO é bem alto, pois envolve o processo de troca, novos profissionais, treinamento, etc... Outro exemplo comum é o da compra de uma impressora. Seu TCO não envolve apenas o custo da impressora, mas também o custo do material consumível, quando uma certa produção é prevista. Por isso é que grandes empresas compram menos impressoras, porém impressoras maiores e mais caras, para baixar o TCO, já que o custo por impressão delas acaba saindo menor.

Para o software desenvolvido vale o mesmo conceito. Qual seu TCO? Envolve o preço pago pelo software mais tudo que vai ser pago para possibilitar a implantação e utilização do produto (instalação, cursos de treinamento, manutenção mensal, etc...). Espera-se, em uma decisão racional, que o TCO seja menor que o benefício trazido pelo sistema.

Um exemplo típico é a análise a ser feita na compra de uma impressora. Impressoras mais caras normalmente tem o preço por página impressa mais barato, porém só a partir de um ponto de uma quantidade grande de impressões.

A Tabela 2.1 e a Figura 2.5 mostram como isso acontece. Nelas vemos qual a melhor alternativa, considerando apenas o preço da impressora e do tipo de impressão, qual a impressão mais barata. É possível, por exemplo, somar o gasto de energia a essa conta.

Anklesaria (2008) divide os custos em possíveis classes:

- **preço de compra**, e
- custos internos, que se dividem em
 - **custos de aquisição**, como transporte, inspeção, armazenagem;
 - **custos de uso**, como manutenção, garantias, e
 - **custos do fim de vida**, como custos para desmontar um projeto, etc.

2. Valor

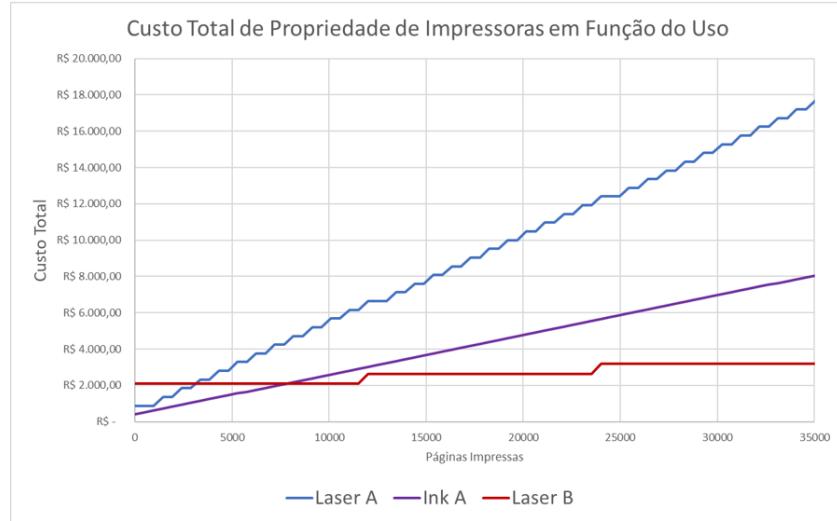


Figura 2.5.: Gráfico do TCO de impressoras em função da quantidade de páginas impressas. Percebe-se que após aproximadamente 3500 páginas é melhor comprar uma impressora a laser mais cara, caso esse fosse o único fator de análise.

2.4.2. Retorno do Investimento

O cálculo do ROI depende de haver uma previsão para a receita corrente R e para o custo C do projeto. Sua fórmula é:

$$ROI = \frac{R - C}{C} \quad (2.3)$$

O cálculo do ROI de um projeto que está previsto custar R\$ 500.000,00 e pretende obter como benefício um valor de R\$ 700.000,00 é:

$$ROI = \frac{700 - 500}{500} = \frac{200}{500} = 40\% \quad (2.4)$$

O ROI pode ser usado para comparações com taxas que o mercado financeiro paga por um investimento do mesmo valor. Assim, só valeria a pena fazer um projeto, financeiramente falando, se ele resultasse em um retorno mais alto do que poderia ser obtido com o investimento em algum título do mercado.

Um cuidado ao analisar o ROI de um projeto é o fato de ser um valor percentual. Nesse caso, um projeto de R\$ 10.000,00 com um ROI de 150% pode parecer mais interessante que um projeto de R\$ 1.000.000,00 com um ROI de 1%, porém o segundo gera muito mais dinheiro que o primeiro.

2.4.3. Valor Presente Líquido

O Valor Presente Líquido é o valor atualizado para o presente de todos os gastos e ganhos de um projeto. Essa atualização é feita com uma taxa de desconto fixa. Isso significa que se você ganhar R\$ 100,00 daqui a um ano, e a inflação for de 5% ao ano, hoje isso tem o valor equivalente ao valor presente que, quando somado de 5%, resultasse em R\$100,00. A conta é $1/1,05$, que é aproximadamente R\$95,24.

A fórmula do VPL é(Kenton, 2019):

$$VPL = \sum_{t=1}^T \frac{C_t}{(1+i)^t} \quad (2.5)$$

onde C_t é o caixa líquido, receitas menos despesas, em um período t e i a taxa de desconto em um investimento alternativo, sendo T o número de períodos.

Esse cálculo é exemplificado na Tabela 2.2, onde usamos o método de calcular o saldo de cada período e atualizá-lo para o presente, com uma taxa fixa de 5%.

Período	Custo	Receita	Caixa	Valor Presente
0	R\$ 50.000,00	R\$ 0,00	-R\$ 50.000,00	-R\$ 50.000,00
1	R\$ 40.000,00	R\$ 52.500,00	R\$ 12.500,00	R\$ 11.904,76
2	R\$ 40.000,00	R\$ 52.000,00	R\$ 12.000,00	R\$ 10.884,35
3	R\$ 50.000,00	R\$ 61.500,00	R\$ 11.500,00	R\$ 9.934,13
4	R\$ 20.000,00	R\$ 31.000,00	R\$ 11.000,00	R\$ 9.049,73
5	R\$ 20.000,00	R\$ 30.500,00	R\$ 10.500,00	R\$ 8.227,02
6	R\$ 20.000,00	R\$ 30.000,00	R\$ 10.000,00	R\$ 7.462,15
7	R\$ 20.000,00	R\$ 29.500,00	R\$ 9.500,00	R\$ 6.751,47
Total				R\$14.213,63

Tabela 2.2.: Tabela de cálculo do Valor Presente Líquido, considerando saldos diferentes em cada período. Foi usada uma taxa de 5% a cada período. A fórmula para cada Valor Presente é $Caixa_i / (1 + Taxa)^{\text{Período}}$

2.4.4. Taxa Interna de Retorno

A Taxa Interna de Retorno é um cálculo um pouco mais complicado, exigindo um processamento de tentativa e erro. Ela indica de quanto deveria ser a taxa de desconto para o VPL ser zero, isto é, o resultado do projeto não ser nem positivo, nem negativo. O processo de cálcula-la é numérico, mas o Excel e calculadoras financeiras fornecem funções para isso.

A fórmula que relaciona o VPL com o TIR é(Hayes, 2019):

$$0 = VPL = \sum_{t=1}^T \frac{C_t}{(1 + TIR)^t} - C_0 \quad (2.6)$$

2. Valor

onde C_0 é o investimento inicial total.

Quanto mais alto for o TIR, melhor será o resultado do projeto, sempre lembrando que como o cálculo é percentual, isso depende do valor do projeto. Uma maneira de entender isso é que você precisaria de uma aplicação melhor que a taxa para o projeto não valer a pena financeiramente.

2.4.5. Priorização Baseada em Valor com o Cliente

A priorização é um processo para “determinar a importância relativa de informações”(IIBA, 2011). Ela permite várias abordagens:

- agrupamento, normalmente em poucas classes pré-determinadas;
- *ranking* ou ordenação;
- orçamento ou *time-boxing*, por meio da alocação de recursos finitos e
- negociação, em busca de um consenso.

Algumas das técnicas que podem ser usadas para criar uma estimativa relativa de valor com clientes e outras partes interessadas são (Satpathy et al., 2016):

- esquemas simples de pontuação, onde cada item do projeto analisado recebe um número entre uma variação pequena, como de 1 a 3 ou 1 a 5, ou uma rótulo como “Alto”, “Médio” ou “Baixo”, criando grupos ordenados;
- prioritização MoSCoW, onde cada item é associado a um conceito do grupo “*Must Have*”, precisa ter, “*Should Have*”, devia ter, “*Could Have*”, podia ter e “*Won't Have*”, não vai ter (IIBA, 2011), criando grupos com significados específicos;
- **dinheiro de brinquedo ou 100-pontos**, onde é dada uma quantidade de dinheiro de brincadeira⁶, como 100 moedas, ou pontos, para o cliente associar aos itens do projeto, algumas vezes submetido a algumas regras, como não ter X itens com o mesmo valor (Leffingwell e Widrig, 1999), criando uma ordem parcial baseada na alocação de recursos;
- **Análise de Kano** (Moorman, 2012), tratada na subseção a seguir;
- ordenação, usualmente feita com cartões com um item cada (J. Robertson e S. Robertson, 1998), mas também realizada em planilhas eletrônicas, possivelmente incluindo um processo em grupo, ou de negociação.

2.4.6. Análise de Kano

O modelo de Kano é um método para avaliar a reação emocional de clientes a características individuais de um produto. Kano detectou cinco respostas emocionais a essas características (Moorman, 2012):

- encantadores ou atrativos (*delighters*), “eu gosto disso”, que trazem satisfação se presentes, mas não trazem insatisfação se não estiverem presentes, normalmente

⁶Facilmente obtida em uma caixa de Banco Imobiliário

- porque são inesperadas e endereçam necessidades não reconhecidas;
- unidimensionais (*satisfiers*), que causam satisfação se estiverem presentes e insatisfação se não estiverem presentes;
 - obrigatórios ou esperados (*dissatisfiers*), que causam insatisfação se não estiverem presentes, mas não causam satisfação se estiverem presentes;
 - indiferentes (indifferent), ou
 - indesejados.

Para classificar em uma dessas cinco respostas emocionais, são feitas duas perguntas ao cliente (Moorman, 2012):

- Como você vai se sentir se a funcionalidade estiver presente?
- Como você vai se sentir se a funcionalidade não estiver presente?

As respostas devem ser em uma escala ordinal:

- eu gosto disso;
- eu espero isso;
- eu sou neutro em relação a isso;
- eu posso tolerar isso, ou
- eu não gosto disso.

Finalmente, a classificação é feita cruzando as respostas na Tabela 2.3.

Tabela 2.3.: Resolvendo a classe de Kano por meio do cruzamento de duas perguntas
(Moorman, 2012)

		Questão Negativa				
Questão Positiva	Gosto	Gosto	Espero	Neutro	Tolero	Não Gosto
	Espero	Conflito!	Atrativo	Atrativo	Atrativo	Unidimensional
	Neutro	Indesejado	Indiferente	Indiferente	Indiferente	Obrigatório
	Tolero	Indesejado	Indiferente	Indiferente	Indiferente	Obrigatório
	Não Gosto	Indesejado	Indesejado	Indesejado	Indesejado	Conflito!

As categorias de Kano não são permanentes, e as características de um produto, ou tipo de produto, vão migrando de atrativas, para unidimensionais, e para obrigatórias, e podem até mesmo se tornar indesejadas. O tamanho do celular, por exemplo, sofreu essa mudança com o tempo.

2.5. Valor em Software

Conhecendo algumas definições na Economia e no Marketing, podemos compreender melhor outras definições que encontramos na Engenharia de Software.

Segundo Erdogan, Favaro e Halling (2006) , **valor** é a “diferença entre benefícios e

2. Valor

custos de um bem, ajustados ao risco, em um certo momento de tempo”.

Ele é dirigido por valores individuais ou coletivos. As partes interessadas esperam obter algum **benefício**, seja ele tangível ou intangível, econômico ou social, monetário ou utilitário, ou ainda estético ou ético (Biffl et al., 2006).

Valor significa esse benefício final, que existe geralmente nos olhos do observado e admite múltiplas caracterizações (Biffl et al., 2006).

O risco, que ainda não tínhamos encontrado em outras definições, é importante, e é parte das técnicas de gestão de projeto modernas considerá-lo. O risco pode ser incluído inclusive indiretamente no custo total previsto de um projeto multiplicando sua probabilidade pelo seu impacto. Isso faz pouco sentido para um risco único, mas já faz sentido para o conjunto total de riscos de um projeto.

Assim, o valor de algo pode ser visto como uma fórmula matemática, de uma forma básica como:

$$\text{benefício} - \text{custos} \quad (2.7)$$

Ou, considerando os riscos:

$$(\text{benefício} - \text{custos}) \times \text{correção por risco} \quad (2.8)$$

A partir dessa interpretação, busca valor é buscar os maiores benefícios a um custo que valha a pena, com risco baixo. Esses benefícios devem ser identificados no início do projeto, e estar alinhados com o planejamento estratégico da empresa.

É importante frisar que o valor deve ser em relação a organização, não ao software propriamente dito. Assim, algumas funcionalidades que poderiam tornar o software muito mais poderoso podem, na prática, não ser úteis, não trazer benefícios, custar caro demais ou ter risco muito alto. E isso depende de cada projeto, ou cada aplicação do software.

Gane e Sarson (1979) introduziram um o acrônimo **IRACIS**, que identifica três formas de agregar valor (Gane e Sarson, 1979; Ruble, 1997):

- Aumentar Faturamento (*Improve Revenue*);
- Evitar Custos (*Avoid Costs*), e
- Melhorar Serviços (*Improve Services*).

2.5.1. Triângulo do Valor

É comum se falar, em projetos de software, que existem três parâmetros, que são qualidade (ou escopo), tempo e custo, formando um triângulo, mas o cliente só pode definir dois. Apesar de ser tratado como um brincadeira, é realmente impossível arbitrar esses três valores de forma independente. Ou seja, a afirmação “eu quero um software com a funcionalidade X em tempo Y e custo Z” não pode ser feita sem conhecimento de causa (BeSeen, 2015; Kerzner, 2017).

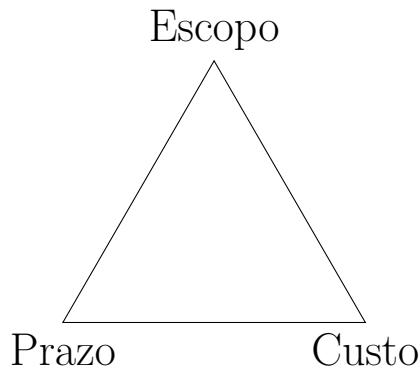


Figura 2.6.: O triângulo do valor para a maioria dos projetos.

A princípio, isso pode ser entendido a partir do fato que Barry W. Boehm et al. (2000) mostraram que há uma relação entre a funcionalidade e qualidade desejada, e o esforço necessário para desenvolver o sistema, o que é óbvio. O que não era tão óbvio é que, além disso, dado um esforço, o tempo necessário para desenvolver o sistema também é determinado. Podem ser feitas certas modificações na previsão estatística, porém se forem feitas no sentido de diminuir o prazo necessário para a entrega do produto, elas implicam em risco maior. O assunto já foi tratado há muito anos por Brooks (1978) em seu livro adequadamente chamado *The Mythical Man-Month*.

Kerzner (2017) trata escopo, tempo e custo como restrições primárias que competem entre si em um projeto tradicional, e ainda apresenta restrições secundárias: reputação e imagem, risco, qualidade e valor. Já em projetos complexos, ainda segundo Kerzner, as restrições primárias passam a ser reputação e imagem, valor e qualidade, enquanto escopo, risco, custo e tempo passam a ser restrições secundárias. Podem aparecer também outros fatores, como segurança e estética, que aparece em parques de diversão, por exemplo.

2.5.2. Valor em Métodos Ágeis

É comum que métodos ágeis, como Scrum (Satpathy et al., 2016), proponham priorizar, de forma contínua, em ciclos curtos, os requisitos baseados no valor de negócio que é entregue aos clientes e usuários. O foco do Scrum, por exemplo, é tornar entregar valor, na forma de software executável, rapidamente, por meio de entregues incrementais a cada ciclo. Não há, porém, uma definição específica do significado da palavra valor, o que leva a necessidade de um entendimento maior do termo. Mesmo assim, cada história do usuário representa um valor a ser entregue ao cliente.

2.6. Conclusão

Não existe uma ideia unificada do que é valor, principalmente quando o conceito é comparado entre áreas distintas.

Como os clientes do analista de sistemas, partes interessadas nos projetos, estão em várias áreas diferentes, mesmo em um só projeto, então podem interpretar valor de formas diferentes também.

Mesmo assim, a principal função do desenvolvimento de software é produzir um software que tenha valor ao usuário, seja esse valor calculado por uma fórmula

Para isso é importante definir o que é valor no início de qualquer projeto, de maneira consensual entre as partes interessadas. Isso é uma prática ágil que deve sempre ser adotada.

A técnica MoSCoW também é muito prática e pode ser associada a qualquer outra. A partir dela pode ser mais fácil fazer uma ordenação de uma grande lista de requisitos, por exemplo.

Já a técnica de Kano é mais difícil de ser usada para projetos sob encomenda, mas é apropriada para o *design* de novos produtos.

O uso dos Elementos de Valor (Almquist, Cleghorn e Sherer, 2018; Almquist, Senior e Bloch, 2016) pode ser uma solução apropriada para o processo de discussão com as partes interessadas em busca de definições mais precisas e completas do verdadeiro valor.

2.7. Exercícios

Exercício 2.7-1: Procure outras definições de valor na Economia ou investigue mais sobre as definições dadas. Discuta a diferença entre essas definições e como elas mostram a evolução do entendimento do que é valor.

Exercício 2.7-2: Busque outras definições de valor, principalmente de outras áreas que não foram citadas neste capítulo.

Exercício 2.7-3: Vamos supor que Alice queira abrir um negócio. Para isso ela precisa de um software que permita registrar suas vendas. Sem esse software, por motivos legais, ela não pode vender. No mercado ela pode encontrar um software básico, que não atende todas as suas necessidades mas atende o requisito legal, por R\$ 100.000,00 por ano. Nesse caso, ela perderia a capacidade de mudar seus preços dinamicamente, o que faria com que ela deixasse de ganhar R\$ 240.000,00 por ano. Porém, a taxa de juros do mercado, se ela investir o dinheiro que usaria para o software novo é de 5% ao ano. Quanto ela pode pagar de aluguel, por ano, pelo software que atende completamente suas necessidades?

Exercício 2.7-4: Vá para o site <http://jogodeanalisedesistemas.xexeo.net/> e visite a Livraria Resolve. Identifique qual pode ser o conceito de valor para o projeto que essa organização deseja.

DRAFT

DRAFT

I keep six honest serving-men
(They taught me all I knew);
Their names are What and Why
and When
And How and Where and Who.

(Rudyard Kipling)

Conteúdo

3.1.	As Perguntas Certas	28
3.2.	Usos do 5W2H	29
3.3.	Onde Está o Valor?	30
3.4.	Várias Perguntas para cada Palavra	31
3.5.	Exercícios	34

Por que 5W2H?

A ideia de usar a técnica 5W2H é mostrar que toda a Análise de Sistemas busca, na verdade, descobrir as perguntas e as respostas corretas em relação a um contexto de desenvolvimento, sendo que essas perguntas são normalmente dentro da forma do 5W2H ou variações. Também objetiva garantir, em cada capítulo, que todas as perguntas corretas tenham sido feitas.

Este capítulo apresenta uma técnica informal, porém extremamente útil, de descrição de fatos, coleta de informações, análise crítica análise de situação, ou registro de decisão

3. 5W2H

conhecida como **5W2H**.

Apesar de informal, ela é usada em vários modelos formais, tanto de forma explícita como implícita. Esta regra será usada em todo este livro, também implícita ou explicitamente, para ajudar a explicar os diversos modelos usados em análise de sistemas.

3.1. As Perguntas Certas

Todo trabalho que vai ser feito precisa ser definido de alguma forma. Para isso é importante que se façam as perguntas certas, e que essas perguntas levem a uma definição precisa, ou seja, não ambígua, do que vai ser feito e quais as condições em que será aceito. Também, quando se descreve algo, é necessário que as informações dadas sobre o que está sendo descrito sejam suficientemente completas para serem úteis. Isso é verdade para a criação de notícias, histórias, acordos de ações a serem tomadas no final das reuniões e, também, para a análise de sistemas, onde estamos na prática descrevendo um sistema de forma a permitir sua implmentação.

Um quantidade muito grande de textos, desde daqueles que tentam ensinar crianças a contar histórias até aqueles que ajudam a profissionais a determinar uma arquitetura de informação empresarial, indica que é necessário responder as seguintes perguntas, conhecidas como 5W2H em inglês:

- Por que? (*Why*)
- O que? (*What*)
- Quem? (*Who*)
- Quando? (*When*)
- Onde? (*Where*)
- Como? (*How*)
- Por quanto? (*How Much*)

Existem variações. Uma lista mais básica só possui 6, exclui o “Por quanto?” (5WH). Listas maiores incluem termos como “ganhos” ou *Wins*, “Quantos”, e variações de “Quem” que são usadas em inglês, como “Whose”, em português “De quem?”, e “Whom”, que é uma versão passiva. Na prática, essa lista de 7 é a mais conhecida na área de negócios e a versão 5WH, em áreas como jornalismo e educação .

A fonte original destas perguntas é o estudo da Ética, e mais tarde da Retórica, na Grécia antiga, e foram chamadas originalmente de **sete circunstâncias** por Aristóteles. Em latim as perguntas eram: *quis, quid, quando, ubi, cur, quem ad modum e quibus adminiculis*. A tradução seria: quem, o que, quando, onde, por que, como e por que meios (Fedotov, 2019; Sloan, 2010).

Por que essas perguntas são boas? Primeiro, quando usadas realmente como perguntas em uma entrevista, são perguntas abertas, que não incluem a resposta dentro delas e obrigam quem responde a responder sem usar apenas uma resposta do tipo “sim” ou “não”. Também são perguntas que, a princípio, não induzem uma resposta. Por

exemplo, perguntamos “Quem merece ganhar o prêmio?” em vez de “Alice merece ganhar o prêmio?”, deixando para o entrevistado decidir sem ser influenciado pelo nome de Alice na pergunta. Além disso, as perguntas, reunidas, mostram um grande quadro informativo sobre um tema.

3.2. Usos do 5W2H

Um das áreas de utilização, por exemplo, é o jornalismo. Se uma notícia vai ser dada, é importante saber responder, no texto da notícia, todas essas perguntas, ou a notícia estará incompleta.

Um exemplo é o seguinte primeiro parágrafo de uma notícia:

BRASÍLIA Um dia depois de vetar a proibição para que as companhias aéreas cobrem pelo despacho de bagagens, o presidente Jair Bolsonaro justificou a decisão, afirmado, nesta terça-feira, que “empresas menores alegavam que seria um empecilho” à operação no país e disse que não pretende enviar outra medida ao Congresso Nacional para restringir a cobrança apenas para as chamadas “low cost”, de baixo custo. (Extra, 2019)

As perguntas 5W2H para essa notícia estão respondidas na Tabela 3.1.

Pergunta	Resposta
Onde	Brasília
Quando	Um dia depois de vetar nesta terça-feira
Quem	Jair Bolsonaro
O que	vetar a proibição para que as companhias aéreas cobrem pelo despacho de bagagens
Por que	empresas menores alegavam que seria um empecilho à operação no país

Tabela 3.1.: Perguntas e respostas a partir da notícia.

Outro uso é para registrar decisões de uma região. Para cada item da reunião deve ser registrado:

- O que deve ser feito?
- Quem é o responsável?
- Quando deve ser feito/entregue?
- Onde deve ser feito/entregue?
- Por que deve ser feito?
- Como vai ser feito?
- Quanto vai ser gasto?

Na análise de sistemas, ou na análise de requisitos, as perguntas mais respondidas são ligadas a **quem** deseja **o que** e **por que** deseja. Perguntas adicionais, feitas em algumas

3. 5W2H

práticas ou que tem que ser respondidas nas fases iniciais de um projeto de software são, por exemplo: para **quando** deseja, **por quanto** deseja, **como** deseja que seja feito e **onde** deseja que funcione. Várias outras perguntas podem ser feitas e que também vão influenciar o projeto, como **quem** paga, **quem** vai ser afetado, até **quando** vai funcionar, etc.

3.2.1. O Framework de Zachman

O *Framework de Zachman* é uma metodologia de descrição de uma arquitetura de sistemas de informação baseado na técnica 5WH, i.e., não incluindo custos (Sowa e Zachman, 1992; Zachman, 1987). Várias das técnicas que usaremos nesse livro geram artefatos que pertencem ao *Framework de Zachman*. Ele é um dos principais exemplos do uso direto da técnica 5WH em Engenharia de Software.

Sua principal caracterização é o quadro da Figura 3.1.

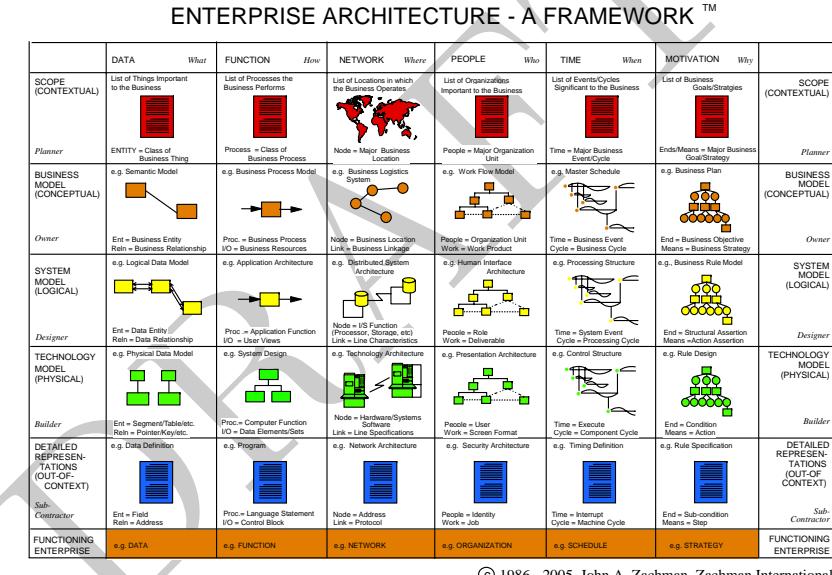


Figura 3.1.: O Framework de Zachman fornece uma visão da arquitetura empresarial a partir de 6 dimensões definidas pelo 5WH. Reprinted with permission by John A. Zachman, Zachman International.

3.3. Onde Está o Valor?

Tendo em vista que a proposta é **desenvolver software tendo em vista entregar valor ao cliente**, onde está o valor no método 5W2H?

3.4. Várias Perguntas para cada Palavra

Normalmente a resposta é razoavelmente simples: o valor está normalmente caracterizado, de alguma forma na resposta a pergunta “por que?” feita a um requisito, pois a justificativa de um requisito indica o motivo pelo qual ele é importante.

É possível que, para descobrir o verdadeiro valor, seja necessário fazer essa pergunta mais de uma vez, isto é, perguntar por que a resposta para a pergunta “por que” foi dada, e então perguntar de novo até que se alcance uma resposta que realmente caracterize o valor. Essa técnica é conhecida como **5 Por QuesSerrat** (2017).

Algumas vezes também é possível que a resposta necessite de perguntas do tipo *O que*, ampliando o conceito tanto do “por que” quanto do “o que”. Por exemplo, se em algum momento a resposta for “preciso dessa função por causa de uma lei”, a próxima pergunta poderia ser “O que (ou qual) é essa lei?”.

Perguntar por que sucessivamente ou perguntar o que sucessivamente, ou misturar as duas sequências, é uma prática chamada *squeeze and stretch Higgins* (1994).

3.4. Várias Perguntas para cada Palavra

Algumas perguntas podem ser feitas mais de uma vez. Por exemplo, “Quem?” podem se referir ao agente ou ao paciente de uma ação, ou a um observador. A pergunta “Quando?” pode se referir ao quando começou ou quando acabou o evento, ou quando ele foi planejado.

Cabe ao profissional entender como usar essas possibilidades ao seu favor. As subseções a seguir mostram algumas opções que podem, e na maioria das vezes devem, ser investigadas em um projeto, sem esgotar as possibilidades.

Além disso, como as práticas de desenvolvimento de software tratada nesse livro podem ser usadas desde o início do projeto, para servir de especificação do mesmo, várias perguntas podem ser feitas não só sobre o produto, mas também sobre o projeto.

3.4.1. Perguntas sobre o que

Descobrir o que é a tarefa principal da análise. Basicamente todo este livro fala sobre isso.

Normalmente as perguntas do tipo “o quê” caracterizam mais o produto da análise que o projeto propriamente dito.

- O que será feito?
- O que não será feito?
- O que é necessário?
- O que é suficiente?
- O que é insuficiente?

3.4.2. Perguntas sobre quem

As perguntas sobre “quem” são muito importantes em várias fases do projeto. No início porque precisamos identificar as partes interessadas (Capítulo 12). Depois porque cada funcionalidade do sistema vai atender a algum usuário.

Uma das técnicas muito usadas em gerência de projetos é, para cada atividade, classificar dos as partes interessadas de acordo com suas responsabilidades, criando uma matriz conhecida como Matriz RACI (PMI, 2017). Essa matriz contém pacotes de trabalho ou atividades nas linhas e partes interessadas nas colunas, sendo que cada célula deve indicar se a parte interessada **realiza**, **aprova**, **é consultada** ou **é informada**.

- Quem deve realizar?
- Quem deve aprovar?
- Quem deve ser consultado?
- Quem deve ser informado?
- Quem deve estar envolvido?
- Quem pode ser afetado?
- Quem pode se beneficiar?
- Quem pode ser prejudicado?
- Quem deve verificar?
- Quem deve validar?
- Quem deve homologar?
- Quem deve pagar?
- Quem deve usar?
- Quem deve operar?
- Quem deve manter?

3.4.3. Perguntas sobre quando

Geralmente as perguntas do tipo “quando” tem relação com prazos. Elas podem se refiliar tanto ao negócio quanto ao projeto, ou seja, existem tempos do negócio, como o prazo para entregar um relatório, e tempos do projeto, como o prazo para entregar uma função no software.

- Quando deve ficar pronto?
- Quando deve iniciar a execução?
- Quando vai acontecer?
- Quando será tarde demais?
- Quando os recursos estarão disponíveis?
- Quando devem ser as entregas?
- Quando deve ser aprovado?

3.4.4. Perguntas sobre onde

Novamente esse tipo de pergunta tem relação com diferentes assuntos. Um é o local do mundo real, por exemplo, um software pode ser usado em uma fábrica, mas não no escritório de uma organização. Outro é em relação a infraestrutura de TI, o software pode rodar na nuvem, em um servidor da empresa ou em celulares. Finalmente, em relação ao projeto, um software pode ser feito em um laboratório ou por uma equipe espalhada pelo mundo.

- Onde será feito?
- Onde será testado?
- Onde será homologado?
- Onde será instalado?
- Onde será utilizado?
- Onde será mantido?

3.4.5. Perguntas sobre como

Geralmente a fase de Análise, objetivo deste livro, não discute muito o “como”, sendo mais preocupada com o “o quê”, sendo o “como” mais tratado na fase de projeto. Mesmo assim, várias perguntas desse tipo já tem que ser respondidas pela análise, tanto em relação ao produto como ao projeto de construí-lo.

- Como será feito?
- Como será utilizado?
- Como será comercializado?
- Como será corrigido no futuro?
- Como sua operação será gerenciada?

3.4.6. Perguntas sobre por que

Apesar de parecer que os porquês não influenciam a execução do projeto, a verdade é que considerá-los traz uma mudança importante na visão que o analista vai ter da solução. Os porquês são na verdade a justificativa, definem valor e são a orientação do caminho a ser seguido a cada decisão.

- Por que aconteceu?
- Por que será feito?
- Por que foi pedido?
- Por que foi aprovado?
- Por que será aprovado?
- Por que existem resistências?
- Por que alguém é a favor?
- Por que alguém é contra?

3. 5W2H

- Por que a funcionalidade é desejada?
- Por que o requisito foi pedido?

3.4.7. Perguntas sobre quanto

Não existe projeto sem orçamento. Mesmo que você vá trabalhar para si, ou de graça, há um custo em horas e outros recursos necessários. Por isso essas perguntas são normalmente respondidas quando todas as outras já foram respondidas, ou, ao contrário, são respondidas antes porque vão servir de limites ao projeto, limitando as respostas aceitáveis para as outras perguntas.

- Quanto custará fazer?
- Quanto custaria não fazer?
- Quanto será obtido em benefício quando feito?
- Quanto será necessário do recurso X¹?
- Quanto tempo será necessário?

3.5. Exercícios

Exercício 3.5-1: Vá para o site <http://jogodeanalisedesistemas.xexeo.net/> e visite a Livraria Resolve. A partir da visita, analise as respostas dadas às perguntas 5W2H e veja se elas são adequadas. Escreva uma lista de perguntas adicionais, sempre no formato 5W2H a serem feitas para cada personagem.

¹pessoas, funções no projeto, equipamentos, etc.

4

Modelos e Abstrações

The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise

(Edsger Dijkstra)

Conteúdo

4.1.	Modelos	36
4.2.	Tipos de Abstrações	39
4.3.	Trabalhando com as abstrações	44
4.4.	Exercícios	44

Por que modelos e abstrações?

Todas as práticas deste livro realização alguma forma de abstração do problema ou solução que fornece modelos aos desenvolvedores e outras partes interessadas.

Este capítulo faz uma introdução aos temas correlatos **modelo** e **abstração**, que são as bases de todos os métodos de Análise de Sistema.

Todas as técnicas estudadas nesse livro implicam na criação de um modelo, seja ele do domínio da aplicação, do negócio, do problema, ou do sistema que será implementado, inclusive modelos que fazem a relação entre modelos distintos.

Apesar de usarmos estes termos de forma coloquial, é importante responder às pergun-

4. Modelos e Abstrações

tas: o que é um modelo? O que é uma abstração?

Um **modelo** é uma representação de algum objeto, conceito, conjunto de conceitos ou realidade. Modelos são criados para que nós possamos estudar, normalmente segundo algum aspecto escolhido, o objeto modelado. Na grande maioria das vezes, um modelo é uma versão simplificada, ou seja, abstrata do objeto modelado. Essa versão simplificada permite uma comunicação com foco em alguns conceitos, evitando o que é irrelevante(Ed Yourdon, 2006).

Abstração é o processo mental de separar um ou mais elementos de uma totalidade complexa de forma a facilitar a sua compreensão por meio de um modelo, eliminando (ou subtraindo) o resto. Segundo Rosen (2018), a “abstração é um processo mental distinto em que novas ideias ou conceitos são formadas pela consideração de vários objetos ou ideias e omitindo características que os distingue.”

A Tabela 4.1 mostra uma sequência de imagens, a partir de uma foto, que ilustram o conceito de abstração. Enquanto a primeira foto¹ é muito detalhada, as seguintes continuam de alguma maneira representando o conceito de mulher, porém de uma forma cada vez mais abstrata, até que se chega em um ícone que não tem nenhuma relação com a imagem de uma mulher, o espelho de vênus, reconhecido internacionalmente como símbolo do sexo feminino.

Tabela 4.1.: Sequência de imagens que mostram abstrações crescentes de uma imagem inicial que representa uma mulher. Foto por Michael Jatremski.



4.1. Modelos

No nosso dia a dia nos deparamos constantemente com modelos. Um mapa é um modelo do território que descreve. Uma maquete de um prédio é um modelo que nos permite ver o prédio a ser construído como um objeto tridimensional. Uma receita de bolo é um modelo do processo para construir o bolo. Mesmo uma foto de um modelo pode ser vista como outro modelo.

Um modelo deve ser simples o bastante para ser fácil de manipular e, simultaneamente, complexo o suficiente de forma a servir para a solução do problema em questão, de acordo com o ponto de vista desejado. Assim, um mapa para navegação tem informações diferentes de uma mapa para estudo do clima, e um modelo de comportamento do

¹Foto original por Michael Jatremski, <https://openphoto.net/gallery/image/view/5539>

4.1. Modelos

software também tem informações diferentes do modelo da informação que o software armazena.

Quanto mais simples o modelo, maior a abstração feita para produzi-lo, ou seja, mais características são suprimidas do objeto original. Chaitin (2006) chega a dizer que se uma teoria, que é também um modelo, é do mesmo tamanho que o dado que ela explica, então não tem valor nenhum. Ainda segundo o autor, uma teoria só é útil quando é uma compressão dos dados, e compreender é comprimir.

No nosso dia a dia utilizamos continuamente a capacidade humana de abstrair para poder trabalhar com toda a informação que o mundo nos fornece. Voltando ao caso do mapa: ele é um modelo de uma região. Dependendo da informação que queremos, colocamos alguns símbolos e tiramos outros do mapa. Um mapa também não pode ser perfeito, tem que “abstrair” as informações que não são necessárias para a utilização que foi construído. Se não houvesse nenhuma abstração, o mapa teria que ter o mesmo tamanho da cidade.

Podemos usar mapas com diversos graus de detalhe, ou seja, mais ou menos abstratos. Um globo terrestre, por exemplo, é um mapa muito abstrato, geralmente com o objetivo de ensinar noções básicas de geografia. Já uma carta náutica é um mapa muito detalhado, que permite a navios ou barcos menores navegarem de forma segura em uma região. Ainda mais detalhada será a planta de uma casa ou prédio. Os níveis de detalhe são infinitos e são usados de acordo com a necessidade do problema a ser resolvido. A Figura 4.1 exemplifica diferentes mapas, com diferentes níveis de abstração adequados ao seu uso.

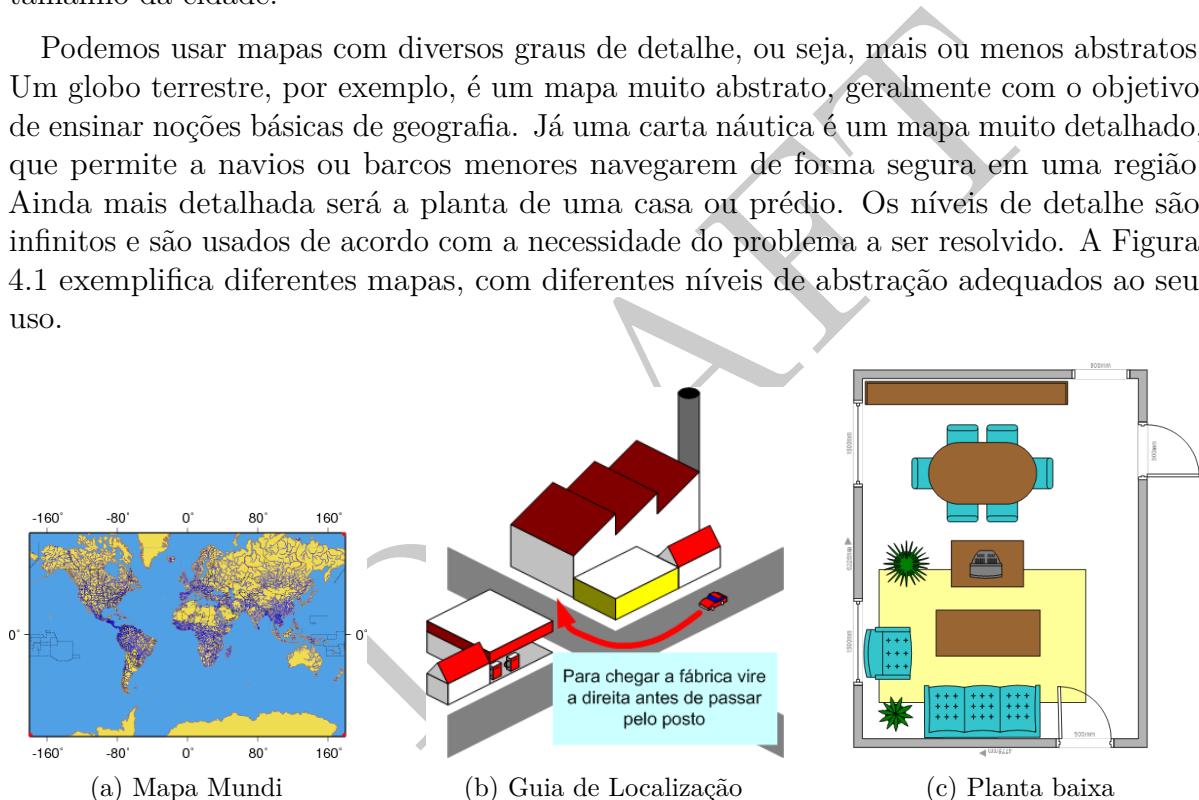


Figura 4.1.: Diferentes tipos de mapas, ou seja, modelos, cada um com um nível diferente de abstração e dedicado a uma utilização distinta.

Um tipo especial de modelo é o protótipo. Um protótipo é um modelo exemplar, no sentido que fornecer um exemplo, onde as simplificações de certo aspecto são muito pequenas ou não existem. Por exemplo, um protótipo de um software pode ter todas as suas telas na versão final, sendo esse o aspecto estudado, mas nenhuma funcionalidade. Em algumas áreas, onde há a necessidade de fabricar algo em série, protótipos são

4. Modelos e Abstrações

um modelo de fabricação, completo em funcionalidade, mas que não foram feitos pelo processo final esperado no momento em que serão produzidos em série.

Finalmente, relativo a aplicação de modelos em nosso problema Ed Yourdon (2006) afirmou que o analista de sistema usa modelos para:

- dar foco as características importantes do sistema e diminuindo a influência de características menos importantes;
- discutir mudanças e correções nos requisitos do usuário a um custo baixo e risco mínimo, e
- verificar que o analista de sistema entende corretamente o ambiente do usuário e o documentou de forma que os outros desenvolvedores possam construir o sistema.

Um típico modelo usado em análise de sistemas, na atualidade, é o Diagrama de Casos de Uso, como o apresentado na Figura 4.2.

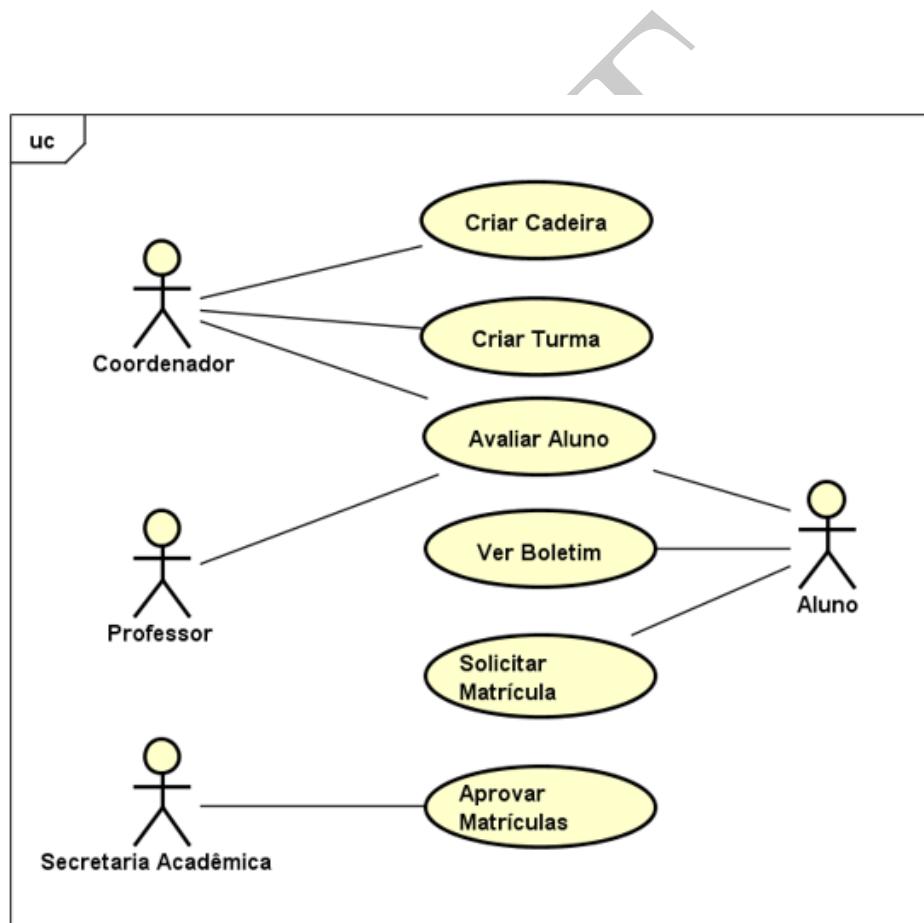


Figura 4.2.: Exemplo de Diagrama de Casos de Uso.

4.2. Tipos de Abstrações

Existem várias formas de abstração praticadas no dia a dia ou em situações especiais. No desenvolvimento de sistemas, utilizamos alguns processos de abstração típicos:

- **ocultação da informação** (ou abstração da caixa-preta, ou encapsulamento);
- **classificação**, que relaciona instâncias e classes;
- **generalização**, ou herança, que relaciona classes;
- **composição**, ou todo-parte;
- **identificação**;
- **simplificação pelo Caso Normal**;
- **foco/inibição**;
- **refinamento sucessivo**, e
- **separação de interesses**.

4.2.1. Ocultação da Informação

Pela abstração de **Ocultação da Informação**, deixamos de nos preocupar com o interior de uma coisa, só prestando atenção a seu exterior observável.

Podemos encontrar esse exemplo facilmente em muitas coisas do mundo real. Quantas pessoas, por exemplo, sabem como funciona um telefone celular? Poucas, certamente. Porém quase todas sabem usá-lo, porque “abstraem” o seu funcionamento interno (isso é, não pensam nisso) e colocam em foco apenas o comportamento externo.

Por isso chamamos também essa abstração de **abstração de caixa-preta**. O conceito inverso (ou seja, abrir a caixa) é chamado de **caixa-branca**.

Na área de Computação é muito comum ocultar informação para facilitar a compreensão. Por exemplo, em programação, se você usa uma biblioteca de funções, não está interessado em saber como uma função é realizada, mas sim apenas nos parâmetros de entrada e em como será o resultado da função. A função funciona como uma caixa-preta.

4.2.2. Classificação

A **classificação** é um mecanismo básico do raciocínio humano. Talvez seja um dos que mais nos habilita a tratar de toda informação que recebemos diariamente.

No processo de classificação eliminamos parte da individualidade do objeto ou sistema analisado e o consideramos como um exemplar de uma **classe**. Quando fazemos isso, aceitamos que esse objeto, agora uma **instância** da classe, divide com todas as outras instâncias da classe um conjunto de características.

Segundo Parrochia (2020), **classificar** é “a operação de distribuir objetos em classes ou grupos, que são, em geral, menos numerosos que eles.”. Essa operação simplifica o

4. Modelos e Abstrações

mundo, tendo tem vantagens principais Parrochia (2020):

- substitui um multiplicidade caótica e confusa por um ordem racional e regular;
- permite trabalhar com um número reduzido de classes de equivalências em vez de trabalhar com os elementos, e
- detectam uma simetria nas classes que diminui a complexidade do problema e simplifica o mundo.

Na classificação o que estamos fazendo é imaginar uma ideia única que descreve, de forma abstrata, todos os objetos de uma conjunto. Ao eliminar a necessidade de tratar cada objeto de forma única, simplificamos o problema em questão.

A classificação é uma relação entre **instâncias** e **classes**. É possível associar uma classe com um conjunto de elementos que possuem algumas de propriedades em comum, como todos os alunos de uma universidade ou todos os animais mamíferos.

O processo reverso da classificação é a **instanciação**. O conjunto de todas as instâncias de uma classe é a extensão dessa classe. Uma classe pode ser descrita por sua extensão, ou por sua intenção, que é uma descrição que permita identificar um conjunto de entidades(Parrochia, 2020).

Exemplos típicos de classificação aparecem na Tabela 4.2.

Tabela 4.2.: Exemplos de Classificação

Instâncias	Classe
Flamengo, Fluminense, São Paulo	Times de Futebol
Brasil, Estados Unidos	País
Pelé, Zidane, Messi	Jogador de Futebol

As classificações mais famosas certamente são a Classificação Científica, que classifica todos os seres vivos, e a Classificação Decimal Universal (CDU), que classifica documentos e é utilizada nas bibliotecas.

É importante notar que na vida real um objeto pode pertencer a várias classes. Uma pessoa pode ser um aluno, um professor, um policial, etc... Normalmente, em modelos teóricos como os que vamos usar, tentamos com que um objeto pertença, diretamente, a só uma classe, de modo a facilitar a manipulação do modelo.

Nada impede que uma classe seja uma instância de outra classe, que pode ser chamada de uma **meta-classe**, de uma outra hierarquia. Em Smalltalk-80, por exemplo, toda classe é uma instância da classe *Class*. Também podemos analisar que **Homo** é uma classe que é uma instância da classe *Gênero*, que reúne todos os gêneros. Na prática a classe “Gênero” e as outras similares, como “Espécie” são meta-classes no modelo da Classificação Científica. Não podemos confundir, porém, essa relação com a próxima, a de generalização.

As classificações também são uma forma especial de Ocultação da Informação, pois

trabalhando com a classe ocultamos as informações como a identidade.

4.2.3. Generalização

Com a **generalização** nós somos capazes de entender como uma classe pode ser descrita por outra classe, mais geral. É importante ver a **diferença entre a classificação e a generalização**: a primeira trata da relação entre objeto e classes, enquanto a segunda trata da relação entre classes.

A generalização é uma relação muito comum entre classes, permitindo que qualquer objeto de uma classe possa ser visto, de uma forma mais geral, como um objeto de outra classe. Por exemplo, “Alice” é uma instância da classe “mulher”, enquanto “Bruno” é uma instância da classe “homem”, porém ambas as classes, “homem” e “mulher” podem ser generalizadas na classe “humano”. Dessa forma, “Alice” e “Bruno” deve ser tratados similarmente quanto membros da classe, mais geral, “humano”, mas podem ser tratados de forma diferente quanto membros das classes, mais específicas, “mulher” e “homem”. Utilizando judiciosamente a generalização podemos simplificar a forma de tratar objetos de classes similares em uma hierarquia.

O processo reverso da generalização é a **especialização**. Ela é uma relação entre classes, ao contrário da classificação que é uma relação entre classes e instâncias.

Exemplos típicos de generalização aparecem na Tabela 4.3.

Tabela 4.3.: Exemplos de Generalização

Classes	Classe mais geral
Funcionário, Aluno, Professor	Pessoa
Automóvel, Avião, Navio	Meio de Transporte
Computador, Rádio, Televisão	Aparelhos Eletrônicos

Novamente a Classificação Científica dos seres vivos nos oferece um grande exemplo da relação de generalização. Cada instância da Espécie *Homo sapiens* é uma instância da Espécie *Homo*, logo a classe *Homo* é uma generalização da classe *Homo Sapiens*.

Na linguística a relação de generalização é conhecida como **hiperonímia**, e a especialização é conhecida como **hiponímia**.

4.2.4. Composição ou Agregação

Na **composição** entendemos um objeto complexo, formado de um conjunto de outros objetos, como um só objeto. É como vemos uma bicicleta ou um carro. Ao eliminar a necessidade de descrever as partes, simplificamos a compreensão do objeto analisado.

4. Modelos e Abstrações

O processo reverso da composição é a **decomposição**. A Figura 4.3 mostra um brinquedo, o todo, e as partes que o compõe. As partes em separado não são o brinquedo.



(a) Partes do brinquedo



(b) Brinquedo completo

Figura 4.3.: Um brinquedo é composto de suas partes. Foto do autor

Exemplos típicos de composição aparecem na Tabela 4.4.

Normalmente, em modelagem de dados, usamos o conceito de composição para dizer que uma classe (como endereço) é uma característica de outra classe, descrevendo um entre seus atributos.

Tabela 4.4.: Exemplos de Composição

Partes	Objeto
Pneus, motor, etc.	Carro
Capa, centenas de folhas, etc.	Livro
Cabelo, pele, ossos, etc.	Homem

Na linguística a parte é conhecida como um **merônimo** e o todo como um **holônimo**. As relações são de **meronímia** ou **holonímia**.

4.2.5. Identificação

Com a **identificação** nós somos capazes de entender como caracterizar unicamente um objeto. Um nome identifica uma pessoa, por exemplo. Ao identificar unicamente um objeto podemos separá-lo de outro objeto semelhante e atribuir a entidades específicas atributos e características que só pertencem a ela, e não pertencem a outros elementos daquela classe.

Há uma diferença entre instanciar e identificar. Uma instância deve possuir uma identificação e uma identificação se aplica a uma instância. A identificação permite a que duas instâncias sejam reconhecidas como distintas ou como representações de um mesmo objeto (normalmente devendo ser reunidas em uma).

4.2.6. Simplificação pelo Caso Normal

Toda aplicação, ao funcionar, deve tratar de casos específicos que ocorrem durante o funcionamento normal, sejam eles alternativas que são aceitas no negócio, como alternativas não aceitas ou mesmo erros do sistema. Porém, é bem mais fácil discutir o funcionamento normal antes e depois os casos específicos.

A abstração de **simplificação pelo caso normal** indica que devemos começar a trabalhar pelo modo comum ou normal de funcionamento, ou ainda melhor, o modo onde tudo ocorre da forma mais simples e depois ir inserindo mecanismos para tratar das variações possíveis, por meio da especificação das **condições** que levam a essas variações.

4.2.7. Foco e Inibição

Uma das características importantes do ser humano é ser capaz de prestar atenção, isto é, por o foco em um detalhe, inibindo parcialmente os outros detalhes ao redor, e assim processar a informação, detalhe a detalhe.

Podemos ver essa forma de abstração acontecer no dia a dia, quando estamos olhando para um local e as áreas ao redor ficam levemente vigiadas pela visão periférica, mas não estamos realmente prestando atenção nas mesmas.

A abstração de simplificação pelo caso normal é uma forma de colocar o foco em uma questão e inibir as outras, i.e., as variações, que depois serão o foco da análise uma a uma, mantendo as outras inibidas.

Isso também acontece em um modelo de dados. Cada parte de um modelo foca em alguma informação que pretendemos registrar, e possui regiões ao redor que nos informam outras informações adicionais, mas não precisamos olhar ao detalhe da outra parte para entender aquela. Por isso separamos o sistema em entidades, como “aluno”, “curso” e “professor”, e depois analisamos cada uma em separado.

Tecnicamente falando, foco e inibição são muitas vezes representados pela modularização e divisão do sistema em partes estanques, com as características de alta coesão, todo um módulo trata apenas de um assunto, e baixo acoplamento, um módulo não interfere no outro. Outra forma de usar o foco e a inibição é no refinamento sucessivo.

4.2.8. Refinamento Sucessivo

A técnica de **refinamento sucessivo** indica que cada problema deve ser tratado de uma forma mais geral para depois ser analisado de uma forma mais específicas, normalmente seguindo o conceito de “explodir” um problema anterior (mais geral) em sub-problemas mais específicos, sendo cada um desses sub-problemas “explodidos” também até chegarmos a um problema de solução simples.

O refinamento sucessivo é uma forma mais específica da abstração de Foco e Inibição

4. Modelos e Abstrações

e faz parte de várias estratégias de abstração baseadas no conceito de **dividir para conquistar**. Equivale a uma estratégia *top-down* de solução de problemas, cuja a inversa é a estratégia *bottom-up*

4.2.9. Separação de Interesses

Separação de interesses é o processo de abstração onde tentamos descrever, ou produzir, um conceito separando-o em conceitos distintos com a menor quantidade possível de interseção, baseado em algum aspecto específico do problema sendo tratado. Esses conceitos normalmente são características ou comportamentos.

A separação de interesses é uma forma mais específica da abstração de Foco e Inibição e também faz parte de várias estratégias de abstração baseadas no conceito de dividir para conquistar.

4.3. Trabalhando com as abstrações

Imagine que precisamos descrever comprar um carro. É óbvio que todo carro possui quatro pneus, um motor, etc. Isso é uma classe bastante geral. Porém, desejamos ainda falar sobre um modelo específico: uma Ferrari Testarossa, por exemplo. Logo, acabamos de especializar nosso modelo, mas ainda não chegamos ao nível de objeto. Quando vemos o carro específico, aí temos o objeto. Ele é identificável como instância daquela classe porque apesar de dividir várias características em comum com outros objetos da classe, também tem algumas características únicas, como o número de série do chassi. Finalmente, desejamos trocar a cor do assento do carro. Nesse instante, já estamos vendendo uma parte do carro, decompondo-o em suas partes.

4.4. Exercícios

Exercício 4.4-1: Vá para o site <http://jogodeanalisedesistemas.xexeo.net/> e visite a Livraria Resolve. Faça um modelo na forma de um diagrama de toda a informação levantada. Pode ser, por exemplo, como um Mapa Mental.

5

Dados e Informação

The goal is to turn data into information, and information into insight.

(Carly Fiorina)

Conteúdo

5.1.	Usando Significados Diferentes	46
5.2.	Conclusão	48

Antes de entender o que é um Sistema de Informação, é preciso entender melhor o que significa a palavra Informação.

Este capítulo trata de quatro palavras que no dia a dia parecem ser sinônimas, mas têm significados diferentes quando usadas na Análise de Dados: dado, informação, conhecimento e sabedoria.

Vamos recorrer a dicionários para ter uma definição inicial. Segundo o American Heritage, **informação**, no contexto de computadores, “é o dado quando processado, guardado ou transmitido”(American Heritage, 2019).

Já no dicionário Aurélio, informação, entre outros significados, pode ser “Conhecimento amplo e bem fundamentado, resultante da análise e combinação de vários informes”, “Coleção de fatos ou de outros dados fornecidos à máquina, a fim de se objetivar um processamento” ou ainda “Segundo a teoria da informação, medida da redução da incerteza, sobre um determinado estado de coisas, por intermédio de uma mensagem”.

Apesar de não estarmos diretamente envolvidos com a teoria da informação, não

5. Dados e Informação

podemos de deixar de notar a importância da definição que diz que a **informação reduz a incerteza por meio de uma mensagem.**

5.1. Usando Significados Diferentes

Dados, informação, conhecimento e sabedoria¹ são quatro palavras que na língua corrente possuem uma interseção de significados, porém são consideradas diferentes em áreas específicas, como Computação e Engenharia de Sistemas e Sistemas de Informação.

A distinção entre essas palavras e a existência de uma hierarquia de entendimento entre elas é uma discussão corrente, construída em cima de uma pirâmide que registra a complexidade do que esses termos significam, conhecida como hierarquia DIKW (Rowley, 2007).

Diversos autores propõe definições diferentes. Zins (2007) chega a apresentar 44 definições diferentes para dados, informação e conhecimento, fornecidas por especialistas no assunto reunidos em um painel, usando a metodologia *Critical Delphi* (Zins, 2012).

Seguindo a classificação de Zins (2007), preferimos uma conceituação baseada em um modelo onde dados e informação são fenômenos externos, enquanto conhecimento, e consequentemente sabedoria, são fenômenos internos. Isso significa que dados e informação são do domínio universal, externos à mente, enquanto conhecimento e sabedoria são do domínio subjetivo, internos à mente(Zins, 2007).

Dados são apenas os símbolos que usamos para representar a informação, o registro de diferentes aspectos de um fato ou fenômeno, como Os números que guardamos em um banco de dados. Dados não são interpretados, eles existem, são adquiridos de alguma forma, via coleta, pesquisa ou criação, guardados de outra forma e, possivelmente, apresentados em uma terceira. O computador é uma máquina programável que manipula dados.

Por outro lado, informação é o dado com significado, normalmente processado de forma a ser útil. Uma informação deve permitir responder perguntas como “quando”, “quanto”, “quem”, etc.

$$\text{Informação} = \text{Dado} + \text{Significado} \quad (5.1)$$

É necessário fazer um mapeamento entre dados e informação. Esse mapeamento pode ser simples ou complexo, dependendo de várias variáveis envolvidas, que vão desde decisões arbitrárias tomadas pelo desenvolvedor até padrões internacionais. Por exemplo, em muitos sistemas é preciso ter a informação do sexo de uma pessoa (masculino ou feminino). Para isso, são guardados um número (1 ou 0) ou uma letra (M ou F), que é o dado que faz o registro da informação.

¹Em inglês, *wisdom*

5.1. Usando Significados Diferentes

Existem outras definições para dados e informação, mas esta diferenciação entre símbolo e significado será suficiente neste livro. Em especial, na Teoria da Informação tem uma abordagem onde o significado não é importante, apenas a quantidade de mensagens possíveis(Claude E Shannon, 1963).

Definir conhecimento e sabedoria é muito mais complexo, sendo conceitos em que há mais incerteza ou mesmo desacordo sobre o que realmente significam formalmente, além do significado ambíguo dessas palavras na língua portuguesa ou inglesa. Porém, vamos tomar algumas decisões sobre esses significados, sempre no contexto deste livro.

Sumarizando vários autores, Rowley (2007) define conhecimento como “uma mistura de informação, entendimento, capacidades, experiências, habilidades e valores.” Além disso, o mesmo autor lembra a diferença entre conhecimento explícito, que está codificado em documentos, bancos de dados e outros registros, e tácito, que é está na mente das pessoas, ou nos processos que elas realizam. Essa diferença foi bastante explorada por Nonaka e Takeuchi (1995), que apontam também a dificuldade de processar computacionalmente conhecimento tácito.

Finalmente, Rowley (2007) afirma que vários autores tratando de dados, informação e conhecimento não falam de sabedoria, e aponta uma publicação anterior sua, que define sabedoria como “a capacidade de por em ação o comportamento mais apropriado, levando em conta o que é conhecido (conhecimento) e o que traz o bem maior (considerações sociais e éticas).”(Rowley, 2006).

Rowley (2007) também nos lembra que enquanto dados são muito fáceis de processar e contém pouco significado, e ficam na base da hierarquia, sabedoria é muito difícil de processar e fica no topo da mesma. Isso resulta em uma pirâmide como na Figura 5.1.

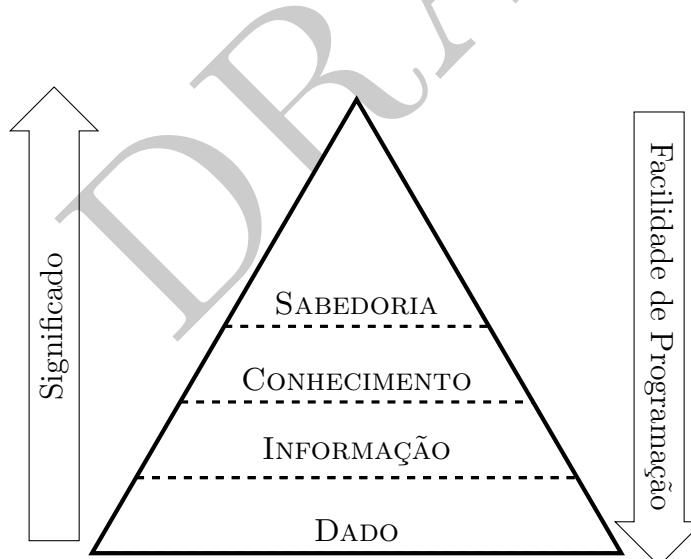


Figura 5.1.: Pirâmide da hierarquia do DIKW. Fonte:(Rowley, 2007)

5.2. Conclusão

O assunto deste livro é análise de sistemas de informação. A tecnologia computacional trata do processamento de dados. Esses dados normalmente são apresentados de forma contextualizada que os transforma em informação. Por exemplo, um sistema pode ter conter milhares de registros representando vendas, com números guardados em um formato binário que nada significa para o usuário, mas o usuário vai poder ver em um relatório o valor total vendido no mês, informação que é relevante para ele.

O conhecimento normalmente está mapeado nos algoritmos do sistema, nas regras de negócio implementadas. Alguns sistemas, que por exemplo usam aprendizado de máquina, podem inferir novos conhecimentos. Um exemplo disso seria o sistema de vendas descobrir que um tipo de item é mais vendido no início do mês, ou sempre junto de outro item, o que é chamado de análise de cesta de compras.

Já a sabedoria parece, por enquanto, estar ainda nas mãos dos seres humanos.

DRAFT

6

Sistema

Sufficiently simple natural structures are predictable but uncontrollable, whereas sufficiently complex symbolic descriptions are controllable but unpredictable.

(Howard Patte)

Conteúdo

6.1.	Tipos de Sistemas	51
6.2.	Sistemas Automatizados	52
6.3.	Princípios Gerais de Sistemas	52

Por que sistemas?

Este livro é sobre análise de sistemas de informação, e para entender o que é um sistema de informação é preciso antes entender o que é um sistema.

O dicionário Houaiss oferece duas definições da palavra sistema, entre várias, que são adequadas a nossa visão:

- “conjunto de elementos, concretos ou abstratos, intelectualmente organizado” (Houaiss, Villar e Mello Franco, 2009), e
- “conjunto de pessoas, procedimentos e equipamento projetado, construído, operado e mantido com a finalidade de coletar, registrar, processar, armazenar, recuperar

6. Sistema

e exibir informação, podendo assim servir-se de diferentes tecnologias”(Houaiss, Villar e Mello Franco, 2009).

A primeira definição é bem geral e se aplica a qualquer sistema. De acordo com essa definição, um sistema existe quando um conjunto de um tipo variados de coisas é entendido como organizado de alguma forma, sejam essas coisas concretas ou apenas conceitos.

A partir dessa definição podemos pensar sobre o Sistema Solar, entendendo como um conjunto de uma estrela, planetas e outros corpos organizados pela lei da gravidade, por exemplo. Ou um relógio, como um sistema de peças organizadas por conexões mecânicas.

A segunda já é uma definição específica que se adapta melhor a um tema que é o principal foco deste livro: sistemas de informações.

Bunge é um importante filósofo argentino que define formalmente o conceito de sistema. Apesar de ser uma definição sofisticada e também traduzida em notação matemática, ela é simples de entender e muito parecida com a primeira definição do Dicionário Houaiss, de forma que atende a visão mais ampla do que é um sistema.

Segundo ele, um sistema é composto de: um conjunto de **componentes**, um **ambiente**, que é um conjunto de itens nos quais ele é conectado, e uma **estrutura**, que é a relação entre os componentes e entre eles e a estrutura.

Um de seus exemplos de sistemas é uma escola, seus componentes são os funcionários e dos alunos, seu ambiente é a sua sociedade, e a estrutura são as relação de ensinar, aprende, gerenciar e outras.

Bunge (1979) também chama atenção que elementos de sistemas precisam agir entre si, o que ele chama de conexão, que é uma relação mais forte do que a simples associação. Existem dois tipos de conexões, a ação de um elemento sobre o outro, no caso um é o agente e outro o paciente, e a interação, onde existem ações recíprocas. Assim, uma família é um sistema, mas não um conjunto de estados de uma coisa, porque não há uma conexão, apenas uma relação. Um sistema (concreto) só existe para Bunge (1979) se possui ao menos duas coisas que se conectam dentro de um ambiente.

A ideia básica de Bunge (1979) para a importância de sistemas é a famosa frase “o todo é maior que as somas de suas partes”. O sistema tem novas propriedades que emergem das ações e interações entre as partes e que não existem nas partes quando isoladas.

Seguindo essa linha, Bertalanffy (1969) defende que os elementos de um “complexo” podem ser distinguidos de três maneiras:

1. pelo seu número;
2. pela sua espécie, e
3. pelas suas relações.

As duas primeiras maneiras de considerar os elementos são dependentes apenas deles, de forma que suas características são mantidas dentro e fora do complexo, já a terceira é dependente do que acontece dentro do complexo, logo para entendê-las não basta

conhecer as partes.

Esse raciocínio é bem aplicado a sistemas de informação, já que as partes devem ser entendidas como um todo, dentro do sistema, e não isoladamente.

6.1. Tipos de Sistemas

Podemos encontrar três divisões básicas sobre sistemas:

- quanto a sua natureza, entre sistemas **naturais** e **artificiais**, ou seja, feitos pelos homens(Ed Yourdon, 2006);
- quanto a sua origem, entre **abstratos**, como sistemas de informação, e **concretos**, como um automóvel, e
- quanto ao seu tipo, entre sistemas **abertos**, que recebem influência do meio externo, como matéria, energia ou informação, e **fechados**, que não recebem(Bertalanffy, 1969).

A primeira divisão é fácil de entender. Sistemas naturais estão presentes na Natureza, como sistemas estelares, o sistema digestivo e o Sistema Solar, representado na Figura 6.1. Já os sistemas artificiais são criados pelo ser humano, como o sistema aeroviário, o sistema elétrico e sistemas de informática. Podemos chamar a atenção que, de certa forma, os sistemas naturais são também imaginados pelos seres humanos, que vêm em um conjunto de coisas inter-relacionadas as características de um sistema, porém humanos não os constroem. Sistemas artificiais são construídos por seres humanos. Assim, humanos decidem que partes compõem o Sistema Solar, e até decidem de que classe alguns objetos são, como Plutão, que deixou de ser um planeta, mas não tem interferência na existência desse sistema.

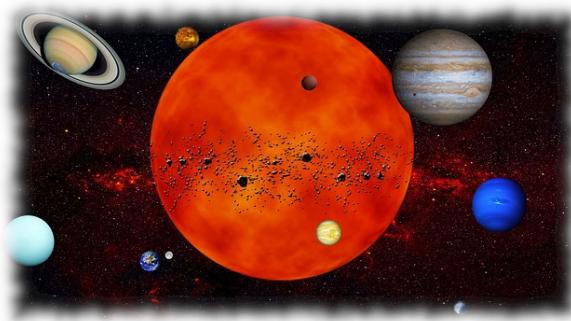


Figura 6.1.: O Sistema Solar é um sistema natural. Imagem por Don Cloud no Pixabay

Já para entender a terceira divisão, entre sistemas abertos e fechados, é preciso levar em conta a forma como são analisados. Um sistema fechado não troca matéria, energia, ou informação, com o ambiente ao seu redor ou outro sistema. Já um sistema aberto faz essas trocas. Sistemas fechados são simplificações teóricas.



Figura 6.2.: Um *smartphone* é um sistema artificial. Imagem CC BY-NC

6.2. Sistemas Automatizados

A raça humana não só criou seus próprios sistemas, como máquinas, fábricas e sistemas de informação, como também os automatizou, isto é, criou formas para que esses sistemas funcionem com uma intervenção muito menor do ser humano. É só comparar uma fábrica automática de biscoitos com o trabalho artesanal de um confeiteiro. A fábrica vai seguir uma receita, misturar ingredientes, bater, assar e empacotar os biscoitos, tendo vantagens sobre o ser humano;

Nós criamos sistemas automatizados para:

- aumentar a produção;
- diminuir o custo;
- acelerar os tempos;
- aumentar a segurança de processos;
- atender a questões políticas;
- diminuir a carga de trabalho insalubre sobre seres humanos;
- diminuir a variação na produção, etc.

6.3. Princípios Gerais de Sistemas

Ed Yourdon (2006) propõe alguns princípios gerais de sistemas¹:

- quanto mais especializado um sistema é, menos ele é capaz de se adaptar a circunstâncias diferentes;

¹E em especial sistemas de informação

6.3. Princípios Gerais de Sistemas

- quanto maior é um sistema, mais de seus recursos tem que ser dedicados a sua operação e manutenção diária;
- sistemas são sempre partes de sistemas maiores e sempre podem ser particionados em sistemas menores;
- é comum que sistemas cresçam, e
- as interações entre componentes de um sistemas são muitas vezes complexas e sutis.

DRAFT

DRAFT

Organizações

Every company has two organizational structures: The formal one is written on the charts; the other is the everyday relationship of the men and women in the organization.

(Harold Geneen)

Conteúdo

7.1.	Funções Típicas de uma Organização	57
7.2.	Estrutura das Organizações	57
7.3.	Estratégia Organizacional	59
7.4.	A Necessidade de Sistemas de Informação	59
7.5.	Processo de Informatização nas Organizações	60
7.6.	A necessidade centralização dos dados	62
7.7.	Organograma	62
7.8.	Conclusão	68

7. Organizações

7.9. Exercícios	68
---------------------------	----

Por que organizações?

Este livro é sobre análise de sistemas de informação, e prioritariamente sobre aqueles que trazem valor para organizações, principais demandantes desse tipo de sistema.

Neste texto, usamos o termo **organização** para representar, de forma geral, todas as pessoas jurídicas e suas subdivisões. Empresas, departamentos e seções, associações, órgãos governamentais, clubes, igrejas, etc. Uma forma melhor de dizer é que queremos discutir sobre todas as pessoas **menos** as pessoas naturais ou pessoas físicas.

Está claro que **uma organização é um sistema**. Ela é compostas de partes, como pessoas, que se interrelacionam e tem relações com o ambiente. Elas são sistemas abertos e artificiais.

Por exemplo, uma escola privada é um sistema composto de:

- proprietários;
- diretores;
- administradores;
- coordenadores acadêmicos;
- professores;
- funcionários administrativos;
- funcionários de serviços gerais;
- alunos;
- pais de alunos;
- currículos;
- material escolar, e
- prédios e suas salas.

Como interação entre as partes temos, entre outras: professores dão aulas para alunos dentro de uma sala de aula seguindo um currículo e usando material escolar.

Segundo Chiavenato (2014), a organização “é um conjunto de cargos funcionais e hierárquicos a cujas prescrições e normas de comportamento todos os seus membros devem se sujeitar”, com o princípio que seus membros se comportam racionalmente. O autor ainda chama atenção para o fato das organizações serem a forma dominante de instituição no mundo moderno. Notamos que essa definição só inclui pessoas.

Se mesmo uma pessoa física já precisa de sistemas de informações, na forma de agenda, caderno de telefone, uso de planilhas eletrônicas para calcular os gastos do mês, mais ainda se pode esperar de uma organização. Isso acontece por vários motivos, entre eles a necessidade de manter uma memória, a de guardar informações para seu funcionamento, obrigações impostas pelo governo e outras.

Uma organização tem um propósito de ser. Para as mais complexas ou mais bem

gerenciadas, este propósito é definido em um planejamento estratégico que conta com a definição de uma missão, uma visão, metas e objetivos. Muitas vezes são definidos *KPI* - *Key Performance Indicators*, que permitem medir o desempenho da organização em relação ao que foi planejado.

Quanto maior a organização, maior a quantidade de dados que ela gera. Estes dados vêm tanto da interação com o mundo externo, quanto da comunicação e gestão interna. Esses dados representam informações que explicam de várias formas o que é a organização e qual o seu desempenho. Para manipular esses dados são usados os **Sistemas de Informação**.

Assim, a organização precisa que os dados estejam disponíveis para se conhecer. Ao mesmo tempo, ela precisa que esses dados sejam de boa qualidade.

7.1. Funções Típicas de uma Organização

Toda organização tem que cumprir algumas funções básicas para seu funcionamento. Chiavenato (2014) faz uma primeira divisão em cinco grandes funções:

- produção, operações ou serviços;
- comercial ou marketing;
- financeira;
- recursos humanos, e
- administrativas.

Várias outras funções podem ser citadas, como a engenharia, a contábil, a de TI, a inovação, etc. Essas funções são normalmente organizadas na estrutura da empresa. É comum, por exemplo, que a fundão de TI fique subordinada a um diretor financeiro¹ ou administrativo. Em organizações mais voltadas para a informação, já existem cargos de diretoria para a área, como o CIO (*Chief Information Officer*).

Essas funções interagem. Por exemplo, marketing faz um plano de propaganda, que faz com que vendas aconteçam, os produtos vendidos são feitos pela produção, de acordo com um projeto da engenharia, a partir de insumos comprados pela compra e guardados pelo estoque, baseado na pesquisa e inovação. Tudo é entregue pela logística. A cobrança é feita pelo financeiro, que informa a contabilidade. O jurídico faz os contratos de todas as vendas.

7.2. Estrutura das Organizações

As organizações, em uma visão neoclássica, podem se estruturar de três formas (Chiavenato, 2014):

¹Mesmo que isso não faça muito sentido.

7. Organizações

- **linear**, a estrutura mais simples e antiga, inspirada nos exércitos e igrejas do passado, hierárquica e com a autoridade única do superior sobre seus subordinados, onde as linhas de comunicação são formais e sempre pela hierarquia da organização, com aspecto piramidal e decisões centralizadas;
- **funcional**, onde a autoridade é relativa ao conhecimento e especialização, cada funcionário se reportando a vários chefes, de acordo com a especialidade, com linhas de comunicação diretas, e decisões descentralizadas, e
- **linha-staff**. resultado da combinação dos dois tipos anteriores, buscando as vantagens dos dois, havendo dois tipos de órgão, de linha, que tratam dos objetivos da organização e seguem uma autoridade linear, e *staff*, que assessoram, planejam, controlam, etc. com autoridade funcional.

Já mais modernamente se reconheceu a estrutura **matricial**, onde se combina a departamentalização funcional com a por produto ou projeto na mesma organização.

Além disso, Chiavenato (2014) também mostra que as organizações tradicionais têm, pelo menos, três níveis:

1. **operações**, de nível técnico e onde existem os executores e supervisores;
2. **planos**, de nível gerencial, onde estão gerentes e chefes, também chamado de **tático**, e
3. **decisões**, de nível institucional, onde estão os executivos a nível de direção, também chamado de **estratégico**.

Mesmo hoje em dia, muitas organizações ainda podem ser caracterizadas dentro dessas formas. As teorias, porém, evoluíram. A era da informação causou mudanças drásticas nas estruturas de poder das organizações, e também causou o achatamento da hierarquia, eliminando a necessidade de vários níveis de gerência. Novas formas de gerenciar apareceram e levaram novamente a estruturas diferentes.

As tendências atuais são Chiavenato (2014):

- cadeias de comando mais curtas, com menos níveis hierárquicos;
- menos unidade de comando, com o crescimento dos relacionamentos horizontais sobre os verticais (estruturas tradicionais de comando);
- maiores amplitudes de controle e autonomia, com delegação de responsabilidade e menos supervisão direta;
- maior participação e empoderamento, com transferência de responsabilidades e delegação;
- ênfase em equipes de trabalho e projetos;
- organização em unidades de negócio;
- menos controles de comportamento e foco nos objetivos e resultados;
- forte infraestrutura de TI;
- foco no negócio essencial, com terceirização das atividades não essenciais;
- consolidação da economia do conhecimento;
- construção de competências;
- envolvimento de terceiros, e

- governança corporativa.

7.3. Estratégia Organizacional

Dentro das mudanças nas organizações citamos o foco no negócio essencial, que implica no desenvolvimento de uma estratégia organizacional, isto é, um “padrão ou plano que integra os objetivos globais de uma organização e as políticas e ações em um todo coerente.”(Chiavenato, 2014).

Uma das principais características do planejamento estratégico é a definição da missão e da visão da empresa. A **missão** é a finalidade da organização, ou seja, responde três perguntas(Chiavenato, 2014):

- quem somos;
- o que fazemos, e
- por que fazemos.

Já a **visão** é declaração de como a empresa se vê no futuro, como um projeto de futuro, em um determinado prazo, sendo “o destino que se pretende transformar em realidade”(Chiavenato, 2014).

Segundo a OMG (2015), na especificação do *Business Motivation Model*, a missão descreve os meios para alcançar a visão, que é o fim. A visão então permite gerar **objetivos** e **metas**, enquanto a missão vai levar a um curso de ação baseado em estratégias e táticas, e diretivas na forma de regras e políticas empresariais.

7.4. A Necessidade de Sistemas de Informação

Se mesmo uma pessoa física já precisa de sistemas de informações, na forma de agenda, caderno de telefone, uso de planilhas eletrônicas para calcular os gastos do mês, mais ainda se pode esperar de uma organização.

Algumas organizações tem nas informações sua operação principal, como uma empresa que vende entradas de cinema, e já tem, no seu nível operacional, a TI como ferramente essencial. Porém todas as organizações têm a necessidade de controlar e gerenciar todas as informações que dispõem sobre si mesmas e sobre o mercado, de modo a permitir que suas decisões produzam ações que gerem benefícios e evitem prejuízos, de curto, médio e longo prazo, nos níveis operacional, tático e estratégico. Além disso, outros motivos existem como a necessidade de manter uma memória, a de guardar informações para seu funcionamento, e obrigações impostas pelo governo. Hoje em dia, por exemplo, as empresas devem fornecer todas as suas notas fiscais de forma eletrônica para os órgãos da receita.

As informações necessárias para a organização aparecem em todo seu ambiente. Cada

7. Organizações

ação realizada, cada decisão tomada, gera, ou pelo menos deveria gerar, um registro que pode ser utilizado mais tarde para a tomada de decisões. Por exemplo, cada venda feita por uma cadeia de lojas de varejo pode ser incluída em um gráfico ou relatório que permita entender o desempenho das vendas por local, por produto, por vendedor e até mesmo pela hora do dia. Em uso mais avançado, esses dados podem ser correlacionados, com o processo de seleção da área de Recursos Humanos, de maneira a tentar desenvolver um perfil do vendedor a ser contratado, tanto de forma geral como para atender necessidades específicas. Com relação aos dados de mercado, podem ser usados para prever o desempenho de uma nova loja em uma localidade específica, de acordo com dados demográficos ou informações sobre outros tipos de comércio ou mesmo de consumo de água ou eletricidade.

Tradicionalmente, quanto maior a organização, maior a quantidade de dados que ela gera. Estes dados vêm tanto da interação com o mundo externo, quanto da comunicação e gestão interna. Esses dados representam informações que explicam de várias formas o que é a organização e qual o seu desempenho. No século XXI mesmo pequenas organizações, até mesmo compostas de um só homem, já possuem, por desejo ou obrigação, algum grau de informatização. Porém apareceram novas empresas, totalmente informatizadas, que operam dentro de ambientes virtuais, como a internet, e manipulam grande quantidades de informação mesmo sendo pequenas.

Para ser possível operar, controlar e gerenciar essas organizações é necessário implantar sistemas de informação.

7.5. Processo de Informatização nas Organizações

Frente a importância da informação para o sucesso da organização, é fácil entender que, ao longo de sua vida, ela deve passar por um processo contínuo de informatização, a fim de atender as demandas cada vez maiores tanto da própria organização, quanto de seus fornecedores, parceiros, clientes e também do Estado.

Esse processo é necessário e geralmente benéfico, mas não sem problemas.

Entre os principais problemas do processo de informatização das organizações está o fado dele ser normalmente de crescimento vegetativo, emergente, *bottom-up* e por demanda imediata. Os sistemas são criados um a um, em função de necessidades específicas trazidas pelas partes interessadas, e construídos de modo a atender principalmente os requisitos dessas partes, muitas vezes desconsiderando como as atividades e informações desses sistemas podem ser úteis para o resto da organização. Isso gera um descasamento entre os sistemas e a necessidade de interfaces complexas. Além disso, são usadas tecnologias diferentes, mesmo porque as mais antigas vão saindo do mercado, não restando alternativa que migrar.

Por outro lado, tentativas de criar grandes sistemas integrados que atendam toda a organização parecem estar fadados ao fracasso. O tamanho de um sistema é um fortíssimo

7.5. Processo de Informatização nas Organizações

fator de risco (Gibbs, 1994; Pressman e Maxim, 2014), já que a relação do tamanho com a taxa de fracasso é mais que linear.

Como sempre, as melhores soluções, ou pelos menos as soluções mais viáveis, parecem estar em um meio termo, ou pelo menos em decisões tomadas de forma consciente dos riscos e benefícios de cada opção.

Não se pode também deixar de levar em consideração que organizações diferentes têm gestões diferentes desse processo. Enquanto uma pode escolher os sistemas de informação a serem implantados de forma *ad-hoc*, outra, mais corretamente, pode possuir um planejamento estratégico que indica em que direção a área de Tecnologia da Informação deve seguir e que sistemas serão estratégicos para o seu progresso, criando programas que comportam vários projetos e analisando frequentemente seu portfólio de software.

Esse quadro bastante variado, evoluiu muito tecnologicamente e no entendimento do negócio informatizado nos últimos anos. Alguns grandes marcos, como o aparecimento do computador comercial, do computador pessoal, das redes locais e dos sistemas cliente-servidor, da Internet e, mais recentemente, das tecnologias em nuvem, causaram grandes mudanças na forma de pensar a própria organização, criando até mesmo a possibilidade de organizações virtuais.

Ao logo desse tempo houve também um aprendizado que trouxe soluções bastante adequadas aos problemas comuns de gestão das organizações, como o uso de sistemas **ERP** e a compra de software **COTS**.

COTS significa **Commercial Of The Shelf**, e indica software pronto que fornece funções específicas com nenhuma ou pouca adaptação para quem o compra ou implementa. Principalmente pequenos negócios e profissionais liberais podem ter quase todas suas necessidades atendidas por meio de esse tipo de software. Algumas áreas de aplicação, mesmo em empresas grandes, podem se beneficiar desse tipo de produto. Deve ficar claro, porém, que nesse caso a organização deve se adaptar as práticas implementadas no produto.

Já os sistemas **ERP**, **Entreprise Resource Planning**, são na prática sistemas de gestão empresarial customizáveis destinados a integrar fortemente as áreas de negócio. Sua principal característica é funcionar em funções do negócio cuja tarefa é bem definida, e cuja experiência de desenvolvimento de software é muito forte, e seu maior sucesso vem da implementação, já no software, das melhores práticas do negócio. Assim, muitas empresas adotam não só os ERP, mas também as suas práticas nessas áreas, abandonando a sua forma de trabalhar em troca de uma melhor prática do mercado.

Soluções ERP hoje são capazes de servir grande parte das necessidades genéricas de todas as empresas, desde funções de Recursos Humanos até detalhes de Contabilidade.

Essas soluções, porém, são genéricas. Mesmo quando customizáveis, dificilmente atendem necessidades importantes dos processos das cadeias de valor das empresas, pois estes são específicos e poucas empresas desejam remodelá-los de acordo com uma prática comum no mercado, até mesmo porque suas diferenças podem fornecer justamente o

7. Organizações

diferencial competitivo que possuem. Assim, o software desenvolvido sob encomenda, interna ou externamente, ainda possui, e certamente sempre possuirá, um presença marcante em todos as organizações.

Todos esses sistemas geram dados. Quase que escondidos nesses dados estão as informações necessárias para a tomada de decisão.

7.6. A necessidade centralização dos dados

O outro processo de grande sucesso foi a centralização dos dados dos sistemas operacionais e transacionais (OLTP) em bases de dados integradas na empresa, principalmente em SGDBs Relacionais de grande porte.

O quadro do primeiro quarto do século XXI é que a maioria das organizações entende que seus dados operacionais devem estar sob controle centralizado. O ideal é que todos esses sistemas compartilhem uma mesma base, porém necessidades como desempenho, custo de implementação, estratégia empresarial, permanência da tecnologia usada no mercado, e ainda outras, podem levar a existência de mais de uma base.

Esta situação também não é sem seus problemas. A centralização dos dados sob um controle único pode tanto auxiliar como trazer dificuldades para o desenvolvimento rápido de sistemas, principalmente quando há exigência de muita burocacia para alterar a base ou estruturar informações de forma a atender novas demandas de negócio.

7.7. Organograma

A forma mais simples de representar uma empresa é provavelmente o organograma.

Um **organograma** é uma descrição da organização de uma empresa, amplamente divulgada, descrevendo as suas áreas e as hierarquias entre elas, de comando ou comunicação. O Organograma é ferramenta essencial na compreensão de uma empresa e suas linhas de poder.

Organogramas são diagramas que descrevem a estrutura formal de uma organização incluindo suas relações hierárquicas, normalmente por meio de linhas e retângulos. São simples de ler, porém alguns organogramas são mais complicados que outros, de forma a representar informações adicionais.

Normalmente um organograma tem o formato de um grafo hierárquico, onde no alto está uma caixa com a posição mais importante da organização, e nos níveis abaixo aparecem caixas com seus subordinados. Cada caixa pode representar um cargo, uma função ou mesmo um departamento. A Figura 7.1 mostra um organograma simples.

Aris, uma linguagem de modelagem de processos apresentada no Capítulo 16, também permite representar organogramas, e isso é feito como na Figura 7.2.

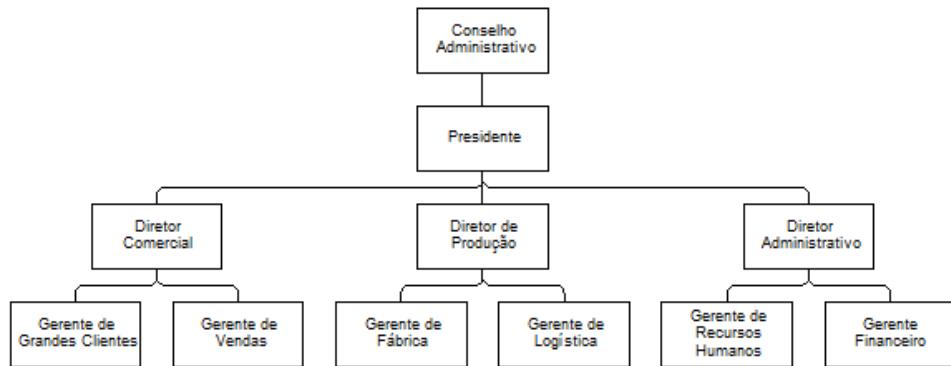


Figura 7.1.: Um organograma simples, de uma empresa hipotética.

Em geral o analista não precisa levantar o organograma, pois a empresa já o possui, mas é comum que haja algumas mudanças não registradas que ele deve corrigir. Na verdade, não é trabalho do analista de sistemas construir o organograma da empresa, porém ele precisa conhecê-lo para melhor desenvolver o seu trabalho. Para isso é importante obter esse documento junto ao cliente, e verificar não só a hierarquia de cargos, mas também quem ocupa cada cargo, e como entrar em contato com cada um dos membros da organização que possa ter interesse no sistema. O organograma é uma boa ferramenta para o levantamento inicial das partes interessadas.

A importância de conhecer o organograma da empresa se reflete tanto na modelagem propriamente dita, pois ele fornece a descrição da empresa que será convertida para objetos do modelo, como no processo de modelagem, pois a partir do organograma é obtido o conhecimento de cargos e responsabilidades, definindo pessoas a serem entrevistadas.

Ao levantar o organograma, pode ser interessante também solicitar documentos que registrem ou levantem com entrevista as descrições e responsabilidades dos cargos, se elas existirem. Apesar de recomendado, a existência desses documentos não só é infrequente, como muitas vezes mostra um quadro fora da realidade, então o analista deve também validar toda a informação.

Este texto não tratará do processo de levantamento do organograma, pois essa prática é mais afeita à administração. Fica, porém, o lembrete de sua importância como documento de referência ao analista. Em todo caso, em poucas entrevistas pode ser levantado um organograma se não perfeito, adequado ao trabalho de análise.

7.7.1. Relações em um organograma

Um organograma pode ser utilizado para representar diferentes formas de subordinação, como a subordinação direta (onde o subordinado deve cumprir as ordens de seu chefe), a assessoria (onde o assessor fornece conselhos e pareceres) e a subordinação funcional (onde o superior pode determinar funções e métodos a outras áreas).

7. Organizações

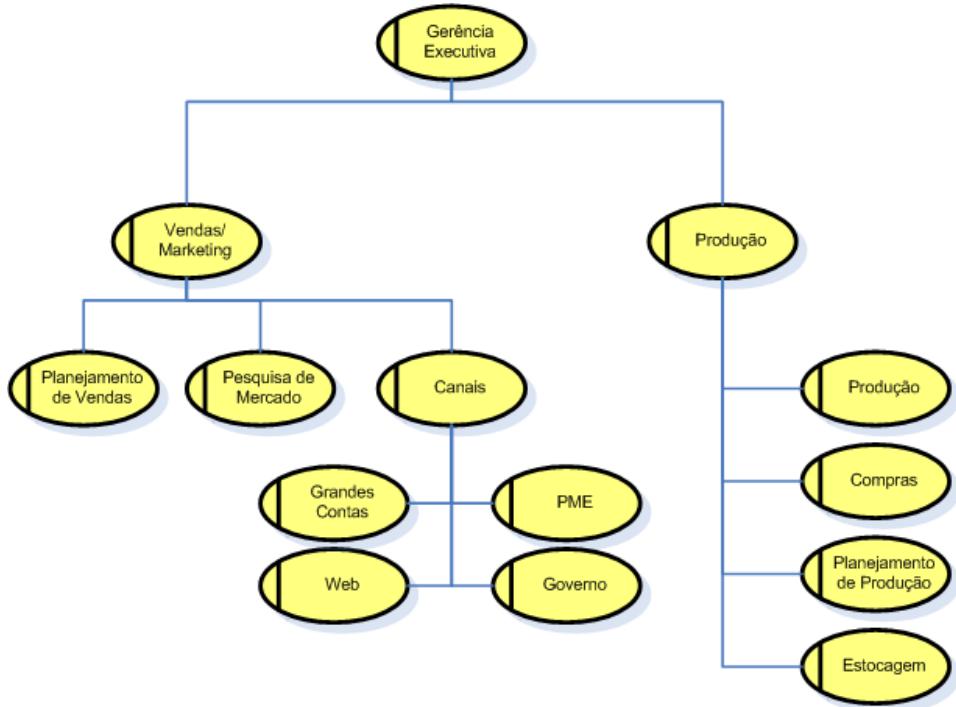


Figura 7.2.: Um exemplo de um organograma de um pedaço de uma empresa hipotética escrito em ARIS.

Normalmente a subordinação direta é representada por uma linha cheia vertical, a assessoria por uma linha cheia horizontal e a subordinação funcional por uma linha pontilhada. As Figuras 7.3 e 7.4 exemplificam alguns desses casos.

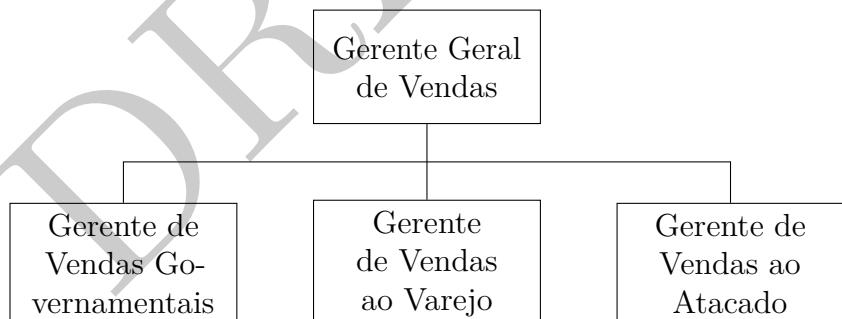


Figura 7.3.: A relação de subordinação, normalmente mantida na vertical, entre o gerente geral e seus gerentes subordinados, que aparecem em um mesmo nível.

7.7.2. Outros formatos de organograma

Na grande maioria das vezes serão encontrados nas organizações organogramas clássicos ou pequenas variações. Existem, porém, algumas formas alternativas. Uma forma

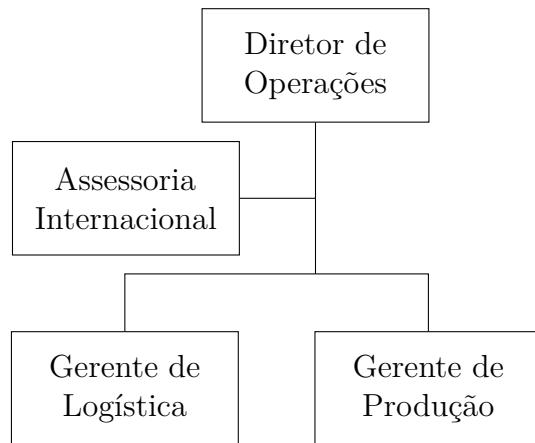


Figura 7.4.: A relação de assessoria é normalmente feita com uma barra horizontal, como a Assessoria Internacional aparece no diagrama acima.

interessante, também com muitas variações, é a radial ou solar. A Figura 7.5 mostra um organograma radial com

Outra forma possível é inverter o organograma. O organograma da Figura 7.6, criado pelo site Guia de Direitos, mostra a estrutura do Poder Judiciário, não tratando de cargos, mas sim de partes desse poder.

Devido à grande variedade de formas usadas, organogramas podem ser desenhados tanto em sistemas específicos como em softwares genéricos de desenho.

7. Organizações

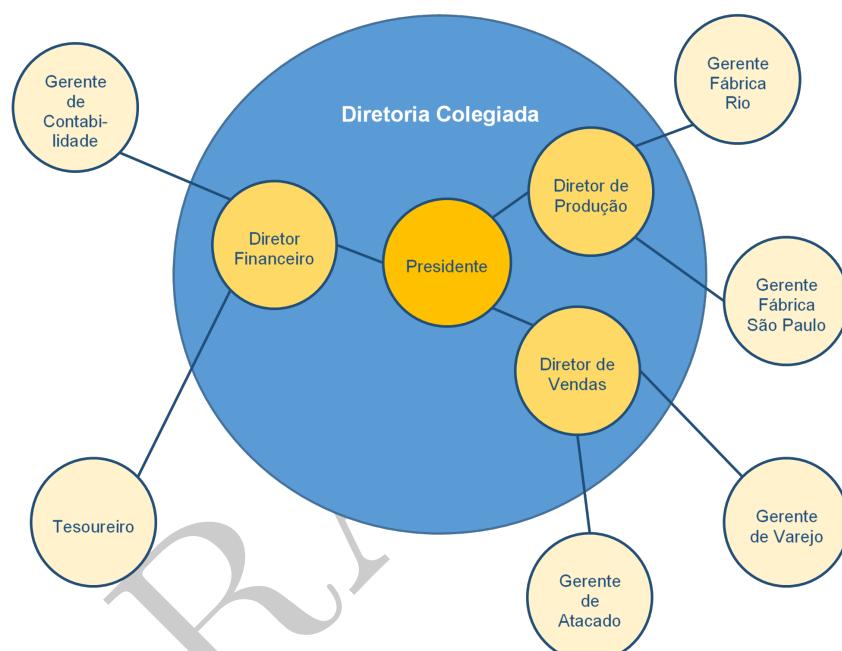


Figura 7.5.: Um exemplo de organograma radial, com uma espécie de “diagrama de Venn” indicando que os diretores e o presidente compõe a diretoria colegiada.



Figura 7.6.: Organograma do Poder Judiciário, em forma inversa, criado pelo site Guia de Direitos, deixado em Copyleft.

7.8. Conclusão

Em todo caso, a organização moderna não pode viver sem a Tecnologia da Informação. Ela é essencial, se não para atender sua função produtiva e suas outras funções, principalmente a administrativa.

Além disso, todo sistema de informação deve estar associado a estratégia da organização, em busca de um valor real para o negócio da organização. Isso é uma mudança de mentalidade importante que vem se fortalecendo cada vez mais.

7.9. Exercícios

Exercício 7.9-1: Vá para o site <http://jogodeanalisedesistemas.xexeo.net/> e visite a Livraria Resolve. Identifique, para cada personagem, qual função da organização ele ajuda a realizar.

Exercício 7.9-2: Vá para o site <http://jogodeanalisedesistemas.xexeo.net/> e visite a Livraria Resolve. Faça um organograma da livraria. Se necessário, planeje mais perguntas que podem ser feitas aos personagens, invente uma resposta razoável, e desenhe o organograma de forma a atender essas respostas.

8

Sistemas de Informação

Information technology and business are becoming inextricably interwoven. I don't think anybody can talk meaningfully about one without talking about the other.

(Bill Gates)

Conteúdo

8.1.	Características dos Sistemas de Informação	71
8.2.	Sistemas de Informação Típicos e a Organização	72
8.3.	Tipos de Projetos de Sistemas de Informação	72
8.4.	Porque são feitos projetos de SI	73
8.5.	O Poder está com o usuário	74
8.6.	Exercícios	75

Por que sistemas de informação?

Este livro é sobre análise de sistemas de informação, logo é preciso entender o que são.

Sistemas de Informação são utilizados em organizações para planejamento, monitoração, comunicação e controle das suas atividades, por meio da manipulação e guarda de informações.

8. Sistemas de Informação

Segundo o Dicionário Aurélio, a palavra **sistema** significa, entre outras coisas, um Conjunto particular de instrumentos e convenções adotados com o fim de dar uma informação. Esta acepção claramente se refere ao que chamamos de **sistema de informação**. Os instrumentos são as ferramentas, os mecanismos, concretos ou abstratos, que utilizamos para fazer funcionar os sistemas, tais como: programas de computador, relatórios, formulários, etc. As convenções são as suas regras de utilização.

K. Laudon e J. Laudon (2011) fornecem uma definição formal para um **Sistema de Informação**: “Conjunto de componentes inter-relacionados, que coletam (ou recuperam), processam, armazenam e distribuem informações destinadas a apoiar a tomada de decisões, a coordenação e o controle de uma organização”

Sistemas de informação são sistemas artificiais, abertos e abstratos.

Um exemplo típico de sistema de informação é um sistema de controle de empréstimos de uma biblioteca. Entre suas várias finalidades, a principal é certamente controlar o empréstimo dos livros, informando quem está com qual livro em um determinado momento (quando). Além disso, o sistema permite outras atividades, como a gerência do estoque de livros, ou seja, que livros estão na biblioteca, a monitoração das livros mais e menos emprestados ou usados, etc.

Outro exemplo de sistema de informações é um *Dashboard* estratégico que informa o valor dos *KPI*, *Key Performance Indicators*, da organização, permitindo um diagnóstico de seu funcionamento e a verificação de sua aderência ao planejamento estratégico

A Figura 8.1 mostra uma tela de um sistema real (BDO) que gerencia uma frota de caminhões. Nesta tela são informados os dados sobre uma viagem do caminhão.

Figura 8.1.: Uma tela de um sistema de informações real

Apesar de estarmos preocupados com o desenvolvimento de sistemas de informação automatizados, implementados na forma de programas de computador, isso não é uma necessidade. Durante séculos as organizações usaram sistemas de informação apenas com o uso de pessoas, papel e tinta. Apenas bem mais tarde, aparecem máquinas como máquinas de escrever e de somar. Não seria exagerado dizer que a escrita e os números

8.1. Características dos Sistemas de Informação

foram criados para suportar os primeiros sistemas de informação, que tratavam, por exemplo, de colheitas e comércio. Muitas das técnicas deste livro podem, e devem, ser aplicadas para entender sistemas de informação manuais.

Uma organização possui muitos sistemas de informação, integrados ou não. Mesmo uma pequena empresa caseira pode ser gerenciada com várias planilhas, enquanto uma grande multi-nacional vai certamente possuir de um grande sistema de *Enterprise Resource Planning* (ERP) , que é o nome dado a sistemas de gestão empresarial, e mais uma miríade de outros sistemas.

Este capítulo apresenta uma breve descrição de como funciona, para que servem e quem usa os sistemas de informação dentro de uma organização.

8.1. Características dos Sistemas de Informação

É importante entender que sistemas de informação são sistemas **interativos** e **reativos**(McMenamin e Palmer, 1984).

Interativo significa que o sistema troca informações com o ambiente, em especial com os agentes externos que fazem parte desse ambiente, pessoas e outros sistemas de computador. O sistema só faz sentido se é capaz dessa interação(McMenamin e Palmer, 1984).

Reativo significa que o sistema funciona reagindo a mudanças no ambiente, e em especial, a mudanças provocadas pelos agentes externos(McMenamin e Palmer, 1984).

Nossos sistemas também são **sistemas de respostas planejadas**. Isso significa que nossas respostas são determinadas e determinísticas, que podemos criar um programa que as produza. Também significa que todas as perguntas que podem ser feitas ao sistema podem, e são, identificadas previamente(McMenamin e Palmer, 1984).

Escolhendo essas regras de modelagem, escolhemos um caminho para decidir quando o sistema vai funcionar: em vez de deixar isso incerto, como em muitos métodos, nós determinamos que o sistema só funciona para responder um evento no ambiente, causado por um agente externo, e que possua uma resposta planejada.

As metodologias de desenvolvimento apresentadas neste livro são feitas sob medida para sistemas interativos e reativos, de respostas planejadas. Nesse caso, somos ao mesmo tempo restritivos, pois se o sistema não pode ser modelado dessa forma não serve para nossa metodologia, como ampliativos, pois a grande maioria dos sistemas pode ser modelada de forma natural com esses princípios.

8.2. Sistemas de Informação Típicos e a Organização

Atualmente já consideremos que vários sistemas de informações típicos de uma empresa são necessidades básicas que podem ser atendidas de uma só vez. Esses sistemas constroem o que é comumente chamado de ERPs de Enterprise Resource Planning ou Sistemas de Gestão Empresarial em português mas que na prática não são sistemas de planejamento (ou de recursos), mas sim de controle e administração de uma empresa.

Entre as características encontradas em ERPs podemos citar a integração das atividades da empresa e o uso de um banco de dados único. O líder mundial do mercado é a SAP AG, com o produto SAP R/3. O custo de implantação de um ERP de grande porte pode chegar até 300 milhões de dólares. No Brasil, existem produtos menos ambiciosos e mais baratos.

Os sistemas de ERP atuais contêm módulos representando os mais típicos sistemas de informações necessários em uma empresa, tais como: Contabilidade Fiscal, Contabilidade Gerencial, Orçamento e Execução Orçamentária, Ativo Fixo, Caixa e bancos, Fluxo de Caixa, Aplicações e Empréstimos, Contas a Receber, Contas a Pagar, Controle de Viagens, Controle de Inadimplência, Administração dos preços de venda, Compras, Controle de fretes, Controle de contratos, Controle de investimentos, Cotações de vendas, Estoque, Exportação, Faturamento, Gerenciamento de armazéns, Importação, Obrigações fiscais, Pedidos, Previsão de vendas, Recebimento, Gestão de informação de RH, Pagadoria, Treinamento, RH scorecard, Planejamento de RH, Planejamento de produção, Planejamento da capacidade, Custos industriais, Controle de chão de fábrica, Controle da produção, Configurador de produtos, Planejamento de Manutenção, Acompanhamento de Manutenção e ainda muitos outros...

8.3. Tipos de Projetos de Sistemas de Informação

Existem três tipos de projeto de sistemas de informação: manual, manual para automático e re-automação. Os processos de re-automação ainda podem se dividir em recodificação, re-projeto e re-desenvolvimento, melhoria ou manutenção.

Todos esses tipos de projeto apresentam ao analista de sistemas o mesmo desafio: descobrir o que deve ser feito. Porém, cada tipo apresenta certas particularidades que facilitam ou dificultam esse trabalho de análise.

O trabalho do analista em sistemas manuais é mais relacionado à formalização, por meio de documentação e padrões, de processos já adotados, a criação de novos processos e a transformação de processos existentes tendo em vista otimizá-los ou possibilitar que atendam novas necessidades da organização. Esses processos podem ser bastante complexos e convolutos em alguns casos, o que exige do analista uma boa capacidade de compreensão e modelagem.

Porém, como não serão transformados em programas de computador, o analista pode

trabalhar com ferramentais mais informais e mais próximas ao dia a dia do usuário.

Os projetos que apresentam maior dificuldade são os de passagem do processo manual para o automático. Isso acontece porque normalmente esses projetos exigem todo o trabalho feito no tipo anterior, e, de forma adicional, a criação de um modelo computacional e com certo grau de formalidade, que possa ser usado pelos desenvolvedores. Não há, a princípio, uma guia que indique a adequação da automação ou que novos resultados podem ser obtidos. O usuário, por não ter acesso a sistemas de informação que executem a mesma atividade, tem pouco conhecimento sobre o que é possível fazer, ou não tem ideia de qual o custo de produzir certos resultados.

Já os projetos de redesenvolvimento apresentam a vantagem de possuir uma base que pode ser utilizado como referência do que deve ser feito (repetido), do que não deve ser mantido (eliminações) e das novas atividades necessárias. O usuário, acostumado e experiente com um sistema existente, pode fornecer informações mais adequadas sobre o que espera do novo sistema, ou da manutenção ou melhoria sendo feita. Há um risco, porém. Quando o sistema antigo ainda funciona, muitos projetos de redesenvolvimento acabam falhando, ou por não atingir claramente a mesma qualidade ou funcionalidade do sistema anterior, por alguma percepção de risco que exista sobre o uso da nova versão, ou ainda por questões políticas da organização, como pressão exercida pelos responsáveis do sistema ainda rodando.

8.4. Porque são feitos projetos de SI

Muitos são os motivos que influenciam o início de um projeto de desenvolvimento de um Sistemas de Informação. Em geral, usando um raciocínio econômico, podemos dizer que um projeto é iniciado quando o benefício do retorno esperado supera o custo do projeto . O problema é que não é fácil converter esses valores em números normalmente.

8.4.1. Benefícios do Sistema

Vários motivos podem ser analisados como benefícios esperados de um projeto. O principal benefício que um sistema de informação pode oferecer é melhorar significativamente o negócio do usuário, aumentando seu lucro. Porém, essa não é a única motivação possível.

Uma motivação comum é modernização de um sistema. Com o tempo a tecnologia de um sistema vai se tornando superada. Isso faz com que o risco e o custo de manter o sistema funcionando naquela tecnologia aumentem, aumentando gradativamente o interesse de se transportar o sistema para uma nova plataforma. Simultaneamente, novas tecnologias apresentam novas oportunidades, como desempenho superior ou facilidade de aprendizado, aumentando também com o tempo o interesse nessa atualização. Chega um momento então que passa a valer a pena o investimento em modernização.

8. Sistemas de Informação

Outro motivo importante é a mudança de premissas básicas do negócio, causada pela atuação da firma no mercado. Essas mudanças tanto podem de dentro da empresa quanto podem ser provocadas por mudanças na legislação ou por ação dos concorrentes. Por exemplo, há alguns anos atrás, no Brasil que convivia com inflações altíssimas, foram feitas várias modificações nos sistemas financeiros das empresas para aceitar a mudança de moeda do país e o convívio com moedas diferentes simultaneamente. Em outro exemplo, com a invenção e grande aceitação dos sistemas de premiação por viagens ou por milhas, as companhias aéreas tiveram que desenvolver sistemas específicos, interagindo com seus sistemas de passagens, para tratar do assunto. Muito comum também é a mudança de uma atividade da empresa, seja por um processo contínuo, como o de qualidade total, quanto por processos radicais como os de reengenharia e *downsizing*.

Os sistemas de informação também são importantes por oferecerem as empresas uma capacidade maior de competição. Com a informação correta e com os processos corretos de tratamento da informação uma empresa pode ter um diferencial de qualidade no mercado. Por outro lado, se todo um mercado já adotou um tipo de sistema, ou se pelo menos um concorrente já o fez, a empresa que não tem um sistema equivalente fica prejudicada na competição. Esse tipo de efeito foi visto quando as companhias aéreas passaram a vender passagens via Internet. No início era mais uma propaganda, depois passou a ser um diferencial positivo. Atualmente todas as companhias aéreas possuem formas de vender passagens diretamente via Internet, sendo uma funcionalidade obrigatória para as empresas.

Hoje em dia um grande motivador de novos projetos e a busca por melhor tratamento da informação que já existe em sistemas de tipo operacional, como a criação de Sistemas de Suporte Executivo. Nesse caso, o dado básico do sistema já está quase que totalmente mapeado e o que ocorreram são consultas e transformações. Isso é tão comum que novas áreas de T.I. apareceram ao redor desses sistemas, identificadas por *buzzwords* como *Data Warehouse* e *Business Intelligence*.

8.5. O Poder está com o usuário

Um dos acontecimentos mais marcantes da computação é a transferência do poder daqueles que operavam as máquinas, os famosos e muitas vezes odiados CPDs os Centros de Processamento de Dados para o usuário final.

Para aqueles que só chegaram ao mundo da informática agora, ou para aqueles que nasceram após a revolução causada pelos microcomputadores, é muitas vezes difícil de entender a complexidade e a mística que cercavam os grandes computadores. Resfriados a água, mantidos em salas seguras, gastando espaço e energia, esses computadores da década de 1970 tinham o poder computacional por vezes menor que um telefone celular do início do século XXI. Eram esses computadores, porém, que mantinham os dados fluindo, as contas e salários sendo pagos, à custa de uma vigilância permanente de seu funcionamento e do uso de recursos. Até hoje, muitos serviços críticos funcionam em

versões modernizadas desses computadores, geralmente a custos altos, por causa do alto risco de transferência para outras tecnologias .

Parte da transferência de poder aconteceu quando os microcomputadores chegaram em grande quantidade as empresas, permitindo que os usuários que não eram atendidos no prazo e na qualidade que esperavam, tomassem a rédea do processo de software, desenvolvendo seus próprios aplicativas, usando planilhas eletrônicas e sistemas simples de banco de dados (como dBase II e Access) e, muitas vezes, passando por cima da estrutura da própria organização e contratando soluções terceirizadas, já que não precisavam do computador central para executá-las.

Isso alterou drasticamente a estrutura de poder das organizações, que eram fortemente dependentes dos dados processados de forma central, em grandes máquinas, com software de ciclo de desenvolvimento muito longo. O processo de mudança não poupou carreiras, sendo que algumas empresas simplesmente fecharam seus CPDs, terceirizando suas atividades e despedindo ou transferindo todos seus funcionários.

Hoje em dia o próprio nome CPD é estigmatizado. Cabe agora ao setor de TI tecnologia da informação manter uma estrutura muito mais complexa que a anterior, unificando sistemas de várias gerações, em redes com grande quantidade de computadores executando sistemas operacionais diferentes e aplicações diferentes. Ainda existem conflitos entre o pessoal de TI e as outras partes da empresa, mas o foco cada vez mais é melhorar o negócio e atender melhor os usuários.

8.6. Exercícios

Exercício 8.6-1: Vá para o site <http://jogodeanalisedesistemas.xexeo.net/> e visite a Livraria Resolve. Descreva que sistemas de informação a livraria precisaria ter para funcionar.

DRAFT

9

Introdução a Engenharia de Software

Most software today is very much like an Egyptian pyramid with millions of bricks piled on top of each other, with no structural integrity, but just done by brute force and thousands of slaves.

(Alan Kay)

Conteúdo

9.1.	O que é Software	78
9.2.	Por que software é importante	81
9.3.	Quais os riscos que software traz?	83
9.4.	Por que é difícil fazer software?	85
9.5.	O que é Engenharia de Software	89
9.6.	Por que a Engenharia de Software é necessária?	90
9.7.	Qual o Segredo do Sucesso de um Projeto de Software	91
9.8.	O Corpo de Conhecimento da ES	92
9.9.	As normas da Engenharia de Software	93
9.10.	SEMAT	94
9.11.	Quem é o Engenheiro de Software	97

Por que Engenharia de Software?

Sistemas de Informação são entregues às organizações na forma de software. Para produzir software são usadas práticas de Engenharia de Software, como as várias práticas que são usadas em análise de sistemas de informação. Por isso é importante entender o que é realmente software, porque é difícil construir-lo e como a Engenharia de Software fornece práticas e métodos para realizar essa atividade.

Este capítulo faz uma pequena introdução a área de Engenharia de Software.

9.1. O que é Software

Software é uma palavra da língua inglesa usada em contraposição a palavra **hardware**, que descreve máquinas e aparatos físicos de forma geral. Assim, um computador seria dividido em uma parte física, composta de circuitos eletrônicos, e uma parte não-física, composta do programa que o faz funcionar.

 Quem inventou a palavra software?

Há dúvidas sobre quem passou a usar a palavra no contexto da computação, sendo que Alan Turing é um dos citados, e Paul Niquete reivindica ter usado o termo ao falar desde 1953. Normalmente o termo é associado a sua primeira publicação, por Tukey (1958). Um registro anterior, de 1956, em um artigo de Richard R. Carhart(Niquete, 2006) usa a palavra software com a acepção de pessoas envolvidas com o computador. Em francês existe a palavra *logiciel*, que os portugueses traduziram para logiciário. Este termo foi tentado no Brasil mas não chegou a ter força

Atualmente o termo software é associado não apenas ao programa de computador, mas também as suas estruturas de dados e sua documentação. Pressman e Maxim (2014) definem **software** como:

1. **instruções** (programas de computador) que quando executadas fornecem as funções, características e desempenho desejados;
 2. **estruturas de dados** que habilitam o programa a manipular informação de forma adequada, e
 3. **informação** que descreve a operação e uso do programa.

Já a IEEE define software como “programas de computador, procedimentos, e, possivelmente, documentação e dados pertinentes a operação de um sistema computacional”(IEEE, 2017).

9.1. O que é Software

A Figura 9.1 mostra uma página do Sistema Acadêmico do CEDERJ e parte do código para sua criação. Apesar de HTML não ser uma linguagem de programação, é parte do que chamamos de software.



Figura 9.1.: Um software executando na web e parte do seu código.

Um computador atual não funciona sem software. Na verdade, o computador que usamos para nossas tarefas diárias possui camadas e camadas de software para funcionar. É o software que instrui a máquina construída com circuito eletrônicos sobre os passos que ela deve seguir. Mesmo em uma CPU moderna, que normalmente vemos como *hardware*, já existe software e até sistemas operacionais instalados(Vaughan-Nichols, 2017).

Software é considerado um bem imaterial. É bastante difícil, por exemplo, calcular o seu **valor**. Quanto foi gasto para produzi-lo, quanto ele traz de lucro, quanto é gasto para mantê-lo, quanto seria gasto para substitui-lo seriam alternativas de fazer isso. Produtos como o sistema de reserva de passagens SABRE, originalmente criado para a American Airlines(Head, 2002), se tornaram tão importantes que se transformaram em negócios isolados(Levere, 2001), adquirindo valor próprio como negócio.

Outra característica de software é que seu custo de desenvolvimento é basicamente composto pelo esforço das pessoas envolvidas em produzi-lo. Software não exige ou consome matéria prima, e o uso de máquinas para produzi-lo é basicamente proporcional ao número de computadores necessários para as pessoas realizando tarefas, mais uns poucos computadores necessários para manter atividades automáticas. Isto não significa, porém, que seja barato desenvolver software.

Software pode ser encontrado de várias formas: texto em arquivos digitais ou impressos, desenhos, arquivos binários ou em outros formatos, bits na memória do computador, documentações em diferentes mídias, como texto, e vídeos. Este documento, por exemplo, é possuí uma versão digital no formato PDF que contém, dentro dele, programas em PostScript que instruem como deve ser impresso ou visualizado. Este é um exemplo da ubiquidade do software, isto é, ele está presente ao nosso redor e não percebemos mais essa presença.

9.1.1. Como cobrar? A pergunta mais comum nos cursos de ES.

Uma expectativa muito comum do iniciante na Engenharia de Software é que ela vai poder ser usada facilmente para responder a pergunta: quanto devo cobrar pelo software que produzo? Esse valor, porém, não é discutido na área. A questão importante para a Engenharia de Software é o **custo**, não o **preço**. O preço depende de outros fatores, como a oferta e a demanda do mercado. Preços de produtos e serviços são estudados por outras áreas do conhecimento, como a Economia e o Marketing. Mesmo assim, é possível apresentar algumas informações sobre preço.

É verdade que muitas vezes o software é cobrado em função do seu custo, ao qual é adicionado uma margem de lucro. Então, quem escolher essa forma de trabalho vai encontrar, nesse livro, algumas informações úteis.

O custo de software poder ser previsto ou calculado de várias maneiras, como por meio de horas trabalhadas, linhas de código, relatórios e telas, as estatísticas que permitem alguma comparação estão relacionadas a uma medida conhecida como Pontos de Função, que será tratada no Capítulo 26. No caso da previsão, sempre haverá uma incerteza, tanto por motivos da essência do software, quanto pela capacidade das técnicas. Além disso, todo projeto tem riscos. A Engenharia de Software tem entre suas metas aumentar a precisão das previsões e diminuir o risco de projeto.

Nesta forma de cobrança, é preciso prever o custo de alguma forma, e isto é feito em função do tamanho do esforço necessário para produzir o software, que pode ser equacionado a partir da funcionalidade entregue ao cliente. A medida mais aceita em todo mundo, principalmente pelos governos e órgãos regulatórios e de padronização é o Ponto de Função(IFPUG, 2010). Esse assunto será tratado no Capítulo 26. No momento é razoável saber que um relatório simples custa 4 pontos de função.

No mercado mundial, segundo Czarnacka-Chrobot (2012), com dados do ISBSG, um ponto de função custa entre US\$ 300 e US\$ 1000, com uma média de US\$750, mas foram encontrados projetos com valores desde US\$ 17 até US\$ 2727 na base consultada, incluindo os *outliers*. São dados que incluem custos não só de desenvolvimento, mas também de apoio.

Com esses mesmos dados, o preço por hora varia entre US\$60 até US\$105, com uma média de US\$ 80, havendo extremos entre US\$7 e US\$ 570.

Um ponto de função pode custar entre US\$ 1600 em média no Japão, até US\$ 125 em média na Índia. Não é de se espantar que tanto software seja terceirizado para a Índia. Em relação ao tamanho dos projetos, é comum que quanto maior o software, maior o preço do ponto de função.

No Brasil, devido a regulamentação do governo, software é muitas vezes precificado de duas formas: pelo uso de Pontos de Função (PF) e por Unidades de Serviço Técnico (USTs). USTs são acordadas por projeto, havendo uma validação com a prática do próprio governo em outros projetos. Os preços oscilam entre R\$ 250,00 até R\$ 1.500,00(FATTO, 2020). É importante frisar que existe um regra de contagem de Pontos de Função

destinada aos contratos do governo brasileiro que vai além do modelo original ou do padrão IFPUG(Brasil, 2018).

Isso significa que, para o governo brasileiro, uma tela com um relatório simples, como a quantidade de pedidos de aposentadoria por estado, por exemplo, pode custar algo entre R\$ 1.000,00 e R\$ 6.000,00.

A variação de preços é explicada por motivos como vários fatores de tecnologia, incluindo linguagem de programação, requisitos de operação continuada e de segurança, e ainda variações do mercado, como a oferta e demanda de profissionais, e outras.

9.2. Por que software é importante

Basicamente, software, na forma de dados e programas de computador, é o que hoje faz funcionar, monitora e controla todo o mundo. Se no início os computadores eram máquinas extremamente pesadas e caras que viviam em porões resfriados, trabalhando em modo *batch*, hoje praticamente todo equipamento que usamos possui alguma automação, seja das grandes usinas geradoras de energia a cafeteira elétrica que usa essa energia.

Como tudo começou

O primeiro programa de computador, criado para um computador mecânico que nunca ficou pronto, foi feito por Ada Augusta King, nascida Byron, condessa de Lovelace. Ela criou e publicou um programa para a Máquina Analítica de Babbage, mostrando suas potencialidades.

Toda infraestrutura depende da geração de energia, que é planejada e controlada por meio de software, em vários níveis da produção. No Brasil, por exemplo, a ONS, o Operador Nacional do Sistema Elétrico, tem sistemas sofisticados que sabem o status de cada peça usada na transmissão de energia. Toda informação é trocada de forma digital, dos telefones às grandes redes de televisão, passando pela internet propriamente dita. Automóveis chegam a ter dezenas de processadores. Aparelhos domésticos podem se conectar à internet.

Por exemplo, a foto da Figura 9.2 mostra a sala de controle de uma usina de energia nuclear de tecnologia original americana, construída antes da massiva transformações das estruturas de comando e controle dessa indústria para software. É possível ver nessa foto dezenas de controles de hardware, porém também é possível ver que parte dos controles já foi modernizada para sistemas de software.

Assim, dentro dessa realidade, toda a economia é extremamente dependente do funcionamento de programas de computador de diferentes graus de sofisticação. Um freio ABS, por exemplo, possui software em várias camadas, uma servindo como defesa dos erros que as outras podem cometer.

Outra caracterização da importância pode ser dado pelo tamanho da indústria de

9. Introdução a Engenharia de Software



Figura 9.2.: Sala de controle de uma usina nuclear. Fonte: foto de Rodrigo Costa dos Santos

software mundial, avaliada em 2014 em 407 bilhões de dólares pelo Gartner group(Group, 2014), enquanto toda a indústria de TI chegaria a 5.3 trilhões de dólares em 2020(CompTIA, 2019). Além disso, em 19 de maio de 2021, cinco das seis maiores empresas em valor de mercado eram empresas de tecnologia da informação cujo principal diferencial é seu software: Apple, Microsoft, Amazon, Alphabet e Facebook. Essas são enormes empresas de software de uso geral, existe um mercado ainda para software dedicado, como os que controlam um aparelho médico ou fazem uma usina de energia funcionar.

O custo de software geralmente excede o de hardware, até mesmo em sistemas complexos. Além disso, várias funções que antes eram feitas por hardware, hoje são feitas por software, o que apresenta vantagens, mas também aumenta a complexidade do que pode ser feito e consequentemente aumenta a complexidade do software, e também gera novos riscos para os sistemas.

Outra tendência é a virtualização do hardware. Isso significa que funções típicas de hardware são repassadas para camadas de software, sendo que isso acaba afetando também a forma de se pensar o hardware. A Figura 9.3 mostra uma arquitetura que pode ser encontrada em um servidor hoje em dia¹.

9.2.1. A Ubiquidade do Software

Imagine uma pessoa qualquer em uma cidade moderna qualquer. Vamos chamá-la de João. João acorda porque toca um alarme em um relógio digital. Esse relógio está ligado na rede elétrica, que é controlada por software. O relógio contém software dentro dele.

¹É interessante notar que também existe um movimento contrário, de trazer funções desenvolvidas em software para o hardware, permitindo por sua vez que o software possa ser mais sofisticado, em um ciclo de *feedback* positivo, o que aconteceu por exemplo na sofisticação das placas gráficas(Savage, 2020)

9.3. Quais os riscos que software traz?

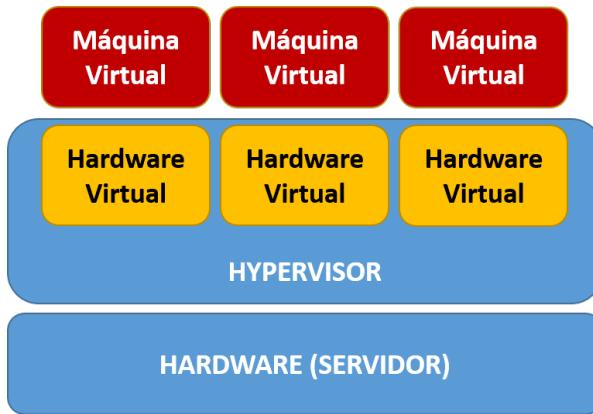


Figura 9.3.: Esquema da virtualização do hardware.

João levanta e liga a TV digital, que contém software, para ver as notícias. Todos os equipamentos da estação de notícias são digitais e contêm software. Ele olha seu celular, com sistemas operacionais e aplicativos altamente sofisticados, para verificar mensagens que vieram em uma rede de dados que contém camadas e camadas de software. Em 10 minutos de vida, João utilizou provavelmente centenas de milhões de linhas de código sem ao menos perceber sua dependência da indústria de Tecnologia da Informação.

A verdade é que o software, e toda a Tecnologia da Informação, se tornou algo ubíquo. Essa palavra indica um tipo de ideia ou objeto que é pervasiva, está em todo lugar, ao ponto que não percebemos mais sua presença.

9.3. Quais os riscos que software traz?

Sendo tão importante, é claro que software também traz riscos às organizações e às pessoas. As duas maiores facetas desse risco são as perdas de vida humana, animais ou bens, por erro do software, e as grandes perdas financeiras causadas por software ou por projetos de software.

Acidentes de aviação onde houve influência do software tiveram grande repercussão na mídia, como o caso dos aviões Boeing 737 Max. Um dos casos mais discutidos tecnicamente, ligado a perda de vidas humanas é o do equipamento de terapia por radiação Therac-25(Levenson e Turner, 1993), onde uma falha em um sistema de proteção feito em software causou que doses altíssimas fossem aplicadas a várias pessoas, causando morte ou lesões em pelo menos 6 pessoas.

Alguns casos são interessantes de citar. O primeiro lançamento do foguete europeu Ariane V, em 1996, resultou em uma autodestruição automática devido a uma exceção não tratada no software(Lions, 1996), não só atrasando o projeto em anos, mas também criando um problema para a imagem do projeto e impactando o prêmio do seguro de lançamento de satélites para toda a indústria. No Brasil, a Volkswagen, em 2006, anunciou

9. Introdução a Engenharia de Software

o *recall* de 123 mil veículos para trocar um software que controlava o funcionamento dos componentes eletrônicos dos automóveis(Folha Online, 2006). No caso, alguns dos automóveis tinham o chip dentro do motor, o que obrigava a abri-lo, operação complicada e que ninguém gosta de fazer em seu carro.

Já em prejuízos financeiros os casos se repetem ano após ano. Pequenos problemas de software, como aplicativos de celular parando de funcionar, ou erros de cálculo em fórmulas que usamos em planilhas eletrônicas, podem ter um grande valor agregado na economia, se somadas todas as horas de trabalho perdidas ou ainda os prejuízos reais causados. A Tricentis (2019) estimou que as perdas devidas a falhas de software somaram US\$1,72 trilhões em 2017, afetando 3,68 bilhões de pessoas e com uma perda acumulada de 268 anos. Já Krasner (2021) identificou um custo total de US\$ 2.08 causado por baixa qualidade de software, com mais US\$ 1.31 trilhões a serem gastos no futuro para correções.

Projetos inacabados com altos custos desperdiçados são também bastante comuns. Um dos maiores fracassos relatados aconteceu nas décadas de 80 e 90, quando a Administração Federal de Aviação americana (FAA) tentou criar um novo sistema de controle do tráfego aéreo. Reconhecido como ambicioso demais, um dos seus requisitos era de 99.99999% de confiabilidade, o sistema foi planejado para custar, na época, US\$ 500 por linha de código, o que já era cinco vezes mais do que o padrão da indústria para processos bem gerenciados. Em 1995 o custo já tinha chegado de US\$ 700 a US\$ 900 por linha de código. Inicialmente orçado em US\$ 2.5 bilhões, em 1994 as estimativas subiram para US\$ 7 bilhões e o projeto foi abandonado, sendo que apenas algumas de suas partes foram adotadas como partes de projetos menores(Gibbs, 1994).

Entre outros grandes fracassos podemos adicionar o *FBI* tendo que jogar fora US\$ 170 milhões em um sistema de apoio a identificação de terroristas, a *Ford Motor Company* abandonando um projeto de US\$ 200 milhões de um novo sistema integrado de compras e o *McDonald's* declarando US\$ 170 milhões em prejuízo por um software para automatizar toda sua operação (Carr, 2005).

Não apenas erros, mas falhas de design na interface, e também falta de dados em software funcionando, também já influenciaram acidentes graves. Os acidentes com aviões B757, do vôo AA965 em Cali, de 20/12/1995, com 159 fatalidades, e do vôo ALW301 em *Puerto Plata*, de 6/2/1996, com 189 fatalidades, foram, em parte, associados a problema na interface homem-computador(Ladkin, 1996).

Problemas de software podem também fazer com que equipamentos não cumpram sua missão de proteção. O acidente do vôo 801 da *Korean Air*, em 3/12/1984, em Guam, que causou 228 mortes, por exemplo, poderia ter sido prevêido se não houvesse um erro no sistema do Radar de Altitude Segura Mínima(Bellovin, 1997).

Existem muitos outros riscos que são relatados diariamente. Equipamentos podem ser controlados por pessoas estranhas a sua operação, câmeras podem ser monitoradas por pessoas sem esse direito, dados pessoais podem ser obtidos de forma ilegal ou legal mas sem o conhecimento dessas pessoas, com grandes ameaças a privacidade, defeitos ou *bugs*

podem fazer equipamentos pararem de funcionar a qualquer momento, etc. Uma boa fonte de notícias sobre o assunto é o *The Risks Digest: Forum on Risks to the Public in Computers and Related Systems*, moderado por Peter G. Neumann².

9.4. Por que é difícil fazer software?

A Engenharia de software nasceu porque fazer software é difícil. O nome foi estabelecido em uma conferência da OTAN em 1968 cujo objeto era descobrir como resolver um problema de engenharia: o projeto, a produção e a manutenção com sucesso de sistemas de software úteis(Naur e Randell, 1969).

Dijkstra descreveu bem o problema de desenvolver software em 1972, na sua palestra de aceitação do prêmio Turing. Se colocando no lugar de um programador do futuro, ou seja, aproximadamente agora, ele conta: “quando não havia máquinas, programar não era um problema de forma alguma; quando nós tínhamos poucos computadores fracos, programar se tornou um problema, e agora nós temos computadores gigantescos, programar se tornou um problema gigantesco”(Dijkstra, 1972). Ou seja, como temos computadores superpoderosos podemos tentar resolver problemas dificílimos.

Além de difíceis, grande parte dos problemas que tratamos são conhecidos como **wicked problems**, cuja tradução poderia ser problemas perversos. Esses problemas se caracterizam por (Conklin, 2005):

- Não serem entendidos até que uma solução seja desenvolvida
- Não haver uma regra de parada para a solução
- Suas soluções não serem certas ou erradas, mas sim melhores, aceitáveis ou não boas o suficiente
- Serem essencialmente únicos e novos
- Cada solução possível ser uma operação única e com consequências
- Não possuírem soluções alternativas

Somam-se a isso outros desafios.

Um deles é a necessidade de atender as demandas das pessoas em um contexto social, compondo diferentes partes interessadas. Isso implica em problemas de comunicação, política organizacional, etc. A complexidade social (Conklin, 2005) geralmente ocorre junto com um problema perverso (*wicked problem*) e é função do número e da diversidade de partes interessadas envolvidas em um projeto.

²<http://catless.ncl.ac.uk/Risks/>

Partes Interessadas

Esse tópico será tratado detalhadamente no Capítulo 12. Um definição simples é “toda pessoa, organização ou entidade que pode ser, ou acredita que pode ser, beneficiada ou prejudicada por um projeto.

Outro problema é o técnico. Projetos de software envolvem pessoas trabalhando em um grande número de partes. Um software de funcionalidade relativamente simples pode ter milhares de linhas de código. Projetos podem durar anos. Tudo isso implica na necessidade de técnicas de gestão de projeto, garantia de qualidade, e outros assuntos que compõe a Engenharia de Software.

Em especial, não só a complexidade de um projeto de software aumenta de forma mais que linear com a quantidade de funcionalidade entregue, quanto os erros de estimativa aumentam mais ainda (Gibbs, 1994). Isso é bem diferente de grande parte da indústria, onde produzir mais pode inclusive baixar os preços.

Para entender o tamanho de um software atual, é possível lembrar que foram necessárias 145.000 linhas de código para o software de orientação da Apollo XI, já o sistema Android versão 2.2 continha 12 milhões de linhas de código(P. Johnson, 2012). A Figura 9.4 mostra o tamanho comparativo de alguns software conhecidos, que chega a 86 milhões de linhas de código para o Mac OS X 10.4. Isso mostra que mesmo um programa pequeno que funcione em sistema operacional como Android, Windows ou MacOS depende de milhões de linhas de código funcionarem corretamente. Para comparação, uma das máquinas mais complexas existente no mundo, o Boeing 747, tem aproximadamente 6 milhões de partes distintas, e um automóvel apenas 30 mil.

Outra questão importante do software é que dificilmente ele se mantém constante ao longo do tempo quando em operação. Dois são os motivos dessa mudança constante: a descoberta de *bugs* que precisam ser consertados e a alteração dos requisitos originais com a evolução do ambiente em que funcionam. Por exemplo, em um software de gerência de recursos humanos, qualquer alteração nas leis trabalhistas pode exigir uma modificação no software para que fique de acordo com as novas leis. O problema é que qualquer modificação no software tem uma chance de inserir novos *bugs*.

A maioria dos autores de Engenharia de Software concorda que a curva de falha do software, que temos dificuldade de tratar, Figura 9.6, é bem diferente da curva de falha do hardware, Figura 9.5, a qual já é bem modelada estatisticamente.

9.4.1. O Relatório CHAOS

Pesquisas de campo também mostram que muitos projetos de software falham ou acabam com problemas. Desde de 1994 o *Standship Group* realiza uma pesquisa chamada “CHAOS Report” que mostra o estado dos projetos de software. As medidas básicas para determinar a resolução de um projeto são (The Standish Group, 2015):

9.4. Por que é difícil fazer software?

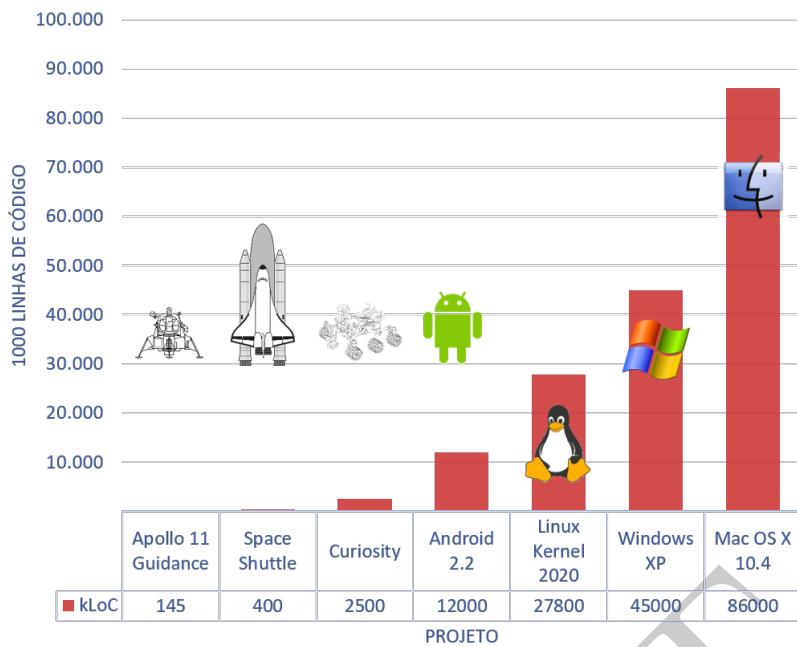


Figura 9.4.: Tamanho de código de alguns projetos conhecidos.

- dentro do **orçamento**;
- dentro do **prazo**, e
- satisfatório em relação a **funcionalidade**.

Essa pesquisa divide os projetos por resolução, em: **fracassos**, que foram cancelados ou não foram usados, **com problemas**, em uma ou duas das três medidas básicas, e **bem sucedidos**³. Na versão de 2015, com uma base de 2011 a 2015, os percentuais mostraram números preocupantes, que podem ser vistos na Tabela 9.1, principalmente para projetos grandes.

Tabela 9.1.: Resolução de projetos de software por tamanho e por método de desenvolvimento, para mais de 10.000 projetos. Fonte:(The Standish Group, 2015)

Tamanho	Método	Bem Sucedido	Com Problemas	Fracassos
Grande	Ágil	18%	59%	23%
	Cascata	3%	55%	42%
Médio	Ágil	27%	62%	11%
	Cascata	7%	68%	25%
Pequeno	Ágil	58%	38%	4%
	Cascata	44%	45%	11%

Como podemos ver, ainda na Tabela 9.1, a taxa de fracassos aumenta muito com o

³Os termos originais são *failed*, *challenged* e *successful*.

9. Introdução a Engenharia de Software

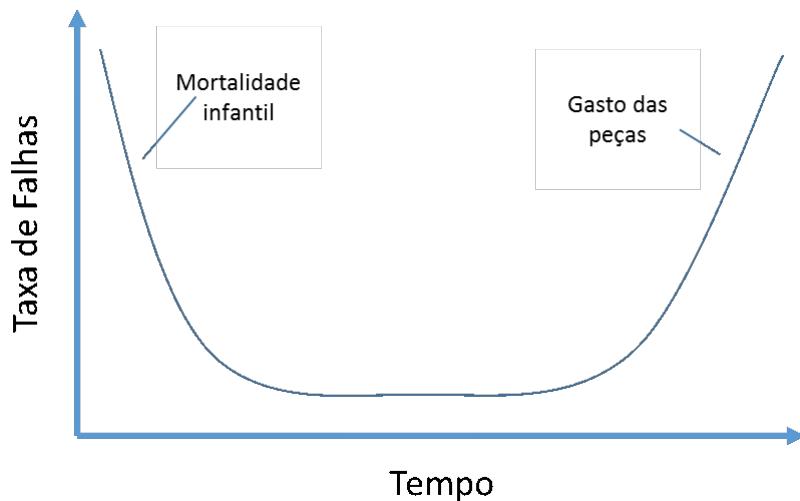


Figura 9.5.: Curva de falha do hardware. Fonte: (Pressman e Maxim, 2014)

tamanho, principalmente para projetos feitos com a metodologia mais tradicional, em Cascata. Além disso, a taxa de sucesso para projetos médios e grandes está abaixo de 30% mesmo para projetos feitos com a metodologia ágil, e menor que 10% para a metodologia tradicional. Isso é uma constatação da dificuldade de fazer projetos de software.

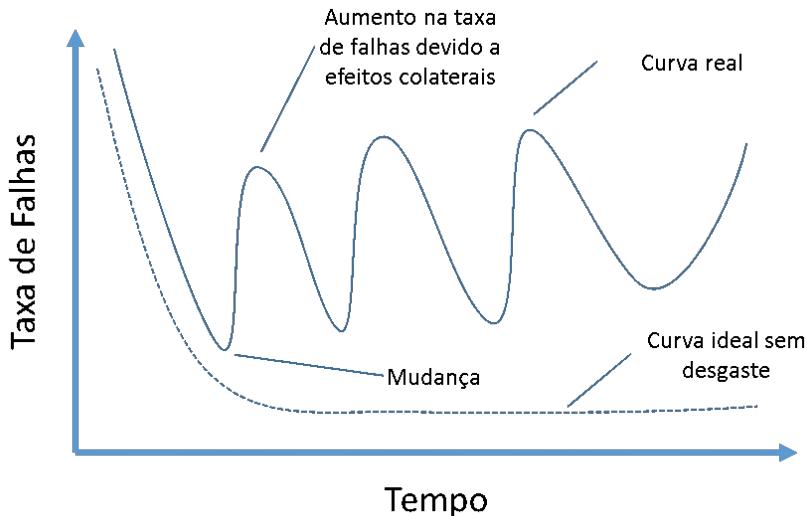


Figura 9.6.: Curva de falha do software. Fonte: (Pressman e Maxim, 2014)

9.5. O que é Engenharia de Software

Engenharia de Software é “a aplicação de uma abordagem sistemática, disciplinada e quantificável ao desenvolvimento, operação e manutenção de software, i.e., a aplicação da engenharia ao software”(IEEE Computer Society, 2014a).

O termo foi criado em torno de 1968, perante a necessidade de melhorar o desenvolvimento de software, que desde o início se mostrou algo bastante difícil. Pesquisas mostram não só que muitos projetos de software falham, parcial ou totalmente, quanto o aumento drástico da probabilidade de fracasso de um projeto software com o aumento da complexidade do mesmo(The Standish Group, 1994).

Segundo o SWEBOK, a “Engenharia de Software é a profissão preocupada com a criação e manutenção de aplicações de software pela aplicação de tecnologias e práticas da Ciência da Computação, Gerência de Projetos, Engenharia, domínios de aplicação e outros campos”(IEEE Computer Society, 2014a).

Uma abordagem mais pragmática é pensar a Engenharia de Software como o estudo das formas e as próprias formas de desenvolver software de maneira custo-efetiva. Fornecer os maiores benefícios possíveis dentro de um conjunto de restrições é um dos principais objetivos da Engenharia que se aplica também a Engenharia de Software.

As metas da Engenharia de Software incluem atender o cliente que precisa do software, gerar valor para as organizações que produzem e utilizam o software e deixar satisfeitos, também, aqueles que produzem o software.



Figura 9.7.: Pirâmide que explica a construção de conceitos na Engenharia de Software.

9.5.1. Uma Visão da Engenharia de Software

Pressman e Maxim (2014) construíram uma pirâmide de 5 níveis para explicar a Engenharia de Software, em uma abordagem bastante técnica. Porém, é necessário mostrar também o que motiva essas técnicas, o que é parte significativa do problema da Engenharia de Software.

Desse modo, a Engenharia de Software pode ser vista por uma pirâmide de 7 níveis, Figura 9.7, onde tudo se baseia em atender as **partes interessadas**, pela agregação de **valor**. Isso só pode ser feito por meio de produtos e serviços de **qualidade**, o que exige **processos**, onde são usados **métodos**, de acordo com **práticas**, que exigem **ferramentas**. Durante o livro, todos esses assuntos serão tratados.

9.6. Por que a Engenharia de Software é necessária?

Precisamos da Engenharia de Software porque não é possível fazer software “de qualquer jeito”. O Ciclo de Vida de um software, de qualquer tamanho, é um projeto que se inicia na descoberta do que se deve fazer e termina quando o software, depois de entregue e usado, é retirado do funcionamento. A fase de suporte e manutenção normalmente dura muito mais que a de desenvolvimento. Para tudo isso é preciso usar técnicas típicas da engenharia, como a construção de modelos abstratos do software que deve ser criado, estimativas de prazo e custo, garantia da qualidade do produto, etc.

9.6.1. Diferenças para outras engenharias

Mas o que há de diferente entre Engenharia de Software e outras engenharias? A princípio, fazer software poderia ser considerado até mais fácil, já que na maioria das engenharias, após construirmos um artefato, temos que reproduzi-lo por um processo industrial. Software pode ser simplesmente copiado, com uma taxa baixíssima de falha

9.7. Qual o Segredo do Sucesso de um Projeto de Software

na cópia e, mesmo em mídias que degradam, de fácil verificação. Assim, a Engenharia de Software se parece mais com a parte das engenharias que costumamos chamar de desenvolvimento do projeto ou desenvolvimento do protótipo final.

Por outro lado, em um projeto de software cada linha programada tem importância. E a quantidade de linhas programadas, ou itens que devem ser analisados separadamente, e ainda a interação entre elas, seja por meio de código ou dados, é enorme. Assim, construir software é, de várias maneiras, muito mais complexo que construir uma casa. Se um tijolo da casa não tem um pequeno pedaço, não há problema nenhum. Se um grão de pó de um elemento químico em um processo químico tem um peso inferior a média, o problema certamente também não existirá. Mas troque uma letra⁴ em um programa de computador e você pode ter um programa que nem compila, ou funciona, porém, executa de forma absolutamente errada⁵.

Outro problema é a dificuldade de saber o que é realmente necessário, devido aos problemas naturais de comunicação entre as pessoas, principalmente usuários e desenvolvedores. Além disso, muitas vezes não se sabe o que deve ser feito, mas sim o problema que deve ser resolvido, o que implica em processos de investigação ainda mais complicados.

9.7. Qual o Segredo do Sucesso de um Projeto de Software

Como vimos, desenvolver software é difícil. A área de Engenharia de Software, então, se preocupa em encontrar formas de melhorar esse desempenho. Segundo o mesmo The Standish Group (2015), se forem utilizadas Metodologias Ágeis, o índice de fracasso cai de 42% para 23% em projetos grandes, e o de sucesso sobe de 3% para 18%.

Esse mesmo relatório relata as razões de sucesso de projetos de software, apresentadas na Tabela 9.2. Observando a Tabela, é possível perceber que os principais fatores não estão ligados diretamente a Processos, Métodos e Ferramentas, porém eles precisam ser alcançados com o uso das técnicas de Engenharia de Software, principalmente o Envolvimento do Usuário e a Otimização, que começa com a gerência de escopo com base no valor de negócio (Hastie e Wojewoda, 2015)

Ainda analisando a Tabela 9.2 e usando conhecimentos de Engenharia de Software, podemos dizer que o envolvimento do usuário é melhorado com o uso de processos ágeis.

⁴A quantidade mínima e discreta de um programa escrito em uma linguagem de programação de terceira geração ou maior

⁵Em um exemplo que aconteceu com o autor, dois espaços de uma identação em Python fizeram que um programa gerasse os dados errados durante dias, até que o erro fosse descoberto.

9. Introdução a Engenharia de Software

Tabela 9.2.: Fatores de Sucesso de um Projeto de Software. Fonte:(The Standish Group, 2015)

Fator de Sucesso	Peso Relativo
Patrocínio Executivo	15
Maturidade Emocional	15
Envolvimento do Usuário	15
Otimização	15
Recursos Capacitados	10
Arquitetura Padrão	8
Processo Ágil	7
Execução Modesta	6
Experiência em Gerência de Projetos	5
Objetivos Claros de Negócio	4

9.8. O Corpo de Conhecimento da ES

Uma das publicações mais importantes da Engenharia de Software é o *Guide to the Software Engineering Body of Knowledge (SWEBOK(R))*. Esse guia, construído colaborativamente por dezenas de pesquisadores ao redor do mundo e mantido pela IEEE Computer Society, define os conhecimentos relativos à área, dividindo-os em 15 áreas do conhecimento(IEEE Computer Society, 2014a), como descritas na Figura 9.8.

Cada área do conhecimento é, por sua vez, subdividida em tópicos e subtópicos. O SWEBOK serve como definição do escopo do que é geralmente reconhecido como Engenharia de Software e ainda como referência primária para cada tópico, devido as citações que ele contém, além de possuir uma referência cruzada entre seus tópicos e as normas ISO/IEC a eles relacionadas.

Analizando criticamente, o SWEBOK é um registro, feito em um momento específico, do que se acordou, entre uma grande quantidade de pesquisadores, do que seriam os tópicos relacionados a Engenharia de Software. Com a evolução da própria Engenharia de Software, o projeto vai ficando desatualizado.

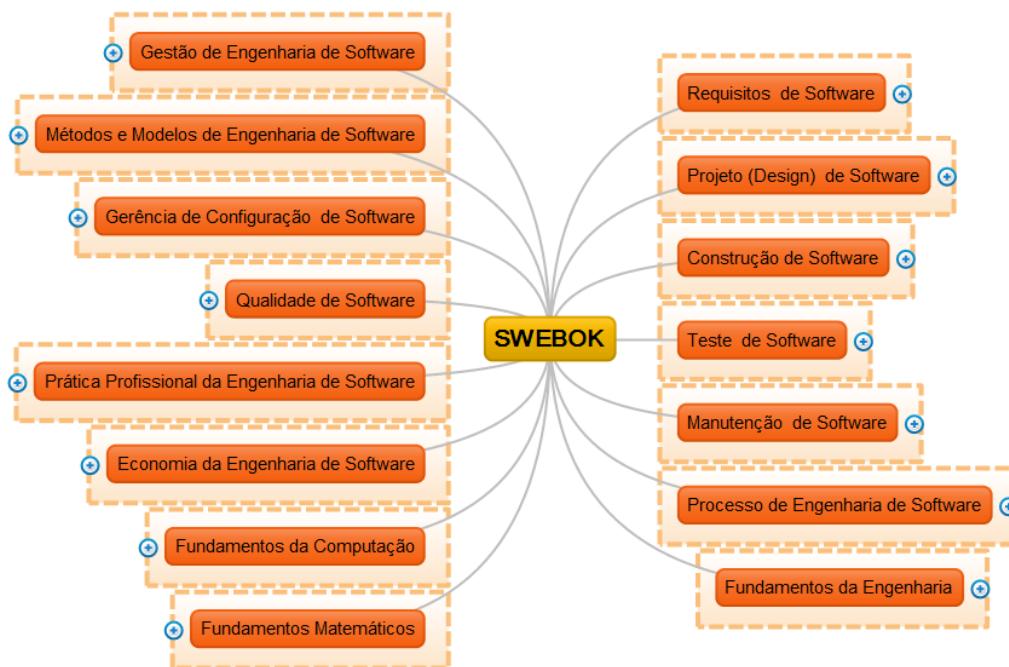


Figura 9.8.: Mind map para as áreas de conhecimento da Engenharia de Software:
Fonte:(IEEE Computer Society, 2014a)

9.9. As normas da Engenharia de Software

Uma parte do conhecimento de Engenharia de Software foi normatizada por instituições nacionais e internacionais, principalmente a *International Standards Organizations* (ISO) e o *Institute of Electrical and Electronics Engineers* (IEEE)⁶. No Brasil, as normas são criadas, normalmente por tradução, pela Associação Brasileira de Normas Técnicas (ABNT) e são numeradas com o código NBR.

As normas geralmente se referem a conceitos estabelecidos, isto é, raramente há normas sobre inovações na Engenharia de Software, porém, nos últimos anos o processo de criação tem se tornado mais dinâmico.

De grande influência na indústria em geral são as normas de qualidade (família ISO 9000), que foram especializadas para software primeiro na família ISO 9216 e depois na família ISO 25000.

⁶Pronunciado i-três-é ou *ai-Triple-i*

9.10. SEMAT

Um movimento interessante na Engenharia de Software é o SEMAT *Software Engineering Method and Theory*. Fundado por Ivar Jacobson, Bertrand Meyer e Richard Soley em 2009, o SEMAT se propõe a “refundar a Engenharia de Software como uma disciplina rigorosa baseada em uma teoria geral da Engenharia de Software e um arcabouço de processo unificado”(Jacobson, Lawson e Ng, 2019; Jacobson, Ng et al., 2013).

Seu principais resultados são o Kernel, composto de Alphas, *Abstract Level Progress Health Attribute*, representações das coisas essenciais que a Engenharia de Software trabalha, Espaços de Atividade e Competências. As Figuras 9.9, 9.10 e 9.11 apresentam, respectivamente, uma representação dessa contribuição.

O SEMAT também define métodos e práticas. Um **método** “provê um guia para todas as coisas que você precisa fazer quanto está desenvolvendo e mantendo um software”(Jacobson, Lawson e Ng, 2019). Métodos são compostos por **práticas**, que são essas coisas que são feitas, ou mini-métodos. Práticas também podem ser compostas por outras práticas(Jacobson, Lawson e Ng, 2019). Segundo Jacobson, Lawson e Ng (2019), enquanto existem muitos métodos no mundo, o conjunto de prática é bem menor. Elas também são reusáveis, podendo aparecer em vários métodos.

9.10.1. O Kernel Essencial

O Kernel Essencial é estruturado em 3 áreas: Cliente, Solução e Empreendimento. Dentro dessas áreas estão os Alphas, como mostra a figura a 9.9.] Os Alphas são as principais dimensões de um processo de software(Jacobson, McMahon e Racko, 2015).

As Oportunidades do Kernel Essencial podem ser diretamente associadas ao conceito de Valor, já que são oportunidades de agregar valor as Partes Interessadas.

As **partes interessadas**, estudadas no Capítulo 12, são “pessoas, grupos ou organização que afetam ou são afetadas pelo sistema de software”(SEMAT Inc., 2019). São as partes interessadas que fornecem **oportunidades**, o “conjunto de circunstâncias que tornam apropriada o desenvolvimento ou mudança de um sistema de software”.

Essas **oportunidades** focalizam os **requisitos**, que são “o que o software precisa fazer para endereçar as oportunidades e satisfazer as partes interessadas”(SEMAT Inc., 2019). O **sistema de software** deve atender os requisitos, sendo o “produto primário do empreendimento de engenharia de software”(SEMAT Inc., 2019). O **time** é o “grupo de pessoas ativamente engajada no desenvolvimento, manutenção, entrega ou suporte de um sistema de software”(SEMAT Inc., 2019). É o time que executa e planeja o **trabalho**, pela aplicação de um ou mais **modos de trabalho**.

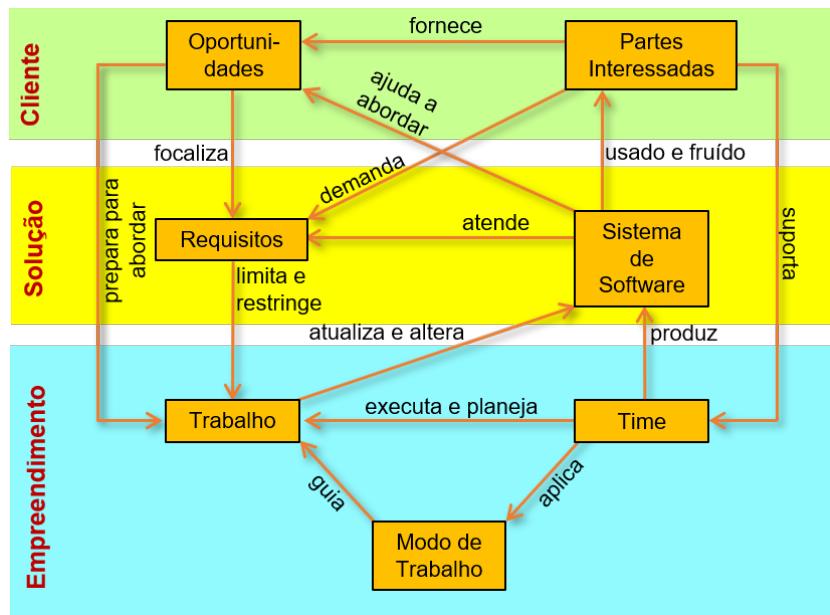


Figura 9.9.: O Kernel Essencial do SEMAT. Fonte:(Jacobson, Ng et al., 2013)

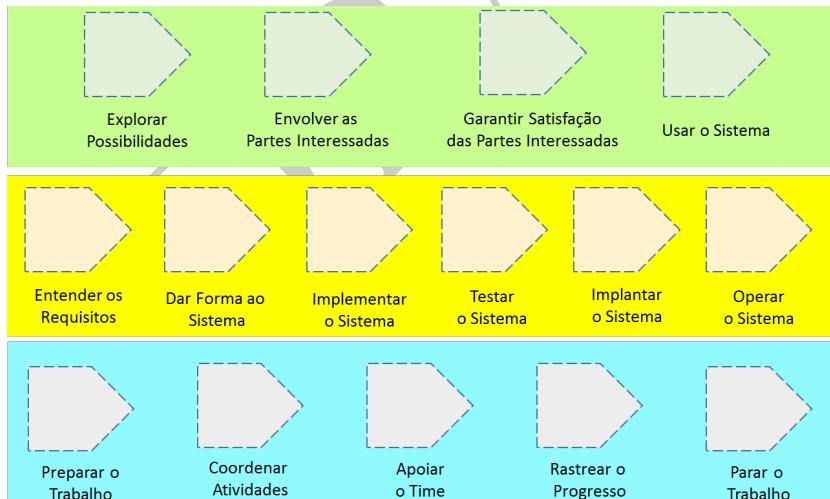


Figura 9.10.: Os Espaços de Atividades no Kernel do SEMAT. Fonte:(Jacobson, Ng et al., 2013)

9. Introdução a Engenharia de Software

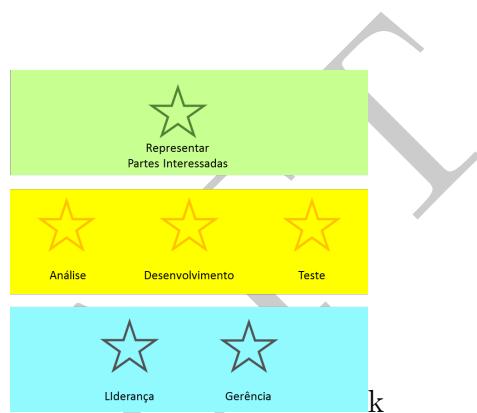


Figura 9.11.: As Competências no Kernel do SEMAT. Fonte:(Jacobson, Ng et al., 2013)

9.11. Quem é o Engenheiro de Software

Não é fácil, hoje em dia, definir exatamente o que faz um Engenheiro de Software, porque a área é muito ampla, como se pode ver na Figura 9.8.

Uma carreira profissional na Engenharia de Software pode começar com funções ligadas a testes e controle de qualidade, ou a programação. A evolução técnica e pessoal do profissional vai levá-lo a cargos de maior responsabilidade técnica e gerencial, como cuidar da arquitetura de software ou gerenciar equipes de desenvolvimento. Com o tempo, deve subir mais na hierarquia e ser responsável por projetos ou mesmo portfólios de projetos.

A IEEE Computer Society lançou em 2014 um documento que fala sobre as competências esperadas de um Engenheiro de Software(IEEE Computer Society, 2014b). Ele divide as competências em cinco grandes áreas, que são detalhadas no documento:

1. Habilidades Técnicas
2. Habilidades e Atributos Comportamentais
3. Habilidades Cognitivas
4. Conhecimento Necessário
5. Disciplinas Relacionadas

Cada uma dessas habilidades possui atividades, que podem ser classificadas em 5 níveis de competência IEEE Computer Society (2014b):

1. **Técnico**, que segue instruções;
2. **Profissional (*Practitioner*) de Nível Inicial**, que faz atividades com supervisão;
3. **Profissional (*Practitioner*)**, que faz atividades com pouca ou nenhuma supervisão
4. **Líder Técnico**, que lidera equipes e indivíduos no desempenho das atividades, e
5. **Engenheiro de Software Sênior**, que serve de Engenheiro Chefe.

Além disso, se espera que todo Engenheiro de Software seja um instrutor e mentor de outros(IEEE Computer Society, 2014b).

9.12. Exercícios

Exercício 9.12-1: Procure outra definição para Engenharia de Software. Discuta as diferenças entre essa definição e as dadas no texto.

Exercício 9.12-2: Procure acidentes ou problemas causados por software funcionando incorretamente? Procure problemas na indústria espacial e médica, por exemplo.

Exercício 9.12-3: Procure prejuízos financeiros causados por projetos de software que sofreram algum problema? Procure relatos sobre grandes projetos governamentais

9. Introdução a Engenharia de Software

ou de empresas que foram interrompidos ou não tiveram sua funcionalidade completa devido a problemas no software.

Exercício 9.12-4: Investigue prejuízos causados pelo mal uso de um software funcionando corretamente, por exemplo, quando planilhas Excel foram mal feitas. Ou acidentes de avião causados pelo erro de digitação. Explique como o software, mesmo funcionando corretamente, pode ter ajudado a causar o erro. Indique uma melhoria que poderia ter ajudado a alertar sobre o erro.

Exercício 9.12-5: Procure um caso onde um símbolo ou espaço errado fez com que um programa não funcionasse de forma drástica.

Exercício 9.12-6: Escolha um objeto ao seu redor que possua software e investigue o que esse software faz e onde ele está. Investigue o tipo de CPU usada por esse objeto.

Exercício 9.12-7: Software também pode ser gerado para produzir erros em máquinas e equipamentos, passando a ser chamados de vírus ou cavalos de Tróia. Procure o relato de um vírus gerado intencionalmente para causar prejuízos específicos para uma organização, ou algo mais genérico que tenha provocado prejuízos a muitos (e talvez lucros a seus criadores).

Exercício 9.12-8: Procure algumas descrições de empregos para Engenheiros de Software e faça um relato de suas atribuições. Avalie o grau de consistência dessas atribuições entre os empregos. Avalie também o que é mais técnico e o que é mais gerencial.

Exercício 9.12-9: Procure o documento (IEEE Computer Society, 2014b) e entenda as habilidades desejadas de um Engenheiro de Software.

10

O Que é um Projeto

Let our advance worrying
become advance thinking and
planning.

(Winston Churchill)

Conteúdo

10.1.	Recursos e Entregas	100
10.2.	Partes Interessadas	101
10.3.	Sucesso, Fracasso e Riscos	101
10.4.	Pontos Críticos ou Pontos Chave de Sucesso	102
10.5.	Conclusão	103

Por que projetos?

O desenvolvimento de software é um tipo de projeto, e a análise de sistemas uma prática usada nesse projeto, algumas vezes sendo tratada como um subprojeto ou até como um projeto *per si*. A compreensão das características de um projeto facilita a entender o contexto onde a análise de sistemas é realizada.

Este capítulo faz uma introdução ao que é um projeto.

Segundo o PMI (2017), um **projeto** é “um esforço temporário empreendido para criar um produto, serviço ou resultado exclusivo. Os projetos e as operações diferem, principalmente, no fato que os projetos são temporários e exclusivos, enquanto as

10. O Que é um Projeto

operações são contínuas e repetitivas.”

Exemplos de projeto, de complexidades diversas, são:

- construir uma casa;
- elaborar um plano de marketing;
- consertar um equipamento;
- fazer uma inspeção de segurança;
- preparar um curso de treinamento, e
- desenvolver um software.

Os projetos tem como características(Kerzner, 2017, pg.2):

- ter um **objetivo específico**, cujo escopo pode ser elaborado tanto no início quanto, mais modernamente, ao longo do seu ciclo de vida(PMI, 2017, pg. 13);
- possuir um **prazo**, no qual deve ser completado dentro de certas especificações;
- ter **início e fim** definidos;
- ter uma **limitação de recursos**, e verba, quando aplicável,
- utilizar recursos, humanos ou não, e
- ser multifuncionais.

10.1. Recursos e Entregas

Projetos entregam alguma coisa aos seus clientes. Existem entregas físicas, como mesas, estradas, protótipos e equipamentos, e entregas abstratas, como software, relatórios e estudos, ou mesmo o resultado de um treinamento, como o aprendizado de um tema como “programação em C” por algumas pessoas. Um projeto sempre tem pelo menos uma entrega, que é o seu objetivo final, mas normalmente existem também entregas intermediárias, principalmente quando há um contrato , e a parte contratada precisa fornecer essas **entregas intermediárias** para comprovar que está fazendo sua parte e receber um pagamento, previamente acordado, da parte contratante.

Para poder fazer suas entregas, um projeto precisa utilizar vários tipos de recursos. Os tipos mais comuns são(Kerzner, 2017, pgs. 11, 12):

- dinheiro;
- pessoas;
- equipamentos e ferramentas;
- locais;
- infraestrutura;
- materiais consumíveis, e
- informação e tecnologia.

Outro item que poderia ser chamado de recurso é o tempo. Um projeto ocorre em um espaço de tempo, partindo de uma data de início e possuindo uma data de fim estimado. Atrasos no fim do projeto afetam negativamente o entendimento do seu sucesso, porém

acabar um projeto antes do previsto nem sempre é considerado um sucesso adicional. Pela necessidade de gastar dinheiro continuamente durante um projeto, para garantir sua execução, um atraso normalmente aumenta o custo total do projeto.

10.2. Partes Interessadas

Todo projeto possui **partes interessadas**, que são indivíduos ou organizações que afetam ou são afetadas pelo projeto(PMI, 2017), seja favorável ou desfavoravelmente(PMI, 2017). Esses efeitos podem surgir ao longo, como o barulho causado em uma obra na rua, ou só ao fim do projeto, como a poluição que pode ser causada quando uma usina térmica de energia começa a funcionar. Vantagens e desvantagens geradas por um projeto podem causar conflitos de interesse entre as partes interessadas. Partes interessadas serão tratadas detalhadamente no Capítulo 12.

10.3. Sucesso, Fracasso e Riscos

O sucesso de um projeto acontece quando ele se completa(Kerzner, 2017, pg. 6):

- dentro do prazo;
- dentro do orçamento;
- com o desempenho apropriado ou no nível especificado;
- sendo aceito pelo clientes e usuários;
- satisfazendo as partes interessadas (PMI, 2017, pg. 552)
- **com mudanças mínimas de escopo acordadas entre as partes interessadas;**
- sem causar distúrbio ao fluxo de trabalho da organização, e
- quando desejado, sem mudar a cultura da organização¹.

As mudanças de escopo são normais em projetosKerzner (2017, pg. 6), provocadas pelo acontecimento dos riscos ou por mudanças não previstas no ambiente ou nas demandas das partes interessadas. Quanto mais longo o projeto, maior o risco de mudanças importantes. Mudanças, porém, podem gerar riscos adicionais, como afetar a moral da equipe. Em projetos que seguem a linha prescritiva de gerência, ou são mantidos sobre contratos fortes, as mudanças são mantidas em um mínimo necessário e exigem a concordância não só do cliente como da equipe de desenvolvimentoKerzner (2017).

10.3.1. A Incerteza dos Projetos e os Riscos

Todo projeto está associado a um nível de incerteza de seus resultados e entregas (Satpathy et al., 2016), não sendo possível garantir com absoluta certeza que um projeto terminará dentro de todos os parâmetros combinados. As técnicas de gerência de projeto

¹alguns projetos tem o objetivo de mudar a cultura

10. O Que é um Projeto

tem como objetivo diminuir essas incertezas, principalmente com o uso de técnicas de análise de risco e do uso de estimativas e margens de erro.

Uma questão importante de projetos são seus riscos, ou seja, os desafios que podem ocorrer ao longo do mesmo e que temos de evitar ou mitigar para completar o projeto com sucesso. Riscos acontecem com uma probabilidade, gerando um custo adicional, se acontecerem, e impactando o projeto de alguma forma, até mesmo com seu cancelamento. Um risco típico de projetos de software é perder pessoal.

Para garantir o sucesso do projeto é preciso gerenciá-lo ativamente. Existem várias formas de gerenciar projetos, e elas incluem atividades para iniciar, planejar, executar, monitorar e controlar e fechar projetos. O PMBOK (PMI, 2017) é uma fonte de referência importante que consolida várias técnicas tradicionais de gerência de projetos, como documentos necessários, diagramas de rede, uso da estrutura analítica de projeto e outras. Ele possui uma parte adicional para métodos ágeis e outra para projetos de software.

10.3.2. Benefícios da Gerência de Projeto

Os benefícios potenciais de gerenciar um projeto são Kerzner (2017, pg. 3):

- identificação clara das responsabilidades funcionais para garantir que todas as atividades estão com responsáveis, mesmo com a troca de profissionais;
- minimizar a necessidade de relatos contínuos;
- identificar os limites de tempo para todos os agendamentos;
- identificar a metodologia para análise de custo-benefício;
- medir o que foi alcançado em relação ao plano;
- identificação antecipada de problemas para permitir ações corretivas;
- melhor capacidade de estimativa e planejamento, e
- conhecimento dos objetivos que podem não ser alcançados ou vão ser excedidos.

10.4. Pontos Críticos ou Pontos Chave de Sucesso

Os pontos críticos (ou chave) de sucesso para um projeto são aquelas questões que, não estando diretamente relacionadas ao desenvolvimento propriamente dito, são essenciais para o bom andamento do projeto.

Exemplos: compromisso de certos funcionários, fornecimento de certa informação, chegada de alguma máquina ou software, compromisso de entrega de dados ou equipamentos pelo cliente, etc.

Os pontos críticos devem ser levantados detalhadamente, pois os compromissos do desenvolvedor só poderão ser cumpridos caso sejam resolvidos satisfatoriamente.

10.5. Conclusão

O desenvolvimento de um software é um projeto, e como tal deve ser gerenciado para aumentar as chances que tenha sucesso. Além disso, desenvolver software é um tipo especial de projeto que tem se mostrado, desde o início, muito desafiador, inclusive criando uma nova área de trabalho, a Engenharia de Software.

A análise de sistemas é uma das tarefas existentes em um projeto de software, e, dentro de todas as outras, tem como finalidade definir os requisitos verdadeiros desse projeto.

DRAFT

DRAFT

11

Análise de Sistemas

O modelo de análise possui 3 dimensões: informação, comportamento e aprensentação(Jacobson, Christerson et al., 1992).

DRAFT

DRAFT

Parte II.

Partes Interessadas

DRAFT

DRAFT

12

Partes Interessadas

Find the appropriate balance of competing claims by various groups of stakeholders. All claims deserve consideration but some claims are more important than others.

(Warren Bennis)

Conteúdo

12.1.	O Que São Partes Interessadas	111
12.2.	Partes Interessadas e Valor	115
12.3.	Gerência das Partes Interessadas	115
12.4.	A abordagem SEMAT para Partes Interessadas	116
12.5.	Caracterização da Parte Interessada	116
12.6.	Classificação das Partes Interessadas	119
12.7.	Representação das Partes Interessadas	123
12.8.	Engajamento das Partes Interessadas	123
12.9.	Matriz RACI	123
12.10.	Conclusão	125

Por que partes interessadas?

Projetos de software afetam e são afetados por pessoas e organizações, as partes interessadas. Entender como modelar e trabalhar com elas é essencial para conclusão satisfatória do projeto.

Todo projeto, inclusive projetos de software, não só atende a pessoas e organizações, compradores, clientes e usuários, mas também **afetam e são afetados** por outras pessoas e organizações de alguma forma. O somatório de todas essas pessoas, direta ou indiretamente ligadas ao projeto, é chamado de **partes interessadas**¹.

As partes interessadas são um Alpha do SEMAT, visto na Figura 12.1, no nível de Cliente(Jacobson, Ng et al., 2013). Neste modelo elas fornecem as oportunidades para um projeto de software, demandam requisitos, suportam o time de desenvolvimento e usam e fruem do sistema de software. Na verdade, elas tem a posição central de todo projeto. O time, na prática, trabalha para as partes interessadas.

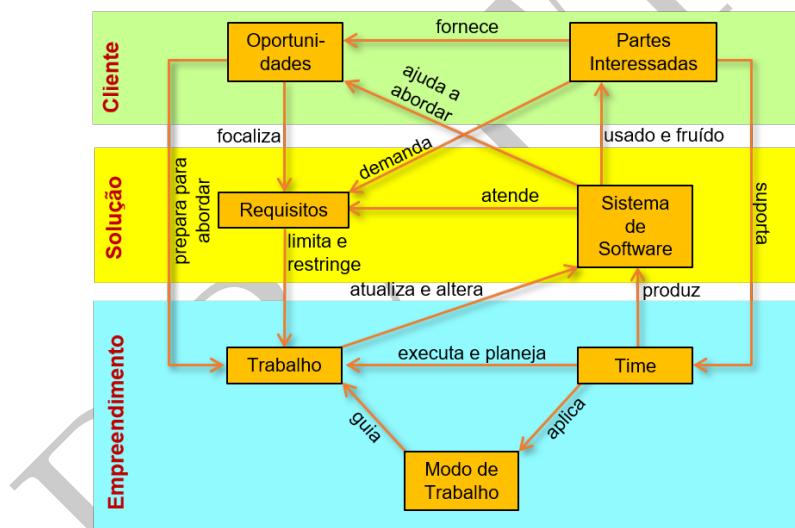


Figura 12.1.: O Kernel Essencial do SEMAT. Fonte:(Jacobson, Ng et al., 2013)

A influência das partes interessadas em um projeto tem um comportamento inverso ao do custo de mudanças, como ilustra a Figura 12.2. Ao longo do projeto, como o trabalho vai ficando pronto, uma mudança provocada pelas partes interessadas começa a custar muito mais, logo elas vão perdendo a capacidade de fazer alterações. Por outro lado, no início, quando o projeto ainda não fez escolhas e compromissos importantes que determinarão o seu futuro, e as mudanças custam pouco, as partes interessadas têm grande influência.

¹O nome em inglês, **stakeholders**, aqueles que possuem algum interesse ou aposta no projeto, é também usado.

A tudo isso se soma o fato que um objetivo geral de qualquer projeto é **satisfazer as partes interessadas**.

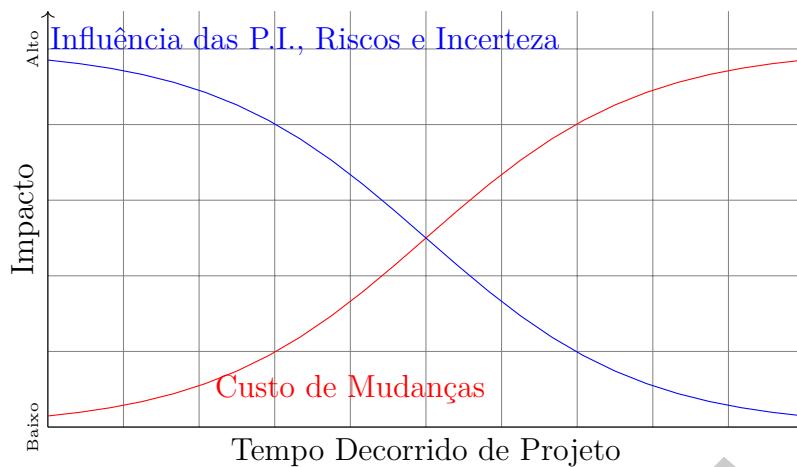


Figura 12.2.: O impacto das partes interessadas.

As práticas ágeis, porém, tentam diminuir, ao mesmo tempo, tanto a perda de influência das partes interessadas, quanto o aumento do custo de modificações. Isso é feito principalmente com os usuários e clientes mais próximos do projeto, ao longo do tempo. Mesmo assim, grandes mudanças têm um custo maior e podem ficar inviáveis.

12.1. O Que São Partes Interessadas

Uma boa definição do que é uma parte interessada é fornecida pelo Guia PMBOK: “é um indivíduo, grupo ou organização que pode afetar, ser afetada ou sentir-se afetada por uma decisão, atividade ou resultado de um projeto. As partes interessadas do projeto podem ser internas ou externas ao projeto, e podem estar envolvidas ativamente ou passivamente, ou mesmo não estar cientes do projeto”(PMI, 2017). Também é possível, para facilitar uma análise, falar de outros tipos de parte interessada, não humanos, como um ecossistema ou o meio-ambiente².

O PMBOK (PMI, 2017) exemplifica várias partes interessadas em um projeto:

- equipe do projeto;
 - gerente do projeto;
 - equipe de gerência do projeto;
 - outros membros do projeto;
- patrocinador;
- clientes e usuários;
- fornecedores;

²No Project Model Canvas, Capítulo 14 há o conceito de Fatores Externos, aplicável nestes casos.

12. Partes Interessadas

- parceiros comerciais;
- gerentes funcionais;
- gerente de portfólios;
- gerente de programas;
- escritório de projetos, e
- outras partes interessadas

Esse mesmo guia avisa que “As partes interessadas do projeto podem ter um impacto positivo ou negativo no projeto, ou ser impactadas de forma positiva ou negativa pelo projeto.”(PMI, 2017).

O grupo dos que afetam e são afetados, com conhecimento de causa ou não, é bem maior que os envolvidos diretamente no projeto. Por exemplo, em um sistema que controla um radar, dedicado a aplicar multas de trânsito automaticamente, podem ser afetados os guardas de trânsito, que podem perder o emprego, os passageiros de todos os veículos e até todo o sistema de saúde do país, devido a diminuição da gravidade dos acidentes. Também podem ser afetados os moradores da região, com menos atropelamentos, ou com um trânsito mais lento que atrasa seus horários. Outro exemplo pode ser um sistema de marcação e monitoração do momento de atendimento de consultas médicas, onde os acompanhantes dos pacientes também são afetados.

Uma avaliação da Tabela 9.2 mostra a importância das partes interessadas para o sucesso de um projeto. Entre os 10 primeiros fatores de sucesso, 5 são ligados a pessoas com interesse no projeto. Entre os 4 primeiros, que estão em igualdade de condição, o envolvimento do Patrocínio Executivo e dos Usuários se mostra essencial.

Nunca diga “usuário”

Gladys S. W. Lam (2015) diz que um dos princípios básicos da análise de negócio é **nunca dizer usuário**. Segundo ele, usuários só existem na perspectiva de um sistema de TI. Nos contextos de negócio as pessoas não são usuárias, mas atores centrais do negócio. Na prática, usuário não significa muito em um mundo onde todos são usuários de sistemas de TI, então ao tratar as partes interessadas devemos usar o termo específico do negócio, como paciente, atendente, médico e enfermeira em um ambulatório, e correntistas, caixa e gerente em um banco.

Deve ficar claro que todo projeto se inicia por meio de das partes interessadas, porém não necessariamente de todas. Descobrir todas as partes interessadas é uma das missões iniciais de um projeto. Na maioria das vezes os usuários, pelo menos em sua maioria, estão identificados no inicio do projeto. Outro caso comum é uma parte interessada advocar a necessidade do projeto para trazer melhorias para outra parte. É comum que software seja desenvolvido para melhorar o atendimento ao cliente da organização que está adquirindo o produto, obviamente a espera de melhorar o resultado global.

Entre as partes interessadas, o usuários do software sendo produzido são um grupo importante, que merece destaque especial em nossos estudos. A princípio são eles, possivelmente por meio de uma representação, que nos fornecerão os requisitos e aceitarão

o produto.

12.1.1. Usuários

Desenvolvemos sistemas para serem usados. Na verdade, uma medida importante da qualidade de um sistema de informação e do sucesso do projeto que o construiu é quanto do sistema é realmente usado pelos seus usuários, ou seja, quanto do que foi feito é realmente útil. Logo, para descobrir o que deve ser feito (o quê), devemos primeiro entender para quem o sistema se destina (quem) e ainda entender o motivo dessas pessoas (por que).

Um tipo especial de parte interessada é o **usuário**. Como diz o nome, um usuário sempre **usa** o sistema. Esse nome pode ser entendido em duas formas. Na forma restrita, indica o **usuário final**, isto é, os que realmente usam o sistema dentro do escopo do seu objetivo, interagindo com suas interfaces, sejam por meio de telas ou relatórios impressos. Já na definição de forma ampla, todos aqueles que usam alguma parte do sistema, incluindo sua documentação e suas estruturas de dados, como os desenvolvedores, que usam o código do sistema, ou os responsáveis por sua operação, também são usuários.

Mesmo com uma interpretação ampla, algumas partes interessadas não são usuários, pois nunca usam alguma parte do sistema, mesmo que sejam afetadas por eles.

Novamente, segundo o PMBOK, o objetivo de todo projeto é gerenciar a satisfação do usuário (PMI, 2017). Por isso é importante garantir o envolvimento de todas as partes interessadas. A forma de fazer isso é por meio de um processo de comunicação contínua, para “entender necessidades e expectativas, abordar as questões conforme elas ocorrem, administrar os interesses conflitantes e incentivar o engajamento apropriado das partes interessadas com as decisões e atividades do projeto” (PMI, 2017).

12.1.2. Perspectivas dos Usuários

As perspectivas básicas que encontramos em entrevistas e reuniões com usuários, são três, como ilustrado na Figura 12.3:

1. o usuário onisciente;
2. o usuário externo ao sistema, e
3. o usuário interno ao sistema.

O **usuário onisciente** fala do sistema como um todo, indicando normalmente coisas que ele deve, ou deverá, fazer. Ele vê tanto o sistema quanto seus usuários de uma perspectiva externa, conhecendo os mecanismos, de forma abstrata, tanto por dentro quanto por fora. Geralmente ele possui uma posição de gerência e muitas vezes é o patrocinador executivo do projeto. Algumas vezes já

É possível que ele não saiba como o sistema funciona realmente, pois seu contato pode ser de outra época em relação a entrevista, desconhecendo então modificações e detalhes

12. Partes Interessadas

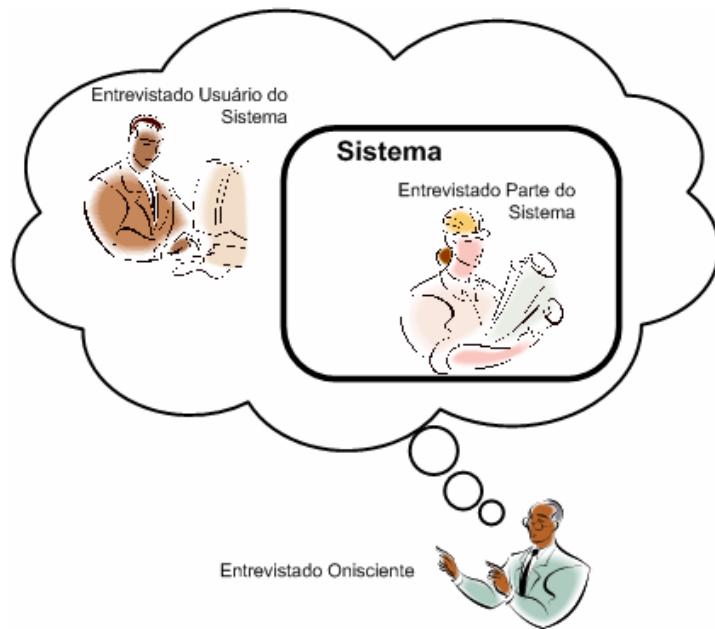


Figura 12.3.: 3 tipos comuns de usuários dos sistemas. Fonte:próprio autor

atuais. Ele normalmente exige funcionalidade gerencial do sistema. Esse usuário muitas vezes quer, e é capaz de ajudar a definir, uma nova abordagem de negócio para o sistema. Tipicamente é um gerente ou diretor e tem algum poder sobre os usuários com outras perspectivas.

O **usuário externo** descreve o sistema como se o estivesse usando, e normalmente já usa uma versão do sistema. Exige funções do sistema, mas principalmente para atender seu nível de atuação, seja gerencial ou operacional. Ele pode apresentar algum nível de desconfiança, pois o novo sistema pode exigir um esforço para conseguir novos conhecimentos ou distribuir conhecimento que hoje apenas ele tem acesso, o que afeta o poder dele na organização. Além disso, ele conhece bem a entrada e a saída do sistema atual, mas não necessariamente os procedimentos e algoritmos, ou seja, como as contas são feitas. Esse usuário, principalmente quando mais próximo do nível operacional da organização, pode levar ao desenvolvimento de um sistema igual ao já existente, apenas com a correção dos defeitos, sem mudança na prática do negócio.

O **usuário interno** descreve o sistema visto por dentro. Algumas vezes, não existe ainda um sistema plenamente informatizado, e é esse usuário que faz a parte de cálculos e tomadas de decisão. Pode apresentar um risco ao desenvolvimento, porque ele é candidato a substituição pelo sistema novo, o que pode gerar desconfiança, franca hostilidade e mesmo sabotagem do projeto. Ele conhece normalmente muito bem como o sistema atual funciona por dentro.

12.2. Partes Interessadas e Valor

Todo valor do projeto é entregue às partes interessadas. Na verdade, seria mais correto dizer que “todos os valores do projeto” são entregues às partes interessadas, já que um projeto pode ter valores diferentes de acordo com as partes interessadas, pois elas possuem interesses diferentes, como será tratado na Seção 12.5.1.

Suponha o seguinte projeto de software: uma conjunto de serviços B2B para uma editora, que serão usados por livrarias *on line* que desejam comprar produtos *on demand* quando forem vendidos em suas lojas virtuais B2C. Esse software está planejado para ter efeito de melhorar o serviço das grandes livrarias *on line*, prejudicando, por consequência, as pequenas livrarias locais. Então, podemos concluir que ele fornece valor positivo para algumas partes interessadas:

- a editora;
- grandes livrarias *on line*;
- o leitor;
- o escritor;
- o serviço de entrega, e
- outros atores, como pessoal de operação do software, vendedores de equipamentos, etc.

Porém ele também fornece um valor negativo para partes interessadas que são afetadas, principalmente de forma indireta, como:

- livreiros e trabalhadores de livraria, que podem perder o emprego;
- proprietários que alugam lojas para livrarias;
- pessoas que querem comprar um presente pessoalmente em cima da hora, e
- pequenas livrarias locais.

É claro que no mercado esse padrão sempre se repetirá: favorecer um negócio sempre prejudica seus competidores, e com isso toda a sua cadeia de produção. Normalmente não há um problema legal ou ético com isso, mas é possível que em condições limites alguma lei de proteção a livre concorrência possa ser ferida.

12.3. Gerênciadas Partes Interessadas

A gerência das partes interessadas aparece em vários documentos da área de Gerência de Projeto e de Engenharia de Software.

Ela é uma atividade do PMBOK que inclui(PMI, 2017):

- “processos para identificar pessoas, grupos ou organizações que podem impactar ou ser impactadas pelo projeto”;
- “análise das expectativas da partes interessadas e seu impacto no projeto”, e
- “o desenvolvimento de estratégias apropriadas de gerência para efetivamente engajar

12. Partes Interessadas

as partes interessadas nas decisões e execução do projeto”.

Para isso é necessária a comunicação contínua com todas as partes interessadas, no intuito de entender e registrar todas as necessidades e expectativas. Além disso, é necessário abordar todas as questões e conflitos de interesse, que podem acontecer a qualquer momento do projeto, e ainda promover o engajamento apropriado nas decisões e atividades do projeto(PMI, 2017).

Essa gestão, então, vai se resumir em 4 partes(PMI, 2017):

1. identificar as partes interessadas;
2. planejar a gerência das partes interessadas;
3. gerenciar o engajamento das partes interessadas, e
4. monitorar o engajamento das partes interessadas.

12.4. A abordagem SEMAT para Partes Interessadas

O SEMAT já considera que o trabalho com as partes interessadas deve evoluir em uma escala de 6 estados, do menos ao mais sofisticado(Jacobson, Ng et al., 2013):

1. reconhecer as partes interessadas;
2. criar uma representação para as partes interessadas;
3. envolver as partes interessadas;
4. fazer com que as partes interessadas concordem;
5. satisfazer as partes interessadas para a implantação, e
6. satisfazer as partes interessadas com o uso.

12.5. Caracterização da Parte Interessada

Após a identificação das partes interessadas, continua-se o levantamento e caracterização.

Para isso, normalmente são respondidas, para cada parte interessada, algumas perguntas, como:

1. Nome
2. Posição na organização
3. Posição/Responsabilidade no projeto
4. Interesse no projeto (Stake)
5. Impacto no projeto
6. Benefícios recebidos do projeto
7. Colaboração necessária ao projeto (o que precisamos dessa parte interessada)?
8. Atitudes percebidas (riscos)

Essas perguntas variam de um autor da área para outro, porém sempre é necessário

ter uma visão de como a parte interessada pode impactar o projeto, sendo apoiando, sendo criando obstáculos. Isso pode ser feito com práticas de classificação das partes interessadas.

12.5.1. Interesses e Objetivos

Como diz o próprio termo, todas as partes interessadas possuem interesses no sistema, mesmo que não estejam cientes do mesmo, ou do seu planejamento. Os interesses têm relação com as expectativas, explícitas e implícitas, que o sistema, em operação, afete a elas, o seu domínio, ou o seu negócio, de alguma forma. Interesses são benefícios que as partes interessadas esperam obter do sistema.

Os usuários finais possuem também **objetivos**, i.e., eles usam o software para obter uma, ou mais, resposta planejada. Neste contexto, objetivos são requisitos funcionais que os usuário esperam encontrar no sistema.

A princípio, se deseja que o sistema seja construído para atingir seus objetivos de forma que todos os interesses sejam atendidos. Porém, pode haver incompatibilidades tanto entre os interesses quanto entre os objetivos das diversas partes interessadas. Tratar casos como esse já foi discutido inclusive como um processo de software específico, conhecido como WinWin (B. W. Boehm, 1988; B. Boehm e Kitapci, 2006).

É importante atentar para a diferença que existe entre **o uso que uma parte interessada fará do sistema** (o que), o seu **objetivo**, com o **interesse** que uma parte interessada tem no sistema, que é o **resultado esperado, ou expectativa** que ela tem. Enquanto todas as partes interessadas tem interesses, só algumas tem objetivos, e essas são as usuárias do sistema.

Estudar os interesses nos permite saber melhor como realizar os objetivos. Por exemplo, um cliente de web site de vendas de produtos tem como objetivos, ao longo de uma compra, várias ações, como escolher um produto, saber seu preço, pagar o produto. Porém ao longo de todo esse tempo, seu único interesse é receber o produto. Mais do que comprar, o usuário quer receber-lo em casa. Então toda a dinâmica do site, ou do software e logística do sistema por trás dele, pode ser construída voltada para esse interesse. Mesmo que quiséssemos atender um interesse menor, como “comprar o produto”, já poderíamos imaginar formas diferentes de implementar as ações correspondentes. Um exemplo disso no mundo real é o *one-click buy*, ou seja, comprar um produto com só um click do mouse e ter certeza de receber-lo, ideia que foi inclusive patenteada pela Amazon³.

Como documentação dos interesses e objetivos das partes interessadas, dois formatos são possíveis. O primeiro é uma tabela com três colunas: parte interessada, interesses e objetivos. Essa tabela tem como defeito não identificar claramente partes interessadas com interesses ou objetivos comuns. A vantagem é a facilidade de gerenciar e preencher,

³O que causou muita polêmica na época. Essa patente já expirou.

12. Partes Interessadas

pois é um formulário muito simples.

Tabela 12.1.: Exemplo de formulário para registro do interesse e objetivo da parte interessada.

Parte Interessada	Interesses	Objetivos
Professor	Não perder seus livros	Registrar um empréstimo
	Recuperar livros emprestados	Registrar uma devolução
	Saber com quem está um livro	Registrar um livro
	Saber os livros que tem	Cobrar um livro atrasado
		Registrar um aluno
Aluno	Pegar livro emprestado com facilidade	Registrar uma devolução
	Não ser cobrado por livro que devolveu	Receber recibo de devolução
Desenvolvedor	Fazer projeto final	

A segunda forma é uma matriz Partes Interessadas vs. Interesses e Objetivos. Dependendo do escopo do sistema ou da quantidade de partes interessadas, escolhe-se um ou outro para coluna ou linha. Nesse caso, a célula que mostra o cruzamento entre parte interessada e interesse ou objetivo pode conter uma avaliação qualquer, como o uso de palavras em uma escala de importância para aquele usuário: baixa, média e alta.

Tabela 12.2.: Exemplo de matriz de cruzamento para registro do interesse e objetivo da parte interessada.

	Professor	Aluno	Desenvolvedor
Interesses	✓	✓	✓
Objetivos	✓	✓	✓
Não perder seus livros	✓		
Recuperar livros emprestados	✓		
Saber com quem está um livro	✓		
Saber os livros que tem	✓		
Pegar livro emprestado com facilidade	✓		
Não ser cobrado por livro que devolveu	✓		
Fazer Projeto Final			✓
Registrar um empréstimo	✓		
Registrar uma devolução	✓		
Registrar um livro	✓		
Cobrar um livro atrasado	✓		
Registrar um aluno	✓		
Receber recibo de devolução	✓		

12.6. Classificação das Partes Interessadas

Além disso, a maioria dos autores propõe classificar as partes interessadas de alguma forma, e contando com apoio de gráficos para entender melhor como gerenciá-las ao longo do projeto.

Um gráfico típico, mostrado na Figura relaciona o interesse da parte interessada com o poder, e indica que tipo de ação deve ser tomada na gerência de partes interessadas.

Essa classificação pode ser feita de várias formas, sendo as mais reconhecidas:

- Poder × Interesse;
- Poder × Influência;
- Influência × Impacto, e
- Poder, Legitimidade e Urgência.

Outra classificação feita é a **Matriz de Engajamento**. Nela se faz uma análise do comportamento atual do usuário em relação a um projeto e o comportamento desejado. São usadas cinco classes possíveis para descrever o engajamento do usuário:

12. Partes Interessadas

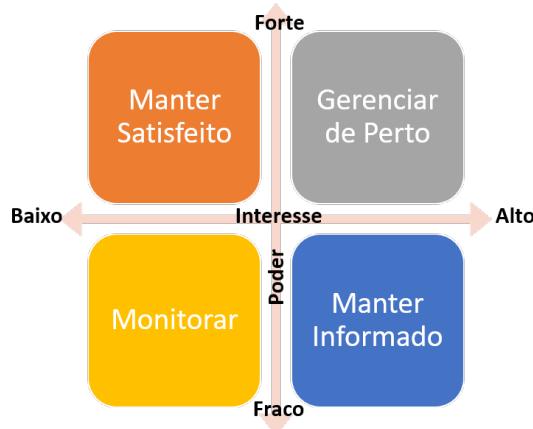


Figura 12.4.: Quadro para o tratamento das partes interessadas. Fonte:(PMI, 2017)

1. desconhece;
2. resistente;
3. neutro;
4. apoiador, e
5. liderança.

O trabalho da equipe é identificar o comportamento atual e, por meio de ações positivas, levar as partes interessadas a um comportamento desejado.

Desconhece	Resistente	Neutro	Apoiador	Líder
Parte Interessada 1	C			D
Parte Interessada 1			C	D
Parte Interessada 1			CD	

Tabela 12.3.: Matriz de Engajamento: C significa comportamento corrente, D o comportamento desejado.

K. E. Wiegers e Beatty (2013), por sua vez, faz uma classificação hierárquica das partes interessadas em:

- clientes, e
 - usuários diretos e indiretos, e
 - ◊ classes de usuários favorecidas, cuja satisfação está fortemente alinhada com os objetivos de negócio do projeto;
 - ◊ classes de usuários desfavorecidas, que não podem usar o produto, por motivos legais, de segurança ou outros, como estelionatários em um sistema bancário;
 - ◊ classes de usuários ignoradas, que não são relevantes ao projeto e
 - ◊ outras classes de usuário, que não se encaixam em nenhuma das outras, mas ainda fornecem requisitos para o projeto .

- outros clientes.
- outras partes interessadas.

12.6.1. Análise das Partes Interessadas

A Análise das Partes Interessadas é uma atividade com o objetivo determinar quais são as partes interessadas, e determinar como são afetadas e como podem afetar o projeto, além de entender quem realmente “conta” em um projeto (Mitchell, Agle e Wood, 1997).

Normalmente análise das partes interessadas se inicia com a identificação das mesmas, como mostram PMI (2017) e Jacobson, Lawson e Ng (2019). Muitas são as formas de identificação, mas certamente a experiência do analista será muito importante, além de documentos formais do projeto.

É importante notar que algumas partes interessadas são essenciais ao projeto de software. Por exemplo, deve haver um responsável final pelo financiamento do projeto, normalmente conhecido como patrocinador. Deve também existir pelo menos um grupo de usuários do futuro sistema, os usuários propriamente ditos.

Técnicas de levantamento podem incluir não só a leitura de documentos e a experiência de pessoas da área de negócio, mas também reuniões de *brainstorm* podem ser usadas para levantar o conjunto de partes interessadas a partir de um núcleo inicial de pessoas.

Existe uma base de análise para identificar partes interessadas (Mitchell, Agle e Wood, 1997) e determinar um grau de atenção que deve ser dado a elas, baseada no conceito de **saliência**, que é formada em 3 atributos: Poder, Legitimidade e Urgência.

O **poder** é a “capacidade de um ator fazer com que outro ator realize algo que não realizaria a princípio” (Mitchell, Agle e Wood, 1997). O Poder por ser baseado em coerção, i.e. o uso de força ou ameaça, em utilitarismo, i.e. o uso de incentivos, e normativo, por influências simbólicas.

A **legitimidade** é a “percepção generalizada ou premissa que as ações de uma entidade são desejáveis, apropriadas dentro de algum sistema socialmente construído de normas, valores, crenças e definições” (Mitchell, Agle e Wood, 1997) Ela se baseia no indivíduo, na organização ou na sociedade.

A **urgência** é “grau que a parte interessada reivindica de atenção imediata” (Mitchell, Agle e Wood, 1997) Ela se baseia na sensitividade ao tempo ou na criticalidade, a importância da reivindicação ou do relacionamento com a parte interessada.

Finalmente, a saliência de uma parte interessada “é o grau que o gerente dá prioridade a partes interessadas que competem” (Mitchell, Agle e Wood, 1997). A análise da posse desses três fatores também leva a denominações como as apresentadas na Figura 12.5.

Mitchell, Agle e Wood (1997) afirma que partes interessadas que possuem apenas uma dessas características normalmente são deixadas de lado pelos gerentes, que aumentam o seu grau de engajamento com as que tem duas das características, até chegar ao grupo

12. Partes Interessadas



Figura 12.5.: Classificação das partes interessadas. Fonte: (Mitchell, Agle e Wood, 1997)

dos “definitivos”, que tem prioridade máxima.

12.7. Representação das Partes Interessadas

Algumas das partes interessadas são formadas por grupos de pessoas. Dependendo do tamanho do grupo, não é possível trazer todos eles para participar do projeto. Nesse caso é necessário criar uma representação para as partes interessadas (Jacobson, Lawson e Ng, 2019). Caberá a essa representação se envolver e tomar decisões em nome do grupo.

Em projetos ágeis é comum que exista um representante único de todo os usuários, que estabelece as demandas e prioridades do projeto, como o *Product Owner* (Rubin, 2013), porém isso não elimina a existência de todas as outras partes.

12.8. Engajamento das Partes Interessadas

O engajamento das partes interessadas se faz por meio de três processos (PMI, 2017):

- inclusão, aumentando o nível de participação das partes interessadas;
- materialidade, pela identificação das questões mais importantes da organização e das partes interessadas, e
- responsividade, pela resposta da organização.

Esse engajamento pode ser obtido de várias formas, dependendo do nível de engajamento que é desejado.

12.9. Matriz RACI

A participação das partes interessadas em cada atividade do projeto pode ser planejada por meio de uma Matriz RACI⁴.

A Matriz RACI, também discutida brevemente no Capítulo 14, faz o cruzamento entre as partes interessadas e alguma forma de entender o projeto, como atividades, casos de uso, histórias do usuário, entregáveis, etc. Nesta seção vamos usar o exemplo de entregáveis, sem perda de generalidade.

Para cada um dos entregáveis, deve ser definida, para cada uma das partes interessadas, uma ação, ou mais, em relação ao projeto. Estas ações formam o acrônimo RACI e significam:

- Responsável, que realiza a atividade;
- Aprovador, que aceita a atividade;

⁴Apesar de ter uma preocupação em descobrir os autores de cada método proposto neste texto, a matriz RACI parece ser um mistério. Ela pode ser uma adaptação das *Responsability Charts*, propostas por (Andersen, Grude e Haug, 2009) originalmente na década de 80, ou uma adaptação de uma *Responsibility Assignment Matrix* que aparece citada em documentações da NASA já em 1977 (Facetation, 2015)

12. Partes Interessadas

- Consultado, que fornece informações para a realização da atividade, e
- Informado, que deve ser informado sobre a atividade.

A Tabela 12.4 é um exemplo de Matriz RACI.

	Diretor	Gerente	Caixa	Contador	Time Devel
Retirar Dinheiro	I	A	C	I	R
Depositar Dinheiro	I	A	C	I	R
Aprovar Empréstimo	AC	C	I	I	R
Cobrar Dívida	AC	C	I	C	R
Gerar Partidas Dobradas	I	I	I	CA	R

Tabela 12.4.: Exemplo de Matriz RACI

A Matriz RACI tem duas regras importantes:

1. Células podem conter mais de uma letra;
2. Apenas uma pessoa pode aprovar uma atividade, e
3. Pelo menos uma pessoa deve ser responsável por realizar uma atividade.

Além disso, duas análises devem ser feitas para verificar a Matriz RACI.

1. A Análise Vertical:

- alguma parte interessada tem um excesso de R's, sendo responsável por uma carga de trabalho excessiva?
- alguma parte interessada tem um excesso de A's, havendo pouca delegação ou responsabilidade excessiva?
- uma parte interessada não tem nenhuma célula em branco, participando de todas as atividades do projeto?

2. A Análise Horizontal

- toda atividade tem pelo menos um responsável;
- toda atividade tem um e apenas um aprovador;
- atividades com mais de um responsável devem ser analisadas para duplicidade de trabalho;
- atividades são realizadas e aprovadas em níveis hierárquicos corretos?
- está correto onde o realizador é a mesma pessoa que o aprovador?
- o número de pessoas sendo informada ou consultada é grande ou pequeno demais?

12.10. Conclusão

As partes interessadas são importantes por dois motivos principais: necessidade de identificar com quem levantar os requisitos e a necessidade do apoio delas durante o projeto.

Para isso um conjunto de técnicas está disponível. Algumas dessas técnicas, como a Matriz de Engajamento, podem causar problemas se divulgadas fora de um âmbito muito restrito no projeto, outras, como a Tabela RACI fazem parte de documentos ostensivos e precisam ser validadas pelas partes interessadas.

É importante que as partes interessadas sejam gerenciadas ao longo do projeto, seguindo procedimentos tradicionais ou ágeis, e o estado dessa gerência pode ser controlado tanto por técnicas tradicionais de gerência de projeto como por um modelo como o do SEMAT.

12.11. Exercícios

Exercício 12.11-1: Descreva as partes interessadas do Sistema de Gestão Acadêmica da sua instituição de ensino.

Exercício 12.11-2: Vá para o site <http://jogodeanalisedesistemas.xexeo.net/> e visite a Livraria Resolve. A partir da sua visita:

1. identifique todas as partes interessadas;
2. classifique-as quanto a:
 - a) poder × interesse;
 - b) poder × influência;
 - c) poder, legitimidade e urgência.
3. construa a Matriz de Engajamento.

DRAFT

Parte III.

Oportunidades

DRAFT

DRAFT

13

Oportunidades e Problemas

A wise man will make more opportunities than he finds.

(Francis Bacon)

Conteúdo

13.1.	Oportunidades	130
13.2.	Problemas	131
13.3.	Tipos de Problemas	131
13.4.	Identificando Problemas	132
13.5.	Caracterização do Problema	141
13.6.	Oportunidade e 5W2H	141
13.7.	Propondo Oportunidades ao Cliente	143
13.8.	Oportunidade e Valor	143
13.9.	Exercícios	144

Por que oportunidades e problemas?

Um projeto de software sempre tem como objetivo aproveitar oportunidades ou resolver problemas. Esses termos, porém, devem ser usados de forma mais precisa para que possamos entender os verdadeiros objetivos do projeto, de acordo com as partes interessadas.

13. Oportunidades e Problemas

Neste capítulo serão discutidas algumas técnicas para descobrir, caracterizar e classificar oportunidades e problemas. As oportunidades são fornecidas pelas partes interessadas, discutidas no Capítulo 12.

13.1. Oportunidades

Um **oportunidade** é uma circunstância que torna apropriado o desenvolvimento de um sistema de software(OMG, 2018). Elas são “uma chance de fazer algo, incluindo consertar um problema existente”(Jacobson, Lawson e Ng, 2019).

Uma oportunidade pode ser uma combinação de problemas, sugestões, percepções de mercado ou diretrizes. Ela articula a razão de criação de um sistema novo ou da modificação de um sistema(OMG, 2018). Ruble (1997), que separa oportunidades de problemas, afirma que enquanto problemas sempre indicam que algo está quebrado, em uma oportunidade isso não é necessário.

As oportunidades devem ser entendidas pela equipe de desenvolvimento de forma a gerar os requisitos. Elas justificam e dão foco aos requisitos(OMG, 2018).

As principais oportunidades se originam das partes interessadas que não são a equipe, porém a equipe de desenvolvimento pode propor também oportunidades, ou reconhecer problemas, sendo necessário então ter o acordo das partes interessadas de que devem ser consideradas no projeto. Um cuidado especial deve ser tomado quando a equipe propõe uma oportunidade, que é a tendência de dar importância excessiva a tecnologia, o que pode gerar um requisito falsoMcMenamin e Palmer (1984).

As oportunidades permitem:

- identificar e motivar as partes interessadas;
- entender o valor que o sistema de software oferece as partes interessadas;
- entender por que o software está sendo desenvolvido;
- entender como o sucesso da implantação do sistema de software vai ser julgado e avaliado, e
- garantir que o sistema de software aborda efetivamente as necessidades de todas as partes interessadas.

É muito importante avaliar continuamente a viabilidade das oportunidades. Jacobson, Lawson e Ng (2019) chamam a atenção para o fato que é necessário:

- confirmar que a oportunidade atende uma **necessidade verdadeira** do cliente;
- tomar decisões sobre que soluções são mais apropriadas para atender a oportunidade, e
- ter a consciência que os benefícios podem demorar algum tempo para ser visíveis para os clientes.

Uma necessidade é verdadeira se o sistema não vai cumprir sua finalidade caso a necessidade não seja atendida(McMenamin e Palmer, 1984).

Oportunidades, positivas ou negativas, são grandes motivadores dos clientes. Se elas não existissem, não haveria necessidade de implementar um novo sistema. A verdadeira motivação de desenvolver o sistema é atender a essas oportunidades, principalmente aqueles que mais afetam o negócio. No *Project Model Canvas* as oportunidades aparecem já no primeiro módulo (Por que), no quadro de justificativas.

13.2. Problemas

Um problema é um estado onde existe uma dificuldade que deve ser resolvida, como o desvio de um padrão aceitável, **uma diferença entre o comportamento desejado e o percebido** ou ainda uma causa, real ou possível, de um ou mais eventos que trouxeram ou podem trazer algum prejuízo.

Problemas incomodam as partes interessadas, trazendo prejuízos financeiros, afetando negativamente a imagem da organização, atrapalhando a produção ou os processos, colocando em risco alguma atividade, e mesmo incomodando apenas psicologicamente.

É comum que os problemas apareçam muito rapidamente nas conversas entre as partes interessadas e a equipe de análise, pois provavelmente estão entre os motivos que causaram a necessidade de um novo desenvolvimento de software, porém não necessariamente as partes interessadas têm conhecimento da causa raiz do problema, pois o que as afeta mais diretamente são os sintomas.

Por exemplo, um restaurante pode estar faturando pouco, mesmo com filas na porta, sendo esse o problema que o cliente quer resolver, procurando um novo sistema de pedidos e fechamento de conta. A causa real, porém, pode ser uma cozinha mal planejada, um salão mal aproveitado ou garçons mal treinados. Nesse caso, um novo sistema dificilmente resolveria o problema.

Deve se tomar cuidado para não procurar uma solução sem que o problema esteja claramente determinado. Por isso é importante descobrir, para cada problema, a sua **causa raiz**. Fazê-lo resulta na continuação do problema e do provável abandono da solução.

Além disso, problemas devem ser definidos em termos de comportamento, não em termos de atitudes ou sentimentos. Para um problema estar realmente descrito, isso deve ser feito na forma de um comportamento que acontece no momento e que é indesejado, colocado em comparação com um comportamento desejado. Segundo Blanchard e S. Johnson (1983), se não há um comportamento desejado não há um problema, mas uma reclamação, e ainda é necessário definir o problema.

13.3. Tipos de Problemas

Podemos identificar três tipos básicos de problemas que são descritos pelos clientes:

13. Oportunidades e Problemas

- **problemas funcionais**, que ocorrem quando um sistema não permite uma funcionalidade que o cliente necessita ou deseja;
- **problemas operacionais**, que ocorrem quando uma funcionalidade funciona de forma errada, e
- **problemas de negócio**, que estão relacionados a manutenção do negócio propriamente dito.

Esses três tipos de problemas se relacionam. Um problema de negócio, por exemplo, pode ser um sintoma da falta de uma funcionalidade adicional no sistema ou de um problema em uma funcionalidade no sistema atual.

13.4. Identificando Problemas

Para analisar o problema, é necessário que os usuários:

- Concordem com a definição do problema;
- Entendam as causas do problema, já que o problema por trás do problema pode ser mais importante, sendo que o problema visto inicialmente é apenas um sintoma;
- identifiquem as partes interessadas e usuários;
- definam a fronteira do sistema de solução. e
- identifiquem as restrições impostas à solução

Algumas práticas para identificar problemas são:

- verificar os resultados obtidos atualmente nos processos e compará-los com objetivos da organização ou padrões do mercado;
- observar o comportamento dos colaboradores;
- levantar a opinião de fornecedores, clientes e vendedores, representantes ou distribuidores, da organização.

Além disso, é importante verificar se as tarefas realizadas:

- contém erros;
- são feitas vagarosamente;
- não são mais feitas como definidas em documentos da organização, e
- não são completadas.

Quanto aos colaboradores, é importante verificar se:

- estão desestimulados;
- não conseguem descrever claramente suas responsabilidades e objetivos, e
- pedem demissão com muita frequência.

Quanto aos parceiros externos, é importante verificar:

- reclamações;
- sugestões de melhorias;
- queda nas vendas, e

- vendas com perdas.

A identificação de problemas às vezes não é fácil se não há um exemplo a comparar. Como saber se seu serviço é lento sem saber como é o serviço da concorrência? Existem práticas específicas para isso, como a compra de relatórios feitos por consultorias especializadas, *benchmarkings*(Stapenhurst, 2009), etc.

Quando um problema é identificado, é importante que todos concordem que esse é o problema a ser resolvido pelo projeto. No Capítulo 14 é apresentado um método prático para iniciar um projeto com esse acordo, o *Project Model Canvas*.

Para estabelecer o problema, é interessante criar uma sentença apropriada, como a seguinte:

O problema de *<descrição do problema>*, afeta *<partes interessadas afetadas>* e resulta em *<impacto nas partes interessadas>*. A solução *<indicar a solução>* Trará os benefícios de *<lista dos principais benefícios>*.

13.4.1. A Técnica de Pareto

Ao lidar com uma quantidade de problemas detectados em um sistema funcionando, é importante determinar os principais problemas, aqueles que tem mais impacto, de modo a dar foco aos esforços e obter melhores resultados.

Uma das técnicas mais conhecidas para isso é a de Pareto.

O **Princípio de Pareto** diz que 20% dos problemas causam 80% dos efeitos. Essa divisão não é real, mas é uma forma de indicar que poucas causas são responsáveis pelos maiores impactos e que é necessário colocar os problemas em perspectiva.

Então, segundo o Princípio de Pareto, devemos calcular o impacto de cada tipo de erro em nosso sistema, selecionando os de maior impacto para resolver.

Devemos tomar cuidado, porém, porque o resultado que obteremos depende dos dados que tivermos¹. A seguir são apresentados dois exemplos, baseados na contagem e custo de erros obtidos em um sistema fictício, onde no primeiro apenas são contados apenas os erros por tipo, chegando a uma análise errada do que é importante, e no segundo é calculado o impacto de cada tipo.

13.4.2. Calculando a Quantidade

Para tratar problemas de acordo com a técnica de Pareto, **supondo que todo problema tem o mesmo efeito**, é necessário seguir os seguintes passos:

1. dividir os problemas em categorias;
2. levantar o número de ocorrências por categoria;

¹Os americanos tem um ditado: *Garbage in, garbage out.*

13. Oportunidades e Problemas

3. classificar as categorias em ordem decrescente por quantidade de ocorrências, como na Tabela 13.1;
4. calcular o total de ocorrências;
5. calcular a porcentagem de cada categoria;
6. determinar uma escala;
7. colocar as categorias como colunas, ordenadamente, a maior a esquerda, e
8. traçar as curvas de porcentagem acumulada, como na Figura 13.1.

A Tabela 13.1 apresenta o primeiro resultado útil, isto é, a tabela ordenada em ordem decrescente da quantidade de ocorrência por tipo de problema, para problemas encontrados em um sistema fictício.

Tabela 13.1.: Uma contagem de tipos de erros para uso da técnica de Pareto

Problema	Ocorrências
Requisito Original Errado	35
Erro de Programação	25
Erro de Projeto	21
Erro de Entrada de Dados	18
Erro de Formato	13
Problema de Segurança	12
Mudança no Negócio	10
Outros (1 Ocorrência)	7
Mudança na Legislação	5
Falta de Rede	3
Falta de Energia	1

A partir da Tabela 13.1 é possível gerar o gráfico da Figura 13.1, que já permite comparar a importância dos problemas, porém ainda considerando que todos os problemas tem o mesmo impacto, o que é raro acontecer.

A análise da Tabela 13.1 e do Figura 13.1 levaria a crer que 82% dos problemas seriam causados por 6 tipos: Requisitos Originais Errados, Erro de Programação, Erro de Projeto, Erro de Entrada de Dados, Erro de Formato e Problema de Segurança. Na prática, não houve um atendimento do Princípio de Pareto, isso porque estamos apenas contando, e não calculando o impacto. No mundo real, é difícil que a premissa que foi feita nesse cálculo seja verdade, dificilmente todo problema tem o mesmo impacto.

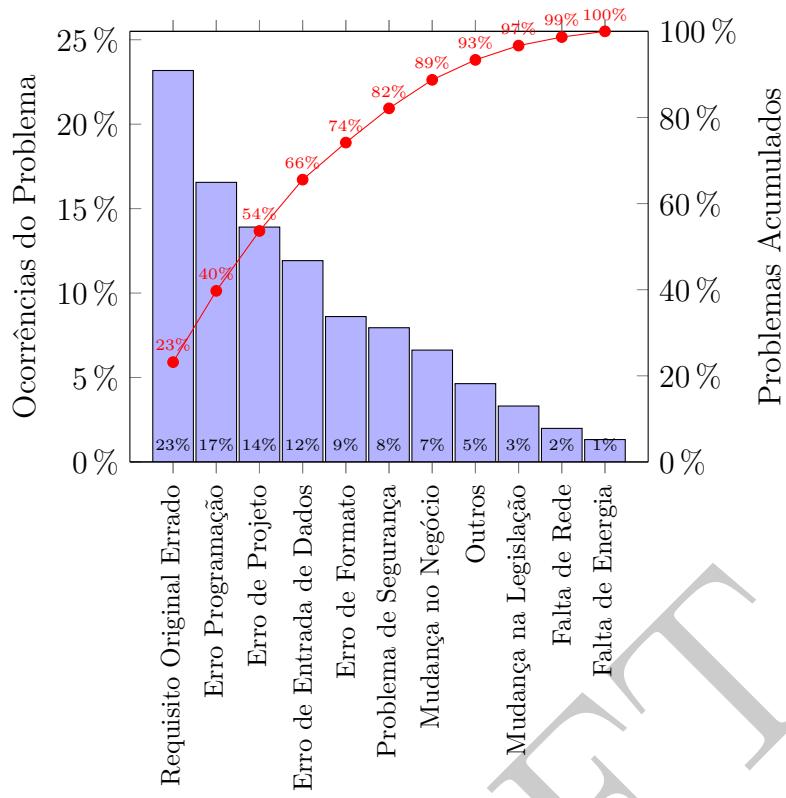


Figura 13.1.: Gráfico com a quantidade de problemas.

13.4.3. Usando o Impacto dos Problemas

A análise anterior pode ser muito melhor se soubermos o impacto de cada problema, seja ele o custo financeiro ou apenas um valor relativo. Dessa forma podemos melhorar a Técnica de Pareto. Vamos supor que os problemas detectados no sistema tenham um custo médio relativo como descrito na Tabela 13.2;

13. Oportunidades e Problemas

Tabela 13.2.: Custo médio relativo dos problemas

Problema	Ocorrências
Requisito Original Errado	100
Mudança no Negócio	80
Mudança na Legislação	80
Erro de Projeto	10
Problema de Segurança	10
Outros (1 Ocorrência)	10
Erro de Programação	2
Erro de Entrada de Dados	1
Erro de Formato	1
Falta de Rede	1
Falta de Energia	1

Quando os problemas tem impacto diferente, é necessário multiplicar a quantidade de problemas de um tipo pelo custo médio do problema, gerando uma tabela de impacto dos erros, como na Tabela 13.3, e o gráfico ficaria como na Figura 13.2.

O novo passo a passo, e o mais aconselhado, para aplicar o Princípio de Pareto com a inclusão do impacto de cada tipo de problema é:

1. dividir os problemas em categorias;
2. levantar o número de acontecimentos por categoria, como na Tabela 13.1;
3. levantar o custo médio por categoria, como na Tabela 13.2;
4. multiplicar o custo médio pelo número de acontecimentos;
5. classificar as categorias em ordem decrescente por custo médio, como na Tabela 13.3;
6. calcular o custo total dos problemas;
7. calcular a porcentagem de cada categoria;
8. determinar uma escala;
9. colocar as categorias como colunas, ordenadamente, a maior a esquerda, e
10. traçar as curvas de porcentagem acumulada, como na Figura 13.2.

Analizando esses resultados vemos a importância de calcular não só a quantidade de erros, mas também seu custo, para melhor priorizar os problemas.

Na Figura 13.2 fica claro que apenas três, ou mesmo dois, problemas precisam atenção, sendo que o primeiro problema é bem mais importante que os outros. Nesse caso, o Princípio de Pareto é atendido.

Tabela 13.3.: Custo total relativo

Problema	Ocorrências
Requisito Original Errado	3500
Mudança no Negócio	2000
Mudança na Legislação	1680
Erro de Projeto	180
Problema de Segurança	130
Outros (1 Ocorrência)	120
Erro de Programação	20
Erro de Entrada de Dados	7
Erro de Formato	5
Falta de Rede	3
Falta de Energia	2

13.4.4. Buscando Causas

Um problema é gerado por várias causas, que podem também ser geradas por outras causas. Descobrir a causa raiz de um problema é um processo de investigação que pode ser documentado em um **Diagrama de Espinha de Peixe**, com o da Figura 13.3, também conhecido como **Diagrama de Causa Raiz**.

Esse diagrama é simples de desenhar ou construir, mas certamente esconde toda uma dificuldade de levantar corretamente as informações que revela. Como muitas propostas de análise, o processo de construção, e o aprendizado dele decorrente, é muitas vezes mais importante que o registro completo final.

O Diagrama de Espinha de Peixe indica as causas para um problema de maneira hierárquica. Para cada causa pode haver outras causas, e assim sucessivamente. O objetivo é a partir de um problema achar as causas raiz.

Ele parte da premissa que normalmente problemas são percebidos por um sintoma, e que o verdadeiro problema está escondido em uma camada de problemas sucessivos.

O processo de criação é mais produtivo se feito por um conjunto de pessoas, em uma reunião semelhante a um *brainstorm*. Nesse caso é importante que todos concordem com o efeito ou problema inicial a ser discutido. Sucessivamente devem ser feitas perguntas sobre “por que” o problema ocorre, ou seja, quais suas causas.

Cada linha de causalidade deve ser seguida até sua causa raiz. Isto é, a prática mais recomendada é trabalhar em profundidade, com perguntas sucessivas sobre por que algo acontece. Isso é similar a técnica dos 5 porquês, usada da mesma forma.

Apesar de no início o diagrama poder ser construído de forma livre, é interessante organizá-lo de alguma maneira para a apresentação. Ramos lotados podem ser divididos, ramos quase vazios podem ser unificados. Também é necessário ter uma noção da

13. Oportunidades e Problemas

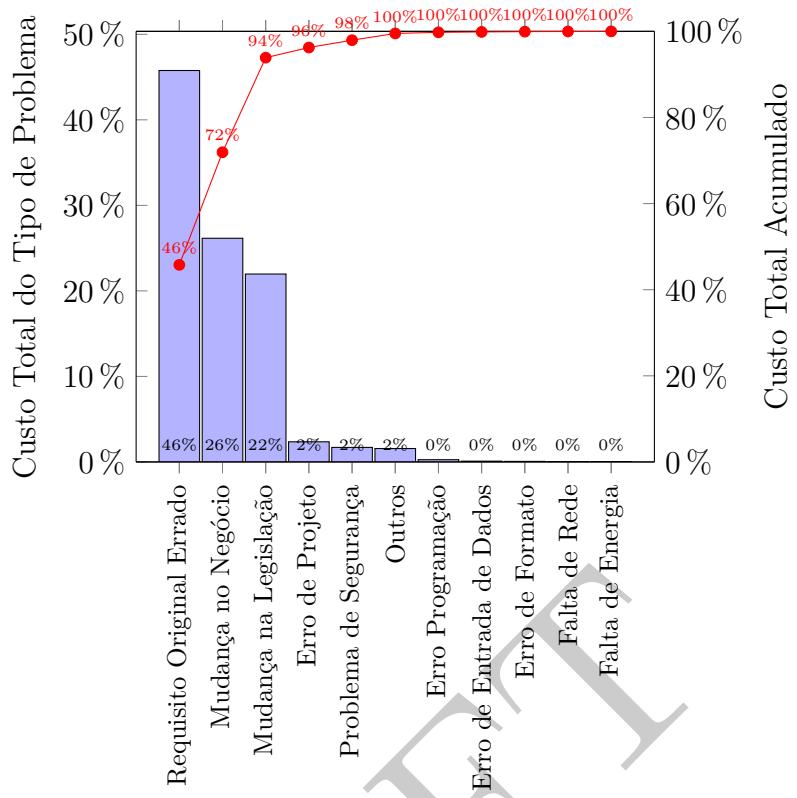


Figura 13.2.: Pareto com custo

importância das causas, porque não é necessário gastar muito tempo em uma causa que não é realmente impactante para o problema.

A construção pode ser feita em um quadro branco ou com um projetor. É importante que todos os participantes tenham acesso completo ao quadro.

O diagrama de Figura 13.3 mostra a investigação de causas para o fato de um restaurante ter longas filas na porta. Pode ser visto que algumas causas conseguiram uma hierarquia de causa raiz mais longa do que outras. Isso não é um problema.

Algumas ferramentas permitem trabalhar com diagramas diferentes mas com a mesma finalidade. Ferramentas de *mind mapping*, por exemplo, permitem não só criar diagramas equivalentes, como permitem também trabalhar com resumos das figuras, como mostrado nas Figuras 13.4 e 13.5

Também é possível usar o diagrama como uma ferramenta pessoal de auxílio a investigação, ou para apoiar a técnicas dos 5 Porquês. Na Figura 13.6, o diagrama é usado para ilustrar o seguinte raciocínio.

1. Tirei uma nota baixa
2. Por que?
3. Porque a prova estava difícil.
4. Por que?

Problemas do Restaurante

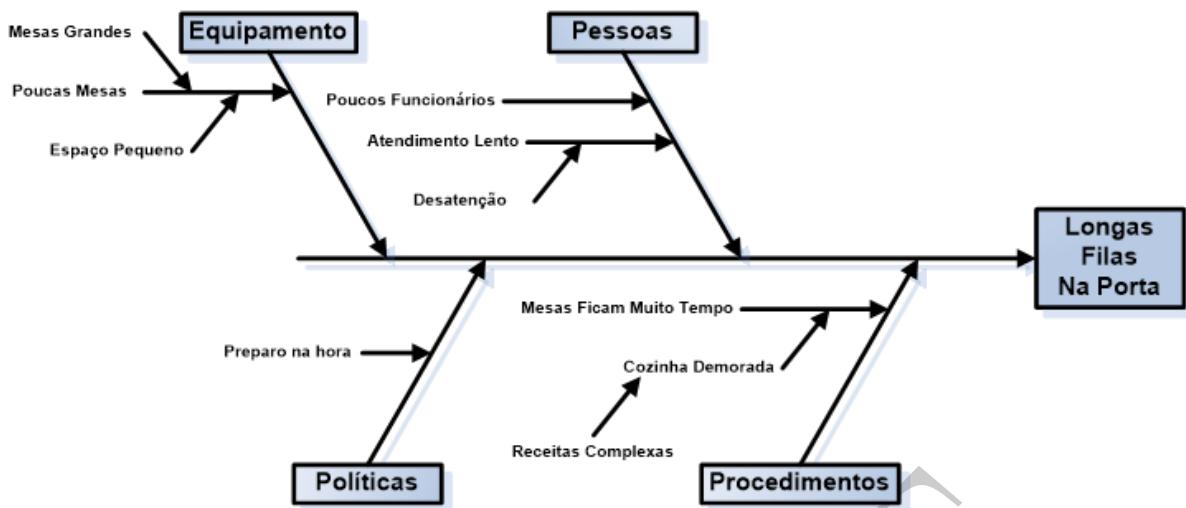


Figura 13.3.: Um diagrama de Espinha de Peixe

Figura 13.4.: Um diagrama de Espinha de Peixe resumido, construído com uma ferramenta de *mind map*.

5. Porque não estudei.
6. Por que?
7. Por que não me organizei.
8. Por que?
9. Acho que não há um motivo para isso.
10. Há outro motivo para tirar uma nota baixa?
11. Sim, não fiz a AD, preparatória para a prova.
12. Por que?
13. Não tive tempo.
14. Por que?
15. Novamente, não me organizei.
16. Por que?
17. Acho que novamente não tenho uma explicação para isso.

13. Oportunidades e Problemas



Figura 13.5.: Um diagrama de Espinha de Peixe expandido, construído com uma ferramenta de *mind map*.

13.4.5. Organização do Diagrama Espinha de Peixe

O Diagrama de Espinha de Peixe pode ser organizado com as causas agrupadas em categorias. Existem pelo menos 3 organizações reconhecidas: para indústrias, para serviços e administração, e para Gestão de Qualidade Total.

As classes para indústrias são:

- pessoal;
- métodos;
- materiais, e
- máquinas.

As classes para serviços e administração são:

- pessoal;
- equipamentos;
- procedimentos, e
- políticas.

As classes para Gestão de Qualidade Total (TQM, *Total Quality Management* são:

- ambiente;
- gestão;
- pessoas;
- materiais;
- equipamento, e
- processo.

Todas essas classes podem ser usadas nas reuniões como inspiração para a busca de causas, fazendo perguntas como “Existe alguma causa de gestão para esse problema?”.

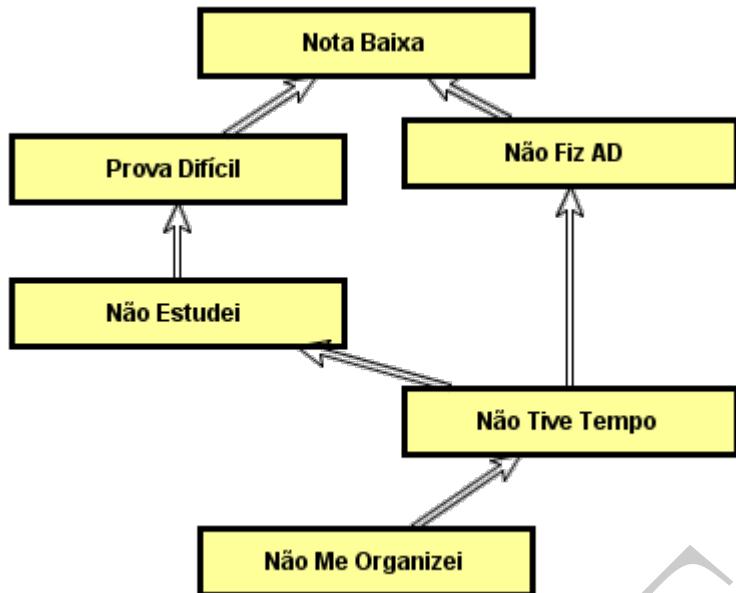


Figura 13.6.: Um Diagrama de Causa Raiz vertical.

13.5. Caracterização do Problema

Identificado o problema é necessário caracterizá-lo claramente. Para isso, é necessário identificar:

- o problema;
- quem é afetado pelo problema;
- seu efeito no negócio e nas partes interessadas;
- o impacto financeiro;
- suas causas e sua causa raiz;
- seu causador, seja ele um processo, um dado, um profissional, etc.;
- pelo menos uma sugestão de solução, e
- os benefícios que uma solução pode trazer.

Alguns desses dados podem ter sido levantados de forma aproximada na identificação do problema. Nesta fase é importante detalhá-los e documentá-los.

13.6. Oportunidade e 5W2H

As oportunidades, ou problemas, bem identificadas devem responder as perguntas 5W2H. Perguntas comuns a serem respondidas são:

- Por que são necessárias? Por que acontecem?
- O que elas são? Qual a causa raiz? Qual o efeito na empresa?

13. Oportunidades e Problemas

- A quem atendem? A quem incomodam?
- Quando elas ocorrem, quando elas devem ser aproveitadas?
- Quanto elas custam? Quanto elas permitem lucrar?
- Onde ocorrem? Onde podem ser resolvidas ou aproveitadas?

13.6.1. Buscando Oportunidades com as Partes Interessadas

A melhor maneira de levantar oportunidades é entender simultaneamente: o que o cliente deseja, o que está funcionando agora, quais os problemas atuais e quais oportunidades existem para um novo sistema.

As entrevistas iniciais para levantamento de necessidades de informação são geralmente feitas com executivos. Um bom conjunto inicial de questões que podem ser feitas é apresentados na lista a seguir (Gillenson e Goldberg, 1984).

- Quais seus objetivos?
- Quais suas responsabilidades?
- Que medidas você usa?
- Que informações você precisa?
- Quais são seus problemas de negócio?
- Que mudanças você vê no futuro (que vão impactar a infra-estrutura do seu negócio)?
- Quais são os fatores críticos de sucesso?
- Qual a informação mais útil que você recebe?
- Como você classificaria as informações que recebe quanto à adequação, validade, duração, consistência, custo, volume, etc.?

Alguns pontos podem ser notados sobre essas perguntas. A pergunta sobre objetivos muitas vezes pode não ser muito bem respondida, assim a segunda pergunta, sobre responsabilidades, permite uma resposta mais prática e mais fácil de ser dada.

A terceira pergunta visa buscar uma compreensão sobre os dados importantes para o entrevistado, abrindo uma sequência de perguntas sobre o tema.

A quinta pergunta, sobre os problemas de negócio, é a mais importante do conjunto e deve ser dada a maior atenção e quantidade de tempo disponível.

É interessante que o problema seja estruturado em um formato causa-efeito, por exemplo: "por causa da falta de dinheiro, o resultado é que não é possível comprar peças de reposição". Também é importante verificar se a causa primordial do problema é um processo ou um dado.

Além disso, o entrevistador deve tentar obter alguma informação de valor (financeiro) sobre o impacto do problema, seja em valores absolutos ou relativos, objetivos ou subjetivos.

13.7. Propondo Oportunidades ao Cliente

O analista de sistemas, com sua experiência tecnológica e de negócio, também pode oferecer oportunidades aos seus clientes.

Uma **oportunidade tecnológica**, como diz o nome, é uma tecnologia específica a ser usada na implementação e que oferecerá alguma vantagem ao cliente. Por exemplo, ao implantar um novo sistema de estoque podemos oferecer a entrada e saída de produtos pelo uso de código de barras. A oportunidade tecnológica não altera a funcionalidade que o cliente necessita, registrar a entrada de um item no estoque, mas sim sua forma de implementação.

Algumas vezes também podemos oferecer **oportunidades de negócio**.

Devemos ter muito cuidado ao propor oportunidades, pois podemos estar criando falsos requisitos, discutidos no Capítulo 19. Um exemplo típico de falso requisito, que costuma ser proposto por analistas, é a proliferação de relatórios em sistemas que na prática usam apenas alguns.

Uma oportunidade de negócio deve ser exaustivamente discutida com o cliente de forma a ficar claro que ela realmente traz benefícios, que esses benefícios são consideráveis, que o risco do projeto não aumenta muito e que ela será realmente utilizada.

Um exemplo que podemos dar é, em um controle de estoque, a funcionalidade de prever que produtos estão próximos de sua data de vencimento. Essa não é uma função normal de sistemas de estoque. Exige um custo adicional não só de desenvolvimento, mas também de operação, como identificar a data de vencimento, controlar diferentes datas para um produto, etc. Em muitos contextos, essa função pode parecer interessante mas ser, na prática, inútil. Em uma oficina mecânica, por exemplo, ela pode ser totalmente inútil. Em um grande mercado, ela pode ser bastante útil. Porém em um pequeno mercado, onde o proprietário tem na verdade um controle mental do estoque e precisa do sistema de estoque apenas para fazer um controle legal ou financeiro, essa funcionalidade pode parecer útil a princípio, mas nunca ser usado no dia a dia.

13.8. Oportunidade e Valor

As oportunidades devem produzir resultados úteis para a empresa, como acelerar processos, eliminar passos desnecessários, reduzir erros na entrada e saída de dados, aumentar a integração entre sistemas, aumentar a satisfação do usuário e facilitar a interação com os parceiros externos da organização (fornecedores, representantes e clientes)(K. E. Kendall e J. E. Kendall, 2013).

A prática IRACIS ajuda a buscar oportunidades, ou a analisar se algo é ou não uma oportunidade, que é tratada na Seção 14.5.2, busca encontrar valor por meio de aumentar as receitas, diminuir os custos e melhorar os serviços.

13. Oportunidades e Problemas

Já os elementos de valor, tratados na SubSeção 2.3.1, permitem também buscar ou analisar outras formas de valor, tanto para pessoas físicas, em sistemas B2C, como para pessoas jurídicas, em ambientes B2B.

Claramente, uma oportunidade (problema) precisa de um valor associado a ela para que se possa reivindicar o seu aproveitamento (solução). Quanto mais este valor puder ser resumido em um valor financeiro, mas fácil é verificar se vale a pena investir em um sistema de software para aproveitá-la (resolvê-lo). Calcular o impacto financeiro de uma oportunidade ou problema é responsabilidade da área de negócio e não do analista de sistemas. Porém, cabe ao analista analisar se o benefício pretendido é factível com o projeto sendo proposto.

Para fazer a decisão financeira final podem ser usadas as técnicas ROI, IRR e NPV apresentadas na Seção 2.4.

13.9. Exercícios

Exercício 13.9-1: Associe as características que devem ser levantadas para um problema às perguntas 5W2H.

Exercício 13.9-2: Analise o local onde você estuda e os serviços de software fornecidos a você. Faça uma lista de problemas e oportunidades. Discuta as causas raiz dos problemas.

Exercício 13.9-3: Vá para o site <http://jogodeanalisedesistemas.xexo.net/> e visite a Livraria Resolve. A partir da sua visita liste os problemas e oportunidades.

14

Project Model Canvas

Unless commitment is made,
there are only promises and
hopes, but no plans

(Peter F. Drucker)

Conteúdo

14.1.	Uso do 5WH	147
14.2.	As Fases da Metodologia	148
14.3.	Requisitos para a Reunião	149
14.4.	Um Exemplo de Projeto	149
14.5.	A Fase de Conceber	149
14.6.	A Fase de Integração	174
14.7.	A Fase de Resolver	174
14.8.	A Fase de COmpartilhar	175
14.9.	Exercícios	175

Por que Project Model Canvas?

Um projeto de software precisa de algum planejamento. O Project Model Canvas fornece uma forma rápida e resumida de planejar, adequada a metodologias prescritivas ou ágeis e também ao contexto deste livro.

14. Project Model Canvas

Project Model Canvas (PMC) (Finocchio Jr., 2013)¹ é uma metodologia de criação de projetos visual e cooperativa, baseado no uso de um Canvas, i.e., um quadro branco com áreas demarcadas que devem ser preenchidas com informações pré-determinadas por um grupo de pessoas em uma reunião de trabalho. A Figura 14.1 mostra o canvas do Project Model Canvas ainda em branco.

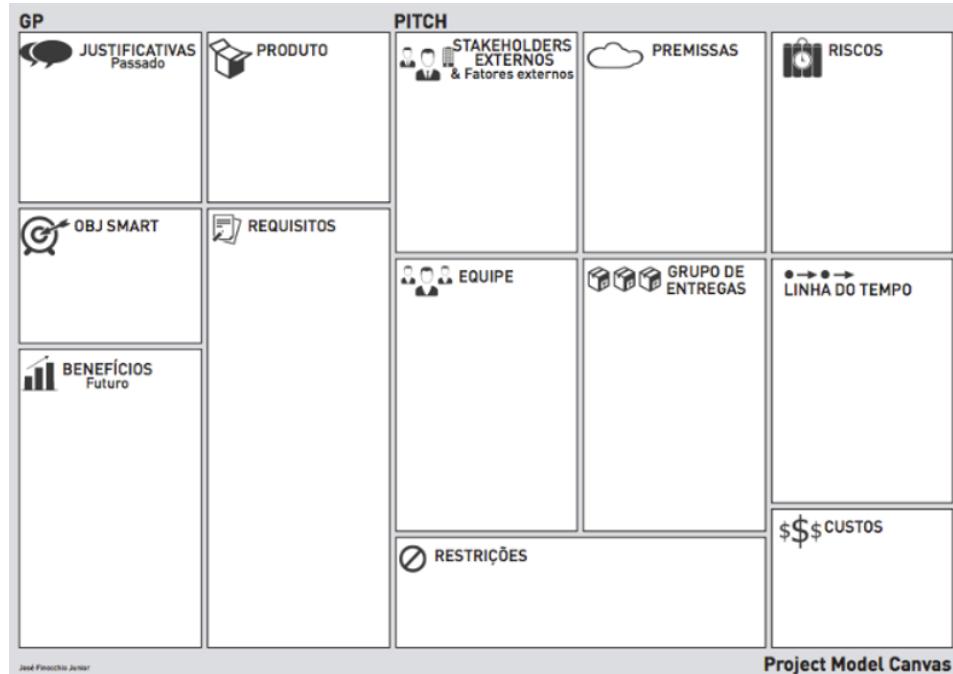


Figura 14.1.: O canvas do Project Model Canvas. Fonte:(Finocchio Jr., 2013)

Essa metodologia é simples, visual e colaborativa. Ela segue um passo a passo sobre um canvas, que vai sendo preenchido e validado no processo. Uma das suas metas é acelerar o processo de definição do projeto, focando no que é mais importante, e diminuindo a burocratização típica dos projetos baseados no PMBOK (Finocchio Jr., 2013).

Os objetivos da metodologia são(Finocchio Jr., 2013):

- focar no que é essencial para o projeto;
- permitir a inovação e a criação de ideias;
- facilitar a colaboração entre as partes interessadas;
- levar as partes interessadas a uma convergência e a criação de um acordo, e
- divulgar as decisões tomadas.

Para alcançar esses objetivos alguns princípios são seguidos(Finocchio Jr., 2013):

- usar um mapa visual que permita a compreensão do todo;

¹Essa metodologia é distribuída com uma versão Creative Commons com atribuição, Não Comercial e Sem Derivações 3.0 Não Adaptada. Isso significa que temos o direito de copiar ou redistribuir o material em qualquer suporte ou formato, mas não com o intuito comercial. Como autor, não entendo como isso pode afetar a versão final comercial deste livro, porém no momento, como ele é distribuído gratuitamente, não há problema em seu uso.

- indicar relações entre os itens necessários por meio de agrupamento e proximidade;
- simplificar os métodos tradicionais;
- estabelecer bases e acordos entre todas as partes interessadas, incluindo clientes e equipe, e
- possuir um passo a passo que guia a metodologia.

Neste capítulo alguns conceitos que são detalhadamente tratados mais a frente no livro são explicados de forma mais simples.

O que é um canvas?

Um canvas é um quadro, geralmente desenhado em um quadro branco ou impresso em formato A3 ou maior, com áreas determinadas que devem ser preenchidas, normalmente com *post-its*, com um objetivo por um grupo de pessoas trabalhando de forma colaborativa. Canvas entraram na moda com o lançamento do Business Model Canvas(Osterwalder, Pigneur e Clark, 2010) e foram se expandindo para várias áreas de aplicação.

14.1. Uso do 5WH

Uma das características mais importantes do PMC é o forte embasamento na prática 5W2H, que define suas regiões e a ordem do passo a passo(Finocchio Jr., 2013) de construção.

Isso é mostrado na Figura 14.2, onde podemos ver que a ordem, da esquerda para direita é: “por que?”, “o que?”, “quem?”, “como?”, e, no final, “quando?” e “quanto?”(Finocchio Jr., 2013).

Esta ordem é típica de projetos onde há um encomenda cliente encomendando algo para um fornecedor. Em projetos onde uma empresa se propõe a desenvolver algo para o mercado é mais comum que as partes interessadas sejam identificadas antes, para definir o público do projeto. Nada impede que isso seja feito com esse mesmo Canvas.

14. Project Model Canvas

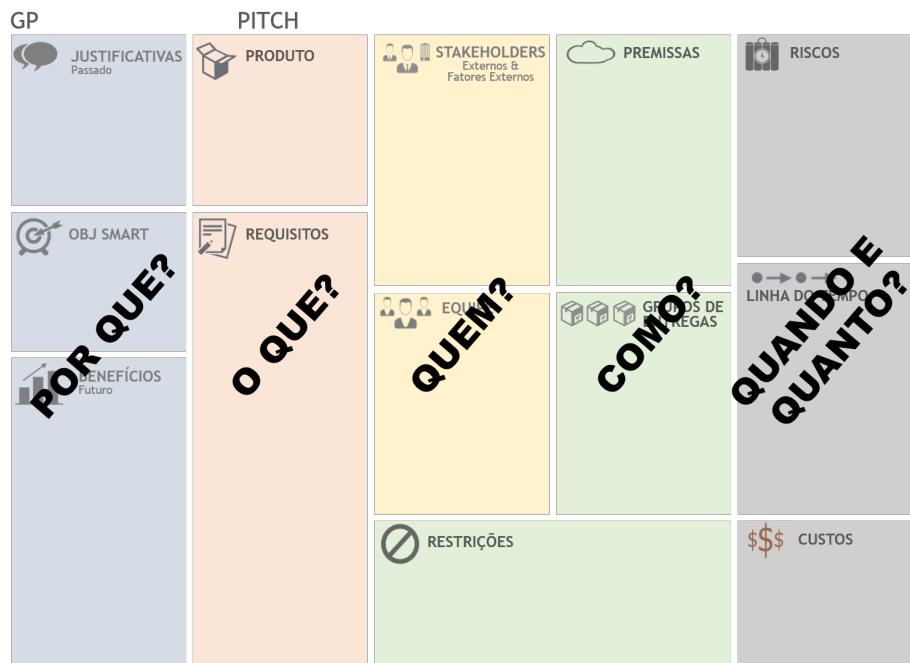


Figura 14.2.: O Project Model Canvas mostrando o 5W2H. Fonte:(Finocchio Jr., 2013)

14.2. As Fases da Metodologia

A Figura 14.3 mostra em BPMN as fases principais da Metodologia PMC, que possui quatro fases em sequência. O canvas é preenchido na primeira parte e depois, na fases seguintes, é usado melhorado e depois usado para criar a comunicação. As fases são(Finocchio Jr., 2013):

1. **conceber**, onde o PMC é preenchido, por meio de um fluxo de trabalho de 12 passos, cada um equivalente a um quadro do canvas;
2. **integrar**, onde são feitas as conferências e amarrações necessárias entre os blocos;
3. **resolver**, onde são resolvidos as questões de balanceamento de projeto entre a equipe, os clientes e patrocinadores, e
4. **compartilhar**, onde são comunicadas as informações levantadas sobre o projeto.

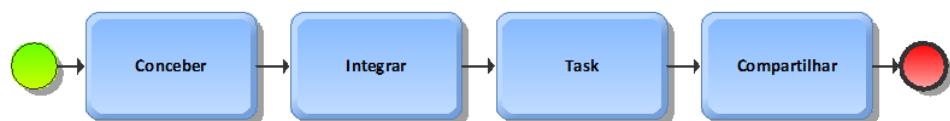


Figura 14.3.: Passos da Metodologia PMC em BPMN. Fonte: próprio autor

Este texto foca na fase de **Conceber**, claramente a fase onde a maior parte das informações é criada.

14.3. Requisitos para a Reunião

Todo canvas é criado com o objetivo de ser ao mesmo tempo uma ferramenta de trabalho e um registro de uma reunião cujo objetivo principal é preenchê-lo, de forma colaborativa, com a participação de várias partes interessadas no projeto.

Assim também é o PMC. Como é um canvas com o objetivo de definir as principais informações de um projeto, três tipos de participantes são importantes nessa reunião:

- pessoas que tenham **conhecimento sobre o que é gerenciamento de projeto** e possam ajudar a manter o foco de cada item tratado dentro das características esperadas, como um gerente de projetos;
- **especialistas da área de negócio**, que possam trazer as verdadeiras necessidades do projeto e concordar e aprovar os resultados propostos, e
- **membros da futura equipe de execução do projeto**, que possam propor soluções, propor e concordar com os resultados do projeto.

14.4. Um Exemplo de Projeto

Para poder explicar como funciona o PMC vamos usar um projeto exemplo. Esse projeto é a realização de um software para celular que ajude um professor a controlar os livros que empresta para os alunos e outras pessoas, porque tem dificuldade de recuperar os livros que empresta porque acaba se saber para quem emprestou.

Basicamente o professor deseja ter uma forma de cadastrar para quem ele está emprestando um livro de sua biblioteca particular, porém ele só quer cadastrar livros e pessoas na hora do empréstimo.

O projeto será realizado como um projeto de final de curso, sendo executado dentro dos regulamentos para sua execução, tendo que ser apresentado a uma banca de professores que vai aprová-lo ou não.

14.5. A Fase de Conceber

A fase de conceber é onde o PMC é inicialmente preenchido. Isso é feito a partir da definição do *pitch* e seguida por um passo a passo pelos quadros que o compõe, da esquerda para direita e de cima para baixo, na ordem indicada na Figura 14.4.

14. Project Model Canvas

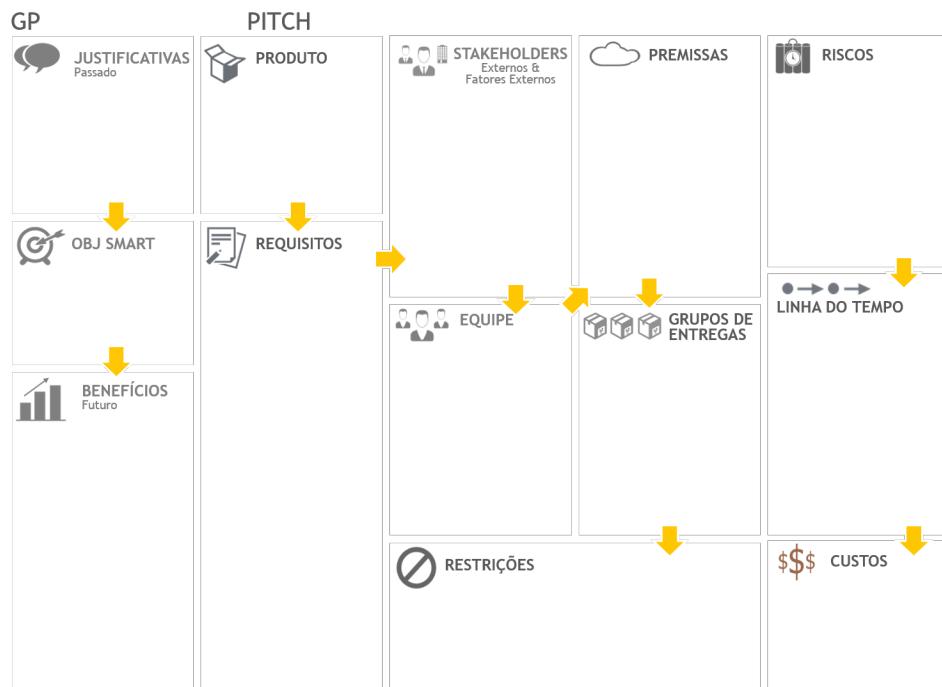


Figura 14.4.: A ordem de trabalho do Project Model Canvas. Fonte:(Finocchio Jr., 2013)

14.5.1. O Pitch

Pitch é uma palavra na moda que significa um fala rápida que é usada para convencer alguém de um projeto. Normalmente é uma fala de 3 a 5 minutos, que pode ser dita em uma viagem de elevador, de onde vem o termo *elevator pitch*. A ideia do *elevator pitch* é aproveitar uma viagem de elevador que você tem com alguém que pode viabilizar seu projeto para convencê-lo que vale a pena ouvir mais sobre ele.

No PMC o pitch é basicamente um nome atraente para o projeto. Existem muitas formas de dar esse nome, porém uma das formas interessantes é garantir que ela tenha 2 características: primeiro ela deve fazer uma referência a uma coisa conhecida, segundo ela deve mostrar como o projeto é diferente dos outros. Podemos citar alguns exemplos de projetos de software que usam essa regra, onde a parte da diferença está sublinhada:

- um sistema de estoque para uma loja de roupas brancas;
- um sistema de controle de compra e venda para um camelô;
- um aplicativo de registro de entrevistas para levantamento de fluxo de veículos entre cidades, e
- um software de aprendizado de máquina para detectar defeitos em impressões

No nosso exemplo, o pitch será “Controlar os livros emprestados”, que é a vontade explicitada pela principal parte interessada, o professor. O gerente do projeto será o autor deste livro. O PMC então deve ser preenchido como na Figura

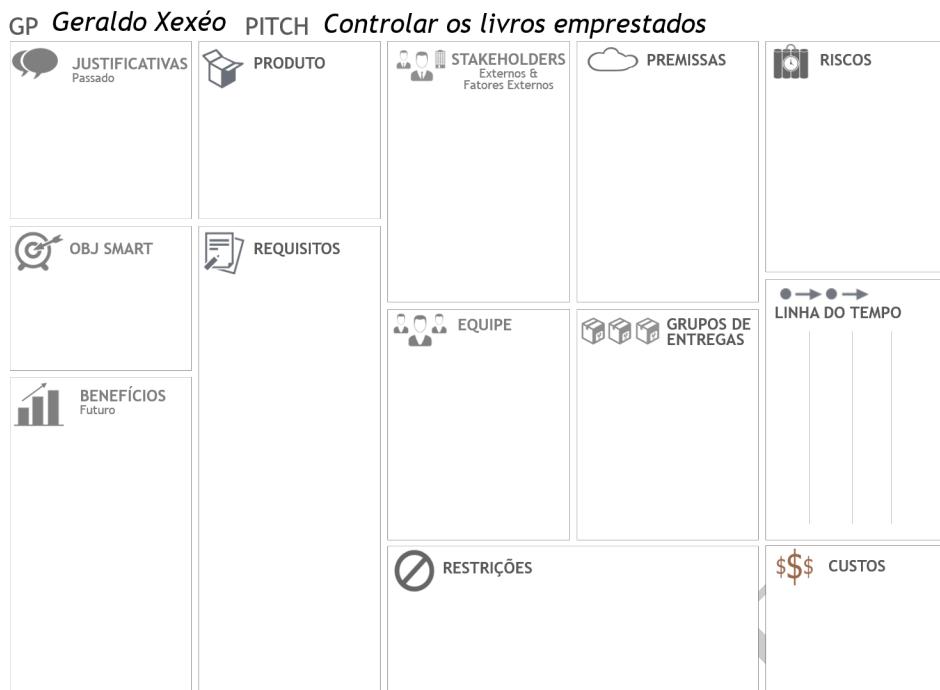


Figura 14.5.: Preenchendo o pitch do PMC.

14.5.2. Por Que?

Nessa primeira fase buscamos um motivo para executar o projeto. Para isso partimos de um estado inicial, que dá a justificativa do projeto, passamos para o objetivo do mesmo e no final definimos os benefícios que o projeto deve trazer(Finocchio Jr., 2013).

Um projeto normalmente acontece por uma necessidade identificada, seja ela na forma de um problema que precisa de solução, uma demanda que precisa ser atendida ou uma oportunidade que pode ser explorada(Finocchio Jr., 2013). Alguns projetos típicos na área de software são correções de falhas encontradas, adaptação a novos requisitos ou requisitos que mudaram, e atendimento a novas necessidades de negócio.

O PMBOK cita as seguintes considerações estratégicas para um projeto(PMI, 2017):

- demanda de mercado;
- oportunidade estratégica ou necessidade de negócio;
- necessidade social;
- consideração ambiental;
- solicitação de cliente;
- exigência jurídica ou de regulamentação, e
- problema existente ou previsto.

Podemos pensar que estamos buscando mostrar um quadro onde de uma realidade, em uma margem de um rio, se passa a uma outra realidade melhor, com benefícios, na outra margem do rio, por meio de uma “ponte”, que são os objetivos que definem a melhoria.

14. Project Model Canvas

Justificativas

A **justificativa** é uma descrição do estado atual que precisa ser modificado por meio dos objetivos.

Basicamente este quadro possuirá uma lista de problemas ou demandas que precisam ser atendidas pelo projeto de alguma forma.

Para o nosso projeto encontramos três justificativas:

- não sei com quem está o livro para pedir devolução;
- alunos não devolvem voluntariamente os livros emprestados, e
- livros não estão catalogados.

Deve ser usado um *post-it* para cada justificativa.

Com essas justificativas adicionadas, o PMC ficará como o da Figura 14.6.

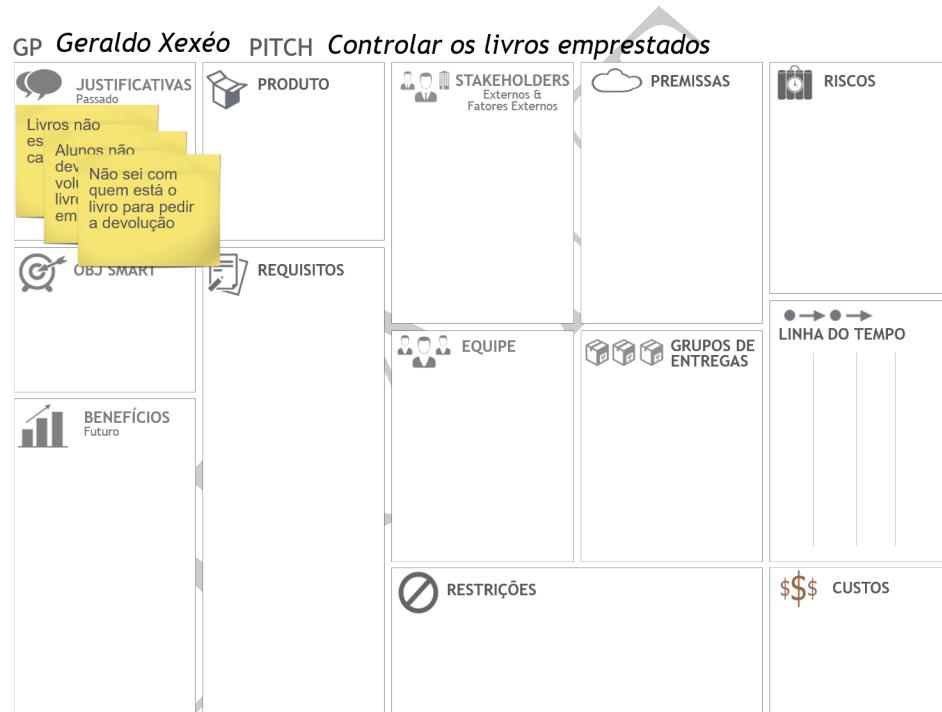


Figura 14.6.: Preenchendo a justificativa do PMC.

Objetivos SMART

Todo projeto tem um objetivo, que é o resultado final que será alcançado, **em termos de negócio**, quando o projeto ficar pronto. Para projetos de software pode haver entre os membros da reunião, ou outras partes interessadas, uma certa confusão entre o objetivo e o produto, no nosso caso sempre um software, seja novo ou modificado, e o objetivo é o que será alcançado com o produto.

Um bom objetivo deve atender aos critérios conhecidos como SMART, uma sigla que significa:

- específico (*Specific*);
- mensurável (*Measurable*);
- alcançável (*Attainable*);
- realístico (*Realistic*), e
- delimitado no tempo (*Time based*).

Há alguma variação para os termos em inglês, como descrito por Kevin L (2016):

- S para *simple* (simples), *strategic* (estratégico) ou *sustainable* (sustentável);
- M para *meaningful* (significativo), ou *motivating* (motivador);
- A para *achievable* (alcançável), *agreed* (acordado), *ambitious* (ambicioso), ou *aligned with corporate goals* (alinhado com objetivos corporativos);
- R para *relevant* (relevante), *resourceed* (com recursos), *reasonable* (razoável), ou *results-based* (baseado em recursos) e
- T para *trackable* (rastreável), *time-based* (baseado em tempo), ou *time/cost limited* (limitado em tempo e custo).

Deve ser usado um *post-it* para cada objetivo e cada projeto deve ter apenas um objetivo ou no máximo dois.

Para o nosso projeto o objetivo vai ser “Controlar o empréstimo de livros a partir de 1/12/2016”. Veja que foi colocada uma data no objetivo dar um limite de tempo ao projeto, mas não foi definido como esse controle de empréstimos será feito, o que virá apenas na próxima fase. Também **não foi dito o que** apoiará esse objetivo a ser atingido, o que será descrito na próxima coluna do PMC.

Com o objetivo adicionado, o PMC ficará como o da Figura 14.7.

14. Project Model Canvas

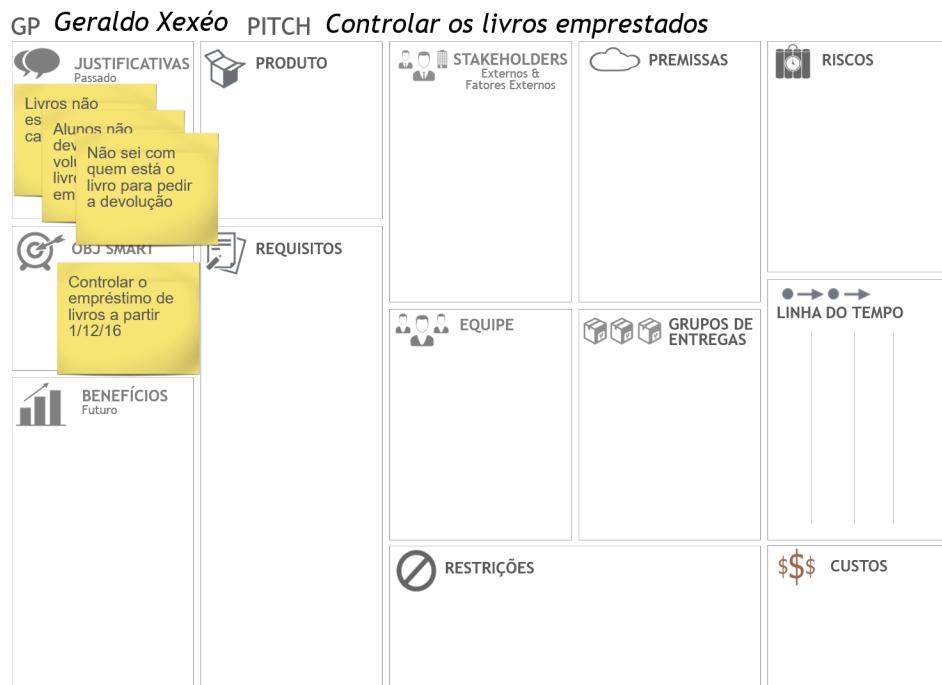


Figura 14.7.: Preenchendo os objetivos SMART do PMC.

Benefícios

Todo o projeto busca agregar algum valor as partes interessadas. Esse valor agregado é definido pelos benefícios a serem alcançados, e descontado do investimento feito no projeto.

Um **benefício** é algo que **gera valor²**, e sempre é **um efeito positivo para a organização**, como(Finocchio Jr., 2013; Gane e Sarson, 1979):

- aumento de receita;
- diminuição de custos;
- melhoria dos serviços prestados;
- uso mais eficiente de ativos;
- melhoria da imagem, e
- atenuação de impactos sociais e ambientais.

Esse valor pode ser tangível ou intangível. Um benefício é considerado tangível quando é material, pode ser tocado ou possuído, ou, mais genericamente, quando podemos associar diretamente a ele uma valor financeiro. Já um benefício intangível é mais difícil de medir, principalmente mais difícil de associar a um valor financeiro, mas ainda importante, como por exemplo a confiança na marca, ou a melhoria no relacionamento com os clientes. Mesmo valores intangíveis podem muitas vezes ser medidos de alguma

²No Capítulo sobre valor, Capítulo 2 são discutidas várias formas de entender o que significa essa palavra para as partes interessadas

forma(Wren, 2003). A confiança na marca, por exemplo, pode ser medida por uma pesquisa de opinião ou mesmo resultados de pesquisas públicas.

Alguns benefícios intangíveis, ou mesmo tangíveis, podem ser tornados mais tangíveis, ou mais ligados a resultados financeiros, por meio da investigação da motivação. Wren (2003) sugere usar a pergunta “Mas por que?” ou “E daí?” (*So what?*). Por exemplo, supondo que a benefício original do cliente seja aumentar o número de contatos, a seguinte sequência de perguntas e resposta é possível(Wren, 2003):

Cliente: Queremos aumentar o número de contatos de 50 para 100 por semana, por vendedor.

Analista: Mas por que?

C: Porque cada 10 contatos trazem 1 venda

A: E daí?

C: Por que a venda média é de R\$10.000,00.

A: E daí?

C: Por que queremos aumentar as vendas em 50.000,00 reais por semana por vendedor.

Os benefícios devem ser relacionados a capacidade do objetivo de resolver os problemas e demandas que aparecem na justificativa. Se não forem alinhados, haverá um problema no projeto, pois ele não estará focado em dar resultados positivos para as justificativas apresentadas(Finocchio Jr., 2013).

Deve ser usado um *post-it* para cada benefício. Adicionalmente, é interessante colocar um *post-it* menor em cada benefício dando o grau de contribuição desse benefício ao projeto. Para isso é interessante usar apenas uma escala verbal, como “baixo”, “médio”, ou “alto”(Finocchio Jr., 2013).

É interessante notar que os benefícios do PMC podem ser associados ao que Ruble (1997) chama de **metas**, sendo que **cada meta precisa ter um métrica que possa ser medida antes e depois da implementação do sistema**. Então, como extensão ao método PMC é recomendável não só listar o benefício, mas no mesmo *post-it* anotar como esse benefício pode ser medido, seja direta ou indiretamente.

Os objetivos esperados no nosso exemplo são:

- os livros emprestados serão recuperados, com o grau de contribuição “muito alto”;
- será possível saber com quem estão os livros emprestados, com grau de contribuição “médio”, e
- ao longo do tempo os livros serão catalogados, com grau de contribuição “médio”.

Nesse exemplo as medidas não foram incluídas na imagem, porém é possível associar a cada benefício uma medida. Por exemplo, “livros perdidos durante o ano”, “livros emprestados sem identificação do emprestador”, e “quantidade de livros catalogados”.

Com a adição dos benefícios, o PMC ficará como o da Figura 14.8.

14. Project Model Canvas

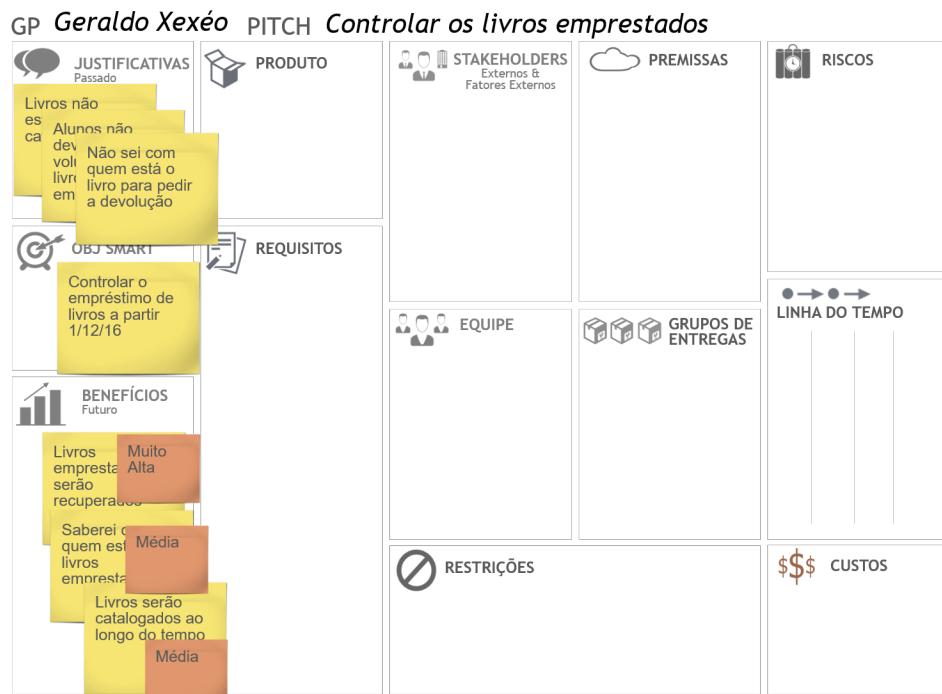


Figura 14.8.: Preenchendo os benefícios do PMC.

Cuidados com os Benefícios e Metas

Devemos tomar cuidado, porém, com as armadilhas que existem na escolha dos benefícios e metas. Primeiro, é muito comum que um cliente deseje um benefício que não pode ser alcançada por um sistema com o objetivo definido. A questão, nesse caso, se resume a negociar a mudança do benefício ou negociar a mudança do objetivo.

Como exemplo, podemos citar o caso de uma proposta de sistema que apresentava um sistema de controle de pedidos para os fornecedores baseado em pedidos de clientes e prometia que o sistema diminuiria o prazo de entrega para os clientes. Analisado o funcionamento da empresa, foi comprovado que era impossível acelerar o prazo meramente controlando os pedidos. Era possível, porém, fazer uma previsão mais acertada do prazo de entrega e estar preparado para atrasos, mediante o acompanhamento dos pedidos.

Outra armadilha é ter uma meta que é afetada por muitas coisas além do sistema. Em um caso simples, podemos citar a proposta de um sistema de reservas de hotel que previa aumentar a ocupação dos quartos. Ora, o nível de ocupação dos quartos de um hotel depende de muitos fatores, inclusive da economia geral. Entendemos que o sistema poderia ter como meta, por exemplo, diminuir o número de reservas não cumpridas. Essa meta é mensurável e poderia ser diretamente influenciada pelo sistema (por meio de verificações com o cliente, por exemplo). É interessante notar que a meta selecionada é um dos fatores que afeta a meta proposta inicialmente. Essa é outra armadilha comum, escolher uma meta (e uma métrica) que na verdade é derivada da meta (e da métrica)

que o sistema tem condições de atingir.

Em relação aos benefícios e metas, se não é possível nenhuma medida que os avaliem, não será possível também saber se foram alcançados, o que os tornam supérfluos.

Buscando Benefícios IRACIS

Uma das formas de buscar os benefícios do PMC é nos guiarmos por três conceitos, ou categorias, lembrados por Ruble (1997), mas originalmente descritos por Gane e Sarson (1979), por meio do acrônimo **IRACIS**³:

1. Aumentar o Faturamento (*Increase Revenue*);
2. Evitar Custos (*Avoid Costs*), e
3. Melhorar o Serviço (*Improve Service*)

A técnica consiste em analisar as justificativas que temos e perguntar como esses três conceitos podem ser atingidos.

Em nosso exemplo, a recuperação de um livro evita um custo, atendendo ao conceito “evitar custos”, saber com quem está os livros e tê-los catalogados o livro melhora o serviço.

Esse método também nos facilita a encontrar métricas, pois elas estarão diretamente relacionadas aos três conceitos.

Buscando Benefícios com Elementos de Valor

Outra forma de buscar benefícios, mais completa que a anterior, a qual contém, é olhar para os Elementos de Valor, discutidos na Subseção 2.3.1.

Cada elemento pode ser entendido como uma classe que representa vários possíveis benefícios específicos. Por exemplo, o elemento “poupar dinheiro” é equivalente ao “evitar custos” do IRACIS, e ambos devem permitir identificar um benefício mais detalhado, como “evitar a necessidade de repor livros perdidos”.

Como os 3 conceitos da IRACIS estão contidos nos 40 elementos, normalmente seria uma vantagem trabalhar com os Elementos de Valor, porém há uma desvantagem dessa prática em relação a IRACIS, que é a dificuldade adicional de trabalhar com 40 conceitos, em vez de simplesmente 3. É possível, porém, estudar antes quais os valores principais da empresa ou da indústria específica e fazer a busca de forma a atender os mais importantes.

Verificando os Benefícios

Wren (2003) sugere que os benefícios sejam verificados *check-list* a seguir.

³Pronunciado como *ear-ack-iss* em inglês.

14. Project Model Canvas

- Está claro se cada benefício é tangível ou intangível?
- É possível identificar cada benefício nas categorias IRACIS, ou como diretamente ligados as KPI da empresa ou ainda infraestrutura básica?
- Os benefícios foram aprofundados por perguntas “E daí?”?
- Cada benefício está identificado como recorrente ou de ocorrência única?
- Todos os benefícios estão claros?
- Todos os benefícios tangíveis estão com o valor bem determinado, possivelmente em um intervalo de valores admissíveis?
- Não existem dependências fora do seu controle, ou fora do impacto do projeto, que podem afetar o resultado dos benefícios

A análise dos benefícios permite ao cliente verificar qual a relação custo/benefício da implantação de um novo sistema. Baseado nas metas o cliente é capaz de fazer uma avaliação econômico-financeira, comparando o preço do sistema, ou melhor, ainda, com o **custo total de propriedade** (TCO Total Cost of Ownership) do sistema, discutido na Seção 2.4.1, com o valor equivalente dos benefícios trazidos pelo sistema.

Por exemplo, um sistema que transforme um trabalho de 1 hora por dia de uma equipe onde cada pessoa que ganha R\$ 10,00 por hora para 15 minutos, poupa R\$ 7,50 por hora por pessoa. Isso corresponde a aproximadamente R\$ 1650,00 por mês, ou ainda R\$ 19800,00 por ano. Como se admite que um investimento tenha retorno em dois anos, o benefício total causado apenas pela aceleração do processo, em dois anos, seria de R\$ 39.600,00 por pessoa. Caso esse fosse o único benefício do sistema, esse valor seria então um limite superior para o TCO.

14.5.3. Ponto de Decisão

A fase “**Por que?**” do PMC é um ponto de decisão do tipo *go-no go* no projeto. Pela análise das justificativas, objetivo e benefícios é possível entender se o projeto traz ou não valor para a organização e qual a importância de fazer esse projeto perante outros.

Isso significa que essa seção do Canvas é sobre o **negócio**, e não sobre o sistema. O objetivo proposto é uma mudança do negócio que será suportada pelo produto, ou resultado, do projeto.

É possível imaginar que uma sessão de trabalho poderia fazer mais de uma proposta desse tipo, para vários possíveis projetos que concorrem por recursos, e em cima disso poderia ser feita uma priorização ou outra decisão que levasse a escolha daqueles que seriam iniciados imediatamente e de outros que seriam deixados para o futuro.

Esse assunto é tratado em áreas da gestão de projetos conhecidas como gestão de programas e gestão de portfólio.

14.5.4. O Que?

Nessa segunda fase definimos o produto do projeto e seus requisitos.

Todo projeto gera um produto, serviço ou resultado único(PMI, 2017). Esse é o entregável principal do projeto, e deve ter características claras e mensuráveis(Finocchio Jr., 2013). Um projeto só alcança o sucesso completo quando esse resultado é entregue de forma completa, atendendo todos os seus requisitos, dentro do prazo, e dentro do custo previsto(Finocchio Jr., 2013)(The Standish Group, 2015).

Para poder iniciar um projeto então é necessário que o produto esteja bem definido, e seus requisitos estejam claros.

Produto

No PMC, o quadro produto identifica de forma sucinta qual o produto, serviço ou resultado que vai ser entregue ao final do projeto e que é o principal entregável e gerador de valor para as partes interessadas.

O projeto deve ser uma demanda clara do cliente, construída sobre a fase anterior, que definiu a justificativa, o objetivo e os benefícios do projeto. Essa demanda deve viabilizar o objetivo e permitir que todos os benefícios sejam alcançados.

Normalmente apenas um produto é feito em cada projeto.

No nosso exemplo o produto é um “aplicativo de smartphone para o controle de empréstimo de livros”.

Com a adição do produto, o PMC ficará como o da Figura 14.9.

14. Project Model Canvas

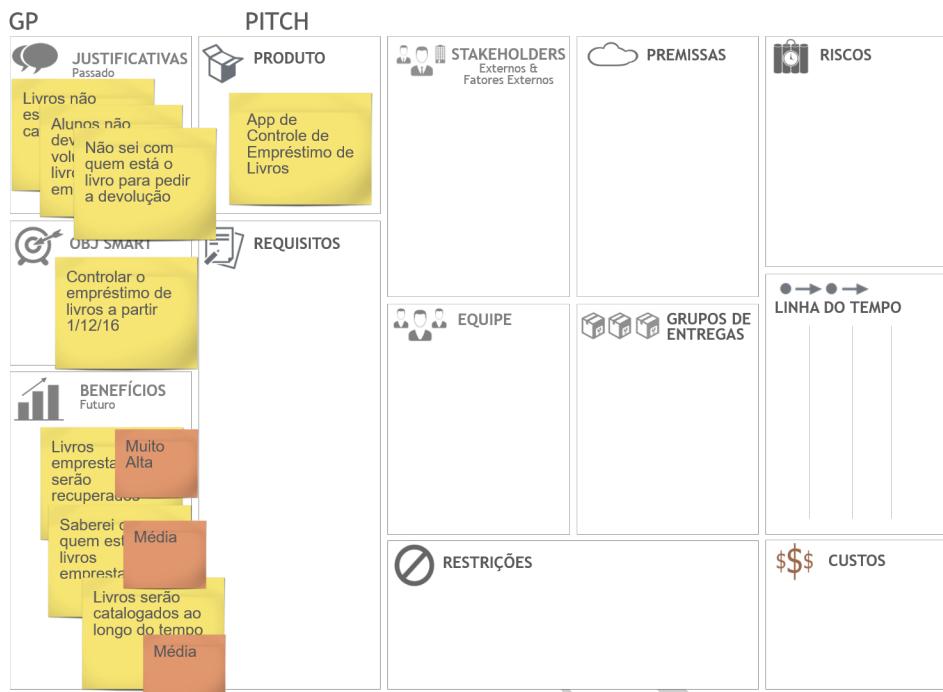


Figura 14.9.: Preenchendo o produto do PMC.

Requisitos

Requisitos são tratadas detalhadamente no Capítulo 19. A definição que adotamos de **requisito** para o PMC é (IEEE, 2010):

- uma condição ou capacidade necessária para um usuário resolver um problema ou atingir um objetivo, ou
- uma condição ou capacidade que precisa ser possuída ou alcançada por um sistema, produto, resultado, serviço ou componente para satisfazer um acordo, contrato, padrão, especificação ou outro documento formalmente imposto.

Os requisitos são a forma como as partes interessadas que agem como cliente comunicam a equipe o que é desejável ou necessário ao produto. Além disso, se um requisito aparece no canvas, significa que foi aceito pela equipe.

Um requisito deve possuir algumas características de qualidade, explicadas no Capítulo 19(IEEE, 2018): ser necessário, apropriado, não ambíguo, completo, consistente, singular, alcançável, rastreável, verificável, correto e conforme.

No caso do PMC, em especial para software, os requisitos podem(Finocchio Jr., 2013):

- especificar um comportamento desejado, sendo um requisito funcional;
- especificar qualidades esperadas do produto, sendo um requisito não-funcional.

Essa é provavelmente a fase mais demorada do método, pois é nela que se define realmente o que é esperado do produto(Finocchio Jr., 2013).

Entre os principais desafios dessa fase está manter os requisitos em um mesmo nível de abstração, e um nível alto sem detalhes desnecessários nessa fase inicial, mas ao mesmo tempo deixar claro o que se deseja do produto.

Antes de passar para o próximo quadro é importante validar esse quadro detalhadamente, observando a relevância de cada requisito, principalmente no contexto do PMC, e se a cobertura dos requisitos deixa claro, no projeto de software, o comportamento esperado do mesmo(Finocchio Jr., 2013).

Os requisitos definem a **demand**a do projeto. No próximo passo serão identificado quem demanda e quem realiza o projeto.

No nosso exemplo os requisitos serão:

- ser um aplicativo *android*;
- registrar os empréstimos;
- registrar os livros ao longo do tempo;
- registrar as pessoas ao longo do tempo;
- mandar avisos para devolver o livro via e-mail e um aplicativo de mensagens;
- enviar arquivo CSV da base por meio de e-mail, e
- facilidade de uso, com poucos passos para o registro.

Com a adição dos requisitos, o PMC ficará como o da Figura 14.10.

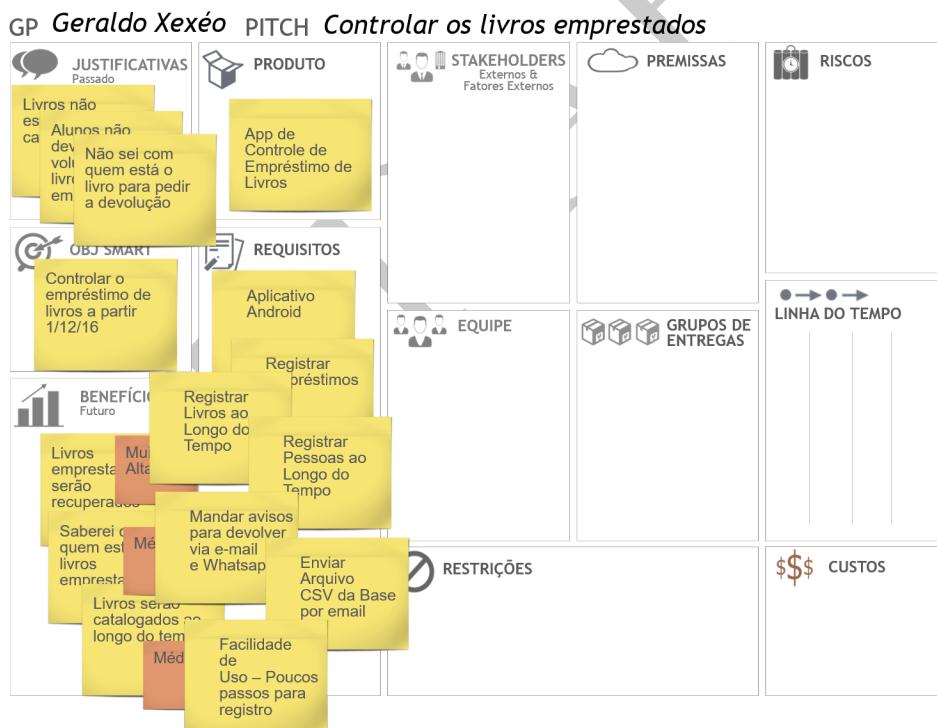


Figura 14.10.: Preenchendo os benefícios do PMC.

É importante notar que enquanto os benefícios estão ligados ao efeito do produto ou

14. Project Model Canvas

serviço na organização, os requisitos são intrínsecos ao produto e serviço. Inclusive, um sistema que cumpre todos os seus requisitos pode não ter os seus benefícios esperados por causa de algum efeito externo.

14.5.5. Quem?

Nessa terceira fase definimos as partes interessadas e os fatores externos, e também a equipe do projeto.

É importante notar que nessa parte temos que ter noção do que pode e o que não pode ser controlado pelo projeto. O que não pode ser controlado, pode ser monitorado(Finocchio Jr., 2013). Os fatores externos só podem ser monitorados, porém devem ser monitorados ativamente.

Partes Interessadas (**Stakeholders**)

Partes interessadas são tratadas detalhadamente no Capítulo 12. A definição que adotamos de **parte interessada** é “é um indivíduo, grupo ou organização que pode afetar, ser afetada ou sentir-se afetada por uma decisão, atividade ou resultado de um projeto. As partes interessadas do projeto podem ser internas ou externas ao projeto, e podem estar envolvidas ativamente ou passivamente, ou não estar cientes do projeto”(PMI, 2017).

Entre as partes interessadas, necessariamente, estão o cliente principal, o patrocinador, os usuários finais do software e todos os outros que de acordo com a definição podem afetar, ser afetados ou se sentir afetados pelo produto e seus efeitos.

O cliente é aquele que vai receber, e aceitar, o produto final. Ele é importante na formulação de requisitos porque é quem vai aprovar ou não a entrega final(Finocchio Jr., 2013).

O patrocinador é aquele que financia o projeto (Finocchio Jr., 2013)(PMI, 2017). Fornece os recursos, principalmente financeiros, e normalmente é quem possui a autoridade direta para liberá-los. Um patrocinador fraco, sem essa autoridade, é um sinal de risco para o projeto.

Todo fornecedor do projeto também é uma parte interessada, pois afetam diretamente o andamento do projeto.

Também são partes interessadas o governo e os órgãos reguladores, que impõe legislações que podem limitar as possibilidades de um projeto, criando restrições(Finocchio Jr., 2013).

Outras partes interessadas que não podem ser esquecidas são clientes e fornecedores do cliente do projeto, pois eles também podem ser afetados pelo produto(Finocchio Jr., 2013).

É importante notar que algumas partes interessadas podem oferecer resistência ao projeto. Por exemplo, em um projeto onde se aumentará a fiscalização do horário de entrada e saída dos funcionários de uma empresa, pode haver reação a sua implementação.

No mesmo grupo de partes interessadas, o PMC inclui fatores externos, que servem para indicar outras entidades que não são pessoas ou compostas por pessoas, como o clima, o que faz sentido em projetos de construção civil, por exemplo(Finocchio Jr., 2013).

Esse fatores externos a serem considerados influenciam o projeto e tem que ser monitorados. Exemplos são(Finocchio Jr., 2013):

- comportamento da economia;
- disponibilidade de uma tecnologia;
- normas regulatórias, ou
- características culturais onde o projeto será implantado.

Para o PMC, as partes interessadas são responsáveis pela demanda do projeto. Logo, é necessário verificar se todos os requisitos podem ser associados a uma parte interessada.

No nosso exemplo, as partes interessadas identificadas são:

- o professor, que é o cliente e usuário principal;
- os alunos que pegam livros emprestados, e
- a banca de projeto final.

Com a adição das partes interessadas, o PMC ficará como o da Figura 14.11.

14. Project Model Canvas

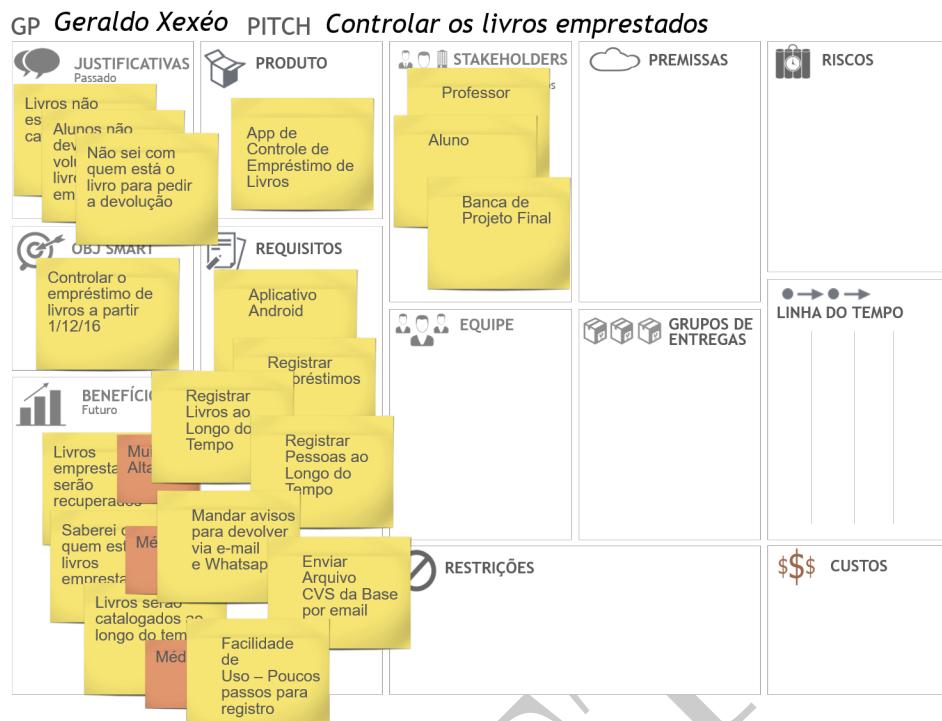


Figura 14.11.: Preenchendo as partes interessadas do PMC.

Equipe

Antes de falar da equipe, é melhor deixar claro que ela, em geral, também é uma parte interessada do projeto. Porém, é comum tratá-la em separado, pois é a equipe que executa o projeto, sendo a responsável por todas suas entregas.

A **equipe** é responsável pela execução do projeto.

A equipe pode ser definida por meio do papel de cada membro, como gerente, programador, testador. Cada papel exige um grupo de habilidades(Finocchio Jr., 2013).

Ela também necessita de ferramentas para executar o projeto, no caso de software computadores e compiladores, por exemplo, parte que não é tratada diretamente pelo PMC.

A equipe também precisa estar disponível durante o tempo do projeto(Finocchio Jr., 2013).

Alguns membros da equipe podem ser diretamente nomeados(Finocchio Jr., 2013). É interessante que os membros nomeados com mais responsabilidade e autoridade dentro da equipe estejam participando da reunião de criação do PMC.

No nosso caso a equipe é formada por:

- o orientador do projeto final, e

- dois alunos.

Com a adição da equipe, o PMC ficará como o da Figura 14.12.



Figura 14.12.: Preenchendo a equipe do PMC.

Tabela RACI

Como citamos na Seção 3.4.2, uma prática interessante em projetos é definir, para cada atividade, quem são os responsáveis por executá-la, os responsáveis por aprová-la, os que devem ser consultados e os que devem ser informados, por meio de uma tabela conhecida como RACI.

Essas duas áreas do PMC, Partes Interessadas e Equipe, podem ser insumo para esta tabela, que já pode ser construída a partir desse nível de detalhe. Isso será mais detalhado na seção 12.9.

Além disso, pensar nesses quatro grupos de partes interessadas pode ajudar a incluir membros esquecidos no PMC. Assim, é importante perguntar: onde estão os decisores, os que vão fazer o trabalho, quem precisa ser consultado e informado.

14.5.6. Como?

Nessa quarta fase definimos como o projeto vai ser realizado, partindo das premissas e restrições e definindo os grupos de entregas.

O PMC segue a estratégia de pensar nas entregas(Finocchio Jr., 2013) e não em atividades. A princípio, estamos seguindo a mesma linha de métodos tradicionais ou ágeis. Entregas podem ser associadas a casos de uso, histórias do usuário, ou outra forma de tratar o que o cliente deseja receber. Atividades implicam em como realizar as entregas e não precisam ser definidas nesse ponto(Finocchio Jr., 2013).

É importante notar que na parte central do PMC podemos ver a existência de uma conexão entre benefícios e requisitos e entre requisitos e entregas. Essa é a linha principal de definição do projeto e tem que estar totalmente consistente.

A parte que define o “Como?” é a que define o trabalho a ser feito pela equipe.

Premissas

Um projeto é um plano que contém incertezas inherentes. O gerente de projeto depende de vários fatores que não podem ser controlados. Por exemplo, qualquer que seja o projeto há um risco (pequeno) de haver um incêndio no local do mesmo e tudo ser perdido⁴(Finocchio Jr., 2013).

Para poder então fazer seu plano, um gerente de projeto precisa assumir que algumas coisas vão acontecer. Em especial, ele deve assumir que algumas partes interessadas cumprirão seus compromissos com o projeto. Assim, por exemplo, se o projeto depende de computadores que vão ser entregues por um fornecedor, o gerente pode assumir que o fornecedor fará a entrega dos computadores na data combinada.

Uma **premissa** é uma suposição que se faz sobre algo que deve acontecer para o projeto ter seu andamento como previsto. Assim, assumimos que as premissas são verdadeiras, mesmo que não estejam sob nosso controle.

Tudo que assumimos e tem impacto no projeto deve ser registrado como premissa.

As premissas são sempre positivas, isto é, supomos que as “coisas vão dar certo”.

Cada premissa pode gerar um risco. Isto acontece porque estamos supondo que o projeto acontecerá com premissas funcionando, mas pode acontecer que uma, ou mais, não aconteçam.

Por exemplo, é possível prever em um projeto de desenvolvimento de software que os recursos, humanos ou equipamentos, estarão todos disponíveis em uma certa data, e todo o projeto ser construído sobre essa premissa, porém há sempre o risco de um recurso não estar disponível na data planejada. Os riscos serão tratados em outro quadro do PMC,

⁴esse tipo de risco, não específico, não deve ser tratado no PMC, a não ser que o projeto envolva um risco específico de incêndio

mais adiante.

As premissas de nosso projeto são:

- o trabalho tem um escopo adequado para um projeto final, e
- celulares *android* são compatíveis entre si.

Com a adição das premissas, o PMC ficará como o da Figura 14.13.



Figura 14.13.: Preenchendo as premissas do PMC.

Grupo de Entregas

A principal técnica de definição do que um projeto entrega é o refinamento sucessivo, sendo que sua técnica tradicional é a Estrutura Analítica de Projeto (EAP), mais conhecida por seu nome em inglês *Work Breakdown Structure (WBS)*(PMI, 2017).

As entregas são componentes menores que compõem o projeto, formando um todo quando totalmente entregues(Finocchio Jr., 2013).

Cada entrega deve, por si, entregar um valor adicional ao usuários. Elas devem, portanto, ser(Finocchio Jr., 2013):

- tangíveis;
- mensuráveis, e
- verificáveis.

14. Project Model Canvas

No PMC devemos focar nas entregas mais relevantes. Essa fase não precisa ser exaustiva, pois ainda estamos em um momento muito preliminar do projeto. Devem ser construídas de forma que, em conjunto, caracterizem a lógica do projeto. Elas devem ser simplificadas em grandes grupos(Finocchio Jr., 2013).

No nosso projeto as entregas são organizadas basicamente pelas telas do aplicativo funcionando:

- tela de empréstimo;
- tela de cadastro de livro;
- outros cadastros, e
- relatórios.

Com a adição das entregas, o PMC ficará como o da Figura 14.14. Veja que as entregas estão organizadas verticalmente, o os *post-it* foram cortados. Usaremos essa organização para trabalhar no quadro linha do tempo.

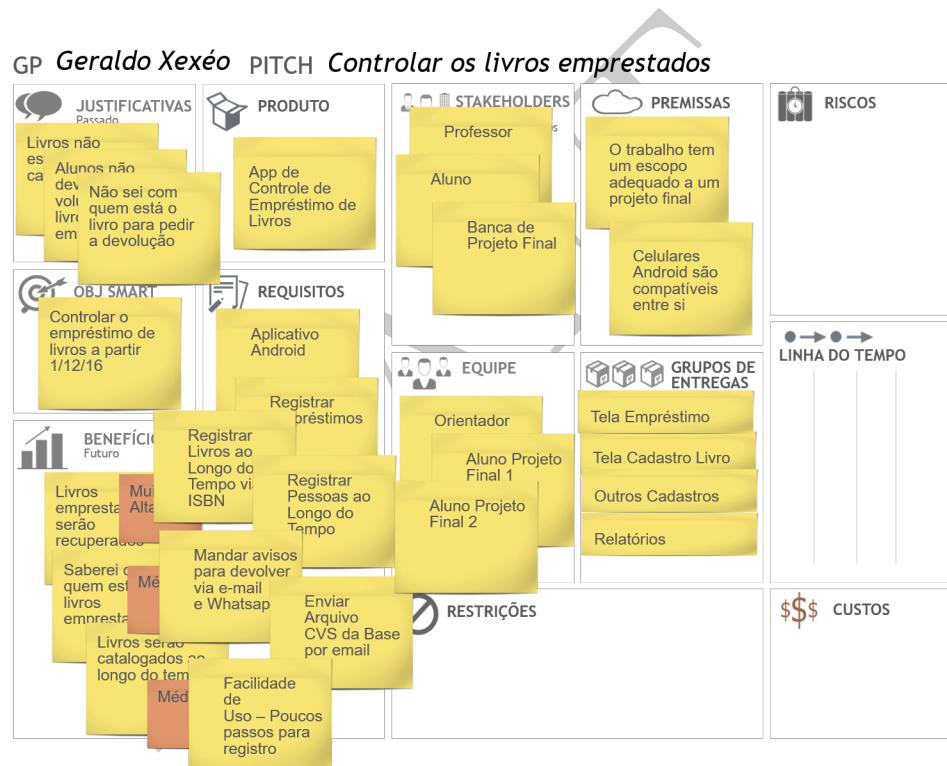


Figura 14.14.: Preenchendo as entregas do PMC.

Restrições

Restrições são limitações, de qualquer origem, que são impostas ao trabalho realizado pela equipe, diminuindo sua liberdade de escolha de opções(Finocchio Jr., 2013). Restrições são discutidas novamente na Seção 19.3.3.

Na verdade, restrições também são um tipo de requisito, mas um tipo tão comum e especial que acabam registradas de outra forma. A principal diferença que podemos colocar é que requisitos são desejados, porém restrições são arbitrariedades impostas por algum motivo.

As restrições devem (Finocchio Jr., 2013):

- ser específicas;
- ser quantificadas;
- indicar o que é limitado, e
- indicar quem impõe a restrição.

Restrições típicas são, por exemplo(Finocchio Jr., 2013):

- períodos de trabalho;
- limites de logística;
- limites de recursos;
- dependências de outros projetos;
- contratos que devem ser seguidos, ou
- padrões tecnológicos.

No nosso projeto as restrições são:

- obrigatoriedade de usar o ambiente cordova, exigida pelo professor;
- limite de tempo de dois períodos letivos, que é o prazo do projeto final, e
- equipe máxima de 3 alunos, também imposta pelo regulamento do projeto final.

Com a adição das restrições, o PMC ficará como o da Figura 14.13.

14. Project Model Canvas



Figura 14.15.: Preenchendo as restrições do PMC.

14.5.7. Quando e Quanto?

Nessa quinta e última fase, definimos riscos e custos do projeto e ainda a linha de tempo de entregas.

O que está sendo feito é um dimensionamento do projeto, por isso é essencial que os riscos façam parte dessa fase(Finocchio Jr., 2013).

Riscos

Riscos são incertezas que podem influenciar o resultado do projeto. Apesar do nome estar normalmente associado a ameaças ao sucesso do projeto, modernamente também tratamos de riscos positivos, também conhecidos como oportunidades(Finocchio Jr., 2013).

Todo **risco** é sempre um evento futuro, que pode ou não ocorrer, revelando uma ameaça ou oportunidade relevante para o projeto.

É importante que os riscos sejam gerenciados, por meio de um processo que inclui a mitigação, a monitoração e o planejamento e a execução da resposta planejada no caso da ocorrência do risco(Finocchio Jr., 2013).

Todo risco possui(Finocchio Jr., 2013):

- uma causa, isto é, o fato ou condição que provoca o acontecimento do risco;
- uma probabilidade de ocorrer;
- um efeito, isto é, as consequências em termos de macro-objetivos do projeto, e
- um impacto no projeto, que determinará sua continuidade, ou não, com ou sem prejuízos ou lucros.

Toda premissa gera pelo um risco, porém esses não são os únicos riscos.

Se um risco ocorre, deixa de ser um risco, para ser um evento do projeto que deve ser tratado com técnicas apropriadas de gestão.

No nosso projeto as causas e os riscos associados são:

- uma mudança drástica no ambiente escolhido no meio do projeto, sendo necessários refazer parte do projeto, com probabilidade de 10%, sendo o efeito um atraso no projeto e impacto 9 em uma escala de 1 a 10, e
- o escopo do projeto ser muito grande para um projeto final e parte do projeto ter que ser abandonada, com probabilidade estimada de 1%, o efeito do projeto ficar incompleto e impacto 8.

Com a adição dos riscos, o PMC ficará como o da Figura 14.16.

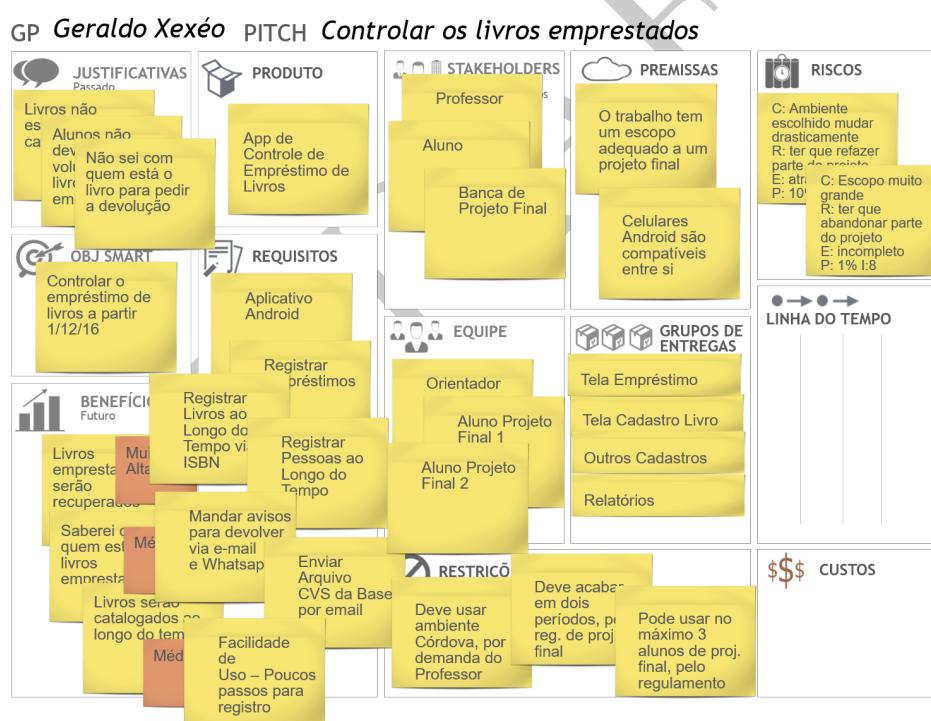


Figura 14.16.: Preenchendo os riscos do PMC.

14. Project Model Canvas

Linha de Tempo

Parte da definição de um projeto diz que ele é um **empreendimento temporário**(PMI, 2017).

Por isso é importantíssimo estimar quanto tempo vai durar um projeto. Essa estimativa global é difícil de ser feita, assim o que fazer é estimar as partes do projeto, definidas no grupo de entregas.

Lembramos que o prazo de alcançar o objetivo SMART do projeto já foi definido.

A sugestão feita pelo PMC é dividir o tempo em 4 partes, que seriam quatro grandes fases do projeto, e posicionar cada entrega passando por essas quatro grandes divisões

Com a adição das estimativas de tempo, o PMC ficará como o da Figura 14.17.

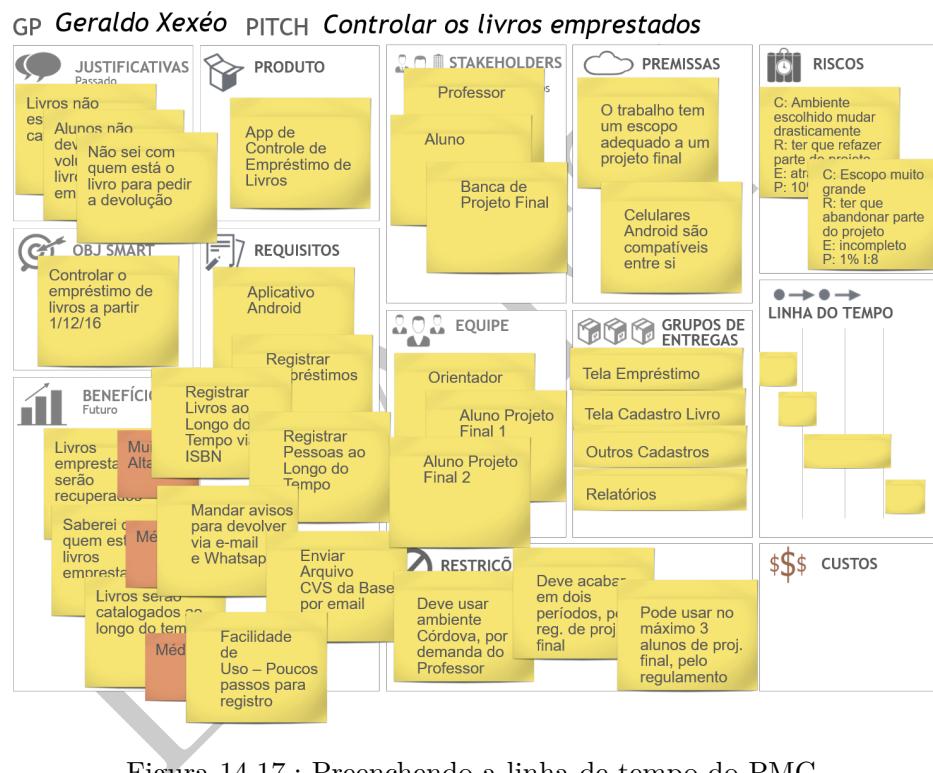


Figura 14.17.: Preenchendo a linha de tempo do PMC.

Custos

A parte mais difícil de fazer nesse momento do projeto é a estimativa de custo. Mais ainda em projeto de sala de aula, onde não há uma estimativa realistica do custo de recursos humanos.

Todo custo é composto de:

- custo de recursos humanos;

- custo de equipamentos utilizados;
 - custo de matéria prima consumida, raro em projetos de software, e
 - custo de infraestrutura, como aluguel de sala, conta de luz, etc.

Além disso, ainda existe uma divisão entre custos fixos, que não são dependentes de outros fatores, e custos variáveis, que podem se modificar ao longo do projeto.

Em projetos de software o custo dominante é a contratação de pessoal.

No PMC é sugerido que o custo seja feito por entregável(Finocchio Jr., 2013). Isso é bastante razoável, mas pode ser difícil de realizar. Essa é a abordagem tomada no PMC da Figura 14.18.

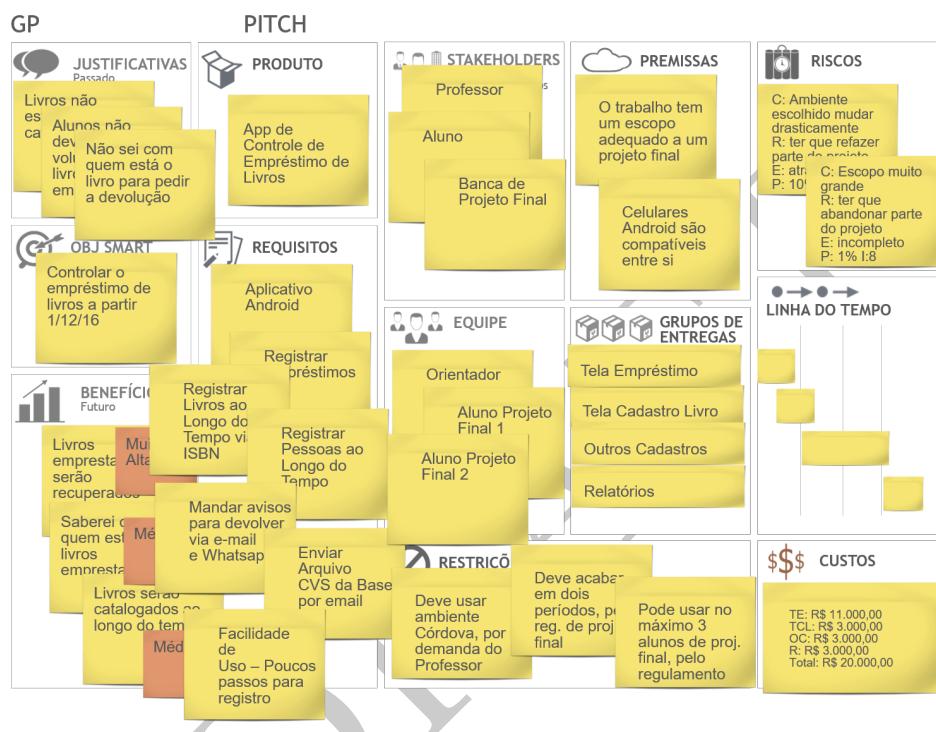


Figura 14.18.: Preenchendo os custos do PMC.

Porém, para calcular o custo, as vezes é melhor fazer outra conta. Na Figura 14.19 são apresentados o custo de compra de celulares, das bolsas dos alunos e dos computadores. Essa fórmula de cálculo permitiu gerar o custo por entrega. Podemos ver que alguns custos ficaram na primeira fase, pois computadores e telefones tem que ser comprados no início do projeto.

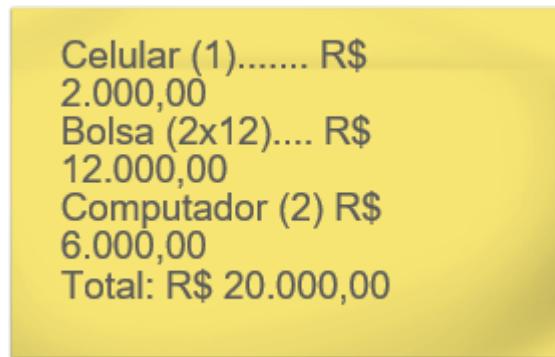


Figura 14.19.: Uma alternativa para calcular inicialmente os custos do PMC.

14.6. A Fase de Integração

Nesta fase deve ser seguido um protocolo (Finocchio Jr., 2013):

1. verificar se as justificativas serão resolvidas;
2. verificar se o objetivo é suficiente e necessários;
3. verificar se todos os requisitos possuem um dono e se, em conjunto, definem o produto;
4. verificar se a equipe está completa, com todos que precisam estar nela;
5. verificar se houve convergência nas premissas;
6. investigar se as restrições foram todas identificadas;
7. investigar se os riscos cobrem o que é sabido e o que não é sabido, e
8. verificar se o cronograma e o orçamento estão orientados por entregas.

Além disso, algumas ligações devem ser garantidas.

As premissas devem estar ligadas a compromissos os expectativas quanto as partes interessadas externas, inclusive fatores externos ligados a natureza, por exemplo. Para cada premissa deve estar associado um risco. Possivelmente restrições são necessárias para mitigar os riscos.

14.7. A Fase de Resolver

Nesta fase são resolvidos quaisquer pontos que estejam ainda em questão para a realização do plano. Muitas vezes isso exige que seja feito um “dever de casa”, e que se retorne mais tarde ao Canvas.

Problemas típicos são (Finocchio Jr., 2013):

- Projeto não gera valor
 - Não traz contribuição significativa para nenhum dos objetivos estratégicos da

- empresa
- Não adicionam os direcionadores clássicos de valor
 - ◊ Aumento de receita
 - ◊ Redução de custos
 - ◊ Melhoria nos serviços
 - ◊ Otimização do uso de ativos
 - ◊ Melhoria da imagem da organização
 - ◊ Atender demandas legais ou regulatórias
 - ◊ Melhorias sociais ou ambientais
 - Gera melhoria, mas elas são subjetivas e difíceis de quantificar
 - O cliente não sabe o que quer
 - Recursos não estão garantidos/allocados para o projeto
 - Gerente não possui autoridade ou influência para tocar o projeto
 - Equipe não consegue identificar as entregas
 - Riscos para inglês ver
 - Equipe insegura quanto aos prazos
 - Parceiros não se integram a equipe
 - Plano não considera outras pessoas/planeta
 - Existe resistência em relação ao projeto

14.8. A Fase de COnpartilhar

Um projeto bem definido, pronto para ser compartilhado, tem as seguintes características (Finocchio Jr., 2013)

- Defende uma causa
- Claramente delineado
- Pessoas identificadas
- Stakeholders externos identificados
- Níveis de influência e posicionamento dos SE identificados
- Ambiente externo identificado
- Entregas definidas
- Avaliação global de risco feita com cuidado
- Riscos específicos identificados
- Compromissos de finalização de entregas acordado
- Orçamento detalhado

14.9. Exercícios

Exercício 14.9-1: Vá para o site <http://jogodeanalisedesistemas.xexeo.net/> e visite a Livraria Resolve. A partir da sua visita desenhe o *Project Model Canvas* de um

14. Project Model Canvas

projeto do Web Site que a empresa deseja e precisa.

DRAFT

15

Modelagem de Processos de Negócio

The problem is that at a lot of big companies, process becomes a substitute for thinking. You're encouraged to behave like a little gear in a complex machine. Frankly, it allows you to keep people who aren't that smart, who aren't that creative.

(Elon Musk)

Conteúdo

15.1.	Processos de Negócio	178
15.2.	Exemplos de Processos	179
15.3.	Representação de Processos de Negócio	180
15.4.	Por que Entender os Processos de Negócio	180
15.5.	Exercícios	181

Por que processos de negócio?

Os sistemas de informação executam dentro das organizações para atender processos de negócio, logo é preciso entendê-los para melhor atender as partes interessadas.

Cada vez mais a Modelagem de Processos de Negócio vem sendo usada como uma ferramenta principal ou de auxílio do processo de desenvolvimento de software, incluindo

15. Modelagem de Processos de Negócio

data warehouses e aplicações relacionadas, visando o levantamento completo dos requisitos do sistema.

15.1. Processos de Negócio

Uma das informações importantes para o conhecimento de uma organização é entender o que ela faz e como ela faz para atingir seus objetivos de negócio.

Normalmente, isso é obtido por meio da descrição de seus processos de negócio.

Um **processo** “é uma coleção de tarefas inter relacionadas que são iniciadas em resposta a um evento, e que tem como objetivo alcançar um resultado específico para o consumidor do processo”(Scheel et al., 2015).

Processos aparecem em todos os lugares, e podem ser documentados de maneira mais ou menos formal. Um processo simples que usamos no dia a dia é o de fazer um café expresso em uma máquina de cápsulas. Os passos desse processo podem ser descritos como:

1. ligar a máquina de expresso;
2. escolher a cápsula de café;
3. colocar água na máquina;
4. colocar a cápsula na máquina
5. colocar a xícara no local indicado
6. apertar o botão de fazer café
7. remover a xícara

É possível notar que mesmo um processo simples que é feito no dia a dia apresenta dificuldades de descrição. Na lista anterior, por exemplo, não se discute o que fazer se a máquina não estiver na tomada, ou se uma cápsula já está na máquina, ou se já tem água na máquina. Isso significa que uma descrição de um processo simples de forma completa pode exigir um cuidado muito grande com detalhes, se tornando, no final, muito complexa.

Todas as organizações realizam processos. Esses processos podem ter nomes como “Atender Pedido do Cliente” ou “Fabricar Peça”.

A **modelagem de processos** demonstra como funciona a organização, passo a passo, no seu dia a dia. A partir dela pode ser possível levantar os pontos a serem automatizados de um processo e como os processos realmente realizados diferem dos processos normatizados da organização.

Um **processo de negócio** “uma coleção de tarefas e atividades (operações e ações de negócio) consistindo de empregados, materiais, máquinas, sistemas e métodos, que são estruturadas de forma a projetar, criar e entregar um produto ou serviço ao consumidor”(Scheel et al., 2015).

Processos de negócio são sempre descritos a partir de um ponto de vista que indica

a profundidade com que são tratados. Um processo de “Calcular o Preço Final”, por exemplo, tanto pode ser visto apenas como uma atividade única, como por várias etapas que passam por partes diferentes da organização, como ainda por um algoritmo detalhado.

Entre as características principais de processos de negócio é possível citar que eles sempre são sequências de passos, sempre respondem a uma necessidade ou demanda que define seu início e sempre chegam a um resultados específico e mensurável, seja ele o sucesso ou o fracasso do atendimento a necessidade ou demanda original.

A **Modelagem de Processos de Negócio** tem como objetivo entender os processos que ocorrem na organização, listá-los, descrevê-los por meio de uma representação adequado, indicando o passo a passo em um nível de abstração adequado ao propósito e, também, verificar se as representações correspondem a realidade.

Linguagens de descrição de processo de negócio, como ARIS e BPMN, se preocupam em fornecer uma abstração razoável para a descrição de processos. Normalmente essa abstração permite descrições tanto de um ponto de vista muito amplo, com a supressão da maioria dos detalhes, que visa mais entender o que é feito, como um ponto de vista razoavelmente detalhado, que permite ao leitor entender realmente como o processo é executado na organização, ou seja, como algo é feito. Cabe ao modelador do processo escolher e manter esse nível de abstração ao longo do trabalho.

15.2. Exemplos de Processos

É interessante ver alguns exemplos de processos sendo descrito na forma de uma narrativa simples que não inclui alternativas.

O processo de entrega de uma pizza pode ser descrito como:

1. O atendente recebe o pedido de pizza e bebidas
2. O pedido da pizza passa para o pizzaiolo, que prepara as pizzas e coloca na fila para assar
3. O pizzaiolo coloca as pizzas do pedido para assar
4. Quando as pizzas estão prontas, o pizzaiolo retira e passa as pizzas para seu ajudante colocar em caixas
5. Se foram pedidas bebidas, o atendente junta as bebidas ao pedido e o separa
6. O entregador pega o pedido completo e passa no caixa para pegar o troco
7. O entregador entrega a pizza
8. O entregador cobra a pizza do cliente
9. O entregador traz o dinheiro para o caixa

Já uma loja de construção pode ter o seguinte processo para atender um pedido:

1. O vendedor recebe o pedido
2. O vendedor levanta o preço de cada produto
3. O vendedor confirma com o cliente os produtos que vão ser comprados

15. Modelagem de Processos de Negócio

4. O vendedor envia a lista de produtos vendidos para o caixa
5. Enquanto o caixa faz a cobrança, O vendedor separa os produtos desejados
6. Enquanto o vendedor separa o pedido, o caixa cobra o pedido do cliente
7. Enquanto o vendedor separa o pedido, O caixa emite a nota fiscal
8. O vendedor junta a nota fiscal ao pedido
9. O vendedor entrega o pedido ao cliente

O que essas duas descrições permitem entender sobre processos de negócio, e mesmo processos em geral?

Primeiro, elas descrevem um passo a passo do que é feito. Todo processo é um passo a passo para realizar algo. Segundo, elas começam com um evento. Todo processo tem que ter um motivo para começar e deve atender uma necessidade definida por esse motivo.

Outra característica importante: os passos de um processo levam a organização a chegar mais perto do resultado desejado.

Além disso, no processo da pizzaria existe um passo opcional, enquanto no processo da loja de ferragens existem passos paralelos. Tanto passos opcionais como paralelos são comuns em processos de negócio.

Finalmente, os dois são descrições parciais do funcionamento do negócio.

15.3. Representação de Processos de Negócio

Várias são as formas de representar processos de negócio. É possível dividir essas formas em dois grupos básicos: formas textuais e formas gráficas.

Entre as formas textuais encontramos os Casos de Uso, as narrativas formais ou informais, linguagens declarativas e outras.

Entre as formas gráficas, podemos encontrar formas comuns mas hoje consideradas ultrapassadas, como fluxogramas e diagramas IDEF0 e IDEF3, formas usadas na academia mas com pouco uso na área de negócios, como Redes de Petri, e formas atuais em uso no mercado, como ARIS, Diagramas de Atividade e a principal ferramenta da atualidade: os diagramas BPMN.

15.4. Por que Entender os Processos de Negócio

Normalmente sistemas de informação estão inseridos e automatizam processos de negócio. Eles atuam garantindo que aconteçam, ajudando na sua gerência, monitoração e controle nos vários níveis da organização.

Durante os processos de negócio dados primitivos são criados e armazenados. São

os sistemas de informação que tratam esses dados, e possibilitam gerar informações adicionais, derivadas a partir deles.

Assim, entender os processos de negócio facilita a entender o ambiente dos sistemas de informação e seus requisitos.

15.5. Exercícios

Exercício 15.5-1: Descreva o processo de fazer o café expresso em uma máquina de cápsulas da forma mais detalhada possível, considerando as seguintes condições iniciais:

- a máquina de café está em cima do balcão, desligada da tomada e sem água;
- as xícaras estão no armário;
- as cápsulas de café ainda estão fechadas em sua caixa com 10 unidades, na despensa;
- as portas do armário e da despensa estão fechadas;
- é possível pegar água no filtro.

DRAFT

16

Introdução ao Aris EPC

Conteúdo

16.1.	Os softwares de desenho e as notações ARIS	184
16.2.	O Modelo Básico do EPC	185
16.3.	eEPC	192
16.4.	Passos para construir modelos EPC/EPCe	195
16.5.	Deadlocks	196
16.6.	Alguns padrões em EPC	197
16.7.	Revisão das regras básicas do EPC	200
16.8.	Exercícios	200

Por que Aris?

Aris é uma metodologia muito usada no mercado para a modelagem de processo de negócios. Normalmente ela é associada a implantação de grandes sistemas ERP, porém é uma linguagem conhecida por muitos e que o profissional de análise de sistemas vai provavelmente encontrar em sua profissão.

EPC é a sigla em inglês para **Event Driven Process Chain** (Cadeia de Processos Dirigida por Eventos). Esse método é parte do método **ARIS** usado para modelagem de processo e tem grande aceitação no mundo, estando muitas vezes associado à implantação de sistemas de ERP SAP/R3.

A Figura 16.1 mostra um exemplo de um processo imaginário de recepção e atendimento de um pedido por uma organização, contendo uma decisão. O diagrama foi desenhado

usando a ferramenta ARISExpress.

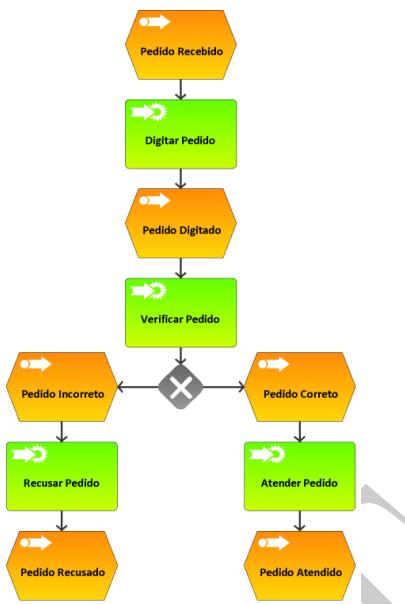


Figura 16.1.: Um processo de recepção e atendimento de pedido escrito em ARIS/EPC, desenhado no ARISExpress

Nesse método, um processo é modelado segundo um fluxo de atividades controladas por eventos. Uma leitura da Figura 16.1 dá a seguinte narrativa para o processo: quando um pedido é recebido, é necessário digitar o pedido, que é então considerado digitado; com o pedido é preciso verificar pedido, o que resulta em o pedido podendo estar correto ou não. Para pedido incorreto é feita a atividade Recusar Pedido, enquanto para um pedido correto é feita a atividade Atender Pedido. Em ambos os casos o processo acaba.

16.1. Os softwares de desenho e as notações ARIS

Apesar de muito divulgado, ARIS é pouco padronizado e existem várias notações, apresentadas em livros, manuais e softwares diferentes. A empresa que mantém o método também já criou notações distintas em função do conhecimento do leitor do diagrama e fez mudanças na notação ao longo do tempo. Atualmente ela mantém pelo menos duas notações diferentes em dois produtos que suporta: o ARISExpress, grátis, e no produto oficial.

As notações mais encontradas atualmente são três, com a notação básica apresentada na Tabela 16.1:

- Uma notação antiga presente em livros e ainda suportada pelo Microsoft Visio.
- Uma notação feita basicamente de caixas coloridas com ícones no topo esquerdo, do software ARISExpress.

- Um notação feita de retângulos com uma marcação colorida no seu lado esquerdo, do software Aris¹.

Apesar deste texto apresentar várias notações, é importante notar que elas não podem ser misturadas. A Tabela 16.1 dá um exemplo de três variações existentes para EPC.

Tabela 16.1.: Objetos básicos do EPC, em várias versões do Aris.

Significado	Original	Aris Express	Aris Business Architect
Atividade	Entregar Pedido	Digitar Pedido	Preparar Ordem de Compra
Evento	Pedido Entregue	Pedido Recebido	Produto Entregue
Regra	XOR AND OR	+ X O	
Interface de Processo		Process interface	

16.2. O Modelo Básico do EPC

O EPC mostra um **fluxo de execução**. Esse fluxo de execução é indicado por **setas direcionadas** em um grafo compostos por nós de vários tipos, como mostra a Figura 16.1.

O fluxo de execução, ou **fluxo de controle**, indica a sequência dos passos descritos no diagrama. Por ele **não trafegam materiais ou informação**. Essa é uma diferença importante entre o EPC e outros modelos mais antigos de fluxograma que misturam o fluxo de dados com o fluxo de execução.

É possível dizer que o diagrama EPC é um grafo direcionado, com vários tipos de nós. Os principais nós desse diagrama são:

- Atividades, também chamadas de funções ou tarefas.
- Eventos
- Conectores
- Interface de Processo

Como diz o nome, o EPC é um cadeia de processo, ou seja, uma sequência de atividades que forma um processo, controlada por eventos. Isso significa que os processos acontecem, ou não, de acordo com os eventos.

¹Esse software pode ser obtido sem custos, por meio de uma licença acadêmica, para alunos e professores

16. Introdução ao Aris EPC

É importante notar que Atividades, Eventos e Interfaces de Processo só podem possuir **apenas um** fluxo de entrada, ou seja, apenas uma seta entrando nelas, e **apenas um** fluxo de saída, ou seja, apenas uma seta saindo delas.

Apenas os conectores podem ser usados para tratar as principais formas de variar os fluxos de execução: a concorrência ou paralelismo de caminhos (noção de E) e a seleção de um caminho entre muitos (noção de XOR).

16.2.1. Atividades

Atividades, também são chamadas de funções ou tarefas. Elas representam passos do processo que precisam ser executadas e suportam um ou mais objetivos de negócio. Não existe processo que não tenha pelo menos uma atividade.

Na prática, atividades podem representar: atividades tangíveis, decisões (mentais), processamento de Informações (automatizadas ou não), ou um subprocesso que é descrito por outro diagrama EPC².

Como atividades registram ações realizadas dentro de um processo, elas possivelmente são iniciadas ou iniciadas ou habilitadas por eventos, geram eventos, consomem recursos e exigem gerenciamento, tempo, e atenção.

O nome de uma atividade é sempre da forma

<verbo no infinitivo><objeto>

como em “Remover detritos” ou “Registrar Pedido”.

Sua representação tradicional é apresentada na Figura 16.2.



Figura 16.2.: Formas de representação de uma atividade em diferentes versões do Aris.

Uma atividade não é completamente definida pelo seu nome. Outros elementos que aparecem no diagrama, se serão apresentados ainda nesse capítulo, detalham sua descrição, como:

- Informação que entra
- Objetivos
- Recursos Consumíveis, como matéria prima

²Nesse caso, as ferramentas CASE fornecem alguma marca indicativa de que a função pode ser expandida.

- Recursos não consumíveis, como máquinas
- Procedimentos realizados
- Capacidades necessárias
- Informação que sai
- Produto ou serviço entregue

Como atividades representam ações, elas sempre envolvem algum gasto de tempo, por mais rápido que seja, e algum recurso, por menor que seja.

Na Figura 16.1 são encontradas quatro atividades: Digitar Pedido, Verificar Pedido, Recusar Pedido e Atender Pedido.

16.2.2. Eventos

Eventos, representam situações, ou estados do sistema, antes ou depois da execução de uma função. Não existem processos sem pelo menos dois eventos: um de início do processo e um do fim do processo.

Os Eventos representam a mudança do estado do mundo enquanto o processo é executado. Três são os tipos de eventos candidatos a serem registrados em um processo :

1. A mudança de estado do mundo que faz com que o processo comece;
2. Mudanças internas de estado enquanto o processo é executado, chamados de eventos intermediários, e
3. O resultado final do processo que tem um efeito externo.

Pelas regras do EPC, **todo diagrama deve começar em um evento**, que indica porque ele começou, e **acabar em um ou mais eventos**, que normalmente indicam o sucesso ou o fracasso do processo em atender ao evento que iniciou o processo. Um diagrama não pode ter outro tipo de nó como final de um fluxo de execução.

Uma definição para **eventos** no EPC é é como “estados comercialmente relevantes, que controlam ou influenciam a progressão subsequente de um ou mais processos de negócio ”(Software AG, 2019a). Não se pode deixar de criticar o uso da palavra evento nesse contexto, quando sua definição mais se assemelha a um estado do sistema.

Em relação as atividades conectados com eles, cada evento pode representar os gatilhos ou os resultados das mesmas(Software AG, 2019a). Porém, em um modelo de processo, a ocorrência de um evento não indica que a atividade a seguir vai ocorrer imediatamente, ela é apenas permitida. Claramente, eventos definem pré-condições para que uma atividade ocorra, e também pós-condições ao fim da atividade.

É importante notar que eventos funcionam como marcos, relacionados com um instante no tempo(Software AG, 2019a), logo eles não gastam tempo ou recursos.

Na Figura 16.1 são encontrados seis eventos: Pedido Recebido, Pedido Digitado, Pedido Incorreto, Pedido Correto, Pedido Recusado e Pedido Atendido.

Assim, na Figura 16.1, o processo começa quando um pedido é recebido e pode acabar

16. Introdução ao Aris EPC



Figura 16.3.: Formas de representação de um evento em diferentes versões do Aris.

de dois modos: com o pedido recusado ou com o pedido atendido. Analisando mais o diagrama é possível perceber que o pedido é recusado porque ele estava incorreto após a atividade “Verificar Pedido”, e é atendido se estiver correto após essa atividade.

Um evento segue a sintaxe: <Sujeito><Verbo na voz passiva>. Ou, mais simplesmente: <Substantivo><Adjetivo>. Essa sintaxe é usada nos exemplos da Figura 16.1.

Os eventos intermediários, isto é, os que aparecem dentro do processo, devem ser escolhido por dois princípios básicos: de forma a cumprir seu papel de explicar uma tomada de decisão, isto é, sempre aparecem depois de todas as regras OU-Exclusivo, indicando o caminho a ser tomado; e também para indicar mudanças significativas de estado. No passado, como cada atividade implica na mudança do estado do processo, era obrigatório colocar um evento entre cada par de atividades. Com o tempo, porém, se percebeu que não havia utilidade em colocar eventos muito simples e óbvios e essa prática deixou de ser exigida. Ainda é importante, porém, usar os eventos para registrar mudanças importantes no estado do processo, ainda marcos de controle para a gerência, ou ainda outro motivo que torne aquele estado interessante. A Figura 16.4 mostra como era obrigado a desenhar um processo no passado e uma alternativa válida no presente.



Figura 16.4.: Exemplos de eventos Segundo a versão original de EPCs, sempre deveria haver um evento entre dois processos. Atualmente é permitido que uma sequência de processos não tenha nenhum evento entre eles. Logo, os diagramas das figuras a seguir podem ser usados como equivalentes.

16.2.3. Regras

Regras, “representam operadores que são usados para especificar as conexões lógicas entre eventos e atividades”(Software AG, 2019b). Eles permitem a unificação e separação

de fluxos segundo os conceitos de E, OU ou OU-exclusivos.

A linguagem oferece três tipos de regras básicas: o que permite que o fluxo de execução siga vários caminhos simultaneamente, conhecido como E, o que permite que um caminho seja escolhido entre vários, conhecido como OU-EXCLUSIVO, e o conector OU, que permite escolher um ou mais caminhos, mas que por sua ambiguidade não é considerado aconselhável e não será usado neste texto.

Esses conectores E, OU e OU-EXCLUSIVO funcionam de duas formas: como divisores de caminho (split) e como junção de caminhos (join). Não é possível utilizar um conector simultaneamente nas duas formas, o que é indicado na Figura 16.5.

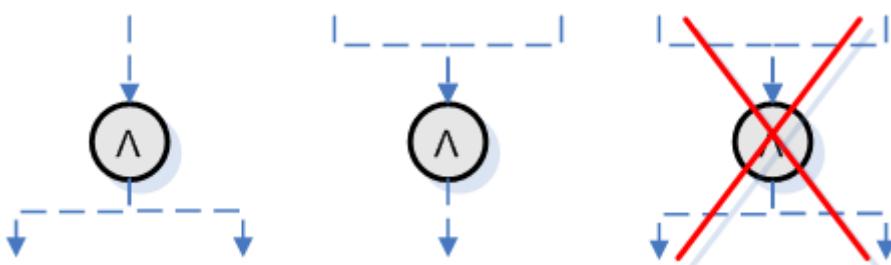


Figura 16.5.: Um conector só deve ser utilizado na forma de divisor ou junção, e nunca das duas formas ao mesmo tempo.

De acordo com as regras sintáticas para EPCs, é possível que, em um processo, uma atividade produza um ou mais eventos, sejam eles simultâneos ou uma escolha. Porém o diagrama não pode ligar diretamente atividades a vários eventos, ou eventos a várias atividades. Então são usadas as regras para indicar de que forma isso ocorre.

A Figura 16.1 apresenta um conector que indica uma decisão (OU-Exclusivo ou XOR). Essa decisão acontece, na verdade, na atividade imediatamente anterior ao símbolo da regra. Nessa decisão, um dos caminhos apenas será seguido, de acordo com o estado atual do processo, isto é, se a situação indica um “Pedido Correto” ou um “Pedido Incorreto”.

16.2.4. Interface de Processo

Interface de Processo, são símbolos de conexão entre diagramas de processo. Elas servem para indicar que um processo, após terminar, tem alguma continuação em outro processo, ou, vice-versa, que um processo começa como a continuação de outro processo.

As **interfaces de processos** são usadas para indicar que um processo acontece quando outro processo acaba. Caminhos são sempre conectados a eventos, devendo aparecer antes ou depois deles, e recebem o nome do processo origem ou destino. Na Figura 16.7 a seguir é possível ver um exemplo de sua utilização.

Elas só devem ser usadas para ligar processos de um mesmo nível.

16. Introdução ao Aris EPC

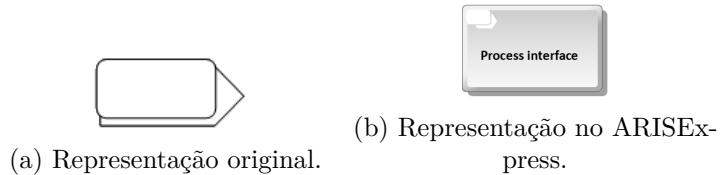


Figura 16.6.: Formas de representação de uma interface de processo em diferentes versões do Aris.

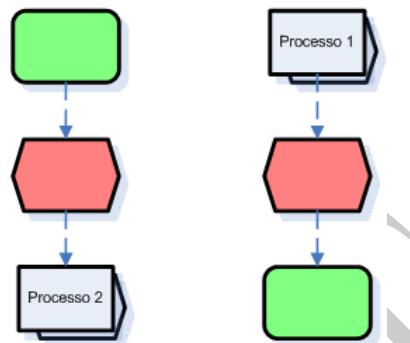


Figura 16.7.: Indicando que um processo é continuação lógica de outro processo. No trecho final do Processo 1 é indicado que ele é seguido pelo Processo 2, e nesse processo é indicado que o caminho é proveniente do Processo 1.

16.2.5. Exemplos Adicionais de EPC

Seguindo essas regras básicas, o processo de entrega de pizza descrito na Seção 15.2 pode ser representado como na Figura 16.8.

Já o processo de atender uma ordem de serviço apresentado na Fig 16.9 apresenta uma decisão e um paralelismo.

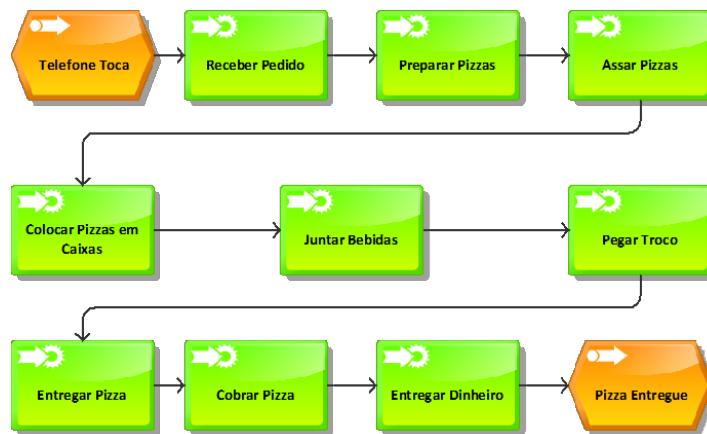


Figura 16.8.: Processo simplificado de entrega de pizzas. Fonte: do autor

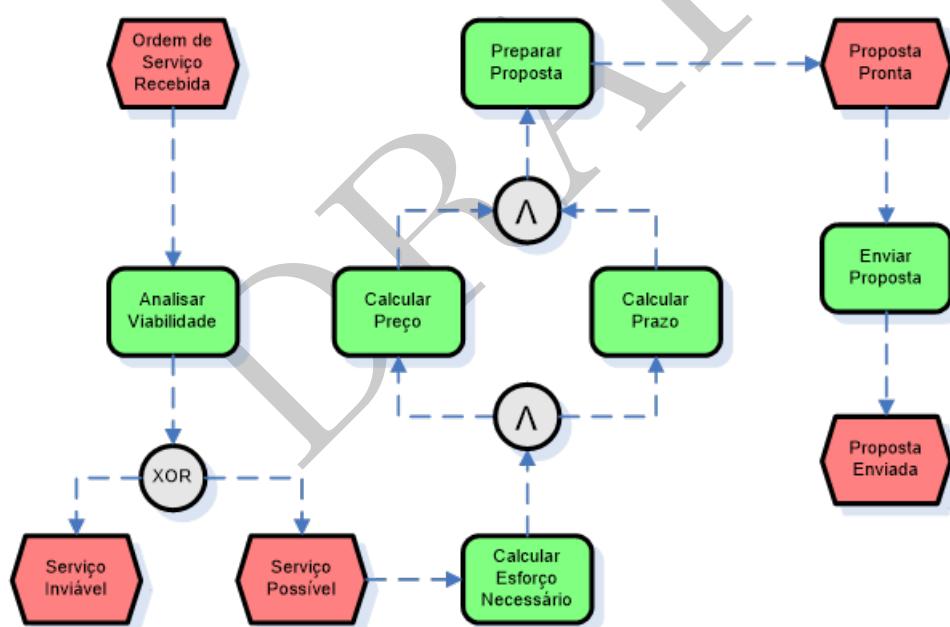


Figura 16.9.: Processo simplificado de atendimento de ordem de serviço. Fonte: do autor

16.3. eEPC

eEPC é a sigla em inglês para Extended Event Driven Process Chain (Cadeia de Processos Dirigida por Eventos). Nessa extensão é possível declarar mais algumas informações sobre o processo sendo descrito.

Esses elementos adicionais funcionam basicamente como comentários ao processo que está sendo documentado. Assim, depois de descrito o processo pelo método não estendido, colocamos sobre eles novos elementos documentando informações como quem realiza o processo, que informação utiliza, que produtos gera ou consome, etc...

Os principais elementos adicionais em um eEPC, segundo a documentação original, são:

- Unidades Organizacionais, que representam departamentos envolvidos em um processo.
- Papéis e Pessoas, que representam pessoas ou papéis envolvidos em um processo.
- Informação ou dados, que representam informação utilizada ou gerada em um processo.
- Produtos ou serviços, que são gerados ou consumidos pelo processo.
- Objetivos, que representam o motivo da realização de um processo ou tarefa

Versões mais modernas inserem vários outros tipos de informações. No ARISExpress, por exemplo, estão disponíveis dez tipos de objetos que podem ser usados, como mostrado na Figura 16.11.

A Figura 16.12, apresenta um exemplo do mesmo processo da Figura 16.1 para demonstrar os usos dos elementos adicionais.

16.3.1. 5W2H no EPC

Um evento indica quando (*when*) algum processo, função ou tarefa deve ser iniciado. Uma atividade indica o quê (*what*) deve ser feito.

Uma unidade organizacional, um papel ou uma pessoa indicam quem (*who*) deve fazer.

Tipo	Símbolo
Unidade Organizacional	
Informação	
Pessoa ou Cargo	
Fluxo de Informação	
Relações Organizacionais	
Produto ou Serviço	
Objetivo	

Figura 16.10.: Elementos complementares dos diagramas eEPC.



Figura 16.11.: Nova lista e imagens dos elementos complementares dos diagramas EPC no ARISExpress.

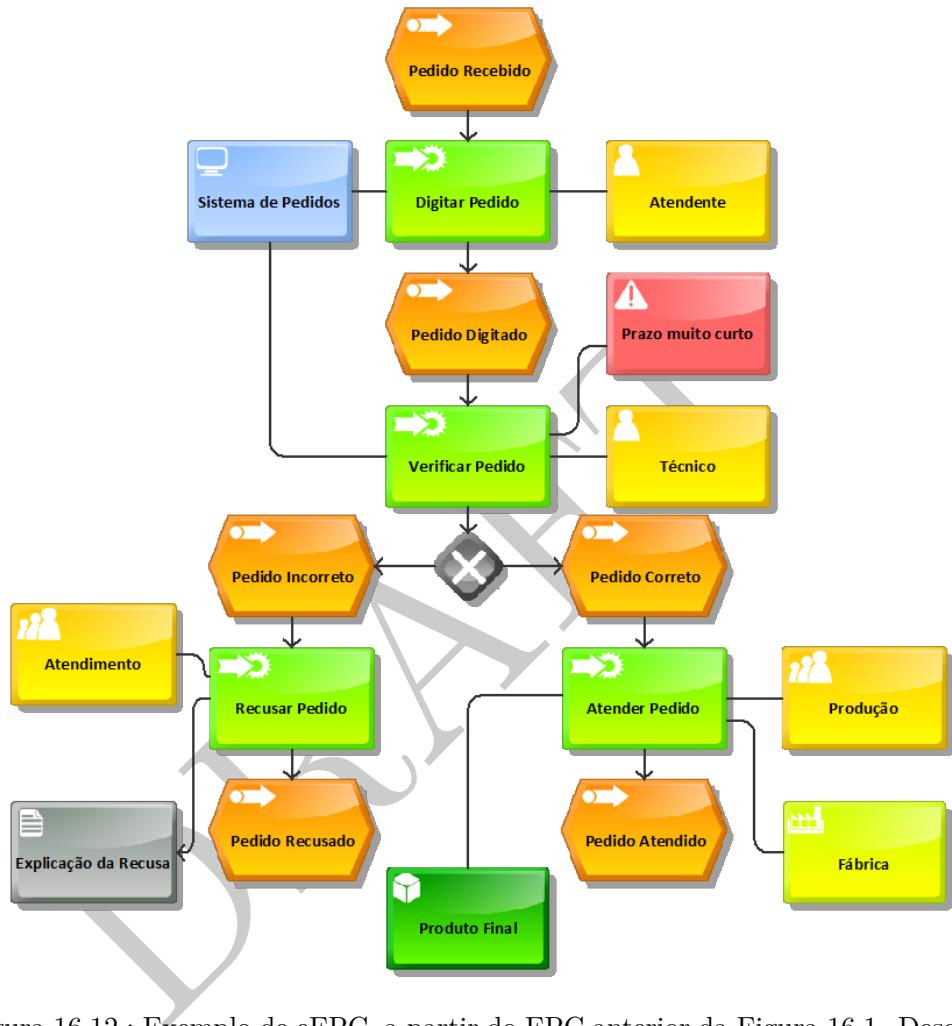


Figura 16.12.: Exemplo de eEPC, a partir do EPC anterior da Figura 16.1. Desenho feito no ARISExpress. Os símbolos suplementares podem variar de acordo com a ferramenta case sendo usada.

16.4. Passos para construir modelos EPC/EPCe

Woods-Wilson (2003) sugeriu os seguintes passos para construir modelos EPC.

1. Identifique os eventos que iniciam as funções, que servem como gatilhos para o processo se iniciar. Normalmente vem de fora para dentro do processo.
2. Identifique as funções do processo, associando-as aos eventos que as iniciam e sua sequência.
3. Decomponha as funções, verificando se são ações lógicas simples ou compostas, executadas por uma ou mais pessoas (ou ainda um sistema de computador). Verifique também se a função é uma transação isolada ou pode ser dividida em partes, se pode ser interrompida em um momento específico e se existe um evento que a interrompa ou que a faça funcionar novamente.
4. Analise os eventos novamente, definindo-os e refinando-os se necessário. Garanta que são necessários e suficientes para iniciar a função. Analise se existem casos especiais nos quais as funções acontecem ou não. Use operadores lógicos para montar as relações entre os eventos.
5. Identifique os eventos de finalização e as saídas (tanto de material quanto de informação). Procure identificar quem processos e pessoas no resto da organização que dependem do processo sendo analisado. EPCs podem ser muito pequenos ou enormes, dependendo unicamente do tamanho do processo que está sendo mapeado.

16.4.1. EPC e Loops/Laços

Por diferentes motivos, laços arbitrários não funcionam bem com EPCs. Alguns textos explicitamente proíbem laços, outros não apresentam exemplos, desconsiderando-os implicitamente. Porém, é possível descrever laços em EPML, a linguagem XML para descrever EPCs.

A questão de modelar laços em PEC é relacionada ao fato que processos de negócio não deveriam ter laços, já que eles devem levar o negócio em direção a solução da necessidade definida na sua origem.

Na prática, isto depende do ponto de vista do modelador e do detalhe que quer dar ao modelo.

Assim, devem ser seguidas seguintes recomendações quanto ao uso de laços nos diagramas EPC:

- Analise a necessidade de laços em função do ponto de vista da modelagem
- Evite os laços se o ponto de vista for mais abstrato
- Use apenas os laços estritamente necessários, não se preocupe com correções de erros simples
- Use apenas laços simples, baseados em XOR
- Verifique sempre se o laço representa a realidade do processo e se ele é significativo.

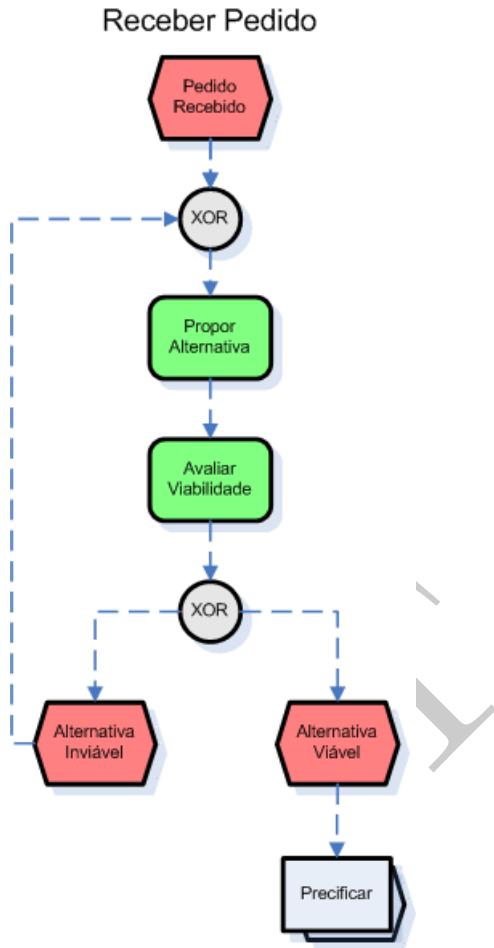


Figura 16.13.: Um EPC válido, com um loop simples baseado em XOR. Também é dado um exemplo de como conectar esse EPC a um outro processo.

16.5. Deadlocks

O uso descuidado de regras do tipo OU-Exclusivo e E sem estruturá-las corretamente gera o risco de deadlocks, isto é, de um estado em que o processo não pode continuar porque um caminho está esperando algo que não pode acontecer.

A Figura 16.14 demonstra um exemplo onde é impossível terminar o processo.

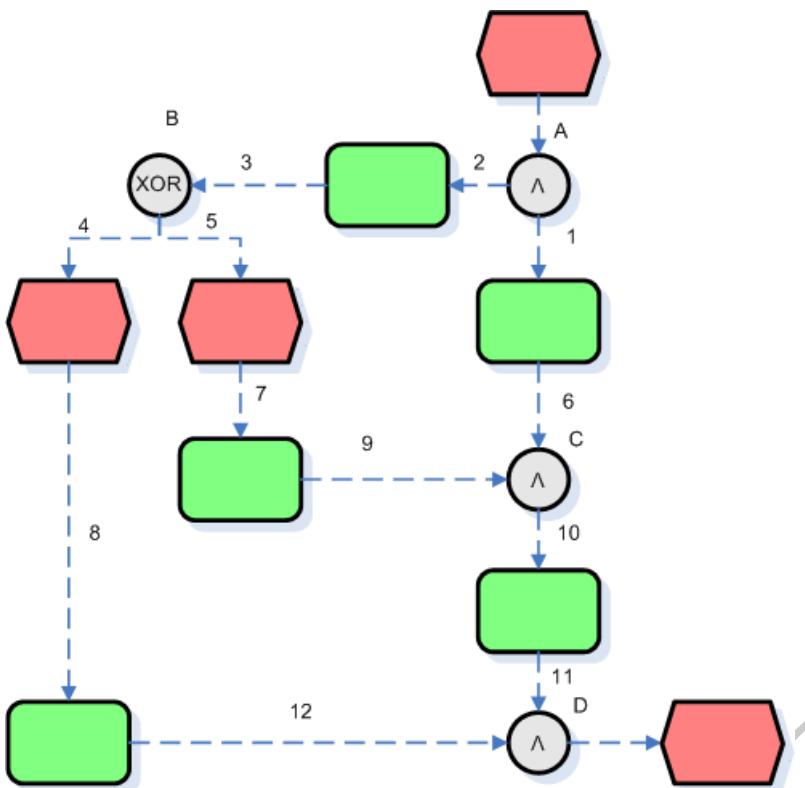


Figura 16.14.: O modelo acima, demonstra um deadlock. Como comentário, os números identificam as setas e as letras identificam os conectores . No conector XOR (B), um dos caminhos 4 ou 5 será escolhido. Porém, se for escolhido o caminho 5, o sistema para de funcionar no conector E (D), pois esse E nunca será satisfeito (já que 4-8-12 não será percorrido). Caso seja escolhido o caminho 4, o conector E (C) nunca será satisfeito (pois 5-7-9 não será percorrido), e o sistema vai parar em (C), esperando 9, e (D), esperando 11.

16.6. Alguns padrões em EPC

Alguns padrões ocorrem em muitos processos.

Um padrão comum é o de fazer algo que é avaliado em outro passo do processo. A Figura 16.13 mostra um exemplo onde uma alternativa é proposta e depois é avaliada, e se não for aceita outra alternativa deve ser proposta. Esse é um dos casos onde o *loop* pode ser mostrado corretamente.

O segundo padrão é escolher um caminho entre duas alternativas. A Figura 16.15 mostra como esse padrão deve ser desenhado. Esse é um padrão extremamente comum, e o número de opções **não** está limitado a dois. O importante é que as condições que permitem escolher o caminho aparecem nos eventos, logo após o símbolo de Ou-exclusivo, que vem logo após a atividade que toma a decisão.

Caminhos paralelos também são um padrão comum, que são descritos como na Figura

16. Introdução ao Aris EPC

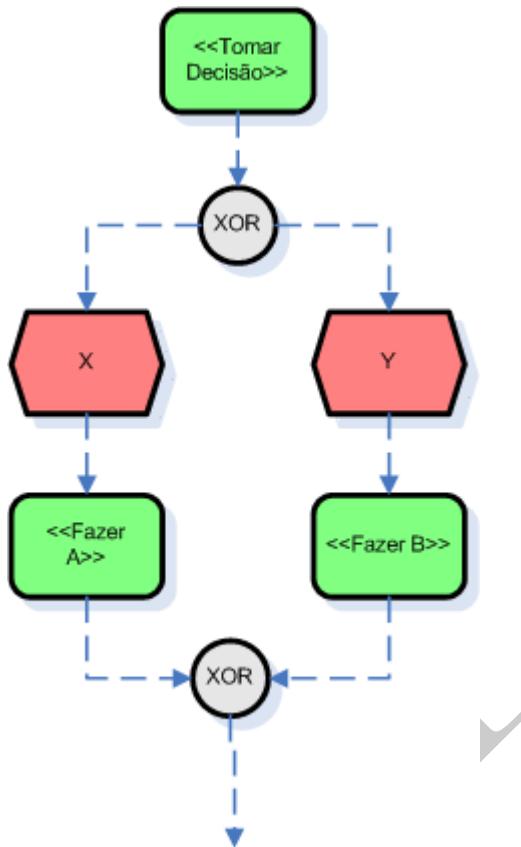


Figura 16.15.: Descrição EPC para Se X então Fazer A, Se Y, então Fazer B

16.16.

Outro padrão é fazer algo se alguma condição não foi alcançada, caso contrário seguir em frente. Isso acontece, por exemplo, se antes de atender um cliente é necessário cadastrá-lo. A Figura 16.17 dá um exemplo desse caso. Esse padrão também é conhecido como “ramo que não faz nada (*do nothing branch*)”.

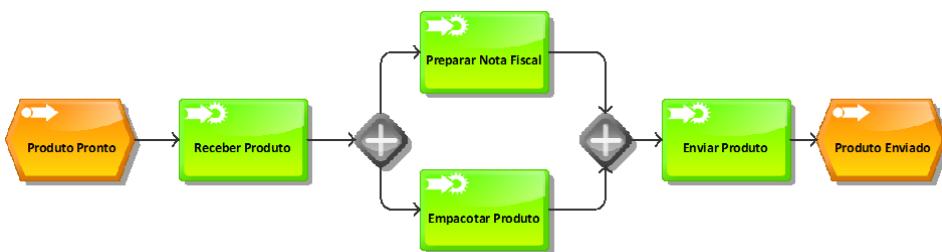


Figura 16.16.: Caminhos paralelos, na notação do ARISExpress

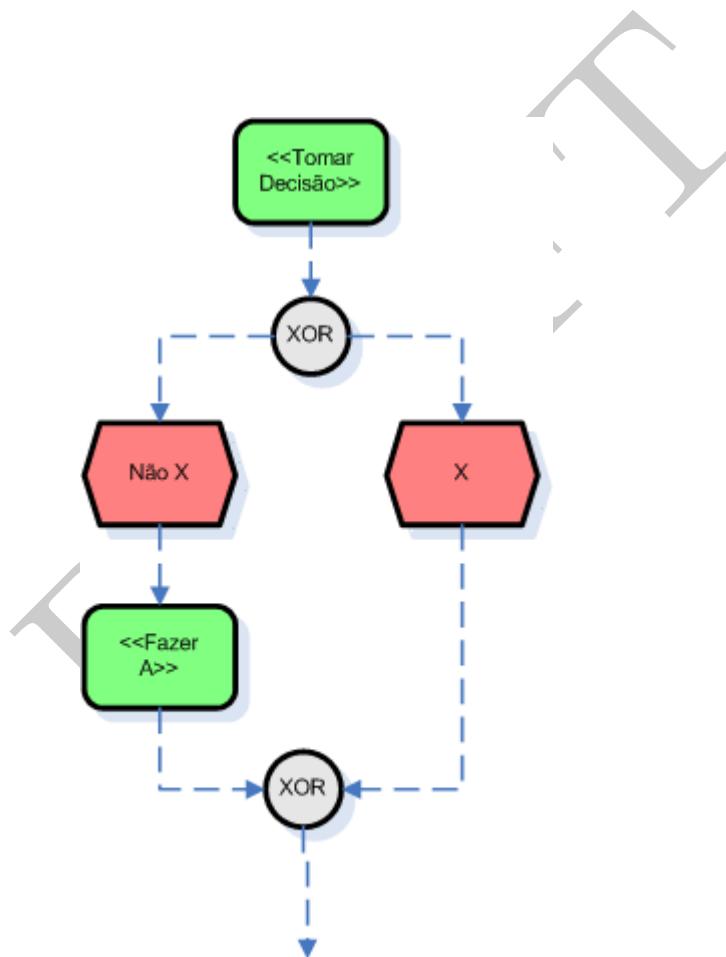


Figura 16.17.: Descrição EPC para Se Não X então Fazer A

16.7. Revisão das regras básicas do EPC

- Não existem nós isolados
- Atividades e eventos têm apenas uma entrada e uma saída
- Operadores lógicos contêm vários fluxos de entrada e um de saída, ou um único fluxo de entrada e vários de saída. Não podem conter vários de ambos.
- Loops devem ser evitados.
- Dois nós só podem possuir um único link entre eles
- Existe um evento inicial e um evento final
- Eventos não tomam decisões, logo só possuem uma saída.
- Evite o uso de OU, pois sua semântica não é clara.
- Cuidado com modelos complexos, que podem levar a deadlocks.
- Mantenha o diagrama estruturado. Sempre que dividir um caminho com XOR ou E, feche o caminho com o mesmo símbolo.
- Não misture notações.

16.8. Exercícios

Exercício 16.8-1: Vá para o site <http://jogodeanalisedesistemas.xexeo.net/> e visite a Livraria Resolve. A partir da sua visita desenhe todos os processos que você identificar, usando para isso o método ARIS-EPCe.

17

BPMN

Conteúdo

17.1.	Usos de BPMN	202
17.2.	Participantes	203
17.3.	Símbolos básicos	203
17.4.	Atividades	203
17.5.	Eventos	206
17.6.	<i>Gateways</i>	209
17.7.	Fluxos	211
17.8.	Alguns Exemplos	212
17.9.	Eventos Anexados a Atividades	212
17.10.	Exercícios	212

Por que BPMN?

BPMN é o padrão *de facto* da modelagem de processos, ferramenta essencial para qualquer projeto de sistema de informação para uma organização

BPMN, sigla para *Business Process Model and Notation*, é uma linguagem gráfica para a modelagem de processos de negócio padronizada pela OMG e que está se tornando o padrão *de facto* da indústria.

A Figura 17.1 mostra um exemplo razoavelmente simples de um processo descrito em BPMN. Nesse diagrama podemos ver um processo que se inicia na tarefa “Receber

17. BPMN

Pedido de Crédito” e continua na tarefa “Aprovar Pedido de Crédito”. Nesse momento encontramos um *gateway*, que nesse caso é uma decisão onde apenas um caminhos pode ser seguido (o losango sem símbolo), e caso o pedido não tenha sido aprovado, seguimos para a tarefa “Avisar Resultado”, ou, caso o pedido tenha sido aprovado, seguimos para a tarefa “Efetuar Transação de Crédito” e então também para a tarefa “Avisar Resultado”, que sempre leva ao fim do processo.

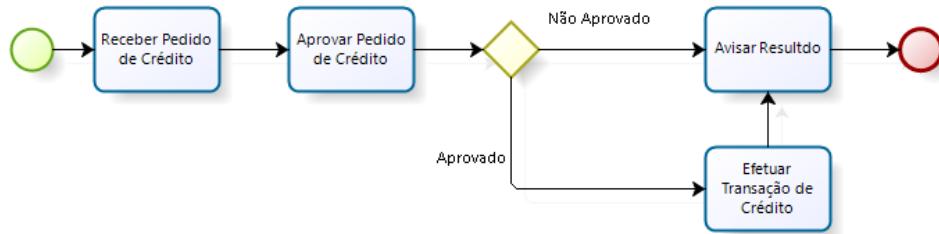


Figura 17.1.: Um exemplo de processo descrito em BPMN. Fonte: imagem do autor

17.1. Usos de BPMN

BPMN foi criado para unificar a representação de processos de negócio, tendo em vista a grande quantidade de linguagens existentes. Além disso, ele mira permitir não só a descrição de processos, mas também sua simulação e execução.

BPMN pode ser usado para criar **diagramas de processo**, **diagramas de colaboração** e **diagramas de coreografia** (OMG, 2013)(OMG, 2013).

Processos, ou orquestrações, podem ser privados ou públicos. Processos privados ainda podem ser executáveis ou não. Processos internos também são chamados de *workflows*, ou *orquestrações* no caso de serem suportados por *web services*(OMG, 2013), e descrevem processos dentro de organizações.

Um processo executável é modelado para ser executado, enquanto processos não executáveis são criados para documentação do comportamento de um processo de negócio da organização (OMG, 2013).

Um processo é **público** quando mostra a interação entre um processo privado e outro processo, ou participante, e sua representação só inclui as atividades usadas para se comunicar com outros participantes (OMG, 2013).

Uma **colaboração** “mostra as interações entre duas ou mais entidades de negócio”(OMG, 2013).

Uma **coreografia** é a “definição de um comportamento esperado, basicamente um contrato procedural, entre participantes interagindo”(OMG, 2013).

Este terá foco maior na descrição de processos privados e não tratará coreografias. Ele também não tem preocupação com a capacidade de executar os processos que apresenta.

17.2. Participantes

Participantes representam organizações ou papéis que participam em uma colaboração. São normalmente responsáveis pela execução de um processo dentro de uma pool (OMG, 2013).

17.3. Símbolos básicos

BPMN representa processos como uma sequência de tarefas, por meio de um fluxo de controle herdeiro do conceito de fluxograma, controlado por *gateways* e eventos.

Para isso são usados quatro 4 símbolos básicos:

1. **atividades**, representadas por caixas de bordas arredondadas;
2. **eventos**, representados por círculos;
3. **gateways**, representados por losangos, e
4. **fluxos**, representados por setas.

Além disso possui alguns símbolos adicionais para representar objetos de dados, armazenamento de dados, e mensagens, anotações e grupos. Também é possível criar novos símbolos representando artefatos, contanto que não sejam similares aos já existentes. Finalmente, processos podem ser organizados em *pools* e raias.

17.4. Atividades

As **atividades** são o trabalho que as organizações fazem. Se incluídas em um processo, passam a ser chamadas de **tarefas** ou **subprocessos**. Um processo é composto por um fluxo de atividades que podem ser genéricas ou possuir significado específico. Elas são representadas por caixas de cantos arredondados e possuem variações descritas pela espessura da caixa, por símbolos no canto superior esquerdo e por símbolos na sua parte de baixo.

Atividades podem ser na verdade chamadas a outros processos ou tarefas (*call activities*) e representam um ponto onde processos ou tarefas globais, como sub-processos, são usados dentro de um processo. Elas são identificados por ter a borda mais grossa.

Atividades são **sempre** nomeadas no formato:

<verbo no infinitivo><objeto>

como em “coletar resíduos” ou “aprovar pedido”.

Tarefas podem ser mais detalhadamente especificadas, por uma marca no canto superior esquerdo, como (OMG, 2013):

- tarefa de envio, onde é enviada uma mensagem a um participante externo;

17. BPMN

- tarefa de recepção, que espera a chegada de uma mensagem de um participante externo, muitas vezes usada para iniciar um processo;
- tarefa de *script*, que é executado por um *engine* de processo de negócio;
- tarefa de regra de negócio, que indica o uso de um *engine* que executa regras de negócio;
- tarefa de serviço, que usam um tipo de serviço, como um *web service* ou uma aplicação automatizadas;
- tarefa manual, feitas de forma totalmente manual, ou
- tarefa de usuário, feitas por uma pessoa com auxílio de uma aplicação.

Além disso, marcadores na base de tarefas e processos podem indicar (OMG, 2013):

- um subprocesso colapsado, por meio de uma caixa com um símbolo de soma;
- um *loop*, por meio de uma seta circular;
- múltiplas instâncias em sequência, por meio de 3 barras horizontais;
- múltiplas instâncias em paralelo, por meio de 3 barras verticais;
- uma compensação, por meio de dois triângulo ligados voltados para a esquerda, e
- *ad-hoc*, por meio de um til, apenas para subprocessos;

Também existem símbolos para subprocessos.

Tabela 17.1.: Símbolos para os eventos que podem aparecer no início de um processo.

Significado	Símbolo	Explicação
Simples		atividade simples
Usuário		atividade feita por um usuário
Manual		atividade feita manualmente
Recebimento		atividade que recebe
Envia		atividade que envia
Script		atividade feita por um script
Serviço		atividade feita por um serviço
Regra de Negócio		atividade que segue regra de negócio

17.5. Eventos

Eventos aparecem em três momentos:

1. **iniciais**, que indicam que um processo vai iniciar, e tem a borda simples;
2. **intermediários**, que afetam o fluxo de processo, e tem uma borda dupla, e
3. **fim**, que determinam onde um processo termina, que tem uma borda grossa.

Os eventos podem estar no fluxo do processo ou ligados diretamente a uma atividade. Nesse caso ele podem interromper ou não a atividade.

O círculo do evento possui duas possíveis bordas e um marcador interno, que pode ser preenchido ou não preenchido. As bordas dão indicação da localização do evento no processo. No caso de um evento ligado a uma tarefa, as bordas normais indicam que o evento interrompe a atividade, e bordas tracejadas indicam que o evento não interrompe.

Eventos podem ser sem tipo, ou especificados detalhadamente em subtipos. Existem vários subespecificações de eventos, mas nem todas podem ser aplicadas a todos os momentos em que o evento pode aparecer (OMG, 2013):

- **cancelamento**, reagindo a cancelamentos ou ativando cancelamentos;
- **compensação**, ativando ou tratando compensações;
- **condição**, reagindo a mudanças em condições do negócio ou relacionadas com as regras de negócio;
- **erro**, capturando ou lançando erros com nomes, semelhantes a exceções em linguagens de programação;
- **escalamento**, indicando a passagem do processo para um nível maior de responsabilidade;
- **ligação**, simplesmente ligando dois pontos como conectores, por exemplo em páginas diferentes;
- **mensagem**, indicando o recebimento ou envio;
- **múltiplo**, capturando um de vários eventos;
- **múltiplo e paralelo**, capturando vários eventos;
- **sinal**, enviados e recebidos através de vários processos e possivelmente tratados múltiplas vezes;
- **temporização**, indicando eventos cíclicos, momentos específicos, prazos e períodos ou faixas de tempo, e
- **terminação**, ativando a terminação final e imediata do processo.

É importante diferenciar um sinal de uma mensagem. Mensagens sempre tem claramente definidas quem as enviou e quem as recebe. Um sinal pode ser recebido por mais de um processo.

O subtipo é especificado por um desenho no interior do círculo que representa o evento. Se o desenho for escuro ou preenchido, significa que o evento está lançando seu sinal, seu erro, ou outra coisa. Se o símbolo for vazado, desenhado apenas com linhas, significa que

está capturando.

17.5.1. Eventos Iniciais

Os eventos iniciais possíveis são apenas o simples e mais seis tipos, como listado na tabela 17.2. Esses eventos nunca emitem nada, só capturam mensagens, sinais e múltiplos eventos de início, ou seja, respondem a gatilhos (OMG, 2013).

Tabela 17.2.: Símbolos para os eventos que podem aparecer no início de um processo.

Significado	Início	Explicação
Simples		evento simples de início
Mensagem		processo se inicia quando recebe uma mensagem
Temporizador		processo se inicia em um certo momento, ou uma data específica, ou outra questão relacionada a tempo, como um prazo
Condicional		processo se inicia quando uma condição ou regra de negócio se torna verdade
Sinal		processo se inicia com um sinal recebido
Múltiplo		processo se inicia quando um de vários eventos acontecem, bastando um acontecer
Múltiplo Paralelo		processo se inicia quando vários eventos acontecem

17.5.2. Eventos Finais

Os eventos finais podem ser simples ou de outros 8 subtipos. Eventos finais nunca recebem ou capturam, sempre enviam algo, ou seja, eles criam resultados (OMG, 2013). A Tabela 17.3 mostra os tipos de eventos possíveis no final de um fluxo de processo. É importante notar que os eventos normalmente só terminam o seu fluxo de processo, a não ser o evento de terminação que termina todos os fluxos.

Tabela 17.3.: Símbolos para os eventos que podem aparecer no fim de um fluxo de processo.

Significado	Início	Explicação
Simples		evento simples de fim
Mensagem		processo termina enviando uma mensagem
Sinal		processo termina enviando um sinal
Múltiplo		processo termina enviando vários eventos
Compensação		processo termina e lança compensação
Escalamento		processo termina e escala para nível superior
Erro		processo termina e avisa de erro
Cancelamento		processo termina e cancela
Terminação		processo termina completamente

17.5.3. Eventos Intermediários

Os eventos intermediários são o simples e mais 13, sendo que 7 recebem mensagens ou esperam algo acontecer e 6 enviam alguma comunicação para outros eventos. Em geral eles podem responder a gatilhos ou gerar resultados (OMG, 2013). A Tabela 17.4 mostra todos os eventos intermediários possíveis.

Tabela 17.4.: Eventos que podem aparecer no meio de um processo.

Significado	Símbolo de Recepção	Símbolo de Envio	Explicação
Simples			evento simples intermediário
Tempo			evento ativado por questão temporal
Mensagem			processo recebe ou envia uma mensagem
Sinal			processo recebe ou envia um sinal
Condição			condição acontece
Múltiplo			processo recebe ou envia vários eventos
Múltiplo			processo recebe vários eventos paralelos
Compensação			processo envia compensação
Escalamento			processo escala para nível superior

17.6. Gateways

Gateways servem para indicar o controle do fluxo do processo e são indicadas por um losango. No diagrama, elas podem dividir um fluxo em vários ou juntar vários fluxos em um único (OMG, 2013).

Um *gateway*, como mostra também a Tabela 17.5 pode ser de 6 tipos (OMG, 2013):

1. exclusivo;
2. inclusivo;
3. paralelo;
4. baseado em evento;
5. exclusivo baseado em eventos
6. paralelo baseado em eventos, e
7. complexo.

Um **gateway exclusivo** representa uma decisão de seguir um entre vários caminhos, ou de chegar a um lugar por um caminho entre muitos. É desenhada como um losango vazio, ou com um x dentro dele, conhecido como *gateway* exclusivo. O fluxo de saída pode ser marcado com uma barra diagonal para dizer que é o caminho *default*. Podemos associar o significado do *gateway* com uma instrução de *if* ou *case* de uma linguagem

Tabela 17.5.: Símbolos para os gateway.

Significado	Símbolo	Explicação
Exclusivo		escolhe um entre os caminhos
Inclusivo		escolhe alguns entre os caminhos
Paralelo		inicia vários fluxos simultaneamente
Baseado em eventos		inicia o fluxo do primeiro evento que acontece
Exclusivo Baseado em eventos		cada ocorrência de um evento subsequente ativa uma nova instância
Parelelo Baseado em eventos		inicia quando todos os eventos subsequentes ocorrem
Complexo		indica uma regra complexa de decisão

de programação. A Figura 17.1 mostra um gateway exclusivo sendo usado para indicar um caminho caso o pedido de crédito seja aprovado e outro caso não seja aprovado. Ela também demonstra o comportamento *default* do BPMN, que é considerar que se vários fluxos que entram em uma tarefa, apenas um precisa chegar a ela para ela iniciar¹.

Já um **gateway paralelo**, desenhado com um losango com um símbolo de uma cruz dentro dele, indica que um fluxo deve ser convertido em vários, que deverão ser seguidos em paralelo, ou vários fluxos devem ser combinados em um, sendo necessário que todos os caminhos cheguem ao *gateway* para poder continuar o processo. Eles são equivalentes as operações de *fork* ou *join*. A Figura 17.2 mostra um exemplo de uso desse *gateway*.

Temos um **gateway inclusivo** se houver um círculo dentro do losango, temos um caso onde um ou mais caminhos podem ser seguidos e todos devem ser avaliados.

O **gateway baseado em eventos**, o **gateway paralelo baseado em eventos**, e **gateway exclusivo baseado em eventos** são *gateways* bem interessantes. Todos eles depende da existência de eventos nos fluxos subsequentes.

¹Esse comportamento é o contrário do ARIS-EPC, onde todos os fluxos devem chegar a ela.

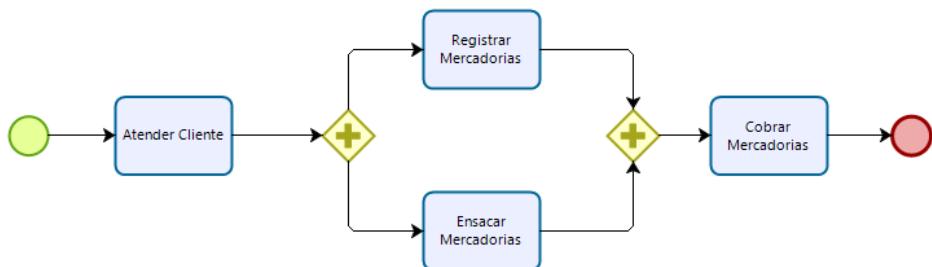


Figura 17.2.: Exemplo de uso de um evento paralelo em BPMN. Após receber o cliente em um caixa o registro de mercadorias e o ensacamento é feito em paralelo, só depois que ambos acabam se faz a cobrança ao cliente. Fonte: do autor.

Por exemplo, com o baseado em eventos, depois dele podemos ter dois fluxos, um com um evento que espera uma mensagem e outro que espera passar um prazo, indicando que a mensagem tem um prazo para chegar. Já com o paralelo baseado em eventos podemos seguir vários fluxos, cada fluxo tendo seu início quando o evento ocorre.

Finalmente o **gateway complexo**, que possui um asterisco dentro de si, significa uma comportamento complexo de condições ou sincronização.

17.7. Fluxos

Existem três seis de fluxos (OMG, 2013):

1. **fluxo de sequência normal**, que indicam a ordem das atividades, e não iniciam em um evento intermediário, são linhas contínuas com uma seta indicando a direção;
2. **fluxo não controlado**, é um fluxo que não é afetado por condições e não passa em um *gateway*,
3. **fluxo condicional**, que pode ter uma expressão de condição que determina se vai ser seguido. Se sai de uma atividade, deve ter um pequeno losango no seu início. Caso saia de um *gateway* é uma linha comum.
4. **fluxo default**, que é usado em *gateways* exclusivos ou inclusivos e mostram o caminho *default* a ser seguido se todas as outras condições nos outros fluxos saindo desse *gateway* forem falsas.
5. **fluxo de mensagem**, que indica o envio de uma mensagem entre dois participantes, indicados por uma linha tracejada que se inicia em um círculo e acaba em uma seta aberta;
6. **fluxo de exceção**, que se inicia em um evento intermediário anexado a um processo,
- e
7. associações, que ligam elementos do processo a informações, representados por linhas pontilhadas, e não são fluxos de controle.

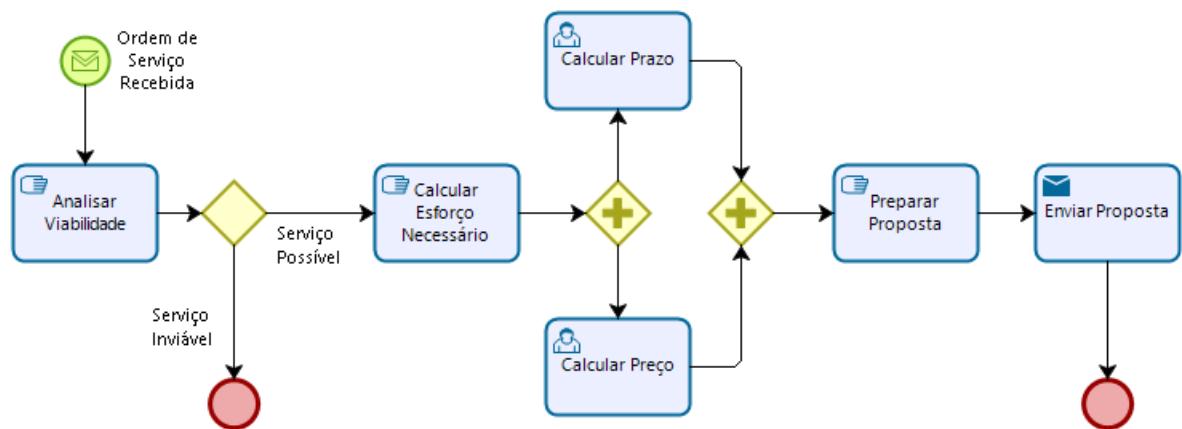


Figura 17.3.: Exemplo do atendimento da ordem de serviço em BPMN, também apresentado na Figura para Aris EPC. Fonte: do autor

17.8. Alguns Exemplos

A Figura 17.3 descreve, em BPMN, o atendimento a uma ordem de serviço. Ela representa o mesmo processo descrito em ARIS EPC na Figura 16.9.

17.9. Eventos Anexados a Atividades

Eventos iniciais e intermediários podem ser associados diretamente a uma atividade. Isso é feito colocando o evento sobre a borda da atividade.

Esses tipos de eventos podem interromper as atividades ou não. Caso não interrompam, seu círculos devem ser tracejados. Compensações e erros sempre interrompem a atividade.

17.10. Exercícios

Exercício 17.10-1: Vá para o site <http://jogodeanalisedesistemas.xexeo.net/> e visite a Livraria Resolve. A partir da sua visita desenhe todos os processos que você identificar, usando para isso o método ARIS-EPCe.

Exercício 17.10-2: Mostre as diferenças do que você conseguiu ou não representar com o EPCe e o BPMN.

18

Regras de Negócio

All action is for the sake of some end; and rules of action, it seems natural to suppose, must take their whole character and color from the end to which they are subservient.

(John Stuart Mill)

Conteúdo

18.1.	Classificações para Regras de Negócio	215
18.2.	Declarações Estruturais	216
18.3.	Declarações de Ações (ou Restrições)	218
18.4.	Derivações	219
18.5.	Regras e Comportamento	219
18.6.	Escrevendo Regras de Negócio	219
18.7.	Outros Modelos de Regras de Negócio	223

Por que regras de negócio?

Toda organização funciona de acordo com regras, que podem ou não estar modelada, por exemplo em modelos BPMN. Saber como tratá-las ou registrá-las é importante para entender o ambiente onde o sistema de informação vai funcionar e também quais dessas regras deverá seguir ou implantar.

Uma **regra de negócio** é “uma sentença que define algum aspecto do negócio. Tem o objetivo de afirmar a estrutura do negócio ou de controlar ou influenciar o comportamento do mesmo”(Hay et al., 2000) Regras de negócio são atômicas, isto é, não deve ser possível quebrá-las quebradas. Assim, se uma regra composta é encontrada, deve ser dividida em regras mais simples.

Segundo a OMG (2019), uma **regra de negócio** “é uma regra sob uma jurisdição de um negócio que tem a autoridade de mudar ou descartar a regra”. Assim, regras ou leis que são impostas ao negócio, sejam leis da física ou leis do país não são, na perspectiva da organização sendo modelada, regras de negócio. A organização vai decidir suas regras de negócio em função das limitações que sofre no seu ambiente, seja ele jurídico ou natural.

Além disso, uma regra de negócio tem que ser possível de ser praticada, sob a jurisdição de um negócio(OMG, 2019).

Levantar as regras de negócio, ou aproveitá-las se já escritas, tem como objetivo subsidiar a análise, fornecendo insumo para a criação de sistemas que as implementem.

Regras de negócio apoiam **políticas de negócio**. Um política de negócio é “um diretiva não acionável que apoia um objetivo de negócio”. A regra de negócio é específica, acionável e testável(IIBA, 2011), derivada da política de negócio(OMG, 2019).

Políticas de negócio guiam uma organização, mas não são diretamente executáveis. Em relação as regras de negócio elas tendem a ser(OMG, 2019):

- menos estruturadas;
 - menos discretas;
 - não atômicas;
 - expressas em um vocabulário menos padronizado e com menor cuidado formal, e
 - não executável diretamente.

Na prática, em relação a estratégia da organização, há uma hierarquia cujo nível mais alto é a missão, seguida das diretivas, que levam a políticas de onde são derivadas as regras de negócio(OMG, 2015)¹

¹Em relação a essa hierarquia, a referência é o *Business Motivation Model*, em que as regras de negócio fazem parte dos meios, enquanto objetivos e metas (*goals* e *objectives*) fazem parte dos fins. É possível que outros modelos considerem que políticas são criadas a partir de objetivos, metas ou outro conceito.

Ao escolher um modelo de regras de negócio é necessário entender se o modelo está sendo construído para um **mundo claro** (*light*) ou **escuro** (*dark*). Em um mundo claro se faz a suposição que tudo que não é expressamente proibido é permitido. Já em um mundo escuro, tudo que não é explicitamente permitido é proibido, ou seja, tudo que é permitido deve ser explicitado. É possível que em um mundo primariamente claro existam áreas escuras, isto é, em geral não há preocupações com proibições não explicitadas, porém em alguns lugares são necessárias **autorizações**, ou seja, uma permissão explícita, que permitam acesso a material perigoso, escarço, de alto valor, privado, etc... Nesse caso, é impossível realizar algo sem a autorização correspondente(OMG, 2019). Autorizações são, também, regras de negócio(R. G. Ross, 2004).

Existem diferenças claras entre regras de negócio e requisitos, já que as primeiras existem no negócio e os últimos são exigências feitas ao sistema, porém elas podem sugerir ou implicar em requisitos(Gladys S. W. Lam, 2015). As regras de negócio também sofrem mudanças com o tempo, o que implica em novas oportunidades, ou problemas, que podem implicar, por sua vez, em novos requisitos para o mesmo ou um novo sistema.

Regras de negócio são muito estudadas hoje em dia. A maioria dos importantes autores americanos da área se reuniu em um grupo conhecido como *Business Rule Group*, que produziu alguns documentos que forneceram a estrutura básica que estamos apresentando aqui, e mais tarde se filiou também a OMG, fonte de vários padrões tratados neste livro. Todas as definições dadas neste texto são versões dos originais em inglês.

18.1. Classificações para Regras de Negócio

Segundo Hay et al. (2000), uma regra de negócio pode ser de três categorias:

- **declarações estruturais**, um conceito ou a declaração de um fato que expressa algum aspecto da estrutura da organização; elas podem ser:
 - definições de **termos atômicos**, ou
 - **fatos** relacionando esses termos;normalmente são descritas por meio de um diagrama de entidades e relacionamentos;
- **declarações de ação**, que descrevem aspectos dinâmicos do negócio, podendo ser uma expressão de:
 - uma **restrição**, ou de
 - uma **condição** que controla as ações de uma organização, ou
- **derivações**, a declaração de um conhecimento que é derivado a partir de outro.

Esta classificação será adotada nas próximas seções para explicar tipos de regras de negócio, porém existem outras propostas.

18.2. Declarações Estruturais

As **declarações estruturais** incluem os termos de negócios e fatos relacionando termos Hay et al. (2000). Todas as regras serão construídas sobre os fatos, que por sua vez serão construídos sobre os termos (R. G. Ross, 2004).

Essas declarações, na prática, equivalem a um registro do vocabulário e linguagem do projeto. “O elemento básico de uma regra de negócio é a linguagem utilizada para expressá-la. A definição das palavras utilizadas nessa linguagem descreve como as pessoas pensam e falam” (Hay et al., 2000).

As declarações estruturais podem declarar atributos, generalizações ou participações. Uma participação, por sua vez, pode ser um papel, uma agregação ou uma associação simples.

18.2.1. Termos

Um **termo** é “uma palavra ou frase que tem um significado específico para ao negócio” (Hay et al., 2000).

Normalmente um projeto só começa realmente a progredir quando a equipe de desenvolvimento comprehende o discurso dos clientes. Para isso, nas primeiras entrevistas ou reuniões, é feito um esforço para levantar um glossário de termos, e, mais tarde, muitos desses termos podem aparecer no Modelo Conceitual de Dados do sistema.

Um exemplo tirado de um sistema governamental:

- contribuinte: é a pessoa física ou jurídica responsável pelo pagamento de um imposto.

A definição de um termo muitas vezes envolve uma relação com outros termos, como no exemplo acima.

Um termo pode ser um tipo (uma classe) ou um literal (uma instância). No caso de regras de negócio costumamos trabalhar principalmente com tipos. Um tipo especial de tipo é um sensor, que representa algo que detecta e reporta valores que mudam constantemente no ambiente (mundo externo). Existe um sensor especial, o relógio, que relata a passagem do tempo, cujo valor é sempre o instante corrente (data e hora corrente).

Os termos definidos são reunidos em um glossário e em um dicionário de dados e podem levar a construção de uma ontologia.

18.2.2. Fatos

A partir dos termos, devemos construir sentenças que descrevam o negócio a partir das relações entre termos e da estrutura criada por essas relações.

Um **fato** “declara uma associação entre dois ou mais termos”(Hay et al., 2000).

Normalmente esses fatos são caracterizados nas entrevistas e reuniões, a partir de declarações do cliente de como o negócio funciona.

Fatos relacionando termos são bastante fáceis de serem encontrados. Muitas vezes encontramos primeiro um fato e depois analisamos o significado dos seus termos.

Tabela 18.1.: Exemplos de Fatos

Tipo de Fato	Exemplo
Atributo	DRE é um atributo de aluno
Generalização	Aluno de graduação é um tipo de Aluno
Papel	Um aluno pode ser representante de classe
Agregação	Uma turma precisa ter alunos
Associação simples	Um aluno deve fazer provas

Duas classificações podem ser feitas com os fatos. A primeira os divide em fatos base, que são simplesmente declarados, e fatos derivados, que são calculados por outras regras de negócio(Hay et al., 2000). Um exemplo de fato básico é: **uma aluno pertence a um curso**. Já um exemplo de fato derivado seria: “A nota final do aluno é baseada na nota das provas e nas notas dos trabalhos.” Fatos derivados implicam na necessidade de uma regra de negócio de derivação. No exemplo, a derivação poderia ser a fórmula:

$$\text{Nota Final} = \frac{\text{média das provas} \times 8 + \text{média dos trabalhos} \times 2}{10} \quad (18.1)$$

A segunda classificação particiona os fatos em atributos, participação e generalização, sendo que participação pode ser dividido em associação, agregação e papel(Hay et al., 2000). Nesse caso há forte influência dos modelos conceituais de dados e dos modelos de orientação a objeto, que podem ser utilizados para documentar essas regras. A Tabela 18.2 mostra essas definições com exemplos de forma compacta. O assunto é similar a modelos tratados nos Capítulos 23 e 24.

18.2.3. Declarações estruturais e o modelo de dados

Termos e fatos estabelecem a estrutura de um negócio. Esta estrutura é normalmente descrita em um modelo de dados. Assim, a maior parte do trabalho inicial com as regras de negócio estruturais pode ser descrita por meio de um modelo de dados conceitual, tema tratado no próximo capítulo.

A seguir damos alguns exemplos de declarações estruturais, para um sistema acadêmico hipotético:

- Alunos são pessoas matriculadas em um curso
- Um Aluno de Graduação é um tipo de Aluno que cursa a graduação

18. Regras de Negócio

Tabela 18.2.: Classificação e exemplos das regras de negócio do tipo fato(Hay et al., 2000)

Tipo	Descrição	Exemplo
Atributo	um termo é atributo de outro	Nome é um atributo de Aluno
Generalização	um termo é uma generalização de outro	Um professor é um funcionário público
Participação/Agregação	relacionamento todo/parte	Uma turma é composta de alunos
Participação/Papel	um termo serve como ator (outro termo) por meio de uma interação com o ambiente	Um professor pode ser responsável por uma turma em um período
Participação/Associação	outros relacionamentos	Um aluno pode pedir revisão da nota

- Um Aluno de Pós-Graduação é um tipo de Aluno que cursa a pós graduação
- Um Curso é implementado por um conjunto de Cadeiras
- Uma Cadeira é ministrada por um Professor
- Um Professor é um tipo de Funcionário que dá aulas e faz pesquisa
- Alunos podem se matricular em Cadeiras durante o período

18.3. Declarações de Ações (ou Restrições)

Representam as restrições ou condições que as instituições colocam em suas ações. Podem aparecer por força de lei, prática de mercado, de decisão da própria empresa ou ainda outros motivos.

Exemplos:

- Um aluno deve ter um DRE
- Um aluno não pode se registrar em dois cursos que acontecem no mesmo horário

Uma Declaração de Ação pode ser uma condição, uma restrição de integridade ou uma autorização. Uma condição diz que se alguma coisa for verdade, então outra regra deve ser aplicada (se - então). Uma restrição de integridade é uma declaração que deve sempre ser verdade. Uma autorização dá uma prerrogativa ou privilégio a um termo, normalmente permitindo que uma pessoa possa realizar uma ação.

Declarações de ações podem ser de uma variedade de outros tipos. Recomendamos a leitura de Uma declaração de ação pode dizer que precisa acontecer (controle) ou o que pode acontecer (influência).

18.4. Derivações

São regras que mostram como transformar conhecimento em uma forma para outro tipo de conhecimento, possivelmente em outra forma, incluindo leis da natureza(Hay et al., 2000). Geralmente são regras ou procedimentos de cálculo ou manipulação de dados. Exemplo:

- O valor a ser pago do imposto predial é 3
- A lista de devedores inclui todos os devedores a mais de dois anos.

18.5. Regras e Comportamento

Cada regra de negócio tem a possibilidade de ser violada em um ou mais momentos onde o estado sistema é alterado. Estes momentos são descritos no modelo comportamental do sistema, por meio de práticas como Casos de Uso, Histórias do Usuário ou outras, que representam as funcionalidades do sistema.

Analizando uma regra podemos ver que tipos de funcionalidades são prováveis de necessitarem de alguma atenção do sistema. Por exemplo, suponha que um sistema de vendas para uma empresa possua as regras(R. Ross, 1998):

- Um aluno cliente solicita um produto
- Um vendedor é designado para atender um cliente permanentemente

A descrição acima faz com que nos perguntemos:

- O que acontece quando um cliente faz seu primeiro pedido(e não tem ainda um vendedor associado)?
- O que acontece quando um vendedor se desliga da empresa.

Em toda funcionalidade, um conjunto de regras deve ser ativado e cada erro deve ser reportado ao usuário.

18.6. Escrevendo Regras de Negócio

Existem muitas formas de descrever regras de negócio, de acordo com o grau de formalismo e a necessidade de execução (ou compreensão por serem humanos) que desejamos. A Tabela 18.3 compara quatro formas básicas de descrição.

18.6.1. Como Definir Termos

R. G. Ross (2017) apresenta um conjunto de orientações para a descrição de definições. Ele parte de alguns critérios gerais para uma definição de qualidade(R. G. Ross, 1999):

18. Regras de Negócio

Tabela 18.3.: Formas de descrever regras de negócio e suas características adaptado de (Von Halle, 2002).

Fragmento de conversação de negócios	Versão em linguagem natural	Versão em uma linguagem de especificação de regras	Versão em uma linguagem de implementação de regras
Pode não ser relevante	Relevante	Relevante	Executável
Pode não ser atômica	Atômica	Atômica	Pode ser procedural
Pode não ser declarativa	Declarativa	Declarativa	
Pode não ser precisa	Não totalmente precisa	Precisa	
Pode ser incompleta	Pode ser incompleta	Completa	
Pode não ser confiável	Confiável	Confiável	
Pode não ser autêntica	Autêntica	Autêntica	
Pode ser redundante	Pode ser redundante	Única	
Pode ser inconsistente	Pode ser inconsistente	Consistente	

- deve ser fácil dar exemplos do que foi definido, e não podem existir contra-exemplos;
- cada definição deve comunicar a essência do que a coisa é, não o que ela faz, como é usado ou porque é importante;
- a definição deve ter foco nas características únicas da coisa;
- cada coisa definida tem que ser distingível de todas as outras coisas definidas;
- cada definição tem que ser tão concisa e pequena quanto possível, sem perda do significado, e
- as definições devem ser legíveis.

As orientações são divididas em seis contextos e são apresentadas a seguir. Para maiores detalhes, consultar R. G. Ross (2017):

1. montar corretamente a definição;
 - a) a definição de um termo deve começar com um substantivo;
 - b) a primeira palavra de uma definição tem que ser um substantivo diferente do termo sendo definido
 - c) a definição não pode ser simplesmente um sinônimo;
 - d) a primeira palavra ou frase da definição de um conceito deve representar um conceito mais amplo do que o conceito sendo definido, o resto da definição deve mostrar como o conceito sendo definido é distinto ou um caso especial

- ou variação da classe mais geral;
2. evitar plurais;
 - a) o termo sendo definido tem que estar no singular;
 - b) a primeira palavra da definição tem que estar no singular;
 3. capturar a essência do conceito;
 - a) a definição do termo deve expressar a essência do conceito, não seu objetivo, uso, ou função;
 - b) uma definição deve ser clara sobre se o conceito sendo definido é individual ou geral;
 4. garantir a qualidade interna das definições;
 - a) a definição não deve ser feita com várias sentenças;
 - b) a definição não deve prescrever ou englobar uma regra de negócio;
 - c) a definição deve fornecer um antecedente claro para cada pronome que usa;
 - d) uma definição deve fornecer um antecedente claro para cada artigo definido usado;
 5. alinhar definições múltiplas;
 - a) a definição de um termo qualificado não pode se afastar da definição do termo sendo qualificado;
 - b) a definição de um termo qualificado não pode repetir texto ou estar em conflito com o termo sendo qualificado;
 - c) a definição de um termo ou uma parte dessa definição não pode ser repetida em uma definição de outro termo;
 - d) uma definição de um termo composto não pode se afastar do significado dos termos usados tomados de forma coletiva
 - e) um conjunto de definições não pode ser circular
 6. contexto;
 - a) Uma entrada cujo significado não pode ser padronizada para todo o escopo do vocabulário deve ser desambiguizada pelo contexto.

Na Tabela 18.4 são apresentadas definições corretas e incorretas segundo as orientações de R. G. Ross (2017).

Tabela 18.4.: Exemplos de definições bem e mal escritas segundo as orientações de R. G. Ross (2017).

Termo	Definição	Avaliação	Explicação
DRE	identifica o aluno quando matriculado em um curso	✗	1.a
DRE	um DRE é um número que identifica um aluno	✗	1.b
DRE	um número que identifica um aluno segundo sua matrícula	✓	

18.6.2. Representações Gráficas

Obviamente, como fizemos aqui, a forma textual é bastante útil. Uma maneira bastante comum é utilizar Diagramas de Entidades e Relacionamentos. Deve ficar claro, porém, que a utilização de DER para definir regras de negócio não é igual à utilização de DER para definir o modelo conceitual de dados de um sistema.

DERs, porém, não representam todas as características das regras de negócio. R. G. Ross (1999), propôs uma notação mais complexa, incluindo (muitos) novos símbolos em um DER. Essa notação, porém, foge do escopo desse texto.

Vejamos a descrição de duas regras para uma livraria (Figura 18.1):

- Um cliente deve pedir livros.
- Livros são entregues por distribuidoras.

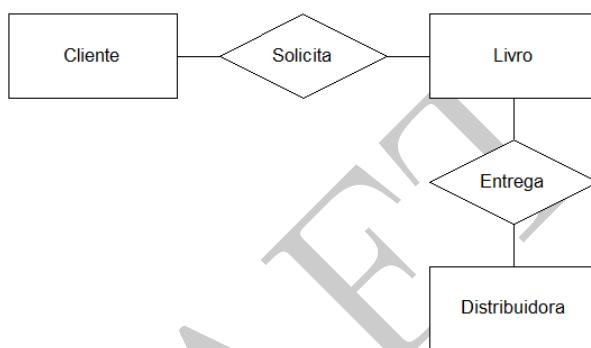


Figura 18.1.: Exemplo de duas regras de negócio descritas juntas e utilizando uma notação de DER simplificada.

Alguns autores fazem uma descrição regra a regra, como na Figura 18.2.



Figura 18.2.: Exemplo da descrição das regras de negócio em separado.

As regras de negócio serão utilizadas nos próximos passos para definir modelos mais formais do sistema. Mas podemos, por exemplo, definir a cardinalidade dos termos em cada relacionamento descrito, como na Figura 18.3 e no texto a seguir:

- Um cliente pede um ou mais livros,
- Livros podem ser pedidos por nenhum, um ou mais clientes,
- Uma distribuidora entrega um ou mais livros,

18.7. Outros Modelos de Regras de Negócio

- Um livro é entregue por apenas uma distribuidora.

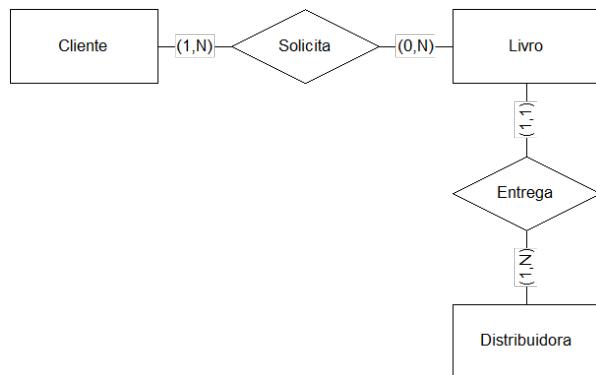


Figura 18.3.: Regras com cardinalidades.

18.7. Outros Modelos de Regras de Negócio

O *Semantics of Business Vocabulary and Business Rules* define o conceito de Elemento de Orientação², que pode ser uma regra de negócio, que remove graus de liberdade, um conselho, que não remove graus de liberdade, ou uma política de negócio, que são as bases para conselhos e regras de negócio. As regras de negócio também se dividem em definicionais ou comportamentais. Já as regras de negócio comportamentais se dividem em obrigações, proibições e permissões com restrições(OMG, 2019). Esse documento é bem preciso em suas definições, mas sua complexidade adicional em relação ao modelo de Hay et al. (2000) foge, no momento, ao escopo deste livro.

Já Von Halle (2002) propõe uma classificação mais detalhada e um conjunto de *templates* que pode ser utilizado para descrever regras de negócio (aqui adaptado para português), apresentado na Tabela 18.5.

Na Tabela 18.5, operadores podem ser os operadores lógicos ou ainda outros como “tem no máximo”, “contém”, etc.

²Element of Guidance

18. Regras de Negócio

Tabela 18.5.: Classificação para regras, definição e templates adaptada de (Von Halle, 2002)

Classificação	Descrição Detalhada	Template
Termo	<p>Nome ou sentença com uma definição acordada que pode ser:</p> <ul style="list-style-type: none"> • Conceito, classe, entidade, • Propriedade, detalhe, atributo, • Valor, • Conjunto de valores 	<termo> é definido como <texto>
Fato	<p>Sentença que relaciona termos em observações relevantes ao negócio</p> <ul style="list-style-type: none"> • Relacionamentos entre entidades • Relacionamentos entre entidades e atributos • Relacionamentos de herança 	<termo1> é um <termo2> <termo1> <verbo> <termo2> <termo1> é composto de <termo2> <termo1> é um papel de <termo2> <termo1> tem a propriedade <termo2>
Computação	Sentença fornecendo um algoritmo para calcular o valor de um termo	<termo> é calculado como <formula>
Restrição obrigatória	Sentença que indica restrições que devem ser verdade em informações fornecidas ao sistema (input)	<termo1> deve ter <no mínimo, no máximo, exatamente n> <termo2> <termo1> deve ser <comparação> <termo2> ou <valor> ou <lista de valores> <termo> deve ser um de <lista de valores> <termo> não pode star em <lista de valores> se <regra> então <restrição> <termo1> deveria ter <no mínimo, no máximo, exatamente n> <termo2> <termo1> deveria ser <comparação> <termo2> ou <valor> ou <lista de valores> <termo> deveria ser um de <lista de valores> <termo> não poderia star em <lista de valores> se <regra> então <restrição> se <termo1> <operador> <termo2, valor, ou lista de valores> ... então <termo3> <operador> <termo4> se <termo1> <operador> <termo2, valor, ou lista de valores> então <ação>
Guideline	Sentença que indica restrições que deveriam ser verdade em informações fornecidas ao sistema (input)	
Conhecimento inferido	Sentenças que expressam circunstâncias que levam a novas informações	
Iniciador de ação	Sentença expressando condições que levam ao início de uma ação	

18.8. Conclusão

É interessante notar que regras de negócio devem ser feitas de forma declarativa, não indicando como vão ser implementadas, onde vão funcionar, quem será responsável ou quando devem ser testadas. Desse jeito as regras serão flexíveis o suficiente para ser utilizadas na modelagem do negócio. Como devem ser declarativas, elas também não devem ser escritas como um procedimento.

Exercício 18.8-1: Vá para o site <http://jogodeanalisedesistemas.xexeo.net/> e visite a Livraria Resolve. A partir da sua visita descreva e classifique todas as regras de negócio que você encontrar.

DRAFT

DRAFT

Parte IV.

Requisitos

DRAFT

DRAFT

19

Requisitos

The hardest part of the software task is arriving at a complete and consistent specification, and much of the essence of building a program is in fact the debugging of the specification.

(Frederick P. Brooks)

Conteúdo

19.1.	Definição Formal	230
19.2.	Requisitos x Benefícios	234
19.3.	Tipos de Requisitos	234
19.4.	O Papel dos Requisitos no Projeto	236
19.5.	Engenharia de Requisitos	237
19.6.	Elicitação de Requisitos	237
19.7.	Níveis de Abstração dos Requisitos	242
19.8.	O Risco de Requisitos Falsos	242
19.9.	Características de um Bom Requisito	243
19.10.	Atributos dos Requisitos	245
19.11.	Priorizando Requisitos	247
19.12.	Requisitos Mudam com o Tempo	248

19. Requisitos

19.13.	Requisitos e Necessidades	248
19.14.	Requisitos de Acordo com Normas IEEE	249
19.15.	A Visão do Novo Sistema	252
19.16.	Descrevendo Requisitos	253
19.17.	Exercício	255

Por que requisitos?

Levantar e especificar os requisitos são a principal função da análise de sistemas, seja qual for a metodologia usada.

Para construir qualquer coisa é preciso saber o que vai ser feito, de maneira a atender da melhor forma possível as demandas das partes interessadas. No caso de software, ou de sistemas em geral, este conhecimento normalmente é registrado na forma de uma **especificação de requisitos**. Em uma interpretação restrita, requisito de software é uma exigência feita ao software no início do projeto.

Como projetos de software normalmente são desenvolvidos em fases, sejam elas sequenciais ou cíclicas, é comum que uma fase anterior construa artefatos de conhecimento que permitem definir melhor o que será feito na próxima fase. Todo artefato criado em uma fase de projeto que define de alguma forma o que é preciso ser feito em uma fase posterior de um projeto pode ser chamado também de requisito¹. Assim, em uma interpretação ampla, qualquer especificação, realizada em uma fase qualquer do processo de software, é um requisito para a próxima fase, já que ela tem que ser cumprida.

Essa ambiguidade, entre a interpretação ampla e restrita, normalmente é resolvida pelo contexto. Outros termos também são adicionados para qualificar o termo requisito e ajudar na interpretação do seu significado exato. Por exemplo, requisitos do usuário é o termo usado normalmente para significar requisitos levantados pelo usuário, com ou sem ajuda de analistas de sistemas, em uma fase anterior, ou no início, da análise de sistemas.

19.1. Definição Formal

Segundo a norma *ISO/IEC/IEEE International Standard - Systems and software engineering – Vocabulary*(IEEE, 2010), um **requisito** é:

- uma condição ou capacidade necessária para um usuário resolver um problema ou atingir um objetivo;
- uma condição ou capacidade que precisa ser possuída ou alcançada por um sistema, produto, resultado, serviço ou componente para satisfazer um acordo, contrato,

¹O termo em inglês é *requirement*, o que levou a algumas traduções usarem o termo requerimento, que é errado em português.

- padrão, especificação ou outro documento formalmente imposto, ou
- Uma representação documental de uma condição ou capacidade no sentido das duas definições anteriores.

Esta definição, múltipla, permite o uso da interpretação ampla de requisitos.

Sommerville e Sawyer (1997) definem requisitos como

uma especificação do que deve ser implementado. Eles são descrições de como o sistema deve se comportar, ou de uma propriedade ou atributo do sistema. Eles podem ser uma restrição ao processo de desenvolvimento do sistema.

Esses autores também afirmam que “não existe uma melhor forma de escrever requisitos”, isto é, cada projeto pode precisar de uma forma específica de acordo com condições como assunto, cultura organizacional, questões de segurança, etc. Por exemplo, é razoavelmente mais caro fazer especificações em linguagens formais, exigindo pessoal com mais formação, software especializado, etc. Além disso, especificações formais são difíceis de serem validadas ou negociadas com o cliente, logo raramente são usadas em desenvolvimento de sistemas comerciais. Porém são usadas em especificações de sistemas que precisam de níveis adicionais de garantias de segurança e robustez, como o *TCAS* (*Traffic Alert and Collision Avoidance System*) e outros (Craigen, Gerhart e Ralston, 1993).

19.1.1. Exemplos de Requisitos

Como visto, a noção de requisito pode ser muito ampla. Eles podem ser encontrados em várias formas, dependendo dos métodos e práticas usados em cada projeto de software específico.

Na sua forma mais tradicional, um requisito é apenas uma sentença que define uma exigência feita ao sistema, na perspectiva do usuário, escrita em linguagem corrente, como em:

“O sistema deverá permitir que o correntista verifique o saldo de sua conta”.

Em casos de requisitos escrito em linguagem natural, muito encontrados em documentos conhecidos como *RFP*, *Request for Proposal*, a forma mais correta é que a exigência seja sempre feita ao sistema sendo especificado, nunca ao usuário, no caso o correntista, porém não é incomum encontrar RFPs contendo erros de perspectiva, já que muitas vezes são construídas por pessoas leigas em relação a área de Engenharia de Software.

Muitas formas já foram propostas para eliciar ou analisar requisitos, ou para fazer uma parte desse trabalho. Algumas deram foram:

- eventos essenciais (McMenamin e Palmer, 1984);
- diagramas de fluxo de dados (DFD) (Gane e Sarson, 1979);
- mini especificações (Gane e Sarson, 1979);
- diagramas de contexto (Gane e Sarson, 1979; K. E. Wiegers e Beatty, 2013);
- cartões volare, como o da Figura 19.1 (J. Robertson e S. Robertson, 2006);

19. Requisitos

- especificações formais (Sommerville, 2015);
- casos de uso (Jacobson, Christerson et al., 1992; Jacobson, Spence e Bittner, 2011);
- diagramas de caso de uso, como o da Figura 19.2 (Jacobson, Christerson et al., 1992; OMG, 2017);
- mapas mentais, como o da Figura 19.3;
- descrições em linguagem natural e hierárquicas (K. E. Wiegert e Beatty, 2013)
- árvores de funcionalidades (K. E. Wiegert e Beatty, 2013), e
- histórias do usuário (Cohn, 2004).

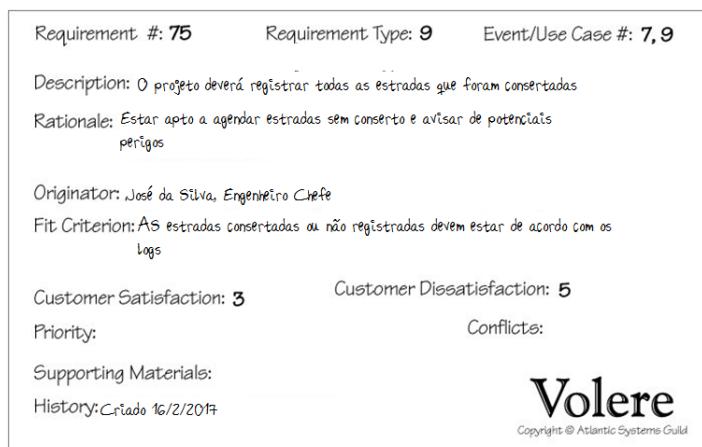


Figura 19.1.: Um cartão Volere.

As duas formas mais em voga no mercado em 2020 serão apresentadas nos capítulos a seguir: histórias do usuário e casos de uso.

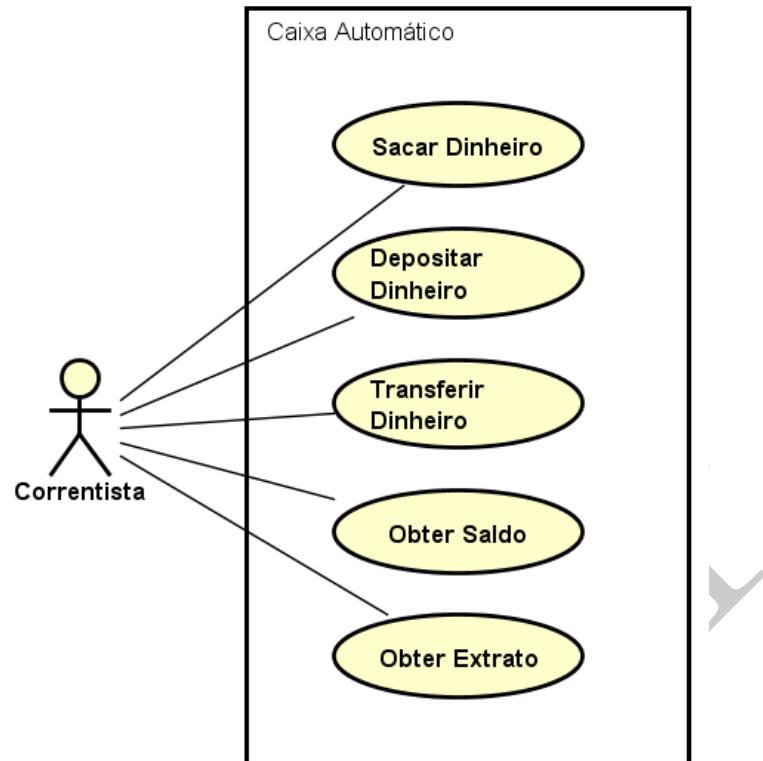


Figura 19.2.: Exemplo de um diagrama de casos de uso



Figura 19.3.: Exemplo de um mapa mental.

19.2. Requisitos x Benefícios

No Capítulo 14 apresentamos brevemente requisitos e benefícios. Um benefício típico seria “diminuir os erros no preenchimento dos pedidos”, enquanto um requisito típico seria “o sistema deve oferecer ao comprador a lista de produtos disponíveis na loja”.

Algumas pessoas podem pensar que um benefício também é algo que se pede ao software, logo também é um requisito. Isto está errado. Um requisito é sempre uma demanda determinada diretamente para o produto ou serviço sendo especificado, enquanto um benefício é uma mudança esperada no negócio quando o produto ou serviço estiver em operação. Não se pode testar um benefício, por exemplo, a não ser que haja, para ele, um requisito específico em paralelo. Um exemplo disso seria um benefício como “permitir compra com cartão de crédito” e um requisito na forma “o sistema deverá permitir que o cliente pague com cartão de crédito”.

19.3. Tipos de Requisitos

Vários autores dividem os requisitos de formas diferentes, porém quase todos concordam que existem dois tipos básicos de requisitos: os **requisitos funcionais** e os **requisitos não funcionais**. Além disso é comum que um tipo de requisito seja colocado a parte, as **restrições**.

19.3.1. Requisitos Funcionais

Um requisito funcional representa algo que o sistema deve fazer, ou seja, uma função esperada do sistema que agregue algum valor a seus usuários. Exemplos típicos incluem a emissão de relatórios e a realização e manutenção de cadastros.

Um requisito funcional tradicional é escrito de uma forma específica, exigindo que o sistema cumpra uma finalidade, como em “o sistema deve permitir que um aluno solicite a emissão do seu histórico escolar”. Também é possível que um requisito funcional seja uma especificação mais detalhada, impondo limites ou exigências a outro requisito, com em: “O CR será calculado a partir da média ponderada das notas em cada curso, sendo que o peso de cada nota será a quantidade de créditos acreditada ao curso.”

19.3.2. Requisitos não funcionais

Requisitos não funcionais falam da forma como os requisitos funcionais devem ser alcançados. Eles definem propriedades e restrições do sistema. Muitos requisitos não funcionais são também requisitos de qualidade, como exigências de desempenho e robustez. Outros são restrições ou exigências de uso de uma ou outra tecnologia.

19.3. Tipos de Requisitos

Muitas vezes não é só difícil descobrir quais são os requisitos não-funcionais, mas também produzir uma especificação do sistema que possa ser cumprida em custo razoável e prazo hábil de forma a atender os usuários. Um exemplo típico seriam dois requisitos não funcionais que, genericamente, são opostos: velocidade e transportabilidade. Ora, para fazer um software muito veloz você precisa adaptá-lo especificamente para o ambiente onde ele está funcionando. Para fazer um software transportável, você precisa implementá-lo de forma a funcionar no maior número possível de ambientes. Normalmente esses dois requisitos se relacionam de forma inversa e para implementá-los simultaneamente é necessário um grande investimento de recursos.

Alguns requisitos não funcionais, quando negados, geram um anti-requisito que nunca seria pedido por um cliente. Por exemplo, um requisito não funcional comum é que o software seja confiável. Ninguém pediria para ser construído um software não confiável, porém diferentes clientes estão interessados em diferentes níveis de confiabilidade e diferentes formas de certificar esse nível.

Deve ser levado em conta que quanto mais esforço é colocado para se alcançar um requisito, maior é o custo agregado ao projeto e isso servirá como referência para o cliente escolher o grau de confiabilidade que deseja. Por isso um requisito deve ser verificável. O requisito robusto, por exemplo, pode ser medido por meio do **MTBF** sistema.

(Sommerville, 2015) mostra uma pequena taxonomia dos tipos de requisitos não-funcionais mais importantes, apresentada na Figura 19.4.

MTBF é o tempo médio entre falhas.

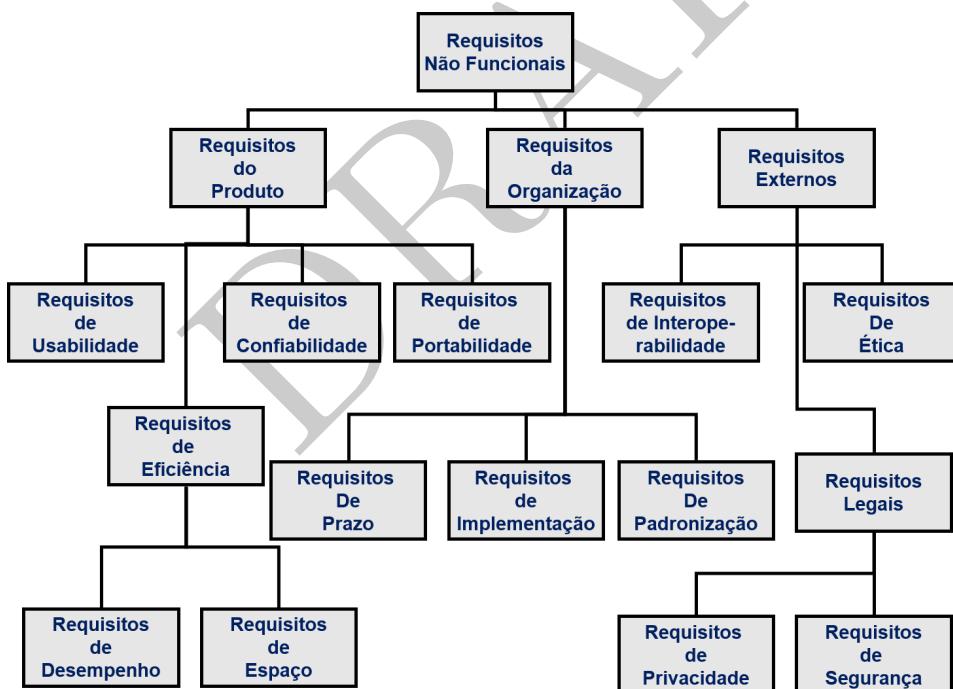


Figura 19.4.: Pequena taxonomia de requisitos não funcionais. Fonte: (Sommerville, 2015)

19.3.3. Restrições

É difícil a discussão para decidir se algo é um requisito ou uma restrição. Essa divisão é muitas vezes artificial e feita a partir de um ponto de vista específico. Por exemplo, se uma especificação obriga o sistema a ser desenvolvido em Java, isso é uma restrição ou um requisito?

Uma maneira de separar restrições de requisitos é analisar se a demanda vem da vontade dos clientes ou usuário diretos, ou é algo obrigatório para eles, como uma lei ou uma regra organizacional que limita as possibilidades do projeto. Ou seja, se o sistema deve ser feito em Java por uma obrigação qualquer, e existem soluções mais fáceis, talvez seja melhor considerá-lo uma restrição.

19.3.4. Outros tipos de requisitos

A literatura de requisitos é ampla e propõe muitas divisões para os requisitos, como requisitos de interface, requisitos de ambiente e etc. Essas divisões são úteis para organização de documentos de especificação de requisitos..

Requisitos de Informação representam a informação que o cliente deseja obter do sistema. Requisitos de informação também são atendidos por eventos. Muitas vezes o cliente expressa requisitos de informação de forma funcional. Outras vezes o cliente está preocupado em conseguir uma informação, mas não sabe como fazê-lo na forma de um requisito funcional. Em todo caso, sempre nos preocuparemos em levantar todos os requisitos de informação, pois eles representam as respostas fundamentais do sistema.

Requisitos de interface são exigências de como o sistema deverá interagir com outros sistemas ou humanos. Também são comuns na maioria das especificações formais. **Requisitos de desempenho** especifica requisitos numéricos estáticos e dinâmicos que são determinados para o sistema ou para a interação com o usuário(IEEE, 1998).

19.4. O Papel dos Requisitos no Projeto

J. Robertson e S. Robertson (2006) mostram o papel dos requisitos em um projeto de software de acordo com a Figura 19.5. O processo de Elicitação de Requisitos parte das Necessidades do Usuário e gera requisitos tanto para a Modelagem de Sistema (principalmente funcionais) quanto para o Projeto de Sistema (principalmente não funcionais). Ele também recebe feedback dessas e outras fases que podem levar a necessidade de correção dos requisitos. Modelos gerados pela Modelagem de Sistemas auxiliam na busca de novos requisitos, por exemplo, permitindo entender que informação está faltando para construir um modelo de dados correto.

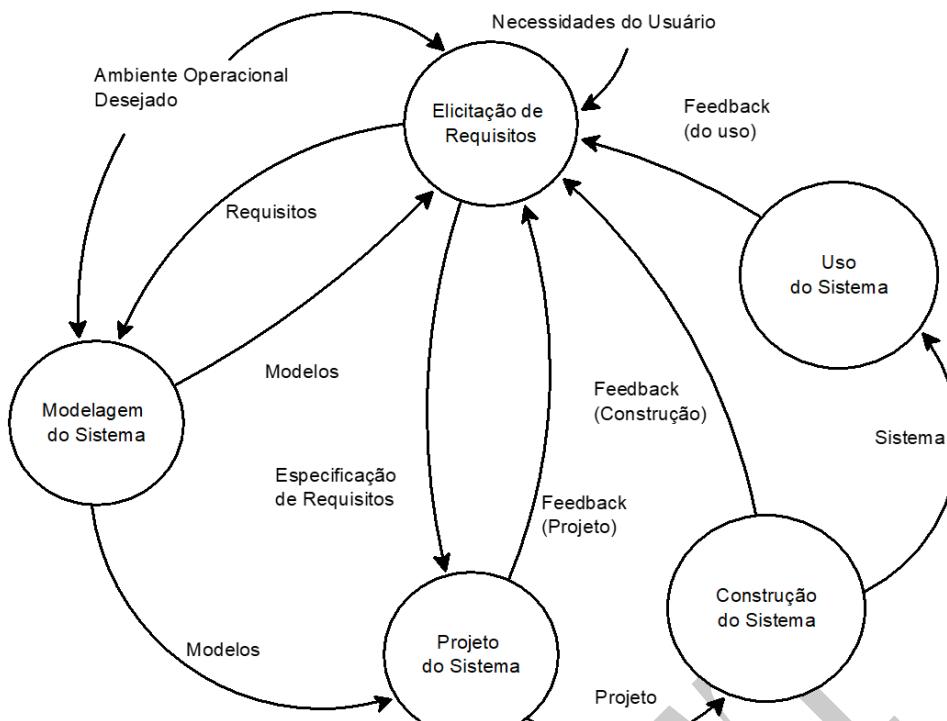


Figura 19.5.: O papel dos requisitos no desenvolvimento do sistema. Fonte:(J. Robertson e S. Robertson, 2006)

19.5. Engenharia de Requisitos

A Engenharia de Requisitos é a disciplina relacionada com o trabalho com requisitos, e pode ser dividida em duas grandes áreas: Desenvolvimento de Requisitos e Gerência de Requisitos, sendo que a última não é tratada neste texto. O Desenvolvimento de Requisitos, por sua vez, pode ser dividido em(K. E. Wiegers e Beatty, 2013):

- Elicitação, que cobre todas as atividades de descoberta dos requisitos;
- Análise, que cobre o entendimento mais rico e preciso de cada requisito;
- Especificação, que trata da representação e armazenamento dos conhecimentos coletados de forma persistente e bem organizada, e
- Validação, que trata da confirmação que os requisitos estão corretos e vão permitir uma solução que satisfaz as partes interessadas.

19.6. Elicitação de Requisitos

A **elicitação** é o levantamento, registro e validação (Christel e Kand, 1992) das expectativas dos diversos interessados no sistema, seguido da consolidação e validação dessas expectativas em requisitos formais. Contemplar essas diferentes visões implica projetar interesses divergentes e conciliá-los.

19. Requisitos

Esse trabalho investigativo de descoberta dos requisitos, é feito com dois processos gerais: entrevistas ou reuniões, ou observação. A diferença básica é que no primeiro tipo de elicitação são feitas perguntas aos clientes, enquanto na segunda o analista observa o ambiente de trabalho da organização. O entendimento gerado pela processo de observação pode exigir entrevistas ou reuniões adicionais para confirmação dos achados. Existem muitas práticas específicas para ambos os processos, como entrevistas fechadas, entrevistas por meio de questionários, reuniões de vários formatos, observação direta, observação de cenas gravadas, análise de documentos, atuação no papel do usuário, etc. (Sommerville, 2015).

Já K. E. Wieggers e Beatty (2013) sugerem 8 técnicas de elicitação e indicam em que tipos de projetos são adequadas, como mostrado na Tabela 19.1.

Tabela 19.1.: Adequação das formas de elicitação propostas por K. E. Wieggers e Beatty (2013) e alguns tipos de projeto.

TIPOS DE PROJETO	TÉCNICAS DE ELICITAÇÃO					
	entrevistas	workshops	grupos de foco	observações	questionários	análise da interface do sistema
software para o mercado consumidor	✓		✓		✓	
software interno da organização	✓	✓	✓	✓	✓	✓
troca de um sistema existente	✓	✓		✓	✓	✓
melhoria de um sistema existente	✓	✓	✓		✓	✓
aplicação nova	✓	✓	✓		✓	✓
pacote de software	✓	✓		✓	✓	
sistemas embarcados	✓	✓				✓
partes interessadas geograficamente distribuídas	✓	✓		✓		

Para levantar requisitos é necessária a interação entre aqueles que conhecem os requisitos ou a necessidades dos usuários e outras partes interessadas e os desenvolvedores. É um processo interativo de comunicação, onde, por aproximações sucessivas, o desenvolvedor constrói um modelo aceito pelos usuários, como ilustrado na Figura 19.6.

O mesmo acontece na investigação que um analista tem que fazer com o cliente em busca do requisito. Inicialmente o cliente será ambíguo, generalista e paulatinamente,

com a ajuda do analista, deverá chegar a uma especificação precisa do que deseja.

Uma receita geral para o levantamento de requisitos pode ser dada em 5 passos:

1. Identificar quem fornecerá os requisitos
2. Levantar lista de desejos para estas pessoas
3. Refinar cada lista de desejos até que seja auto-consistente
4. Integrar as listas de desejos de forma que sejam consistentes
5. Determinar os requisitos não funcionais.

Apesar de tudo, não é uma tarefa fácil levantar os requisitos. Muitos problemas podem acontecer. É comum que os clientes não saibam exatamente o que querem ou não saibam articular propriamente suas ideias, especialmente quando o desenvolvedor não possui o linguajar típico da área de aplicação (jargão). Muitas vezes, também, os desenvolvedores pensam entender melhor do problema que seus clientes, propondo supostas “melhorias” que afetam custo e aumento o risco dos sistemas propostos.

19.6.1. Processo de Elicitação de Requisitos

Dentro do Processo de Desenvolvimento de Software existe um Processo de Elicitação de Requisitos. Christel e Kand (1992) sugeriu o processo descrito na Figura 19.7 e na Tabela 19.2:

19. Requisitos

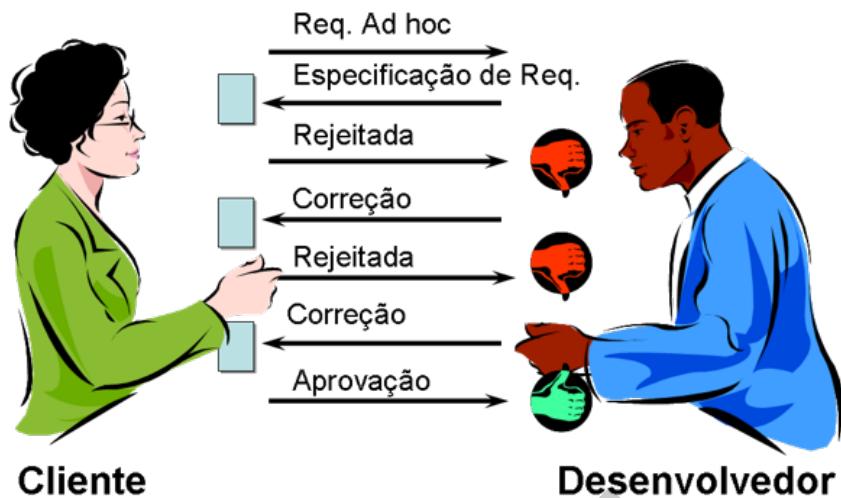


Figura 19.6.: A elicitação de requisitos é um processo interativo onde, por aproximações sucessivas, o desenvolvedor consegue a aprovação de uma especificação de requisitos.

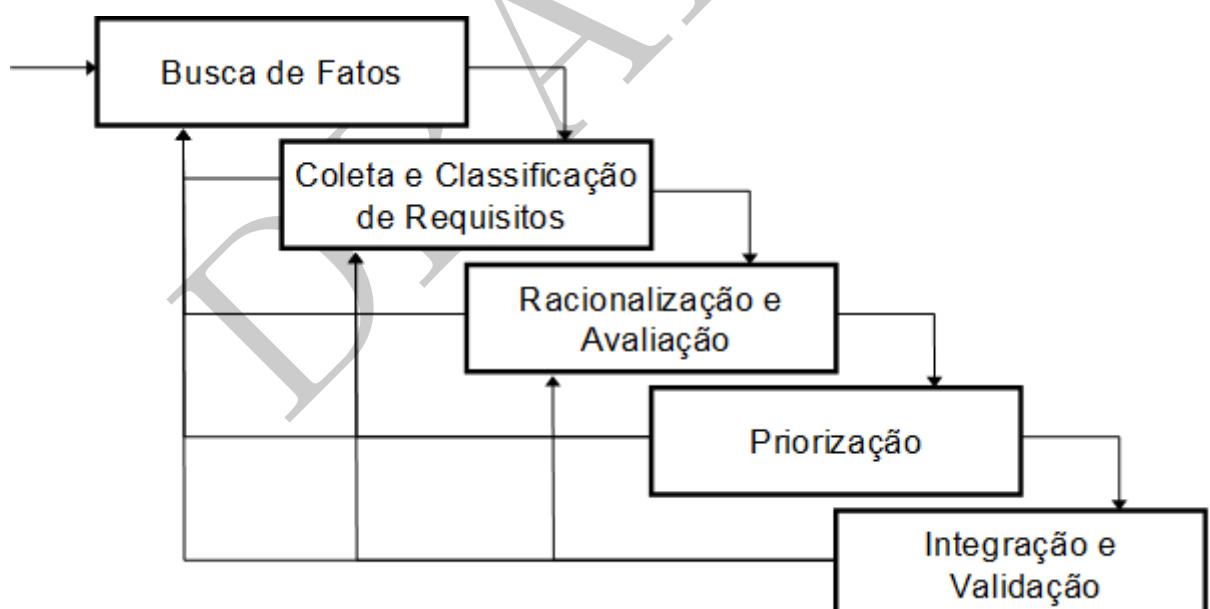


Figura 19.7.: O processo de elicitação de requisitos, adaptado de Christel e Kand (1992).

Tabela 19.2.: Processo de Elicitação de Requisitos segundo Christel e Kand (1992).

Atividade	Tarefas da Elicitação de Requisitos	
	Tarefas orientadas ao usuário	Tarefas orientadas ao desenvolvedor
Busca de Fatos	<ul style="list-style-type: none"> Identificar grupos relevantes através de múltiplos níveis da organização. Determinar os contextos operacional e do problema, incluindo a definição dos modos operacionais, objetivos e cenários de missão como apropriados. Identificar sistemas similares. Realizar análise de contexto. 	<ul style="list-style-type: none"> Identificar especialistas do domínio da aplicação e de desenvolvimento. Identificar modelo de domínio e modelo de arquitetura. Conduzir pesquisas tecnológicas, para mais tarde fazer estudo de viabilidade e análise de risco. Identificar custos e restrições à implementação impostas pelo patrocinador.
Coleta e Classificação dos Requisitos	Levantar a lista de desejos de cada grupo.	Classificar a lista de acordo com funcionais, não funcionais, restrições de ambiente, restrições de projeto e ainda de acordo com as partições definidas pelo modelo de domínio e pelo paradigma de desenvolvimento.
Racionalização e Avaliação	Responder questões da forma Por que você precisa de X?, a partir de raciocínio abstrato. Isso auxilia a transformar o raciocínio das questões sobre como? para as questões sobre o quê?. Determinar funcionalidades críticas para a missão.	Realizar uma análise de riscos, investigando técnicas, custos, prazos e incluindo análise de custos e benefícios e viabilidade baseado na disponibilidade da tecnologia.
Priorização		Priorizar requisitos baseados em custo e dependência. Estudar como o sistema pode ser implementado de forma incremental, investigando modelos arquiteturais apropriados.
Integração e Validação	<ul style="list-style-type: none"> Resolver a maior quantidade possível de pontos em aberto. Validar que os requisitos estão concordando com os objetivos iniciais. Obter autorização e verificação para passar ao próximo passo de desenvolvimento (e.g. a demonstração e a validação). 	Resolver conflitos e verificar consistência.

19.7. Níveis de Abstração dos Requisitos

Qualquer que seja o sistema, existem várias formas de entendê-lo. Similarmente, existem vários níveis de abstração de requisitos em que eles podem ser definidos, dependendo da visão necessária naquele instante. Alguns níveis de abstração reconhecidos para o início do projeto são(K. E. Wiegers e Beatty, 2013), na sequência em que normalmente aparecem:

1. **requisito do negócio** descrição de alto nível do objetivo do projeto que justifica a necessidade do projeto para o negócio, com visão dirigida para os objetivos do negócio;
2. **requisito do usuário** é algum comportamento ou característica que o usuário deseja do software ou o sistema como um todo; o que o usuário quer. São escritos pelo próprio usuário ou levantados por um analista de sistemas que consulta o usuário.
3. **requisito do sistema** é algum comportamento ou característica exigido do sistema como um todo, incluindo hardware e software. O comportamento desejado do sistema. São normalmente levantados por engenheiros ou analistas de sistemas, refinando os requisitos dos usuários e os transformando em termos de engenharia.
4. **requisito do software** é algum comportamento ou característica que é exigido do software. São normalmente levantados por analistas de sistemas.O objetivo da análise é capturar todos os requisitos para o software sendo desenvolvido e apenas aqueles requisitos verdadeiros.

19.8. O Risco de Requisitos Falsos

Uma especificação de software deve conter todos os requisitos e apenas requisitos verdadeiros(McMenamin e Palmer, 1984).

É óbvio que a falta de requisitos implica que o sistema não atenderá seus clientes, não atingindo o valor desejado. Já o requisito falso, ou seja, aquele que não é necessário para que o sistema cumpra sua finalidade(McMenamin e Palmer, 1984), implica em custos e riscos maiores ao projeto.

Segundo McMenamin e Palmer (1984) há dois tipos de requisitos falsos em uma especificação: os tecnológicos e os arbitrários.

Uma das formas do requisito falso tecnológico acontecer é pela transmissão de uma tecnologia de um sistema já existente para outro, quando ela não devia mais existir. Por exemplo, um sistema antigo que precisa avisar os usuários que algo aconteceu pode enviar avisos por *pager*, fax, SMS e correio eletrônico. Em uma nova versão, se adiciona o envio por *Whatsapp* e *Messenger*, mas não se remove o envio por *pager* e fax, tecnologia que não são mais usadas.

A segunda forma é pelo especificação antecipada de um requisito tecnológico. Requisitos

tecnológicos falsos tendem a levar o sistema novo em uma direção de desenvolvimento que pode não ser a mais adequada(McMenamin e Palmer, 1984). Um exemplo seria em um projeto de longo prazo especificar já no início do projeto que deveria ser usado um *framework* específico de software, sem avaliar se o *framework* atende todas as capacidades exigidas. Ao longo do projeto, por exemplo, pode ser descoberto que é necessário que o software seja um web site responsivo, tecnologia que não necessariamente estaria coberta pelo *framework*.

Os requisitos falsos arbitrários aparecem quando o analista inclui um requisito por vontade própria ou por causa do uso de uma técnica de modelagem(McMenamin e Palmer, 1984).

Requisitos falsos tem dois efeitos no desenvolvimento de um projeto de software: eles podem esconder os requisitos verdadeiros e eles aumentam o tamanho do sistema, gerando custos e riscos maiores.

19.9. Características de um Bom Requisito

Um requisito, ou melhor, seu registro, deve ter as seguintes características (IEEE, 2018):

necessário, significando que, se retirado, haverá uma deficiência no sistema, isto é, ele não atenderá plenamente as expectativas do usuário;

- em especial, não devem existir requisitos do tipo seria interessante ter², ou o requisito é necessário ou é dispensável;
- deve ser levado em conta que cada requisito aumenta a complexidade, o custo e o risco do projeto, logo, não podem ser introduzidos de forma espúria;

apropriado, a intenção e a quantidade de detalhe é apropriada ao nível de abstração da especificação e a entidade a que se refere;

não-ambíguo, possuindo uma e apenas uma interpretação, sendo muito importante prestar atenção na linguagem sendo utilizada;

completo, ou seja, não precisa ser explicado ou aumentado, garantindo capacidade suficiente do sistema;

consistente, não contradizendo ou mesmo duplicando outro requisito e utilizando os termos da mesma forma que outros requisitos;

singular, ou seja, sendo apenas um requisito e não uma composição de requisitos;

alcançável, ou seja, realizável a um custo definido por uma ou mais partes interessadas no sistema ;

rastreável, de forma que sua fonte ou origem possa ser encontrada;

verificável, não sendo vago ou geral e sendo quantificado de uma maneira que permita a verificação na forma de inspeção, análise, demonstração ou teste, e

correto, sendo uma representação adequada, e

conforme, de acordo com um padrão e estilo aprovados, quando aplicável.

²Isso está em desacordo com a proposta de priorização da técnica MoSCoW

19. Requisitos

Além desses fatores, quando um requisito for funcional, deverá ser também **independente da implementação**, ou seja, o requisito define o que deve ser feito, mas não como.

Um requisito mal elaborado pode ser transformado em um requisito por meio de seu detalhamento. Por exemplo, o requisito “O sistema deverá responder rapidamente” tem grande ambiguidade. O que é rapidamente? Em que contexto deve ser entendido? Esse requisito pode ser melhorado para “O tempo de resposta do sistema deverá ser de 2 segundos”. Esta forma, apesar de mais detalhada, gera uma dúvida: se o sistema for mais rápido do que isso, o tempo de resposta deve ser aumentado até chegar a 2 segundos? Queremos exatamente dois segundos? Podemos melhorar então para “O tempo de resposta do sistema deverá ser de no máximo de 2 segundos”. Já está bem melhor, mas isso se aplica a todas as partes do sistema? Alguns relatórios, por exemplo, não podem tomar mais tempo. O requisito é alcançável? Então podemos melhorar para dois requisitos “O sistema deverá buscar manter o tempo de resposta menor do que 2 segundos em todas suas funcionalidades” e “O sistema deverá avisar ao usuário de qualquer operação cujo tempo de resposta previsto for maior do que 2 segundos”.

19.9.1. Efeitos de Requisitos com Problemas

A importância de obter requisitos corretos pode ser compreendida rapidamente se imaginarmos que um requisito, quanto mais básico e importante ele é, mais partes do sistema vai afetar.

Basicamente, cada requisito do negócio vai gerar um ou mais requisitos de usuário, e cada requisito do usuário vai gerar um ou mais requisitos do sistema, que vão, por sua vez, gerar outros requisitos com mais detalhes, e isso vai se repetindo a cada fase do projeto de software. O efeito, com a evolução do projeto, é que cada requisito inicial representa não só um compromisso, mas um fator crítico em uma sequência de decisões que são tomadas ao longo do projeto e influenciam várias de suas partes.

Por exemplo, se um requisito for “O sistema deverá enviar uma mensagem de felicitações no dia do aniversário do usuário”, será necessário ter um campo com a data de aniversário no banco de dados, campos em telas que permitem entrar e corrigir a data de aniversário, uma função para mandar a mensagem, possivelmente uma tela, ou campo em uma tela existente, para cadastrar essa informação, etc. Várias partes do código então se referem a um só requisito.

Assim, os erros de requisito afetam drasticamente todo o projeto. Dados empíricos indicam que um problema nos requisitos, se detectado nas fases finais do projeto, pode custar 100 vezes mais para ser corrigido do que se for detectado na fase de elicitação de requisitos. Mesmo que essa detecção se faça mais cedo, em uma fase intermediária, esse custo ainda seria 10 vezes maior.

Um requisito que não caiba em uma das características de qualidade citadas será propenso a defeitos e aumentará o risco do projeto.

Se não for necessário, gastaremos esforços em sua definição e implementação que seriam mais bem gastos tratando de outros requisitos. Além disso, cada requisito adicional aumenta o risco do projeto como um todo, ainda mais se a adição for dispensável.

Se for ambíguo, corremos alto risco de implementá-lo errado, desagradando o cliente.

Se não for verificável, não poderemos testar de maneira inequívoca se está corretamente atendido, possibilitando discussões sobre sua realização.

Se não for completo, faltará ao projeto alguma coisa que o usuário necessita. Se não for consistente, o projeto chegará a encruzilhadas ou talvez se torne impossível.

19.10. Atributos dos Requisitos

A norma IEEE 29148 diz que requisitos bem formados devem ter incluir um conjunto de atributos descritivos que permitam entendê-los e gerenciá-los, e dá como exemplo(IEEE, 2018):

- identificação;
- versão;
- proprietário, isto é, quem mantém o atributo;
- prioridade para as partes interessadas;
- justificativa;
- risco;
- dificuldade, e
- tipo.

Veremos que a maioria desses atributos são normalmente informados nos métodos atuais, como histórias do usuário ou casos de uso, porém normalmente o risco³ não é tratado requisito por requisito, sendo normalmente resultado de outra atividade de gerência de projeto ou análise de sistemas.

19.10.1. Volere e o snowcard

O Cartão Volere ou *snowcard* (J. Robertson e S. Robertson, 2006) é uma forma de registrar requisitos de software que atende parcialmente esta especificação, foi apresentado na Figura 19.1. O registro de requisitos em cartões possibilita que realização de algumas atividades interativas sejam realizadas com facilidade, como a ordenação por prioridade. O cartão apresentado na Figura 19.1, contém os seguintes campos:

- **número identificador**, o para facilitar a discussão, identificamos todos os requisitos unicamente;
- **tipo**, classificando-o como funcional, não funcional, ou outra classificação mais detalhada usada na organização;

³Lembramos que o risco aparece no Project Model Canvas, no capítulo 14

19. Requisitos

- evento, história do usuário, caso de uso, etc. que o atende;
- **descrição**, uma sentença que descreve o requisito;
- **justificativa**;
- **donte do requisito**, a pessoa, grupo, organização, documento ou outra que o origem para o requisito;
- **critério de aceitação**, uma medida que possa ser usada para garantir que o requisito foi alcançado;
- **satisfação do usuário**, um grau, de 1 (nenhum interesse) a 5 (extremamente satisfeito), por exemplo, indicando a satisfação do cliente se esse requisito for alcançado;
- **insatisfação do usuário**, um grau, de 1 (nenhum interesse) a 5 (extremamente insatisfeito), por exemplo, indicando a satisfação do cliente se esse requisito não for alcançado;
- **dependências**, referências a outros requisitos que dependem de alguma forma desse requisito;
- **conflitos**, referência aos requisitos que de alguma forma conflitam com esse
- **material de apoio**, listagem de material de apoio para atender esse requisito, e
- **histórico**, documentação da criação e das mudanças efetuadas.

Um diferencial do método Volere é que ele associa o valor a duas perguntas: a satisfação com a implementação do requisito e a insatisfação com sua ausência no produto. Isso permite classificar cada requisito em um de quatro quadrantes, de acordo com a nota que recebem: crítico, decoração, conflituoso ou dispensável (não fazer). A Figura 19.8 demonstra essa ideia.



Figura 19.8.: Cada requisito se posicionará em algum ponto desse gráfico, de acordo com as notas de avaliação.

19.11. Priorizando Requisitos

Existe uma tendência grande de o sistema crescer muito durante a análise. Principalmente se entrevistamos um grande número de pessoas, existe uma facilidade natural que elas têm para propor novas funcionalidades para um sistema que ainda não existe, por imaginarem alguma utilidade nessas funções propostas. Assim, muitas vezes nos vemos envolvidos com uma quantidade de requisitos tão grande que é óbvio que o sistema a ser feito não poderá ser entregue no prazo ou pelo custo combinado, ou que se pensava em combinar.

Nesse caso, algumas técnicas podem ser utilizadas para caracterizar o que deve ser realmente feito ou, pelo menos, em que ordem as coisas devem ser feitas. A priorização de requisitos é atividade essencial em qualquer projeto e natural em métodos ágeis que usam o *product backlog*.

A primeira técnica disponível é associar a cada requisito do sistema uma importância. Uma escala de três ou cinco valores é adequada para isso, como em: Imprescindível para o sucesso do sistema, Funcionalidade Importante, mas podemos esperar algum tempo, Ajudaria ter, mas é possível viver sem essa funcionalidade, Benefícios mínimos, Desnecessário.

A segunda técnica disponível é planejar o sistema para ser entregue em várias **versões**, mesmo que nem todas as versões estejam incluídas nesse contrato, e pedir para o cliente determinar que funcionalidades devem aparecer em cada versão. Nesse caso pode ser interessante dar um peso ou custo para cada requisito, de modo que o cliente possa controlar seus gastos.

Uma terceira técnica disponível é dar uma **moeda virtual** para o cliente, por exemplo, 1000 dinheiros, e pedir para ele distribuir quanto pagaria por cada função, priorizando no desenvolvimento aquelas funções que o cliente decidir pagar mais.

Uma quarta técnica é conhecida como **MoSCoW**, onde cada item proposto é associado a um conceito do grupo “*Must Have*”, precisa ter, “*Should Have*”, devia ter, “*Could Have*”, podia ter e “*Won't Have*”, não vai ter (IIBA, 2011), criando grupos com significados específicos;

A técnica **Volere** também separa os requisitos dentro de seu quadro de importância, como visto na Figura 19.8.

Todas essas técnicas, porém, ficam dependentes de uma outra priorização importante dos requisitos: a priorização por dependência.

Devem ser levados em conta os vários fatores que influenciam nessa determinação de prioridades, entre eles os citados por (Volare, 2020):

- Diminuir o custo da implementação
- Valor para o comprador
- Tempo para implementar
- Facilidade técnica de implementar

19. Requisitos

- Facilidade do negócio para implementar
- Valor para o negócio
- Obrigação por alguma autoridade externa

19.12. Requisitos Mudam com o Tempo

Também é importante perceber que os requisitos de um software mudam com o tempo. Essas mudanças ocorrem porque o ambiente em que o software reside também muda. Os países alteram suas leis, as organizações alteram suas práticas, a tecnologia evolui, os usuários exigem novas funcionalidades até então não imaginadas ou desnecessárias.

Jones (2005), afirmou que a taxa de mudanças de requisitos para um software comercial em desenvolvimento chega a 3,5% ao mês, e para um software para Web, chega a 4,0% ao mês. Considerando o tempo médio de execução desses e outros tipos projetos, ele chega a valores médios de 2,58% de mudanças ao mês, 14 meses em média de execução e 32,33% de mudança total nos requisitos. É importante notar, porém, que todas essas afirmações são baseadas em cálculos com Pontos de Função para manutenção, onde pequenas alterações podem causar grandes efeitos. Além disso, modificações ao longo do projeto podem representar não uma mudança do requisito verdadeiro, mas sim uma compreensão melhor do que é esse requisito. Mesmo assim, uma mudança corretiva em um requisito tem o mesmo impacto de uma mudança por novas necessidades.

Esse fato tem efeitos em nossa prática. O primeiro é que devemos estar preparados para a mudança, pois ela vai acontecer. O segundo é que quanto maior o tempo de duração do projeto, maior a quantidade de mudanças de requisitos, o que aumenta ainda mais o risco, que já é afetado pelo projeto ser longo e provavelmente complexo.

Assim, não só é importante conhecer os requisitos, mas também conhecer qual a sua estabilidade. Requisitos pouco estáveis devem ser tratados de forma diferente dos requisitos mais estáveis. Requisitos mais críticos devem ser tratados de forma diferente dos requisitos opcionais. Isso não tem relação com o software apenas, mas sim com atender o negócio.

19.13. Requisitos e Necessidades

Requisitos são originários de necessidades das partes interessadas. Enquanto os requisitos vivem no mundo das soluções, as necessidades vivem no mundo dos problemas. Os requisitos indicam as características que devem ser observadas no sistema, que atendem as necessidades.

Voltando ao discutido no Capítulo 2, as necessidades causam desejos, que por sua vez causam demandas. Os desejos são influenciados pelo ambiente, as demandas pela capacidade financeira do comprador. Assim, se uma organização tem como necessidade

tornar mais rápido seu processo de vendas, baixando de 15 minutos para 3 minutos o tempo de concluir uma venda, como estratégia de negócios ela pode desejar desde um aplicativo para celular para seus vendedores usarem na loja, como faz a Apple, ou uma loja toda automatizada onde o que você coloca no carrinho, ou tira do carrinho, é identificado por câmeras e posto, ou removido, automaticamente de sua conta, o que acontece no mercado da Amazon. Porém, o que ela pode pagar? Pode ser que apenas uma nova versão do seu software de caixa registradora, de forma que ele seja mais rápido. Os requisitos virão dessa demanda.

Como é visto no Capítulo 13, necessidades podem ser provenientes de problemas, e um problema é uma diferença, vista com uma perspectiva comportamental, entre uma situação percebida e uma situação desejada. No momento, nos basta a noção que o problema incomoda o usuário (ou a parte interessada) ao ponto dele considerar necessário investir alguma quantia ou esforço de forma a evitar que o problema aconteça.

É comum ouvirmos o ditado “Em time que está ganhando, não se mexe”. Quando somos chamados para desenvolver um sistema, devemos imaginar imediatamente que, se alguém deseja mexer em sua organização, então é porque “não está ganhando”, pelo menos da maneira que supõe possível. Faz pouco sentido imaginar que alguém iria fazer um investimento de risco, e sempre há risco no desenvolvimento de software, sem que haja uma necessidade clara. Se isso for identificado, talvez seja mais adequado cancelar o projeto. Não se deve esquecer o adágio contrário: “em time que está vencendo que se mexe”. Essa frase representa a ideia que para continuar “ganhando” é necessário estar em constante evolução, e isso pode ser também uma motivação importante para um projeto de software.

Muitas vezes o analista se depara com a necessidade de, antes de definir requisitos, definir propriamente qual o problema a ser tratado. Para isso, deve fazer com que os clientes cheguem a um acordo sobre qual é o verdadeiro problema, ou o problema mais importante. Faz pouco sentido construir um sistema se o verdadeiro problema de negócios não for endereçado pela solução planejada. Isso é tratado no Capítulo 13

Um sistema pequeno certamente produzirá a solução para uma pequena quantidade de problemas, um sistema grande certamente atingirá uma grande quantidade de problemas. O ponto em questão é que deve existir alguma não conformidade, seja ela atual ou futura, para valer a pena o investimento, e o risco, de tentar um sistema novo.

19.14. Requisitos de Acordo com Normas IEEE

A IEEE tem um conjunto de normas que tratam de requisitos. Esta seção trata de alguns dos assuntos levantados por essas normas.

Segundo a norma IEEE 24765:2010 (IEEE, 2018) requisitos devem dizer o que (*what*) é desejado, e não como (*how*), sem incluir nenhuma decisão de projeto nos seus níveis mais abstratos.

19. Requisitos

A linguagem da especificação de requisitos deve evitar termos gerais e abstratos, ou que geram requisitos difíceis de verificar. Devem ser evitados(IEEE, 2018):

- adjetivos e superlativos, como “bom” ou “melhor”;
- linguagem subjetiva, como “amigável ao usuário” ou “custo eficiente”;
- pronomes vagos, como “aquele”;
- termos ambíguos como advérbios e adjetivos, como “quase sempre” ou “mínimo”;
- termos lógicos que em português podem ser confusos, como “e” ou “ou”;
- termos não verificáveis, como “fornece suporte”;
- frases comparativas ambíguas, como “melhor que” ou “de melhor qualidade”;
- termos que não são um compromisso definitivo, como “se possível”
- termos que implicam totalidade, como “nunca” ou “sempre”, pois são difíceis ou impossíveis de verificar, e
- referências incompletas, como falta de uma versão ao especificar um software a ser usado.

Ainda segundo a norma IEEE 29148 (IEEE, 2018) é comum estipular o significado uso de palavras que definem obrigatoriedade, futuro ou proibição⁴. A norma ainda diz que requisitos ágeis, como histórias do usuário, não precisam seguir essa regra.

A norma IEEE 29148 (IEEE, 2018) apresenta uma sintaxe para os requisitos com a forma:

[condição][sujeito][ação][objeto][restrição]

onde:

- **condição** é uma regra para ativação do requisito, seja ela o acontecimento de um evento ou um fato que se torna realidade;
- **sujeito** descreve um ator, normalmente o próprio sistema sendo especificado;
- **ação** descreve a ação principal do requisito;
- **objeto** descreve o objeto da ação, e
- **restrição** limita a forma como a ação funciona.

As sintaxes de exemplo apresentadas para requisitos na norma IEEE 29148 são(IEEE, 2018) são:

- [sujeito] [ação] [objeto] [restrição a ação], como em “O sistema acadêmico [sujeito] deve calcular [ação] a aprovação de cada aluno [objeto] segundo as regras do curso específico [restrição]”, ou
- [condição] [sujeito] [ação] [objeto] [restrição a ação], como em “Ao fim do período [condição] o sistema acadêmico [sujeito] deve calcular [ação] a aprovação de cada aluno [objeto] segundo as regras do curso específico [restrição]”.

⁴Em inglês são usadas palavras como “*shall*”, “*must*”, “*should*” e “*shall not*”.

19.14.1. Requisitos Funcionais na Norma IEEE

Segundo a norma IEEE 830:1998 (IEEE, 1998) os requisitos funcionais devem “definir as ações fundamentais que devem acontecer no software ao aceitar e processar as entradas e no processamento e geração das saídas”. Normalmente, devem ser listados com uma sentença que se inicia com “O sistema deve”⁵. No contexto dessa norma, os requisitos funcionais incluem(IEEE, 1998):

- validações de entradas;
- sequência exata de operações;
- respostas a situações anormais, como sobrecarga, falhas de comunicação e tratamento e recuperação de erros;
- efeitos dos parâmetros, e
- relacionamentos das saídas com as entradas, incluindo sequências de entradas e saídas e fórmulas de conversão de entradas para saídas.

Exemplos de requisitos funcionais seguindo essa norma são:

- o sistema acadêmico deve permitir que um aluno solicite seu boletim;
- o sistema acadêmico deve permitir que um professor dê as notas dos alunos para as turmas em que está alocado, e
- o sistema acadêmico deve permitir a um coordenador incluir um aluno em uma turma apesar de ser contra as regras de inclusão.

Para evitar confusões entre um sistema funcionando hoje em dia e um sistema sendo especificado para o futuro, é possível colocar o verbo deve no futuro, ficando o início do requisito “O sistema deverá”.

19.14.2. O Documento de Especificação de Requisitos

Uma especificação de requisitos de sistema é um documento que reúne os requisitos definidos e aceitos por todos os interessados naquele sistema.

Os assuntos que devem ser tratados em uma especificação de requisitos são, segundo a norma IEEE ISO/IEC/IEEE International Standard - *Systems and software engineering – Life cycle processes – Requirements engineering* (IEEE, 1998) são:

- funcionalidade, ou o que o software tem que fazer?
- interfaces externas, com quem ou com que sistemas o software interage?
- desempenho, qual a velocidade, disponibilidade, tempo de resposta das várias funções, etc.?
- atributos (de qualidade), quais são as considerações de portabilidade, correção, manutenibilidade, segurança, etc.?
- restrições de projeto impostas a implementação, como padrões a serem seguidos, políticas de integridade da base de dados, limites dos recursos disponíveis, ambiente

⁵Em inglês, “The system shall”

19. Requisitos

de operação, etc.?

Segundo o mesmo padrão, um sumário adequado a uma especificação de requisitos seria(IEEE, 2018):

1. Introdução
 - a) Objetivo do Sistema
 - b) Escopo do Sistema
 - c) Visão Geral do Sistema
 - i. Contexto do Sistema
 - ii. Funções do Sistemas
 - iii. Características dos Usuários
 - d) Definições
2. Referencias
3. Requisitos do Sistema
 - a) Requisitos Funcionais
 - b) Requisitos de Usabilidade
 - c) Requisitos de Desempenho
 - d) Requisitos de Interface
 - i. Requisitos de Interface Externa
 - ii. Requisitos de Interface Interna
 - e) Operações do Sistema
 - f) Modos e Estados do Sistema
 - g) Características Físicas
 - h) Condições Ambientais
 - i) Requisitos de Segurança
 - j) Requisitos de Gerência de Informação
 - k) Requisitos Regulatórios e Legais
 - l) Requisitos de Sustentação do Ciclo de Vida do Sistema
 - m) Requisitos de Empacotamento, Trato, Envio e Transporte
4. Verificação (paralelo as subseções da seção 3)
5. Apêndices
 - a) Suposições e Dependências
 - b) Acrônimos e Abreviações

19.15. A Visão do Novo Sistema

Dar uma visão do sistema sendo proposto é uma maneira de definir requisitos de altíssimo nível com facilidade de entendimento para as partes interessadas.

A Visão do Sistemas é uma descrição de como vai funcionar o novo sistema em uma descrição simples, em linguagem corrente. Essa visão deve ser escrita com forte apoio do cliente, senão pelo próprio cliente.

Possivelmente, essa visão é apresentada junto com uma visão do sistema atual, com

efeito comparativo, sendo ambas no mesmo nível de abstração e dentro de um mesmo documento inicial do projeto.

A visão pode incluir o protótipo de algumas telas do novo sistema, com a finalidade de mostrar a diferença do sistema novo para o velho ou ainda mostrar como será o comportamento de uma nova finalidade.

A visão do sistema pode incluir não só o funcionamento do sistema, mas também expectativas de comportamento e de efeitos do sistema no negócio. Deve ficar claro que a visão do sistema é uma declaração do usuário, não necessariamente um comprometimento do desenvolvedor.

A visão do sistema inclui também os requisitos já detectados, informalmente, pelo analista. Esses requisitos podem ser divididos em requisitos funcionais, requisitos de informação e requisitos não funcionais. Obviamente só estamos interessados em requisitos verdadeiros.

19.16. Descrevendo Requisitos

Normalmente as especificações de requisitos são escritas em linguagem natural (inglês ou português, por exemplo). O problema é que a forma como falamos e normalmente escrevemos é bastante ambígua. Isso exige que adotemos algumas técnicas básicas, principalmente um formato padronizado, um estilo de linguagem e uma organização que facilite a manipulação do conjunto de requisitos.

Algumas dicas para escrever requisitos são (K. Wiegers, 1999):

- use sentenças e parágrafos curtos;
- use a voz ativa;
- use verbos no futuro;
- use os termos de forma consistente e mantenha um glossário;
- para cada requisito, avalie se a partir de sua definição é possível determinar se ele está pronto ou não;
- garanta que todos os requisitos são verificáveis imaginando (e possivelmente documentando) uma forma de fazê-lo;
- verifique requisitos agregados (termos como e e ou são uma boa indicação) e divida-os, e
- mantenha um nível de detalhe único em todos os requisitos.

19.16.1. Exemplos de Requisitos

A seguir listamos alguns exemplos de requisitos bem descritos, seguindo o estilo de exemplos originais de Karl Wiegers:

- O sistema deverá permitir que um paciente marque uma consulta.

19. Requisitos

- O sistema deverá confirmar que a consulta foi aceita pelo paciente.
- O sistema deverá permitir que um condômino solicite a segunda via de sua conta condominal.
- O sistema deverá permitir um aviso ao condômino que não pagar sua no prazo correto.
- O sistema deverá permitir que um fornecedor cadastre um produto no catálogo.
- O sistema deverá informar ao sistema de estoques que um produto foi vendido.

Todas as sentenças acima usam, na verdade, uma expressão do tipo deverá, tradução do inglês shall, que é uma forma tradicional do americano definir uma ordem. Talvez as seguintes descrições sejam mais adequadas a nossa língua:

- O sistema permitirá que um paciente marque uma consulta.
- O sistema confirmará ao paciente que a consulta foi marcada.
- O sistema permitirá que um condômino solicite a segunda via de sua conta condominal.
- O sistema permitirá um aviso ao condômino que não pagar sua no prazo correto.
- O sistema permitirá que um fornecedor cadastre um produto no catálogo.
- O sistema informará ao sistema de estoques que um produto foi vendido.

Também devemos notar que as entradas do usuário são descritas como deverá permitir. Alguns autores, como o próprio Wiegers, admitem que uma especificação de requisito exija algo do usuário, como em:

- O paciente deverá especificar qual a especialidade médica que deseja.

Essa abordagem não é adequada, pois está fazendo uma exigência ao usuário e não ao sistema. Uma abordagem mais apropriada seria:

- O sistema exigirá que o usuário especifique a especialidade médica que deseja.

Desta forma deixamos claro que:

1. O requisito é do sistema
2. Cabe ao sistema exigir do usuário a resposta
3. Não cabe ao usuário saber o que fazer, mas sim ao sistema saber o que o usuário tem que fazer.
4. A especificação de requisitos do software não requer nada ao usuário.

Em conclusão, uma especificação de requisitos só deve exigir funcionalidade do sistema sendo definido. Entendemos que muitas vezes um sistema faz exigências ao ambiente. Essas exigências podem algo como o usuário deve falar inglês ou o computador deve ser do compatível com chips XYZ/2001. Todas as exigências que o sistema faz devem ser documentadas. Uma localização específica do documento para isso é a seção de suposições ou dependências, mas demandas que o sistema faz não podem ser vistas realmente como um requisito do sistema, apesar de possivelmente definir características do sistema.

19.17. Exercício

Exercício 19.17-1: Escreva um conjunto de requisitos funcionais na forma de “O sistema deve...” para um aplicativo de celular que sirva como relógio e despertador, permitindo vários alarmes.

Exercício 19.17-2: Procure na internet os seguintes documentos:

- *Software Requirements Specification for Cafeteria Ordering System prepared by Karl Wiegers v1.0 de 2002*, e
- *Cafeteria Ordering System Vision and Scope Document prepared by Karl Wiegers v1.0 de 2002*.

Preste atenção que o primeiro documento tem duas versões na rede, uma com casos de uso, outra com especificações em linguagem natural. Baixe os dois. Analise os três documentos e guarde-os para exercícios futuros.

Exercício 19.17-3: Baseado no texto *Cafeteria Ordering System Vision and Scope Document prepared by Karl Wiegers v1.0 de 2002*, feito com linguagem Natural, escreva um documento semelhante para o sistema descrito no Capítulo 14

Exercício 19.17-4: Baseado no texto *Software Requirements Specification for Cafeteria Ordering System prepared by Karl Wiegers v1.0 de 2002*, feito com linguagem Natural, escreva um documento semelhante para o sistema descrito no Capítulo 14

Exercício 19.17-5: Vá para o site <http://jogodeanalisedesistemas.xexeo.net/> e visite a Livraria Resolve. A partir da sua visita especifique no formato IEEE todos os requisitos que você encontrar.

Exercício 19.17-6: Identifique todos os problemas na sua lista de requisitos para a Livraria Resolve. Defina perguntas no formato 5W2H para resolver esses problemas.

DRAFT

20

Histórias do Usuário

Since users don't know how to solve their problems, we need to stop asking ... and to involve them instead.

(Mike Cohn)

Conteúdo

20.1.	Conceituação de Histórias do Usuário	259
20.2.	Um <i>Template</i> Padrão para Histórias de Usuário	260
20.3.	INVEST	262
20.4.	Estados de Uma História do Usuário	266
20.5.	Exercícios	266

Por que histórias do usuário?

Junto com casos de uso, histórias do usuário são uma das duas principais formas de especificar os requisitos de um software sendo usadas atualmente, principalmente em projetos ágeis.

Histórias do usuário são uma prática ágil de registro das necessidades funcionais das partes interessadas, normalmente escritas em um cartão. O nome tem certa ambiguidade, sendo usado tanto para a prática como um todo quanto para o texto principal escrito no cartão. Em métodos ágeis como *eXtreme Programming (XP)* e *Scrum* elas são a principal forma de representação de um **requisito de software**,

20. Histórias do Usuário

O termo **história do usuário** foi cunhado por Kent Beck (Beck, Cockburn et al., 2014), com a intenção de descrever uma prática com maior equilíbrio entre a área de negócio e a área de desenvolvimento, sendo uma maneira mais ágil de especificar requisitos. Um dos objetivos era que, para que os usuários tivessem acesso facilitado ao método, a linguagem deveria ser bem próxima a deles, evitando as formalidades e os jargões de TI dos métodos tradicionais.

Dessa forma, histórias do usuário são uma narrativa na perspectiva do usuário (Beck, Roden et al., 2014) escrita em linguagem natural. Segundo Adzic e Evans (2014), comparando histórias do usuário com requisitos levantados unicamente por desenvolvedores ou pessoas do negócio, “histórias do usuário implicam em um modelo totalmente diferente: requisitos por colaboração”.

Inicialmente histórias do usuário tinham uma definição muito informal, e até controversa (Beck, Cockburn et al., 2014; Beck, Roden et al., 2014). O entendimento e o acordo entre os praticantes dessa metodologia foram evoluindo com o tempo e a divulgação de seu uso, por meio de artigos e livros, e acabaram sendo de certa forma padronizadas no mercado em três conceitos principais que são tratados neste texto: o *template*, a sigla *INVEST* e a regra *Card, Conversation and Confirmation*.

Beck (1999) definiu histórias do usuário, dentro do contexto de *XP* como:

- uma coisa que o cliente quer que o sistema faça;
- que possa ser feita entre uma e cinco semanas, e
- que seja testável.

Nessa definição original, o prazo de uma a cinco semanas tem mais relação com o fato da história poder ser feita junto com outras nesse tempo. Alguns autores falam das histórias isoladamente como podendo ser feitas de meio dia até 2 semanas dias (Cohn, 2004). O prazo exato, porém, não é importante, mas sim o fato que uma história do usuário que é consistente com os prazos dos métodos ágeis, e com a prática de *time-box*.

Uma outra definição, dada por Jacobson, Lawson e Ng (2019) é “Alguma coisa que o sistema de software pode ser estendido para fazer, expressa em termos do valor que vai fornecer para o usuário do sistema.”

É importante compreender, porém, que **não há nenhuma limitação formal de como são escritas**, e o importante é que sejam manipuláveis e comprehensíveis pelos usuários. Além disso, apesar da semelhança entre os nomes, **histórias do usuário não são casos de uso**, ou uma versão simplificada de casos de uso (Beck, Cockburn et al., 2014; Cohn, 2004).

Tipicamente, a parte mais visível das histórias do usuário são escritas em cartões como o apresentado na Figura 20.1.¹

Como em vários métodos ágeis, é comum que os cartões fiquem afixados a um quadro, como o quadro *Kanban*, ou ainda no *Sprint Backlog* ou no *Product Backlog* do *Scrum*.

¹São usados normalmente fichas pautadas de fichário

Corrigir Nota	Alta prioridade
<i>Como um professor eu desejo alterar a nota de um aluno para poder corrigir um erro de digitação ou um pedido de revisão de prova</i>	
GX	31/1/2020
1 dia	

Figura 20.1.: Exemplo de uma história do usuário em um cartão. Fonte: do autor.

Por que cartões?

Vários métodos de análise de sistemas, como *Volare*, CRC e Histórias do Usuário usam cartões, ou fichas de fichário. Essa prática é comum nos EUA, mas encontra até um pouco de resistência no Brasil. A questão é que nos EUA cartões são usados em várias técnicas de estudo, como o uso de *flashcards*, ou o fichamento (que também é usado no Brasil), então o profissional normalmente já tem certo hábito com eles.

20.1. Conceituação de Histórias do Usuário

Uma história do usuário “descreve uma funcionalidade que terá valor” para uma parte interessada de um sistema ou software (Cohn, 2004). Ela é composta de três aspectos, conhecidos como Cartão, Conversação e Confirmação(Cohn, 2004; Jeffries, 2001):

1. o **cartão** que contém uma descrição textual da história **usada para planejamento e como um lembrete**, ou a história do usuário propriamente dita, como a que está na Figura 20.1;
2. as **conversações** entre desenvolvedores e usuários sobre a história, e que levam a descoberta de seus detalhes, e
3. as **confirmações**, as quais garantem que a história está completa e funcionando como esperado.

Segundo K. E. Wiegert e Beatty (2013, p. 146), conversações servem para refinar as histórias do usuário e também para levar dessas histórias refinadas para testes de aceitação. Essas conversações podem levar ao desmembramento da história original em histórias menores e a detalhes adicionais.

Devemos deixar claro que os cartões são apenas parte de usar histórias do

Card, Conversation and Confirmation

20. Histórias do Usuário

usuário como método, servindo como **ponto focal** do trabalho. A principal mudança, na verdade, é ter uma ambiente participativo, onde usuários e desenvolvedores tem encontros face a face onde se estabelecem as **conversações**. Possivelmente, essas conversações podem levar a alguns documentos que serão usados pelos desenvolvedores, mas não há nenhum padrão pré-concebido a seguir na documentação.

Isso causa uma pequena confusão entre o método e o artefato histórias do usuário. O método é composto pelos três aspectos, o artefato, ou a descrição da história do usuário, é registrado em um cartão. Deve ficar claro, pelo contexto, qual dos significados é tratado em cada parte do texto.

O cartão propriamente dito pode ter várias informações. A principal é a descrição da história, possivelmente na forma de um dos *templates* apresentados a seguir neste texto, ou um definido na organização.

Outra prática comum é **dar um título a história**. Adzic e Evans (2014) recomenda nomear as histórias bem cedo e pensar se outras partes interessadas podem estar interessadas na história, deixando os detalhes para o final.

A Figura 20.2 mostra um cartão adaptado do modelo usado originalmente na Connextra (Patton e Economy, 2014), com suas várias partes identificadas.

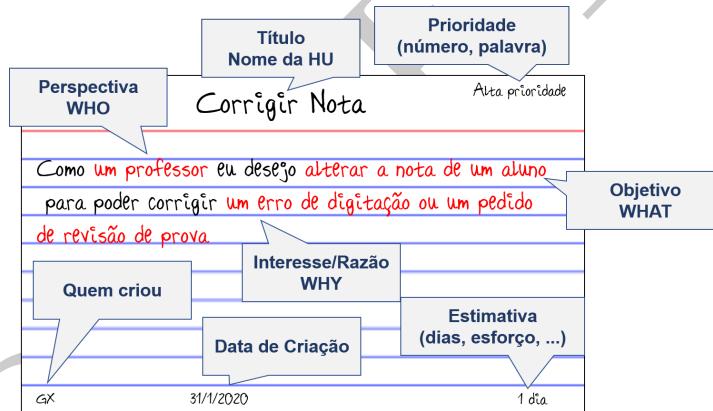


Figura 20.2.: Partes encontradas em um cartão de história do usuário, segundo um modelo original da Connextra (Patton e Economy, 2014).

20.2. Um *Template* Padrão para Histórias de Usuário

North (2013) sugeriu um formato para histórias de usuário em um grupo do Yahoo, conhecido como o *template* da Connextra (Adzic e Evans, 2014), que acabou se tornando uma referência na área:

User Story
Template

Como um <papel de usuário> eu deseo <objetivo> de forma a <razão>. onde o *objetivo* descreve uma funcionalidade do sistema e a *razão* descreve porque aquela

funcionalidade traz valor ao usuário. É **importante que a razão não seja apenas uma nova maneira de dizer o objetivo**, pois nesse caso o valor não está claro.

Exemplo de histórias do usuário que seguem esse padrão são:

- como um sócio do clube eu desejo marcar a quadra de futebol de forma a jogar futebol com amigos, e
- como um gerente eu desejo emitir o relatório de vendas por cliente de forma a descobrir quais são os clientes que mais gastam.

Adzic e Evans (2014) ainda recomenda que as histórias do usuário tenham duas propriedades: descrever uma mudança de comportamento, para as partes interessadas, e descrever uma mudança no sistema, para a equipe de desenvolvimento.

20.2.1. Outros formatos possíveis

O formato padrão responde 3 perguntas sobre o caso de uso: quem, o que e por que. É possível modificar esse formato para responder mais perguntas do 5W2H. Dois formatos possíveis são:

Como um *<papel de usuário>*, *<prazo>*, eu desejo *<objetivo>* de forma a *<razão>*.

o que responde ainda quando, como em “Como gerente, até o dia 30 de maio,eu quero o relatório de vendas por cliente ordenado por valor, porque quero conhecer meus melhores clientes.” ou ainda

Como um *<papel de usuário>*, *<prazo>*, eu desejo *<objetivo>* de forma a *<razão>*.

o que responde ainda “por quanto?”, como no exemplo “Como gerente, até o dia 30 de maio,eu quero o relatório de vendas por cliente ordenado por valor, porque quero conhecer meus melhores clientes, gastando no máximo 1 semana de desenvolvimento.”

Cohn (2004) propõe também histórias escritas em formatos mais livres, como em “Um usuário pode postar seu currículo no *website*”. Já Adzic e Evans (2014) lembra que o importante é que as histórias do usuário estimulem uma boa discussão sobre o requisito, e sugere experimentar com diferentes formatos, como perguntas, imagens.

Exemplos de histórias do usuário escritas de forma mais livre são fornecidos por Cohn (2004) para um sistema de busca de empregos:

- um candidato pode incluir seu currículo;
- um candidato pode editar seu currículo;
- um candidato pode apagar seu currículo;
- um candidato pode marcar o currículo como inativo, e
- um resumo pode incluir educação, empregos anteriores, publicações, salários anteriores e um objetivo.

20. Histórias do Usuário

Chamamos a atenção para a diferença entre essas histórias do usuário e como requisitos do sistema segundo a norma IEEE. Os requisitos demandam funcionalidades do sistema, no formato “o sistema deve”, enquanto as histórias do usuários dizem o que o usuário pode fazer, no formato “um usuário pode”.

20.3. INVEST

Ao desenvolver as histórias é necessário levar em consideração que uma boa história deve seguir alguns princípios que são definidos pelo acrônimo INVEST (Cohn, 2004; Wake, 2003):

- Independente;
- Negociável;
- com Valor;
- Estimável;
- Pequena (*Small*), e
- Testável.

20.3.1. Independente

Cada história do usuário deve ser independente, sempre que possível, de outras. O benefício da independência é evitar problemas de estimativa, priorização e planejamento (Cohn, 2004). Elas devem poder ser implementadas em qualquer ordem (Wake, 2003). Isso não é sempre possível, já que algumas funcionalidades, como a produção de relatórios, precisam ser feitas uma de cada vez, e a primeira vez vai afetar, normalmente facilitando, as outras.

Também no caso de estimativas de história com alta dependência fica difícil acertar, pois a implementação de uma vai afetar a implementação da outra, mas pode não ser claro qual o impacto final. Isso pode implicar em juntar algumas histórias de modo a deixar as que as co-dependências (Cohn, 2004) estejam todas na mesma história.

Por exemplo, suponha que em um caixa automático para pagamento de estacionamento existam várias histórias, com os seguintes nomes:

- o motorista deseja **pagar com cartão de crédito** para poder sair do estacionamento com seu carro;
- o motorista deseja **pagar com cartão de débito** para poder sair do estacionamento com seu carro;
- o motorista deseja **pagar com notas** para poder sair do estacionamento com seu carro, e
- o motorista deseja **pagar com aplicativo** para poder sair do estacionamento com seu carro;

Claramente, a implementação de uma dessas histórias pode afetar as outras, tornando-as mais fáceis. A solução de juntar as histórias criaria uma só no lugar delas, usando como ação “pagar estacionamento”. Por outro lado, essa história seria grande, talvez até mesmo um épico, o que pode levar a necessitar de um outro tratamento, mais tarde no projeto.

Outra opção seria manter as histórias em separado, mas criar duas estimativas, uma para a história sendo feita isoladamente, outra para a história sendo feita depois de uma outra estar feita (Cohn, 2004).

Wake (2012) descreve 3 formas de dependência:

1. sobreposição;
2. ordem, e
3. contenção.

Dependência de Sobreposição

A **sobreposição** acontece quando existem um conjunto de funcionalidades e as histórias do usuário cobrem algumas dessas funcionalidades, com interseção entre elas. Um sinal de que isto está acontecendo são histórias com a conjunção “e”, como em “Cliente compra e vende itens” e “Cliente compra e devolve itens”. Isso deveria ser dividido em histórias diferentes como nomes como (Wake, 2012):

- cliente compra itens;
- cliente vende itens, e
- cliente devolve itens

Dependência de Ordem

Nesse caso há uma história que precisa ser implementada antes da outra. Os motivos podem ser muitos, inclusive ser da natureza do problema, já que um item só pode ser devolvido se for comprado. Esse tipo de dependência não pode ser eliminado, mas normalmente a prioridade do negócio vai estar na mesma ordem. ordem (Wake, 2012).

Dependência de Contenção

Nesse caso existe uma organização hierárquica das histórias, onde uma contém outras. Todos os métodos tem que tratar isso de alguma forma, já que a abstração de refinamento sucessivo é muito prática e utilizada praticamente por todos para entender melhor como um sistema se organiza.

Histórias de usuário de nível muito alto de abstração recebem nomes como temas ou épicos (Wake, 2012) e existem formas específicas de tratá-las.

20. Histórias do Usuário

A característica principal da dependência de contenção é que a decomposição hierárquica não é uma estratégia de sequenciamento (Wake, 2012). Então é importante atacar as partes principais, seguindo normalmente uma estratégia de desenvolver o **menor produto viável** e construir versões melhores, agregando o maior valor possível, em sequência.

O **menor produto viável (MVP)** é a versão mais simples de um produto que pode ser lançada com uma quantidade mínima de esforço e desenvolvimento

20.3.2. Negociável

Uma história do usuário é na verdade um tópico de conversação entre usuário e desenvolvedores, logo elas devem ser escritas de forma a serem negociáveis. A negociação promove o entendimento e compromisso entre as partes (Jacobson, Lawson e Ng, 2019).

20.3.3. Com Valor

Cada história precisa ter um valor para o cliente, ou outra parte interessada que não seja da equipe de desenvolvimento (Wake, 2003), como o comprador. Histórias são sempre escritas com a perspectiva do usuário.

Esse conceito está relacionado com a prática geral dos métodos ágeis de desenvolver os sistemas por fatias verticais, isso é, sempre entregando funcionalidade ao cliente, em vez de camadas onde funcionalidades internas, de base, são feitas antes, e mais tarde as camadas externas, que atendem os clientes, são feitas ().

A questão do interesse de outras partes interessadas é importante, porque usuários não estão interessados com certos requisitos do sistema. Cohn (2004) cita, por exemplo, a necessidade de seguir alguma norma internacional, que pode ser do interesse do comprador mas não dos usuários. Um história válida, então, seria “A equipe de desenvolvimento irá fornecer documentação que atende as normas de auditoria da ISO 9001”(Cohn, 2004).

20.3.4. Estimável

Para poder ser útil no processo de planejamento, é importante que as histórias do usuário sejam estimáveis Wake (2003) em relação ao esforço necessário para implementá-las.

Estimativas são sempre feitas pelos desenvolvedores. É preciso reafirmar a prática que as pessoas do negócio decidem o que deve ser feito, mas são os desenvolvedores que decidem como e o prazo.

Estimar histórias e dar uma avaliação do esforço necessário para desenvolvê-las, em alguma medida combinada no projeto. Dois métodos usados no mercado são usar pesos relativos, como a sequência (1,2,3,5,8,13,20,40,100) usada no *Planning Poker* ou tamanhos de camisa, como (XP,P,M,G,XG).

Os principais motivos para histórias não poderem ser estimadas são (Cohn, 2005):

O *Planning Poker* é uma técnica de estimativa baseada na técnica de Delphi que usa reuniões face a face e estimativas pessoais por meio de cartas com números que

- os desenvolvedores não entenderam o domínio, isto é, não entenderam como a história do usuário deve acontecer no seu contexto, e devem discutir mais a história com os usuários;
- os desenvolvedores não sabem como implementar, por exemplo, por falta de domínio da tecnologia, e
- a história é muito grande e é muito difícil estimar o esforço necessários, sendo um épico.

É importante notar que normalmente as histórias do usuário são usadas junto com *time-boxes*, o que implica em elas terem um tamanho máximo, correspondente ao que pode ser feito dentro desse intervalo de tempo.

20.3.5. Pequena

O tamanho da história do usuário é um fator importante na sua utilidade tanto no projeto quanto no planejamento do mesmo. Histórias muito pequenas são fáceis de implementar, mas podem trazer pouco valor, enquanto histórias muito grandes trazem muito valor, mas também causam dificuldades para estimá-las ou completá-las dentro de um *time-box*. Encontrar um tamanho ideal para uma história é parte do trabalho de colaboração entre desenvolvedores e clientes. Esse tamanho deve ser pequeno de forma que ela possa ser implementada em um ciclo de desenvolvimento (Jacobson, Lawson e Ng, 2019)

Histórias grandes demais podem ser de dois tipos (Cohn, 2004):

- **histórias compostas**, que são épicos com várias histórias menores dentro dele, ou
- **histórias complexas**, que são inherentemente grandes e difícil de ser particionadas em histórias menores.

Um exemplo de história composta é a história “Um correntista pode movimentar sua conta” em um banco, que pode ser facilmente convertida em um conjunto de histórias como:

- um correntista pode depositar valor em conta;
- um correntista pode sacar dinheiro da conta;
- um correntista pode transferir valor de sua conta para outra, e
- um correntista pode pagar boleto.

Histórias grandes podem, e normalmente devem, ser quebradas. Cada história menor devem manter uma interação significativa com o usuário, mesmo que pequena ou especializada. Uma dica para escolher um pedaço de uma história do usuário durante o processo de divisão é olhar os testes e verificar seus objetivos (Jacobson, Lawson e Ng, 2019).

20.3.6. Testável

Esse critério é um dos mais importantes. Normalmente é usado com técnicas como *Test Driven Development*, onde o teste é implementado antes. Também ajuda a que desenvolvedor e usuário concordem com o que vai ser feito (Jacobson, Lawson e Ng, 2019).

20.4. Estados de Uma História do Usuário

De acordo com o SEMAT, uma história do usuário pode passar por 4 estágios (Jacobson, Lawson e Ng, 2019):

1. identificada, onde tem seu valor claramente expresso e está no *backlog* do produto;
2. pronta para desenvolvimento, onde os detalhes foram discutidos e está claro o que é necessário para cumprir seus requisitos;
3. em progresso, onde o time está desenvolvendo a história, e
4. verificada, onde um representante dos usuários verificou a história.

Já um cartão de história pode passar por 3 estados (Jacobson, Lawson e Ng, 2019):

1. valor expresso, o que pode ser feito em alguns dos formatos aceitos;
2. critério de aceitação expresso, e
3. conversação capturada, onde foram feitas as discussões que detalham os requisitos da história e elas foram possivelmente registradas no cartão ou em um sistema a parte.

Finalmente, um caso de teste passa também por 3 estados (Jacobson, Lawson e Ng, 2019):

1. critério de aceitação capturado;
2. roteiro de teste pronto, e
3. roteiro de teste automatizado.

20.5. Exercícios

Exercício 20.5-1: Escreva um conjunto de histórias do usuário para um aplicativo de celular que sirva como relógio e despertador, permitindo vários alarmes.

Exercício 20.5-2: Vá para o site <http://jogodeanalisedesistemas.xexeo.net/> e visite a Livraria Resolve. A partir da sua visita especifique no todas as histórias do usuário que encontrou.

21

Casos de Uso

We thus do not model reality as it is, as object orientation is often said to do, but we model the reality as we want to see it and to highlight what is important in our application

(Ivar Jacobson)

Conteúdo

21.1.	Conceituação de Caso de Uso	269
21.2.	Conceitos Importantes nos Casos de Uso	271
21.3.	A Narrativa do Caso de Uso	278
21.4.	O Nível de Abstração	278
21.5.	O Escopo do Caso de Uso	279
21.6.	Casos de Uso Especiais	280
21.7.	Partes do Caso de Uso	281
21.8.	Exemplo de Um Caso de Uso	282
21.9.	Levantando Casos de Usos	283
21.10.	Diagramas de Caso de Uso	286
21.11.	O Que o Diagrama de Caso de Uso Não Informa	293

21. Casos de Uso

21.12. Exercícios 293

Por que casos de uso?

Junto com histórias do usuário, casos de uso são uma das principais formas de especificar requisitos, sendo ainda mais detalhadas que as primeiras. São usados em projetos ágeis ou prescritivos e um documento muito comum de se achar em fábricas de software.

Este capítulo apresenta Casos de Uso e Diagramas de Caso de Uso. É importante não confundir os dois conceitos. Um caso de uso é uma narrativa textual detalhada, acompanhada de informações que ajudam a entendê-la, normalmente apresentado na forma de um documento. Um diagrama de caso de uso é um gráfico que resume todos os casos de uso do sistema, não fornecendo nenhum detalhe, e que informa apenas uma visão global do escopo do sistema.

Casos de uso foram criados por Ivar Jacobson em 1986, aparecendo em um artigo da OOPSLA 87(Jacobson, 1987), mas só foram publicados em livro em 1992 (Jacobson, Christerson et al., 1992). Uma das melhores descrições de seu uso foi feita por Cockburn (2000), no livro *Writing Effective Use Cases*.

A parte principal de um caso de uso é uma sequência numerada de atividades que representam as ações que acontecem durante a interação de usuários com um sistema, como na descrição do cenário principal do caso de uso fictício “Avaliar Alunos” da Figura 21.6, que é um objetivo do ator professor e manda informações para o ator aluno.

1. Sistema Acadêmico apresenta tela de opções
2. Professor escolhe Entregar Notas
3. Sistema Acadêmico apresenta lista de cadeiras disponíveis
4. Professor escolhe cadeira
5. Sistema Acadêmico apresenta lista de alunos com espaço para notas
6. Professor inclui notas para todos os alunos
7. Sistema Acadêmico registra notas
8. Sistema Acadêmico envia alteração de notas para todos Alunos por email
9. Sistema Acadêmico informa ao professor que as notas foram alteradas
10. Professor sai do Sistema Acadêmico

Figura 21.1.: Exemplo de cenário principal de caso de uso.

Casos de uso são uma das técnicas de maior sucesso na análise de sistemas, e apesar de serem identificados com a Análise Orientada a Objeto, não há nenhuma obrigação de serem usados apenas nesse contexto. A principal vantagem dos casos de uso sobre técnicas

anteriores é serem descrições do comportamento concreto do software a ser desenvolvido, quando no passado se dava muita importância a descrições abstratas que eram confusas para o usuário final e deixam grande margem de interpretação, e consequentemente eram ambíguas para o desenvolvedor(Jacobson, 2004).

Casos de uso contam como o sistema funciona, ou funcionará, na visão de seus usuários, o que é sempre uma visão externa ao sistema. Eles devem ser facilmente lidos tanto pelos clientes, e outras partes interessadas, quanto pelos desenvolvedores. São, na prática, histórias contada na forma de um passo a passo, de como o usuário usa o sistema e dos efeitos dessa utilização no estado do sistema.

Já um diagrama de caso de uso tem a aparência da Figura 21.2.

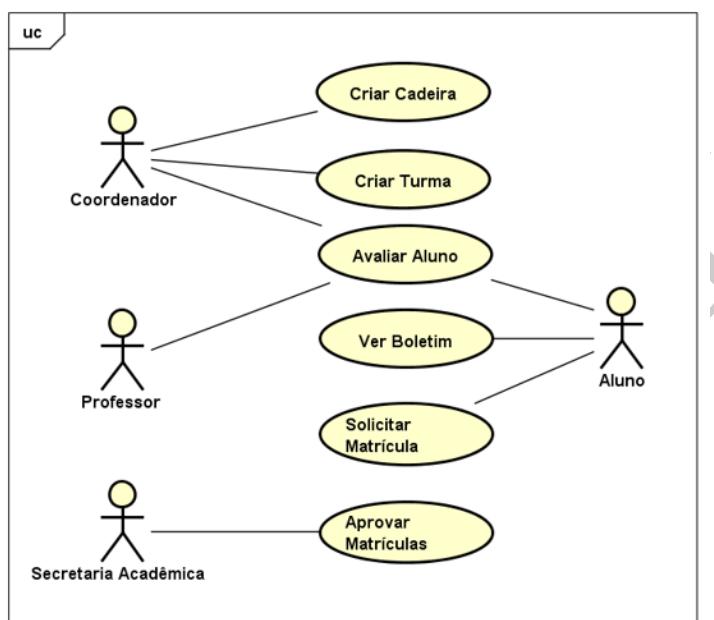


Figura 21.2.: Exemplo de um caso de uso.

Diagramas de caso de uso permitem ao seu leitor ter uma ideia do escopo do sistema. Eles apresentam os atores envolvidos e as funcionalidades esperadas do sistema de forma simples. Desde a origem da análise de sistemas é comum existir um diagrama cujo objetivo é mostrar o escopo do sistema, e o diagrama de caso de uso cumpre essa função na Análise Orientada a Objeto, sendo parte da especificação UMLOMG (2017). Basicamente eles substituíram os Diagramas de Contexto Gane e Sarson (1979) e Ruble (1997) nessa finalidade.

21.1. Conceituação de Caso de Uso

Um **caso de uso** é uma especificação, em forma de narrativa, de uma sequência de interações entre um sistema e os atores (agentes externos) que o usam de forma a alcançar

21. Casos de Uso

um objetivo. Esse sistema pode ser um software ou uma organização complexa.

Cada caso de uso responde a uma solicitação de uma parte interessada, o **ator principal**, e descreve como o sistema vai se comportar enquanto interage com esse ator, de acordo com várias condições que podem ocorrer durante a interação(Cockburn, 2000).

Casos de uso podem ser simples ou complexos, devendo descrever, em um nível de detalhe desejado, algo que um usuário ou cliente quer que o sistema faça. Cada caso de uso descreve e define parte da funcionalidade de um sistema.

As definições de caso de uso, pelos seus criadores, evoluíram com o tempo. Jacobson, Spence e Bittner (2011) citam duas definições, que são apresentadas a seguir.

- Um **caso de uso** mostra todas as formas de usar um sistema para alcançar um objetivo específico para um usuário específico. O conjunto de casos de uso fornece todas as formas úteis de usar o sistema e obter o valor que ele oferece.
- Um **caso de uso 2.0** é uma prática ágil e escalável que captura um conjunto de requisitos e guia o desenvolvimento incremental de um sistema para atingi-los.

Um caso de uso bem escrito deve descrever, de forma completa, um processo executado pelo sistema, contando uma história de como o ator principal tenta alcançar um objetivo específico ao usar o sistema. Na sua forma mais completa, um caso de uso apresenta vários cenários possíveis de sucesso ou falha na busca por esse objetivo(Cockburn, 2000) e ainda informações adicionais para o time.

O conjunto completo de casos de uso forma a especificação funcional de um sistema. Essa especificação é facilmente legível por usuários ou desenvolvedores, permitindo também sua validação e verificação. É importante notar que os diagramas de caso de uso têm um valor muito pequeno frente ao documento textual que é a descrição do caso de uso.

Um caso de uso deve:

- descrever um tarefa de negócio que serve a um único objetivo de negócio;
- não ser orientado a uma linguagem de programação;
- ter o nível de detalhe apropriado;
- ser curto o suficiente para ser implementado por um desenvolvedor de software em um versão do produto;
- ser descrito do ponto de vista externo, e
- ser consistente, tanto no nível de abstração quanto na escolha entre mostrar o sistema como uma caixa branca ou uma caixa preta.

Casos de uso também podem ser vistos como “contratos entre as partes interessadas e o sistema sobre o seu comportamento” Cockburn (2000).

Para isso, casos de uso devem descrever três tipos de ações(Cockburn, 2000):

- **interações entre dois atores**, sendo um deles normalmente o sistema;
- **validações**, que protegem uma ou mais partes interessadas, e
- **mudanças do estado interno do sistema**, que atendem uma ou mais partes

interessadas.

21.2. Conceitos Importantes nos Casos de Uso

Os principais conceitos envolvidos em um caso de uso são:

- **atores**, e entre eles o **ator principal**, que participam do caso de uso;
- um **objetivo** do ator principal, que nomeia o caso de uso;
- os **cenários** que permitem atingir o objetivo ou levam a falhas, entre eles o **cenário principal** e vários possíveis **cenários alternativos**;
- as **condições** que geram os cenários alternativos;
- cada cenário é um **sequência de ações** que acontece durante a interação entre atores e o sistema, e
- cada ação é:
 - uma troca simples e unidirecional de informações entre dois atores;
 - uma validação, ou
 - uma mudança do estado interno do sistema.

21.2.1. Ator

Um ator é uma entidade externa ao sistema com comportamento próprio. Com isso queremos dizer que o sistema não pode controlar o comportamento do ator.

Os atores interagem com o sistema para alcançar seus objetivos. Em cada Caso de Uso, um ator é o ator principal, que visa alcançar com sucesso o seu objetivo principal por meio daquele caso de uso. Muitas vezes, o sistema invocará outros atores, que deverão cumprir suas responsabilidades para o ator principal alcançar seu objetivo, ou receberão informações.

Um ator não é a mesma coisa que um usuário. Um ator representa um papel dos usuários de um sistema. Tanto um ator pode representar um papel assumido por vários usuários, como o papel de correntista em um banco, para o qual existem milhares de usuários, quanto um usuário (pessoa real) pode ser representado por vários atores, como no caso de um funcionário do banco que é também correntista. Conceitualmente, em UML, usuários são instâncias de atores e cada ator define uma classe de usuários(OMG, 2017).

Os principais tipos de atores são:

- pessoas;
- organizações;
- equipamentos, e
- sistemas.

Um ator, nos diagramas de caso de uso, é representado por um “boneco de pauzinhos”

21. Casos de Uso

sobre o nome que identifica o ator, como na Figuras 21.3 e Figura 21.4.



Figura 21.3.: Representação de um ator em um diagrama de caso de uso.



Figura 21.4.: Representação de dois atores identificados.

O sistema sendo descrito também é um ator, que conversa com o ator principal e com os outros atores.

É importante notar que existem partes interessadas que não são atores, pois nunca interagem com o sistema. Por exemplo, o patrocinador ou os próprios desenvolvedores, mas ainda assim podem ser afetadas pelas ações do sistema. Um exemplo possível seria um sistema de segurança pública que guarda informação sobre criminosos, mas esses criminosos nunca tem contato com o sistema.

21.2.2. Objetivos

Todo ator possui um ou mais objetivos ao usar o sistema. Alguns objetivos são ativos, isso quer dizer que o usuário invoca o sistema, se tornando o ator principal de um caso de uso, e outros podem ser ditos passivos, ou seja, o sistema vai informar o ator de algo ou invocar o ator para alguma ação.

Cada objetivo define e dá nome a um caso de uso. Esse objetivo deve deixar bem claro o que o ator deseja obter como funcionalidade do sistema, definindo de forma implícita o comportamento que ele espera do sistema.

Exemplos de objetivos são:

- em um sistema bancário, depositar dinheiro em conta corrente, sacar dinheiro de conta corrente, pagar boleto, transferir dinheiro para outra conta corrente, requisitar cheques;
- em um sistema de seleção de profissionais, enviar currículo, marcar entrevista, avaliar currículo, solicitar classificação dos candidatos, e

21.2. Conceitos Importantes nos Casos de Uso

- em um sistema acadêmico, solicitar matrícula, avaliar alunos, informar presenças, emitir boletim.

Nomeando Casos de Uso

O objetivo é usado para nomear o caso de uso. Isso é feito no formato

<verbo no infinitivo><objeto>

como em “emitir boletim”. Para isso devem ser usados verbos que indiquem uma ação do ator, pois ele entra no sistema com a intenção de fazer algo. A Tabela 21.1 apresenta alguns verbos comumente usados para nomear casos de uso.

Tabela 21.1.: Verbos comumente usados para nomear casos de uso

Alterar	Descobrir	Permitir
Analisar	Encontrar	Preencher
Aprontar	Entregar	Preparar
Arranjar	Especificar	Projetar
Avaliar	Estabelecer	Providenciar
Buscar	Executar	Realizar
Classificar	Garantir	Recuperar
Completar	Identificar	Requisitar
Conseguir	Informar	Selecionar
Consultar	Monitorar	Solicitar
Definir	Mudar	Ver
	Notificar	

21.2.3. Cenários

Um caso de uso pode acontecer de acordo com vários cenários. Cada **cenário** descreve como uma instância específica do caso de uso pode acontecer, ou seja, as sequências específicas de ações que ocorrem na interação entre atores e o sistema.

Um desses cenários é o **cenário principal**, que narra como um ator alcança seu objetivo da forma mais fácil ou comum. O cenário principal é descrito de forma integral em todos os casos de uso e mostra como é esperado que a interação normal que atinge o objetivo aconteça. Por exemplo, para o caso de uso “sacar dinheiro”, solicitado pelo ator “correntista” para o sistema “caixa automático”, o cenário principal poderia ser o da Figura 21.5:

O cenário acima é descrito na forma de uma **narrativa numerada**. Cada passo dessa narrativa é uma troca simples, e unidirecional, de informações, ou de uma mensagem, uma validação ou uma ação. Essa narrativa é o que deve acontecer se tudo der certo, ela

21. Casos de Uso

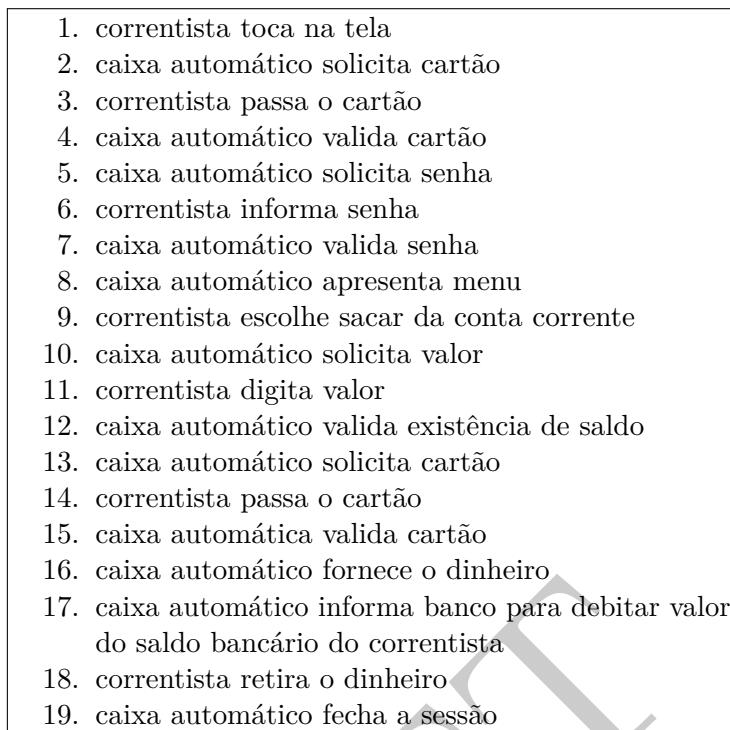


Figura 21.5.: Exemplo de cenário principal do caso de uso **sacar dinheiro**.

não se preocupa, por exemplo, com o que fazer se não há saldo para retirar a quantia ou se a senha está errada. Também não se preocupa se o usuário quer fazer outra ação, como imprimir cheques.

Veja que no passo 17 o caso de uso envolve um outro ator, o banco, para o qual é enviada uma mensagem de alteração de saldo. Uma alternativa a esse passo seria “caixa automático debita valor do saque do saldo do correntista”. Esse é um tipo de passo que indica uma mudança no estado interno do sistema e também devemos considerar obrigatório, pois indica o contrato que o caso de uso cumpre.

Casos de uso também apresentam passos de validação, que tem como função proteger os interesses das partes interessadas. Os passos 4, 7, 12 e 15 fazem a validação e serão usados como locais onde condições definem alternativas.

Cenários, além do cenário principal, podem representar variantes normais do fluxo principal, casos raros, exceções e erros. Dessa maneira, podemos compreender um caso de uso como a descrição de uma coleção de cenários de sucesso ou falha que descrevem um determinado processo do sistema com a finalidade de atender um objetivo do usuário.

Na maior parte das vezes, os casos de uso também possuem vários **cenários alternativos**, alguns que também levam ao objetivo desejado, outros que levam a versões parciais desse objetivo e ainda cenários de falhas, onde o objetivo não é alcançado. Todos esses cenários também são descritos nos casos de uso, porém normalmente de forma compactada, sendo mostrado apenas como eles diferem do caso de uso principal.

21.2. Conceitos Importantes nos Casos de Uso

Os cenários diferem em função de condições específicas que podem acontecer na execução de uma instância do caso de uso. Por exemplo, se o caso de uso descreve uma retirada de dinheiro de uma conta bancária, o cenário principal considera que existe saldo suficiente na conta, enquanto cenários alternativos podem considerar que não existe saldo suficiente, que será necessário usar o cheque especial, ou que é tarde demais para retirar a quantia pedida.

No nosso exemplo, uma condição que leva a um cenário alternativo poderia ser poderia ser “senha não confere” e outra poderia ser “não há saldo suficiente”. Nesse caso, cada condição levaria a um cenário alternativo.

Uma forma possível de descrever o cenário alternativo “senha não confere” seria:

1. caixa automático informa que senha não confere
2. voltar ao passo 5

Nesse caso ainda é possível terminar o caso de uso principal, o sistema volta para um passo anterior ao que aconteceu a alternativa e o ator tem mais uma chance.

Uma condição adicional seria “senha não confere pela terceira vez”, e o cenário relativo seria:

1. caixa automático informa que senha não confere
2. caixa automático informa ao banco para bloquear o cartão
3. caixa automático termina atendimento

Nesse caso não foi possível voltar ao cenário principal, o ator não conseguiu seu intento e o caso de uso terminou em falha.

Novamente, nesse último cenário, teríamos a opção de que o passo 2 fosse “caixa automático bloqueia cartão”, que indicaria uma mudança no estado do sistema.

As Figuras 21.6 e 21.7 ilustram o conceito de um caso de uso contendo um caminho principal e quatro condições que podem alterar a execução de uma instância do caso de uso. As figuras mostram, de forma abstrata, possíveis caminhos que formam cenários de execução desse caso de uso. Podemos ver, na representação da figura, que alguns caminhos alternativos podem permitir ainda alternativas adicionais (o caminho 2 é uma alternativa do caminho 1), e que em uma mesma execução de um caso de uso vários caminhos podem ser seguidos (os caminhos 1 e 3, por exemplo).

Com essas figuras deve ficar claro um conceito: os cenários alternativos podem ser compostos de várias formas, mas com uma descrição compartimentada, não precisamos usar todas as instâncias possíveis dos cenários, por exemplo listar todos os cenários possíveis onde a alternativa 3 é chamada 1, 2, 3 ou infinitas vezes.

Também é importante notar que **não são usadas estruturas se-então-senão** ao escrever casos de uso. O que se usa são condições que podem acontecer em passos de um cenário e ordens como “vá para”. Cada cenário considera que todos seus passos de validação resultam em verdade.

Já o uso de laços, isto é, estruturas de repetição, pode ser feito de algumas formas,

21. Casos de Uso

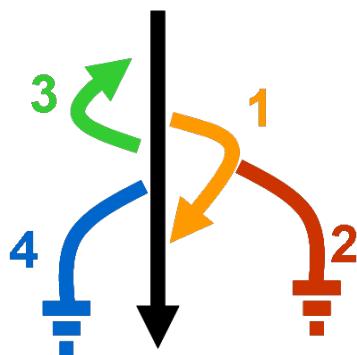


Figura 21.6.: Representação abstrata de um cenário principal e dos fluxos alternativos

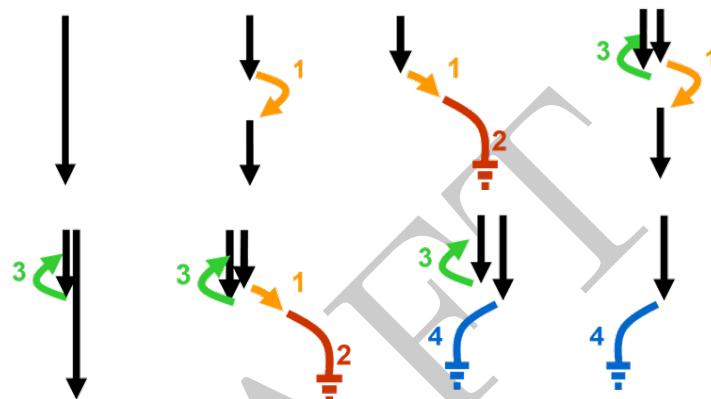


Figura 21.7.: Representação abstrata de cada alternativa investigada nesse caso.

mas nunca na forma de um *for* ou *while* de uma linguagem de programação, mas sim como uma linguagem mais coloquial. (Cockburn, 2000) sugere:

- usar um passo onde algo pode ser feito várias vezes, como em “o comprador seleciona um ou mais produtos”, ou
- um passo não numerado que indica que alguns passos podem ser repetidos, como em “o comprador repete os passos 4-6 até ficar indicar que terminou”, que pode aparecer antes ou depois dos passos;

Por exemplo, um caso de uso onde o ator “professor” dá a nota para vários alunos poderia ser do da Figura 21.8.

A Tabela 21.2 mostra uma notação alternativa para explicar os cenários possíveis a partir de alternativas.

21.2.4. As Alternativas

As alternativas estão sempre associadas a uma condição ocorrendo no cenário principal, possivelmente em um momento de validação, ou, mais raramente, a um outro cenário

21.2. Conceitos Importantes nos Casos de Uso

1. professor seleciona DAR NOTAS
os passos 2-7 acontecem até o professor indicar que terminou
2. sistema mostra lista de alunos
3. professor escolhe aluno
4. sistema mostra registro do aluno
5. professor dá nota ao aluno
6. sistema altera a nota do aluno
7. sistema pergunta se o professor terminou
8. professor sai do sistema

Figura 21.8.: Exemplo de cenário principal do caso de uso **atribuir notas**.

Tabela 21.2.: Uma forma alternativa de representar os cenários possíveis a partir de um cenário principal e as alternativas associadas

Cenário 1	Cenário Principal			
Cenário 2	Cenário Principal	Alternativa 1		
Cenário 3	Cenário Principal	Alternativa 1	Alternativa 2	
Cenário 4	Cenário Principal	Alternativa 3		
Cenário 5	Cenário Principal	Alternativa 3	Alternativa 1	
Cenário 6	Cenário Principal	Alternativa 3	Alternativa 1	Alternativa 2
Cenário 7	Cenário Principal	Alternativa 4		
Cenário 8	Cenário Principal	Alternativa 3	Alternativa 4	

importante.

Um cenário alternativo tem pelo menos quatro partes:

- a localização, mostrando o cenário, normalmente o principal, e o passo onde ocorre a condição;
- a condição, especificada na forma de uma expressão que pode ser avaliada como verdadeira ou falsa (lógica);
- a sequência de ações que corresponde a essa condição, e
- uma finalização que deixa claro o retorno ao fluxo principal ou a finalização de outra forma.

As finalizações possíveis para uma alternativa são:

- voltar ao início do caso de uso, comum em casos de falha onde é possível tentar outra vez;
- voltar a um passo já executado do caso de uso, trazendo o estado do sistema naquele passo;
- ir para um passo posterior ao passo onde ocorreu a condição, possivelmente o final;
- encerrar o caso de uso no cenário alternativo, com sucesso ou sucesso parcial;
- encerrar o caso de uso no cenário alternativo com falha, e

21. Casos de Uso

- abortar o caso de uso, em caso de falha grave.

21.2.5. Casos de Uso em UML

Em UML casos de uso descrevem o comportamento de um sistema, ou parte de sistema, como um componente ou classe, em consideração, denominado *subject*. Um caso de uso pode se aplicar a vários *subjects*, especificando um conjunto de comportamentos para os *subjects* a que é aplicado. Os atores são pessoas e outros sistemas que interagem com o *subject*. Um caso de uso gera um resultado que tem valor para os atores e outras partes interessadas no *subject*(OMG, 2017).

21.3. A Narrativa do Caso de Uso

A narrativa numerada é a forma de básica de descrição dos cenários do caso de uso. Essa forma de narrativa apresenta várias vantagens sobre outras formas já utilizadas de modelagem de processos, como manter o contexto visível, eliminar dificuldades de compreensão para o usuário (causadas por linguagens técnicas com forte grau de abstração) e deixar claro qual o valor de cada função para os usuários.

Cada caso de uso possui uma narrativa principal, que mostra o que acontece no caso normal, e narrativas alternativas que tratam de atender a condições especiais que acontecem durante uma possível execução do caso de uso e fazem ele se afastar da narrativa principal.

Essa narrativa é uma especificação de uma sequência de ações que acontece interações entre o sistema e os agentes externos que usam esse sistema.

Segundo Cockburn (2000), cada passo da narrativa deve:

- usar uma gramática simples;
- mostrar quem está agindo;
- mostrar o processo andando para frente;
- mostrar a intenção do ator, não os movimentos, e
- conter um conjunto de ações razoável;

Outras formas de escrever o caso de uso são usadas no mercado, sendo que é comum que no início da análise seja usado um texto informal na forma de um parágrafo, muitas vezes tirado diretamente de uma entrevista.

21.4. O Nível de Abstração

Uma das formas mais interessantes de entender como desenvolver casos de uso é entender que eles podem ser descritos em diferentes níveis de abstração, desde um nível

bastante abstrato até um nível detalhado em seus mínimos detalhes.

A idéia básica em torno desse conceito, proposto por Cockburn (2000), é que casos de uso de um nível mais abstrato explicam o porquê de um caso de uso de um nível mais baixo, enquanto o caso de uso de um nível mais baixo explica como o caso de uso do nível mais abstrato é realizado.

Podemos identificar cinco níveis de abstração para casos de uso (Cockburn, 2000):

- sumário de alto nível;
- sumário;
- objetivo do usuário;
- sub-função, e
- muito detalhado.

Tabela 21.3.: Ícones que representam o nível de abstração dos casos de uso.

Fonte:(Cockburn, 2000)

Nível	Ícone
Sumário de Alto Nível	
Sumário	
Objetivo do Usuário	
Sub-Função	
Nível Muito Detalhado	

A abordagem adotada neste texto é trabalhar no nível de objetivo do usuário. Se necessário, para projetos grandes, é possível criar resumos no nível de sumário. Por exemplo, os casos de uso “sacar dinheiro”, “depositar dinheiro” e “transferir dinheiro” poderiam ser resumidos em um caso de uso de nível superior chamado “movimentar conta”.

Raramente são usados casos de uso mais detalhados do que no nível do objetivo do usuário, a não ser os casos de uso incluídos, que são normalmente do tipo sub-função.

21.5. O Escopo do Caso de Uso

Da mesma forma que definimos o nível de abstração de um caso de uso, podemos também definir o escopo do caso de uso em relação a organização ou sistema que ele

21. Casos de Uso

descreve em níveis (Cockburn, 2000)

- organização, caixa preta;
- organização, caixa branca;
- sistema, caixa preta;
- sistema, caixa branca, e
- componente.

Tabela 21.4.: Ícones que representam o nível de escopo dos casos de uso. Fonte:(Cockburn, 2000)

Nível	Ícone
Organização, caixa preta	
Organização, caixa branca	
Sistema, caixa preta	
Organização, caixa branca	
Componente	

O nível que trabalhamos nesse texto é normalmente o nível de sistema de caixa preta. Nesse nível, tratamos de atores nomeados usando um sistema que é sempre chamado por seu nome.

No nível de sistema com caixa branca, passamos a nomear a parte do sistema que está sendo usada.

No nível organização e caixa preta, em vez de tratar do uso do sistema, se trata da interação do usuário com a organização. Assim, um correntista passa a conversar com o banco, um paciente com o hospital. No caso da caixa branca, o correntista passa a falar com o caixa ou com o gerente, o paciente com o médico ou a enfermeira. Também é possível usar a caixa branca em relação a setores da organização, como tesouraria e atendimento.

21.6. Casos de Uso Especiais

Alguns casos de uso não tem origem nos usuários ou cliente, mas em outras partes interessadas que precisam implantar e operar o sistema. Esses casos de uso são especiais e podem ser usados poucas vezes, mas podem ser bastante complexos. Alguns casos possíveis são:

- início do sistema;

- parada do sistema;
- manutenção da informação, inclusive com operações do tipo CRUD adicionais;
- adicionar nova funcionalidade com o sistema funcionando, em especial para sistemas que não devem parar;
- transferência do sistema para um novo ambiente, etc.

Create, Read, Update e Delete,
operações que sempre podem ser realizadas sobre os dados mesmo sem ter uma semântica do negócio

21.7. Partes do Caso de Uso

Um caso de uso completo é um documento com várias partes. Entre as partes possíveis que podemos usar estão:

- nome;
- identificação(número, etc.);
- escopo e nível de abstração(Cockburn, 2000);
- ator principal;
- outros atores;
- outras partes interessadas;
- pré-condições;
- gatilhos;
- pós-condições, em caso de sucesso;
- garantias, em caso de falha;
- cenário principal;
- cenários alternativos;
- requisitos especiais;
- variações tecnológicas ou de dados;
- justificativa;
- fonte ou origem;
- casos de uso relacionados;
- questões em aberto.

Já conhecemos a importância dos atores. A seguir discutiremos as outras partes do documento.

As **partes interessadas** a serem tratadas em um caso de uso são aquelas que não fazem parte do caso, mas podem ter interesse em seu resultado. Por exemplo, em um laboratório de exames clínicos, se um ator “técnico em análises clínicas” usa o caso de uso “registrar resultado de exame”, o ator “paciente” é uma parte interessada, mesmo que só receba o resultado após o ator “médico” liberar o resultado.

As **pré-condições** de um caso de uso definem o que é sempre verdadeiro quando um caso de uso acontece. São condições que **não precisam ser testadas**, ou seja, são assumidas como verdadeiras no caso de uso. Um exemplo no caso de uso “registrar resultado de exame” seria “exame já está cadastrado”. Só devem ser comunicadas nessa seção questões dignas de nota, que constituam informação útil para o desenvolvimento e funcionamento do sistema.

21. Casos de Uso

Os **gatilhos** são eventos, como ações ou condições, que causam o início do caso de uso. Origem ou fonte registra quem ou o que, como um documento, demandou ou indicou a necessidade do caso de uso.

As **pós-condições** estabelecem o que deve ser verdadeiro após o caso de uso, no caso de sucesso, por exemplo garantir que caso um caixa automático tenha dado o dinheiro ao correntista que esse dinheiro tenha sido debitado da conta do mesmo. As **garantias** são o estado final caso haja um erro, por exemplo não alterando o saldo do correntista caso ele não tenha recebido o dinheiro.

Pré-condições, pós-condições e garantias são uma especificação do estado anterior e posterior do sistema e servem como parte do contrato do caso de uso.

Justificativa indica a motivação, o interesse para aquele caso de uso.

Requisitos especiais estão relacionados a demandas adicionais que são feitas a esse caso de uso específico e que não são gerais do sistema. Por exemplo, tecnologias específicas que devem ser usadas, velocidade, etc.

21.8. Exemplo de Um Caso de Uso

Caso de Uso: 1	Solicitar Boletim Oficial
<i>Escopo:</i>	Sistemas, caixa-branca
<i>Nível de abstração:</i>	Objetivo do usuário
<i>Autor principal:</i>	Aluno
<i>Autor secundário:</i>	Secretaria Acadêmica
<i>Outras partes interessadas:</i>	<ul style="list-style-type: none">• Coordenador: Tem que assinar o documento
<i>Pré-condições:</i>	<ul style="list-style-type: none">• O aluno está logado no sistema• O aluno tem notas em ao menos um período
<i>Pós-Condições:</i>	O pedido do aluno está incluído na lista de tarefas da Secretaria Acadêmica
<i>Gatilho:</i>	Aluno escolhe solicitar boletim oficial no menu
<i>Cenário Principal:</i>	

1. Sistema pede para o aluno confirmar o pedido
 2. Aluno confirma o pedido
 3. Sistema verifica se o aluno não tem nenhum pedido na fila
 4. Sistema coloca o pedido do aluno na fila de tarefas da Secretaria Acadêmica de seu curso
 5. Sistema avisa a Secretaria Acadêmica que há uma nova tarefa na fila
 6. Sistema informa ao aluno que o pedido foi completado
 7. Aluno aceita a mensagem
-

Alternativas:

- 3.a Aluno já tem um pedido na fila:

1. Sistema avisa Secretaria Acadêmica que um pedido foi refeito
 2. Sistema avisa aluno que seu pedido já estava na fila e que um aviso foi dado a Secretaria Acadêmica
 3. Aluno aceita a mensagem
-

Requisitos Especiais: O sistema deve responder ao aluno em até 5s

Variações de tecnologia: Essa função deve funcionar na versão para telefone celular

Assuntos em Aberto: Será necessário comprovar o pagamento de uma taxa?

21.9. Levantando Casos de Usos

O processo de levantar casos de uso é um processo investigativo e iterativo. Ele pode ser resumido em 6 passos, que devem ser feito iterativamente até alcançar a especificação desejada:

1. identificar os atores;
2. identificar os objetivos de cada ator, nomeando seus casos de uso;
3. escrever o cenário principal para cada objetivo;
4. determinar as condições que geram as alternativas nos cenários existentes;
5. escrever o cenário alternativo para cada condição, e
6. detalhar as variações de dados.

Para identificar os atores é necessário investigar não só os atores que são pessoas físicas, mas também que sistemas, subsistemas, organizações, grupo de pessoas vão interagir com o software. Muitas vezes um caso de uso que começa com um ator precisa se comunicar de alguma forma com outro ator, por exemplo, enviando mensagens. O resultado do primeiro passo é uma lista de atores, que poderá ser estendida nos próximos passos.

Sempre que um ator novo for encontrado, durante a elicitação de um caso de uso, ele

21. Casos de Uso

deverá ser investigado para que se possa avaliar se ele também vai demandar casos de uso, isto é, se terá objetivos para usar o sistema.

Atores devem ser nomeados com um nome específico do negócio ou da área de aplicação. Assim um sistema acadêmico possui aluno, professor, coordenador, etc. Um consultório ou ambulatório possui enfermeiro, auxiliar de enfermagem, médico, paciente, atendente, acompanhante, etc. Uma loja de roupas possui cliente, estoquista, gerente, vendedor, etc. Usar nomes genéricos, como usuário, demonstra má qualidade do modelo. O resultado do primeiro passo é uma lista de atores, que responde a pergunta “quem?”.

Para identificar o objetivo é importante entender o que cada ator precisa do sistema, quais suas necessidades e como elas devem ser demandadas ao sistema. O resultado do segundo passo é uma lista de casos de uso, que responde a pergunta “o que?” e um diagrama de casos de uso, que fornece uma visão geral do escopo do sistema. Com isso é possível ter uma lista delimitada e usável das funções do sistema.

Objetivos são escritos no formato verbo-objeto. Alguns exemplos são: solicitar boletim (para o ator aluno usando um sistema acadêmico) e lançar notas (para o ator professor no mesmo sistema).

No terceiro passo a função do analista de sistemas é determinar como o usuário alcançará seu objetivo usando o sistema. Esse passo a passo só deve ter as mensagens trocadas entre os agentes (atores e sistema). Não pode ser usada nenhuma estrutura de controle, como laços e decisões. O cenário deve capturar a intenção e responsabilidade de cada ator até alcançar o objetivo. Cada passo do cenário deve deixar bem clara a informação que é passada entre os atores. O resultado é uma descrição legível das funções do sistema na forma que seriam executadas se tudo ocorresse normalmente, sem nenhuma condição especial ser ativada.

Um cenário principal para um Sistema Acadêmico, que descreve o caso de uso “emitir boletim”, para atender o ator “aluno”, poderia ser descrito, nessa fase, como na Figura 21.9:

1. aluno informa DRE
2. sistema valida DRE
3. sistema acadêmico solicita senha
4. aluno informa senha
5. sistema acadêmico valida senha
6. sistema acadêmico apresenta menu
7. aluno escolhe emitir boletim
8. sistema acadêmico emite boletim
9. aluno sai do sistema

Figura 21.9.: Exemplo de cenário principal do caso de uso **emitir boletim**.

O quarto passo usa como referência o cenário principal, e outros cenários alternativos do caso de uso em uma interação mais avançada, determinando as alternativas a partir dos passos já descritos. Cada passo pode ter alternativas que levem ao fracasso, ao

sucesso parcial ou ao sucesso total. Cada condição deve ser anotada, gerando uma lista de cenários alternativos que indica o cenário, o passo e a condição que pode ocorrer nesse passo.

No exemplo anterior, a lista de condições poderia ser:

- DRE não é validado
- senha não é validada
- se passam 2 minutos sem o aluno fazer uma ação

No quinto passo são resolvidas todas as alternativas. Isso é feito escrevendo o cenário, seguindo sempre as mesmas regras do cenário principal, e também indicando, ao final do cenário, se foi sucesso, parcial ou total, ou falha, e se é necessário voltar ao fluxo original.

A técnica do caso de uso descreve as alternativas apenas no que elas são diferentes do cenário principal, isto é, elas são normalmente menos passos do que o cenário principal e se referem a ele. O resultado dessa fase é uma versão completa dos casos de uso do sistema.

Finalmente, no último passo, são tratadas as questões que são de baixo nível para a análise, mas precisam ser resolvidas para a implementação. Por exemplo, como será emitido o boletim, que dados serão usados, etc.

Um bom caso de uso corresponde a um processo elementar da organização. Ele descreve a interação de um ator principal durante uma sessão única, onde um objetivo é atingido. Essa sessão **não** leva dias ou se divide em várias utilizações do sistema. Ele deve ser uma conversação, e não um passo único como “apagar um item”. Deve ser uma tarefa, concluída em uma sessão e que produz um resultado mensurável e partindo de uma situação onde as informações estão em um estado consistente e terminando em uma situação onde as informações também estão em um estado consistente, possivelmente diferente do estado inicial.

Casos de uso devem ser pensados como os motivos atômicos, isto é, mais simples, que o usuário tem para usar o sistema. Assim, em um sistema acadêmico, um ator “aluno” pode ter como objetivo de entrar no sistema “emitir o boletim” ou “fazer matrícula em cadeira”. É claro que um aluno específico pode entrar no sistema para fazer essas duas coisas, porém no processo de análise de sistemas elas são tratadas em separado, porque é possível entrar no sistema para fazer essas atividades, mas dentro delas não há nada menor para ser feito, logo elas são atômicas.

O mesmo exemplo pode ser dado com um correntista de banco. Ele pode usar o caixa automática para sacar dinheiro, ou para emitir cheques, logo esses são casos de uso. Ele também pode fazer as duas coisas em uma sessão, mas como essa atividade seria uma composição de outras duas que podem ser feitas isoladamente, isto é, podem ser um objetivo único de um correntista, então as duas atividades isoladas são casos de uso, mas a atividade composta não. Além disso, não faz sentido um usuário entrar no sistema só para fazer um passo menor do que isso, como “entrar senha”.

A lista a seguir mostra algumas perguntas que podem ajudar a eliciar os casos de uso.

21. Casos de Uso

- Quais são as tarefas de um ator no negócio?
- O ator precisa ser informado de que ocorrências que ocorrem no sistema?
- O ator precisa informar o sistema de mudanças no ambiente externo?

Cockburn (2000) propõe uma receita mais detalhada, com 12 passos:

1. encontre as fronteiras do sistema, usando um diagrama de contexto ou uma lista de entradas e saídas;
2. faça um *brainstorm* e list os atores primários, gerando a **lista de atores**;
3. faça um *brainstorm* e gere os objetivos principais dos atores perante o sistema, gerando uma lista de atores e objetivos;
4. escreva os casos de uso no nível de sumário de todos os casos encontrados;
5. considere e revise os casos de uso estratégicos, adicione, retire, junte e separe objetivos;
6. escolha um caso de uso para expandir ou escreva narrativa para ficar acostumado com o material;
7. preencha as partes interessadas, interesses, pré-condições e garantias, e verifique esse trabalho;
8. escreve o cenário principal de sucesso, verifique frente os interesses e garantias;
9. faça um *brainstorm* e liste as principais condições de sucesso e falha alternativas;
10. escreva como os atores e os sistemas devem se comportar em cada alternativa;
11. extraia qualquer sub caso de uso que se mostre necessário, e
12. inicie do início e reajuste os casos de uso, adicione, retira, junta ou separe casos de uso, verifique para completude, legibilidade e condições de falha.

21.10. Diagramas de Caso de Uso

Um diagrama de caso de uso representa de forma muito abstrata todos os casos de uso de um sistema. Seus principais componentes são atores e casos de uso.

Um diagrama de caso de uso pode ser construído com apenas 3 figuras básicas: atores, casos de uso e associações, apresentadas na Figura 21.10,

Outros símbolos, característicos dos casos de uso, também existem, representando relações entre atores e casos de uso. Esses símbolos são: herança, inclusão e extensão.

Finalmente, símbolos genéricos de UML como sistema/partição, comentário, etc., também podem ser usados.

Atores e casos de uso podem se relacionar de várias maneiras, como é mostrado na Tabela 21.6.

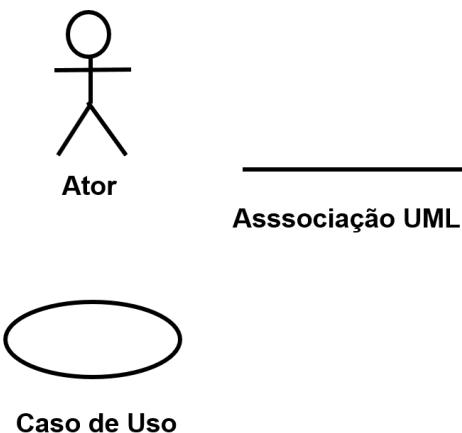


Figura 21.10.: Os elementos básicos de um diagrama de caso de uso são: atores, casos de uso e associações. Fonte: do autor

Tabela 21.6.: Associações que podem ocorrer entre atores e casos de uso.

	ator	caso de uso
ator	generalização	associação
caso de uso	associação	«include» «extend» generalização

21.10.1. Especificando o *Subject*

De acordo com o padrão UML(OMG, 2017), é possível indicar a que sistema caso de uso se refere, o que o padrão chama de *subject*, por meio de uma caixa, como feito na Figura 21.11.

21.10.2. Generalização entre Atores

Muitas vezes diferentes atores tem características comuns. Se estas características comuns puderem ser descritas como uma relação de especialização/generalização, podemos representar isso diretamente em um diagrama de caso de uso por meio da relação de generalização (usando uma seta com ponta triangular e linha cheia).

Múltiplos atores podem ter papéis comuns ao interagir com um caso de uso. A relação de generalização pode ser usada para simplificar relações entre muitos atores e um caso de uso. Ela também mostra que uma instância de um ator especializado por fazer tudo que outro tipo de ator faz.

Um exemplo típico é o da existência, em uma organização, de funcionários e gerentes.

21. Casos de Uso

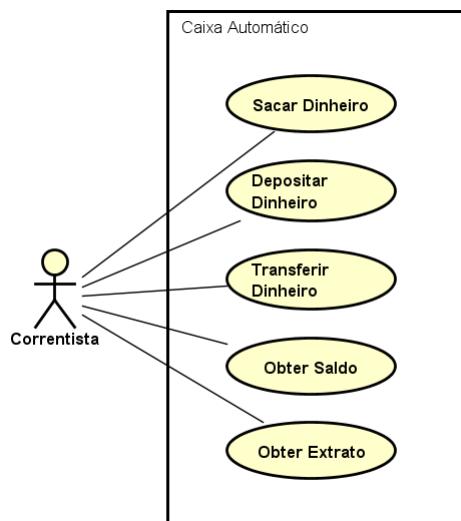


Figura 21.11.: Indicando o sistema ao qual os casos de uso se referem. Fonte:do autor.

Normalmente, tudo que um funcionário pode fazer um gerente também pode fazer, mas a recíproca não é verdadeira. Assim, é possível criar vários casos de uso para o ator funcionário e fazer o ator gerente herdar de funcionário, adicionando-se então os casos de uso exclusivos de gerente. A Figura 21.12 mostra uma visão parcial de um sistema dedicado ao controle de funcionários, onde um ator gerente é uma especialização do ator funcionário. Dessa forma o gerente também pode fazer solicitar férias ou solicitar dispensa médica, mesmo que essa ligação não apareça diretamente no diagrama.

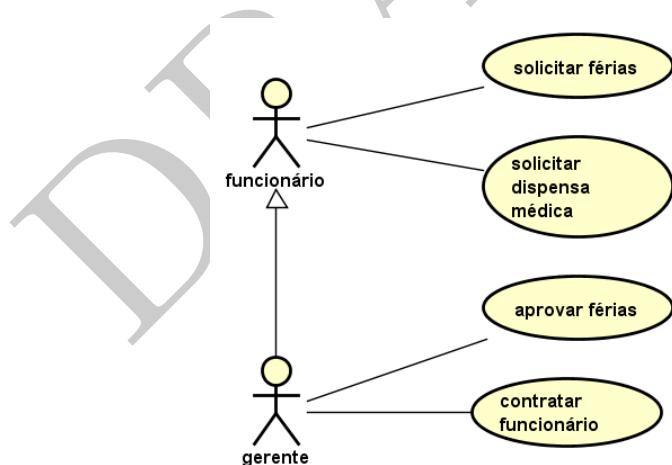


Figura 21.12.: Nesse diagrama de casos de uso, o gerente é um especialização de funcionário, logo pode fazer tudo que o funcionário faz, mais alguns casos de uso que são exclusivos dele. Fonte: do autor

Também é possível usar a relação de generalização para criar atores abstratos, que representam um comportamento comum entre vários atores concretos. Um ator abstrato

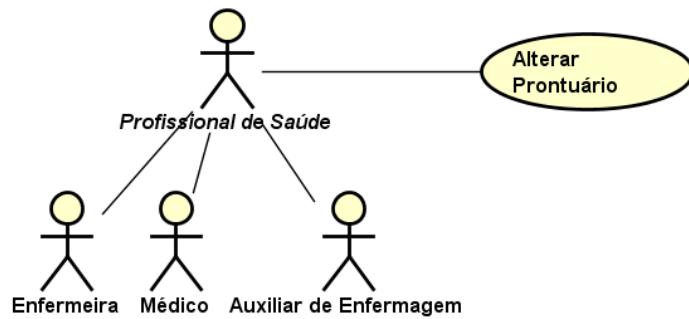


Figura 21.13.: Um *Profissional de Saúde* é a generalização de Enfermeira, Médico ou Auxiliar de Enfermagem, mas nenhum ator é um profissional de saúde se não for de um dos outros tipos. Fonte: do Autor.

não existe na prática no mundo real, como o que é demonstrado no exemplo a seguir. No exemplo da Figura 21.13, 3 atores diferentes são considerados especialização de “*Profissional de Saúde*”, que é um ator abstrato, com o nome em itálico, que existe conceitualmente mesmo no mundo real, porém não existe nenhum exemplo de um profissional de saúde que não seja enfermeira, médico ou auxiliar de enfermagem.

21.10.3. Relações Entre Casos de Uso

Casos de uso podem se relacionar de 3 formas:

- pela inclusão de outro caso de uso;
- pela extensão de outro caso de uso, e
- pela generalização/especialização de outro caso de uso.

Em UML, esses relacionamentos são conhecidos como include, extend, e a generalização propriamente dita, sendo representados como na Figura 21.14.

Inclusão de Casos de Uso

O relacionamento de inclusão é o mais simples de se compreender, pois representa uma relação entre um caso de uso básico e um caso de uso incluído. Isso significa que caso de uso incluído é explicitamente inserido no caso de uso base, de forma semelhante a uma chamada de função. A Figura 21.15 apresenta um exemplo dessa associação.

O relacionamento de inclusão é usado para fatorar um comportamento comum entre dois ou mais casos de uso. Ele evita que tenhamos que descrever o mesmo comportamento duas vezes dentro dos respectivos casos de uso, aumentando a consistência e permitindo o reuso.

Também é possível usar o relacionamento de inclusão apenas para fatorar e encapsular comportamento de um caso de uso base, de forma a simplificar fluxo complexo de eventos

21. Casos de Uso

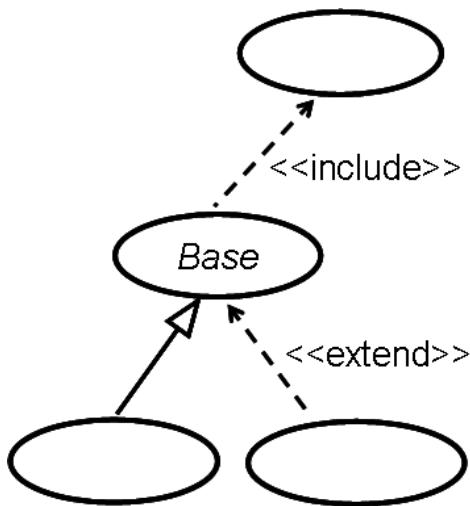


Figura 21.14.: Representações das associações entre casos de uso.

ou remover da parte principal do caso de uso um comportamento que não é parte do objetivo primário. Nesse caso deve se considerar a complexidade do caso de uso descrito de forma completa em comparação com a complexidade de ter mais de um caso de uso sendo descrito para representar só um objetivo do usuário.

É importante entender que um caso de uso incluído é executado totalmente quando é chamado e, se não deve ser executado, a decisão é do caso de uso chamador. Alguns autores dizem que o caso de uso incluído é obrigatório, mas isso só é verdade dentro do fluxo onde ele ocorre, ou seja, um caso de uso incluído pode ocorrer em um cenário alternativo e, no caso, não ser aleatório¹.

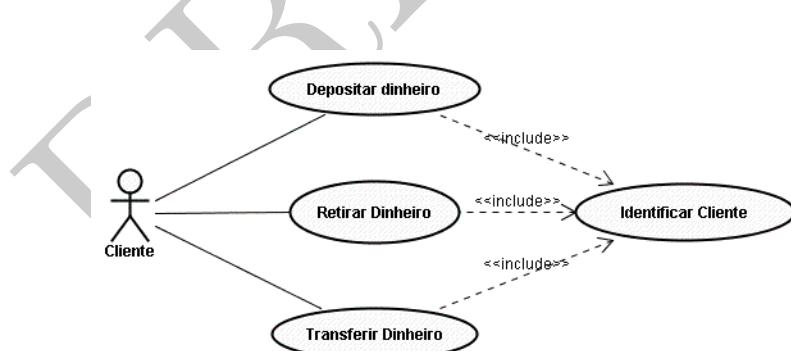


Figura 21.15.: Exemplo da associação de inclusão entre casos de uso

¹Mesmo assim, alguns lugares usam como regra que casos de uso incluídos estão sempre no cenário principal e casos de uso que estendem estão sempre em um cenário alternativo. Isso é apenas uma decisão local que pode tornar o trabalho mais fácil e consistente entre várias pessoas do grupo de análise de sistema, mas não tem base no padrão UML

Extensão de Casos de Uso

A relação de extensão descreve que um caso de uso estende o comportamento de seu caso base, o que acontece apenas se uma condição de extensão for verdade. A relação de extensão é originária do uso de *patches* em sistemas que não podiam ter seus casos de uso originais alterados para aceitar novos comportamentos e pode ser compreendida como uma forma de *patch*, do mesmo jeito que a relação de inclusão pode ser compreendida como uma chamada de função. A Figura 21.16 apresenta um exemplo de relação de extensão em um caso de uso.

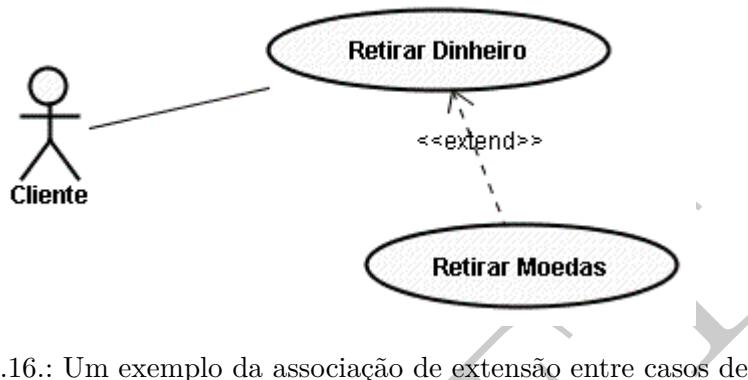


Figura 21.16.: Um exemplo da associação de extensão entre casos de uso.

Uma diferença principal entre inclusão e extensão é que a extensão sempre é chamada pelo caso de uso base, e só na extensão é que é decidido se ela deve fazer algo ou simplesmente retornar para o caso de uso base.

Extensões são tipicamente chamadas para fazer funções adicionais em cenários alternativos ou para descrever uma correção, um *patch*, no cenário principal. Por exemplo, uma possível extensão a um cenário do caso de uso “pagar com cartão de crédito”, em um terminal de ponto de venda², seria “pagar a prazo com cartão de crédito”.

Generalização de Casos de Uso

O relacionamento de generalização, como estamos habituados, descreve um comportamento geral compartilhado pelo caso de uso que herda (filho) com o seu parente. Ela descreve que o “filho” tem o mesmo comportamento geral do pai, porém com alguma diferenciação (especialização).

Esse relacionamento deve ser utilizado para mostrar comportamento, estrutura ou objetivos comuns entre diferentes casos de uso; para mostrar que os casos de uso “filhos” formam uma família de casos de uso com alguma similaridade; ou para assegurar que o comportamento comum se mantém consistente. A figura a seguir mostra um exemplo de herança.

²A famosa máquina de cartão de crédito

21. Casos de Uso

Uma instância de caso de uso executando um caso especializado vai seguir o fluxo de eventos descritos pelo caso parente, inserindo comportamento adicional e modificando seu comportamento como definido no fluxo de eventos do caso especializado.

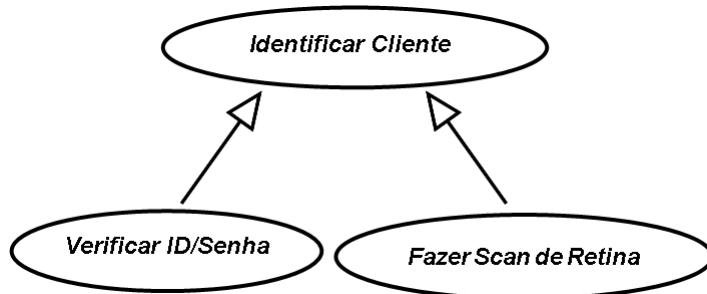


Figura 21.17.: Exemplo da associação de herança entre casos de uso

21.10.4. Usando Cardinalidades em Casos de Uso

O uso mais comum de casos de uso, tendo em vista sua utilidade, é sem cardinalidades nas associações, porém é correto usá-las segundo UML.

Com o uso de indicadores de cardinalidade, o que é parte de UML, algumas informações adicionais podem ser dadas. Quando essa cardinalidade é maior que um, a especificação UML não define o que isso significa(OMG, 2017). Podem ser que os atores, múltiplos, tenham que agir em paralelo, ou em sequência, ou em outra forma.

No diagrama da Figura 21.18, existe um caso de uso apenas: aprovar empréstimo. Toda instância desse caso de uso precisa de um gerente para ser executado, porém em alguns casos especiais precisa também de um gerente regional. Gerentes e gerentes regionais, isto é, suas instâncias específicas, não precisam participar desse caso de uso. As cardinalidades marcadas indicam isso.

Também é possível que um ator execute várias instâncias de um caso de uso, ou que um caso de uso exija mais de uma instância de um ator.

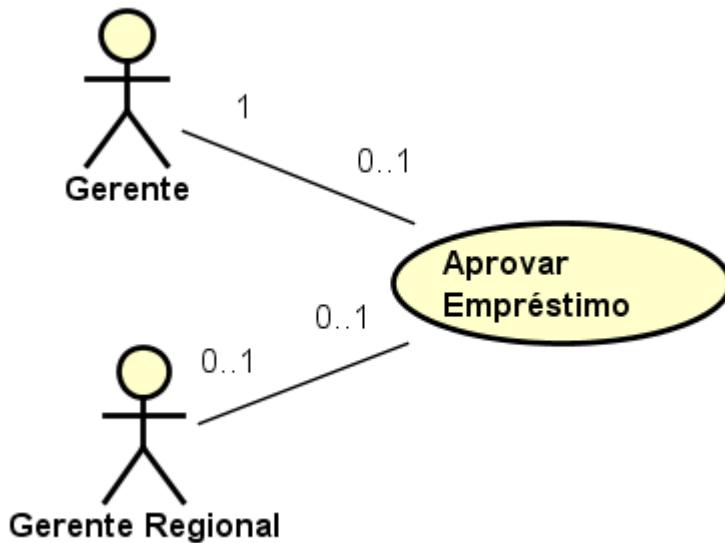


Figura 21.18.: Exemplo do uso de cardinalidades em casos de uso. Fonte: do autor

21.11. O Que o Diagrama de Caso de Uso Não Informa

Um diagrama de caso de uso não dá algumas informações.

Os atores principais não são informados em um diagrama de caso de uso. Mais que isso, não há nenhuma informação que ajude a entender qual a participação de um ator em um caso de uso que esteja associado a vários atores. Para facilitar o entendimento, muitos diagramas, principalmente em livros, colocam o ator principal à esquerda do caso de uso e os outros atores à direita.

Assim vendo um ou mais atores associado a um caso de uso, não é possível saber:

- se ele é o ator principal;
- se ele envia ou recebe informações para o caso de uso;
- se ele participa de todas as instâncias do caso de uso, o que pode ser resolvido em alguns caso com a indicação de cardinalidade na associação, e
- se ele está no cenário principal ou só em algum cenário alternativo.

21.12. Exercícios

Exercício 21.12-1: Escreva um conjunto de casos de uso para um aplicativo de celular que sirva como relógio e despertador, permitindo vários alarmes.

Exercício 21.12-2: Vá para o site <http://jogodeanalisedesistemas.xexeo.net/> e visite a Livraria Resolve. A partir da sua visita identifique todos os atores e defina o

21. Casos de Uso

nome de todas os casos de uso que pensa necessário. Para cada caso de uso imagine um cenário principal.

DRAFT

22

Métodos Simples de Elicitação de Requisitos

Conteúdo

22.1.	A entrevista	295
22.2.	A Reunião de JAD	304

Por que técnicas de elicitação?

Esse capítulo fornece orientações para quem vai começar na análise de sistemas, não tendo uma base formal, mas sim apresentando uma coleção de recomendações de como agir.

Existem muitas técnicas avançadas de elicitação de requisitos que não cabem no contexto desse livro. Aqui trataremos de duas técnicas básicas que coleta de informações: a entrevista e reuniões de JAD.

22.1. A entrevista

Entre as técnicas mais importantes de elicitação de requisitos está a entrevista. Ela está constantemente presente dentro de outras técnicas porque, quase sempre, a Elicitação de Requisitos em algum ponto - exige comunicação direta com os usuários e outros interessados e a forma básica de comunicação, seja de que forma esteja disfarçada, é a entrevista.

A entrevista procura explicitar o pensamento do entrevistado a respeito das suas relações com seu universo em determinada instância, tanto como indivíduo quanto como

22. Métodos Simples de Elicitação de Requisitos

profissional, revelando o conhecimento do entrevistado sobre as pessoas, objetos, fatos e procedimentos com os quais interage.

Os objetivos são os mais diversos, por exemplo: estabelecer expectativas do consumidor, verificar níveis de satisfação e necessidades atuais e futuras; estudar tendências na satisfação do cliente ao longo do tempo ou avaliar programas em andamento.

O primeiro passo de uma entrevista é determinar, entre outros aspectos: seus propósitos ou objetivos; a informação necessária, e de quem obter, para alcançar esses objetivos e os recursos disponíveis para implementar e manter o processo de pesquisa. Outros fatores merecem destaque: precisão na determinação da amostra; abrangência e relevância do conteúdo da pesquisa e, essencial, a validação. Os resultados da entrevista são sumariados em um relatório interpretativo que inclui relato sobre os achados e as recomendações mais importantes. A análise pode ser qualitativa ou quantitativa. Normalmente, as entrevistas abertas estão no primeiro caso, enquanto que os questionários são analisados quantitativamente.

A responsabilidade do entrevistador também é grande. Normalmente, além das entrevistas, é ele quem integra as diferentes interpretações, objetivos, metas, estilos de comunicação, e o uso de terminologia. Há também outras tarefas complexas como decidir a oportunidade ou não de incluir certo tipo de informação no conjunto inicial de requisitos. Finalmente, entrevistar e integrar toma tempo. Como os requisitos são voláteis, quanto mais longo o processo mais facilmente os requisitos deixam de atender as necessidades e expectativas dos interessados. Todos esses encargos recomendam que o analista conheça tanto as técnicas de desenvolvimento quanto o domínio no qual se insere.

Existem vários tipos de entrevistas. Durante o processo de análise, elas normalmente seguem o seguinte processo:

1. Introdução inicial
2. Familiarização com o ambiente
3. Busca de fatos
4. Verificação de informações conseguidas de outras formas
5. Confirmação de informações conseguidas com os candidatos
6. Acompanhamento, amplificação ou clarificação de entrevistas anteriores.
7. Outra grande variável da entrevista é o seu objetivo, entre outros:
8. Levantar informações sobre a organização
9. Levantar informações sobre uma função de negócio
10. Levantar informações sobre um processo ou atividade
11. Descobrir problemas
12. Verificar fatos previamente levantados
13. Fornecer informação
14. Obter dicas para entrevistas futuras

Há várias formas de entrevista, entre elas: entrevista por questionário; entrevista aberta; entrevista estruturada.

22.1.1. Entrevista por Questionário

O questionário é muito usado como técnica de entrevista, principalmente em pesquisas de mercado e opinião. Exige preparação elaborada. Alguns aspectos particulares do processo merecem destaque: emprego de vocabulário adequado para o público entrevistado; inclusão de todos os conteúdos relevantes e de todas as possibilidades de respostas; cuidado com os itens redundantes ou ambíguos, contendo mais de uma idéia ou não relacionados com o propósito da pesquisa; redação clara; execução de testes de validade e confiabilidade da pesquisa.

Há uma tensão não resolvida entre o uso do questionário como um evento interativo ou como instrumento neutro de medida. Por um lado, como entrevista, é visto como uma interação. Por outro lado, no interesse de torná-lo um instrumento, muitos recursos da interação existentes na conversação não são permitidos, suprimindo recursos de medida de incertezas de relevância e interpretação.

Dificuldade importante é o fato das palavras possuírem significados diferentes para pessoas diferentes em diferentes contextos. Em interações normais essas questões de interpretação são negociadas entre os participantes, mas em entrevistas com questionários o treinamento e o método utilizados proíbem essa negociação. Além disso, há necessidade do uso de técnicas específicas nem sempre do conhecimento dos projetistas - para a construção e aplicação de questionários. A menor ambiguidade é uma das principais vantagens da entrevista via questionário.

Para gerar bons itens de questionário, devemos:

- evitar palavras ambíguas ou vagas que tenham significados diferentes para pessoas diferentes;
- redigir itens específicos, claros e concisos e descarte palavras supérfluas;
- incluir apenas uma idéia por item;
- evitar itens com categorias de respostas desbalanceadas;
- evitar itens com dupla negação;
- evitar palavras especializadas, jargões, abreviaturas e anacronismos;
- redigir itens relevantes para a sua pesquisa;
- havendo necessidade de opiniões francas sobre assuntos controversos, evitar itens demográficos que identifiquem os entrevistados.

A construção e análise de questionários é uma técnica estudada em algumas áreas, como Marketing e Psicologia.

22.1.2. Entrevista Aberta

Esse tipo de entrevista evita muitos dos problemas dos questionários, porém também cria outros. O entrevistador formula uma questão e permite que o entrevistado responda como quiser. O entrevistador pode pedir mais detalhes, mas não determina os termos da entrevista. Permanecem, entretanto, as questões: as perguntas podem ser respondidas?

22. Métodos Simples de Elicitação de Requisitos

A resposta faz parte do repertório normal do discurso do entrevistado? Há muitas coisas que as pessoas sabem fazer, mas tem dificuldade de descrever, como há também o conhecimento tácito, que é de difícil elicitação.

Os benefícios das entrevistas abertas são: a ausência de restrições, a possibilidade de trabalhar uma visão ampla e geral de áreas específicas e a expressão livre do entrevistado.

Há desvantagens também. A tarefa de entrevistar é difícil e desgastante. O entrevistador e o entrevistado precisam reconhecer a necessidade de mútua colaboração ou o resultado não será o desejado. Há falta de procedimentos padronizados para estruturar as informações recebidas durante as entrevistas. A análise da informação obtida não é trivial. É difícil ouvir e registrar simultaneamente; principalmente, porque há fatos que só tomam importância depois de outros fatos serem conhecidos, e aí ele já não foi registrado. Daí a importância da gravação e da respectiva transcrição; fica mais fácil selecionar e registrar o que é relevante e validar com o entrevistado.

São exigências para o relacionamento entre os participantes de uma entrevista: respeito ao conhecimento e habilidade do especialista; percepção de expressões não verbais; sensibilidade às diferenças culturais; cordialidade e cooperação.

22.1.3. Entrevista Estruturada

A entrevista estruturada extrai informações sobre perguntas específicas. Nesse tipo de entrevista, é importante entrevistar a pessoa certa. É uma boa técnica para ser usada após uma pesquisa com questionário, quando é possível selecionar, entre as respostas, as partes interessadas com maior potencial de geração de outras informações. Suas vantagens são: Respostas diretas, com menos ambiguidade, informação detalhada. Sua desvantagem básica é que as questões relevantes precisam ser identificadas com antecedência.

22.1.4. O processo da entrevista

O processo de entrevista não se resume ao ato específico da entrevista. Na verdade ele começa muito antes e acaba muito depois. O processo normal da entrevista inclui:

1. Determinação da necessidade da entrevista
2. Especificação do objetivo da entrevista
3. Seleção do entrevistado
4. Marcação da entrevista
5. Preparação das questões ou do roteiro
6. A entrevista propriamente dita
7. Documentação da entrevista, incluindo os fatos e a informação conseguida durante a entrevista.
8. Revisão da transcrição da entrevista com o entrevistado
9. Correção da transcrição
10. Aceitação por parte do entrevistado

11. Arquivamento

22.1.5. Preparando a entrevista

A preparação é uma necessidade básica da entrevista. Não só precisamos preparar a entrevista propriamente dita, mas também preparar a nós mesmos, como entrevistadores, e ao entrevistado.

Uma entrevista deve ter um objetivo. As perguntas ou o roteiro devem ser coerentes. Para isso é importante a determinação desse objetivo,. O entrevistado deve ter noção clara da finalidade da entrevista e perceber sua utilidade. Isso se faz por meio de palestras, textos de divulgação e, principalmente, se explicando ao entrevistado, no início da entrevista, seu objetivo e importância.

Muitas vezes esse objetivo não é específico, principalmente na fase inicial do projeto. Mas deve ser claro, isso é, quando expressado deve permitir que entrevistador e entrevistado compreendam o motivo da entrevista. Assim, no início do projeto os objetivos podem ser: Conhecer o ambiente de trabalho, Levantar expectativas iniciais dos usuários. Já com o passar do tempo do projeto o objetivo se torna mais detalhado, como em: Levantar os documentos utilizados no processo de compra ou Avaliar as telas relativas ao cadastro de bens.

A escolha do entrevistado é o segundo aspecto importante. Devem ser escolhidas as pessoas que permitam obter no final das entrevistas uma visão clara e o mais completa possível do problema, das diversas formas de analisá-lo e solucioná-lo. Nunca se deve tratar um problema a partir de um único nível funcional, nem de uma única visão organizacional, pois estaríamos correndo o sério risco de obter uma visão distorcida.

Devemos lembrar que o sistema afetará todos os níveis funcionais e departamentos da instituição. Dependendo do tipo de entrevista, será necessário um roteiro ou um questionário. No início da análise os roteiros levam a execução de entrevistas abertas, no final geralmente temos entrevistas por questionários. Entrevistas estruturadas são preparadas principalmente para esclarecimento de processos e atividades. Todos os roteiros e questionários devem seguir um modelo padrão, incluindo a apresentação e a conclusão da entrevista. Quanto maior o número de entrevistadores, maior a importância de seguir um padrão.

Outros aspectos fundamentais a serem preparados são:

- A linguagem
- A coerência das perguntas
- A programação dos horários

É importante estar preparado para a linguagem a ser usada na entrevista. Nisso influenciam vários fatores, como nível cultural do entrevistado, terminologia do trabalho, jargão da área, etc. Devemos evitar ao máximo usar os nossos termos técnicos e aproveitar ao máximo a oportunidade de aprender os termos técnicos do entrevistado. Se necessário,

22. Métodos Simples de Elicitação de Requisitos

ler um pequeno texto esclarecedor sobre a área e, sempre, ler o glossário do projeto. O entrevistador deve sempre esclarecer com o entrevistado todas as dúvidas quanto ao vocabulário utilizado no ambiente onde o sistema será implantado.

Marque a entrevista com antecedência, com confirmação de data, hora, duração e local por todas as partes.

As seguintes regras devem ser observadas quanto ao horário:

- As entrevistas devem ter 30, 60 ou 90 minutos e, no máximo, duas horas.
- As entrevistas iniciais podem ser mais longas, enquanto as entrevistas finais devem ser mais rápidas.
- Evite horários perto da hora do almoço ou no final de expediente, ou em uma tarde de sexta-feira ou véspera de feriado.
- Obtenha o telefone do entrevistado, para poder avisá-lo de sua ausência em caso de urgência.
- Chegue sempre 10 minutos adiantado e esteja preparado para esperar e para ter que encerrar a entrevista mais cedo, principalmente com a alta gerência.
- Se possível, caso a entrevista seja mais curta que o combinado, marque imediatamente a sua continuação.
- Quanto ao material necessário para uma entrevista, além do roteiro:
- Prepare e teste o equipamento, principalmente um gravador. Atualmente existem bons gravadores digitais a preços razoáveis no mercado.
- Tenha pelo menos 2 horas de gravação e um jogo de pilhas extras.
- Tenha um caderno de anotações (é melhor que um bloco) reservado para o projeto. Canetas de várias cores, lápis, borrachas.

22.1.6. Realizando a entrevista

O objetivo normal de uma entrevista é conseguir informações do entrevistado, para isso devemos fazer não só que o usuário fale, mas também que ele pense. É importante para o entrevistador não assumir nada e não fazer pré-julgamentos, caso contrário correrá o risco de fazer uma entrevista viciada.

O entrevistador deve manter o controle do assunto da entrevista. Não deixe o entrevistador mudar de assunto ou tergiversar, mantendo suas perguntas direcionadas para o objetivo da entrevista.

As duas principais armas do entrevistador são a **pergunta** e o **silêncio**. Para perguntar devemos ter consciência do tipo de pergunta que escolhemos. Se quisermos que o usuário explique algo, então devemos utilizar uma pergunta aberta. Isso é muito comum em entrevistas abertas no início da análise.

O importante é fazer o usuário pensar, para isso, o entrevistador deve evitar perguntas que contenham a própria resposta ou as que podem ser respondidas apenas com um sim ou não. As perguntas fechadas devem ser utilizadas para tirar dúvidas do entrevistador.

Use questões começando com quem, qual, quando, onde, porque e como sempre que possível. Tente completar o ciclo (quem qual quando onde porque como) para todos os assuntos.

Em dúvida, pergunta novamente de outra forma. O entrevistador deve pedir que processos complicados sejam explicados mais de uma vez, preferencialmente sob perspectivas diferentes. Desenhá-los em quadros brancos ou em papel pode ser uma boa solução para eliminar qualquer dúvida.

É importante estabelecer exemplos concretos para o que está sendo descrito pelo usuário. Também, em caso de uma dúvida, é melhor descrever um exemplo concreto (o que aconteceria se ...) do que uma dúvida abstrata. O entrevistador deve estar consciente que é muito difícil encontrar um entrevistado capaz de raciocinar plenamente de forma abstrata sobre um problema. Mesmo nesse caso, normalmente a forma abstrata se resume ao caso perfeito, sendo que as exceções são melhores explicadas com exemplos.

Não tenha pressa, não responda pelo entrevistado. Não se preocupe com a demora para responder ou o silêncio. O silêncio, inclusive, é uma boa tática para fazer o entrevistado continuar falando. Deixe o entrevistado pensar, olhe para ele curiosamente.

Antes de mudar de assunto, verifique sua compreensão, explicando de forma resumida o que acabou de ouvir. Isso permite ao entrevistado pensar e dar uma clarificação se necessário. Esteja atento para a ausência de críticas por parte do candidato. Isso pode ser causado pela falta de confiança do entrevistado em você ou porque o problema é constrangedor demais para ser tratado. O analista deve constatar esse fato no processo de análise, mas não durante a entrevista.

As interrupções causadas por fatores externos (telefone, pessoas que entram e que saem, etc.) podem ser importantes em um processo de entrevistas. Se uma entrevista for prejudicada por muitas interrupções, isso deve ficar no relatório da entrevista. No relatório, também devemos separar o que é fato do que é opinião.

22.1.7. O Comportamento do Entrevistador

Esteja atento ao próprio comportamento. Lembre-se que não importa sua intenção ao fazer ou deixar de fazer algo, mas a interpretação que o entrevistado dará ou que fizer ou não fizer.

No passado era comum que consultores sempre se vestissem de terno, até mesmo apenas ternos escuros. A maioria das empresas hoje utiliza um código de vestimenta mais informal. A regra mais atual é que o entrevistador ou consultor tome cuidado para não provocar um grande desnível entre a sua roupa e a roupa do entrevistado ou cliente, se adaptando as normas de vestimenta do cliente (ou do mercado ao qual o cliente pertence).

Fisicamente, não faça movimentos desnecessários como bater o lápis na mesa, mexer as chaves no bolso, sacudir ou bater os pés, etc. Movimentos automáticos e cacoetes distraem o entrevistado e, além disso, são interpretados como falta de atenção. Não

22. Métodos Simples de Elicitação de Requisitos

fume, mas também não evite que seu entrevistado fume. Não constranja o entrevistado comentando sobre os males do fumo. Não peça bebidas ou alimentos, como café, mas pode aceitar o que for oferecido. Se necessário, pode pedir água.

Estabeleça um horário para a entrevista e o cumpra rigidamente. Devido aos constantes problemas de trânsito da cidade, e a necessidade de se identificar para seguranças e secretárias, o entrevistador deve sempre planejar chegar ao local com uma folga de tempo, algo em torno de 15 minutos.

Mantenha o interesse. Tome notas, mas não seja obsessivo, principalmente não interrompa o candidato para manter suas notas atualizadas. Se autorizado, grave a entrevista e a reveja mais tarde se necessário. Escute ativamente sem interromper. O entrevistado é quem deve falar a maior parte do tempo.

Utilize um tom educado e cortês. Não seja engraçado, sarcástico ou depreciativo. Não faça comentários pejorativos ou preconceituosos. Não faça comentários sobre política e religião, ou outro tema controverso. Seja cordial, mas sem deixar de ser profissional. Pergunte e responda com cortesia e honestidade. Não de opiniões particulares, mesmo quando pedido. A entrevista é o momento de levantar informações, não de emitir-las.

Não deixa um entrevistado informações passadas por outros entrevistados. Educadamente, responda que não cabe a você decidir ou opinar.

Evite, de toda a forma, confrontar o entrevistado. Não torne a entrevista um interrogatório. Evite discutir, mesmo que não concorde com o usuário. Em caso de discussão, defina claramente o motivo do desacordo, seja ele motivado por fato ou por opinião. Utilize perguntas para restabelecer a comunicação em caso de desacordo. Se necessário, peça desculpas.

Se necessário elucidar algo que foi mal explicado, lembre-se que você não entendeu e que precisa de maiores detalhes. Se os detalhes não estiverem disponíveis, dê ao entrevistado a chance de enviá-los mais tarde.

Basicamente o entrevistador deve ser muito educado.

22.1.8. Roteiro Básico

1. Apresente-se ao entrevistado: Olá, muito prazer, eu sou fulano-de-tal, responsável por parte do projeto XYZ. Apresente seu cartão de visitas se for o primeiro encontro.
2. Informe ao entrevistado o motivo da entrevista e porque foi selecionado: Estou aqui para levantar o funcionamento da sua área, e seu nome foi escolhido por ser o funcionário mais experiente ou Estou aqui para levantar o funcionamento da atividade X, que é de sua responsabilidade.
3. Deixe clara a idéia que o conhecimento e as opiniões do entrevistado são importantes e serão úteis no processo de análise
4. Diga o que vai acontecer com a informação levantada
5. Garanta que o entrevistado lerá a transcrição da entrevista e terá a oportunidade

de corrigi-la, garanta que nada será passado a outras pessoas sem a revisão e verificação do entrevistado.

6. Determine os assuntos confidenciais ou restritos a serem tratados na entrevista
7. Deixe claro que não haverá consequências negativas em função do resultado da entrevista
8. Solicite permissão para gravar a entrevista
9. Se autorizado, inicie a gravação com um texto de apresentação: Entrevista realizada no dia X....
10. Faça a entrevista até faltarem 5 ou 10 minutos para o tempo determinado
11. Avise ao entrevistado que o tempo está acabando e pergunte se gostaria de adicionar alguma informação
12. Solicite ao candidato que responda as perguntas de conclusão
13. Conclua a entrevista de forma positiva, relatando brevemente o resultado positivo alcançado.
14. Se necessário, marque outra entrevista.
15. Entregue ao candidato o formulário de avaliação de entrevista e o envelope correspondente. Ensine-o a enviar a avaliação preenchida.
16. Despeça-se educadamente, agradecendo a atenção e o tempo dispensado.

Muitas vezes a entrevista é precedida por um bate-papo informal de apresentação. Tente manter essa conversação em um tempo mínimo razoável.

22.1.9. Documentando a Entrevista

A entrevista deve ser documentada logo após sua realização. Ao documentá-la rapidamente, estará garantindo que recuperará mais informação.

A documentação da entrevista deve fornecer a seguinte informação.

- A data, hora e local da entrevista.
- Nome do entrevistador
- Cargo do entrevistador
- Nome do entrevistado
- Função do entrevistado e a descrição desse cargo
- Se necessário, informações de background do entrevistado, como experiência no cargo ou com computadores.
- Organograma do entrevistado (superior imediato, colegas do mesmo nível, subordinados).
- O objetivo da entrevista
- Nomes e títulos de todos os outros presentes na entrevista
- Uma descrição completa dos fatos descritos e opiniões do entrevistado
- Opcionalmente, uma transcrição da entrevista, possivelmente expurgada das falas que não tinham relação com o assunto da entrevista.
- Todas as conclusões tiradas dos fatos e opiniões como apresentados
- Todos os problemas de negócio levantados durante a entrevista

22. Métodos Simples de Elicitação de Requisitos

- Exemplos de todos os relatórios, diagramas, documentos, etc., discutidos durante a entrevista.
- Todos os desenhos e diagramas feitos a partir ou durante a entrevista
- Qualquer comentário relevante feito pelo entrevistado
- Todos os números relevantes (quantidades, volume de dados, etc.) coletados durante a entrevista.

É importante notar que o relatório da entrevista deve ser aceito pelo entrevistado. É normal o entrevistado remover alguma coisa ou colocar algo a mais. O analista deve ficar atento aos motivos do usuário em fazer modificações.

Se houver discussão quanto à interpretação de algo e o analista achar essencial manter sua versão no relatório, deve também permitir que o entrevistado coloque sua versão.

22.1.10. As perguntas de conclusão

Ao final da entrevista, é importante realizar uma avaliação da percepção do entrevistado sobre a entrevista que acabou de ser realizada. Para isso é necessário que seja respondido um formulário, contendo perguntas como:

- Você acha que essa entrevista cobriu tudo que era necessário?
- Você acha que foram feitas as perguntas certas?
- Você acha que era a pessoa mais certa para responder essas perguntas?

22.2. A Reunião de JAD

Outra técnica importante de elicitação de requisitos, que merece um tratamento separado, é o JAD. Muitas vezes, quando um grupo de informática entrega um sistema de informação aos seus clientes escuta a frase: não era isso que eu queria! Isto acontece porque os desenvolvedores não conseguiram levantar com os usuários suas verdadeiras necessidades.

Este problema de comunicação pode ter diversas causas: linguagem especializada de ambas as partes, desconhecimento da área de atuação pelos desenvolvedores, preocupações com a tecnologia empregada ao invés das necessidades dos usuários, etc. Na fase inicial do projeto, a dificuldade de comunicação entre clientes e equipe de desenvolvimento é a principal causa de defeitos que serão encontrados no produto final.

Para enfrentar os problemas de comunicação entre desenvolvedores e usuários, foram desenvolvidos, ao final da década de 1970, vários métodos onde o desenvolvimento de sistemas é baseado na dinâmica de grupo.

Na forma tradicional de desenvolver sistemas os analistas entrevistam os usuários, um após outro, tomando notas que são mais tarde consolidadas e então validadas com o usuário, finalmente se transformando em um documento de análise.

O **JAD, Joint Application Design**, ou Método de Projeto Interativo, substitui as entrevistas individuais por reuniões de grupo, onde participam representantes dos usuários e os representantes dos desenvolvedores.

Quando o método é aplicado de forma correta, os resultados alcançados ultrapassam os objetivos imediatos da obtenção de informação dos usuários e cria um ambiente de alta sinergia na equipe, onde todos se sentem comprometidos com as soluções encontradas. Esse comprometimento permite que todos se considerem proprietários e colaboradores no desenvolvimento do sistema.

O objetivo do método é extrair informações de alta qualidade dos usuários, em curto espaço de tempo, através de reuniões estruturadas para alcançar decisões por consenso.

O **Líder de sessão** tem como tarefa número um conduzir o grupo para soluções de consenso. Esse Líder de sessão age como facilitador, um servidor neutro dentro do grupo que não avalia nem contribui com idéias. A responsabilidade do facilitador é sugerir métodos e procedimentos que ajudem o grupo a concentrar energia em tarefas específicas, garantindo a todos os membros do grupo condições de participar.

O **documentador** é um auxiliar imparcial do líder de sessão, responsável pelo registro das decisões e especificações produzidas. Apenas as informações relevantes são documentadas, segundo orientação do líder de sessão.

O **patrocinador** detém a autoridade formal sobre as áreas afetadas pelo sistema, estabelecendo diretrizes e objetivos do projeto.

A **equipe** é responsável pelo conteúdo da sessão, representando as áreas envolvidas no projeto.

A base de trabalho é a equipe presente na reunião. Devemos combinar algumas “regras de jogo”, de modo a alcançar o máximo de produtividade. É natural que em um grupo de 15 pessoas surjam discussões, conversas paralelas, interrupções, etc. Em respeito ao tempo precioso dos participantes vamos é necessário estabelecer um código de cooperação.

O Ambiente físico da reunião é fundamental para a produtividade dos trabalhos, e uma sala como da Figure 22.1. Os seguintes aspectos devem ser considerados:

- os participantes devem estar organizados de forma a poderem se olhar e olhar para o líder de sessão, A melhor arrumação é a em forma de “U”;
- não devem acontecer interrupções aos participantes, e
- todos devem cumprir a agenda, principalmente o início e o fim da reunião.

A forma mais produtiva de decisão do grupo é aquela obtida por **consenso**. Consenso não é a unanimidade de opiniões, mas, sim, a situação em que cada membro concorda que a solução encontrada é a melhor para o grupo e que é possível conviver com ela sem ferir convicções ou valores essenciais.

22. Métodos Simples de Elicitação de Requisitos

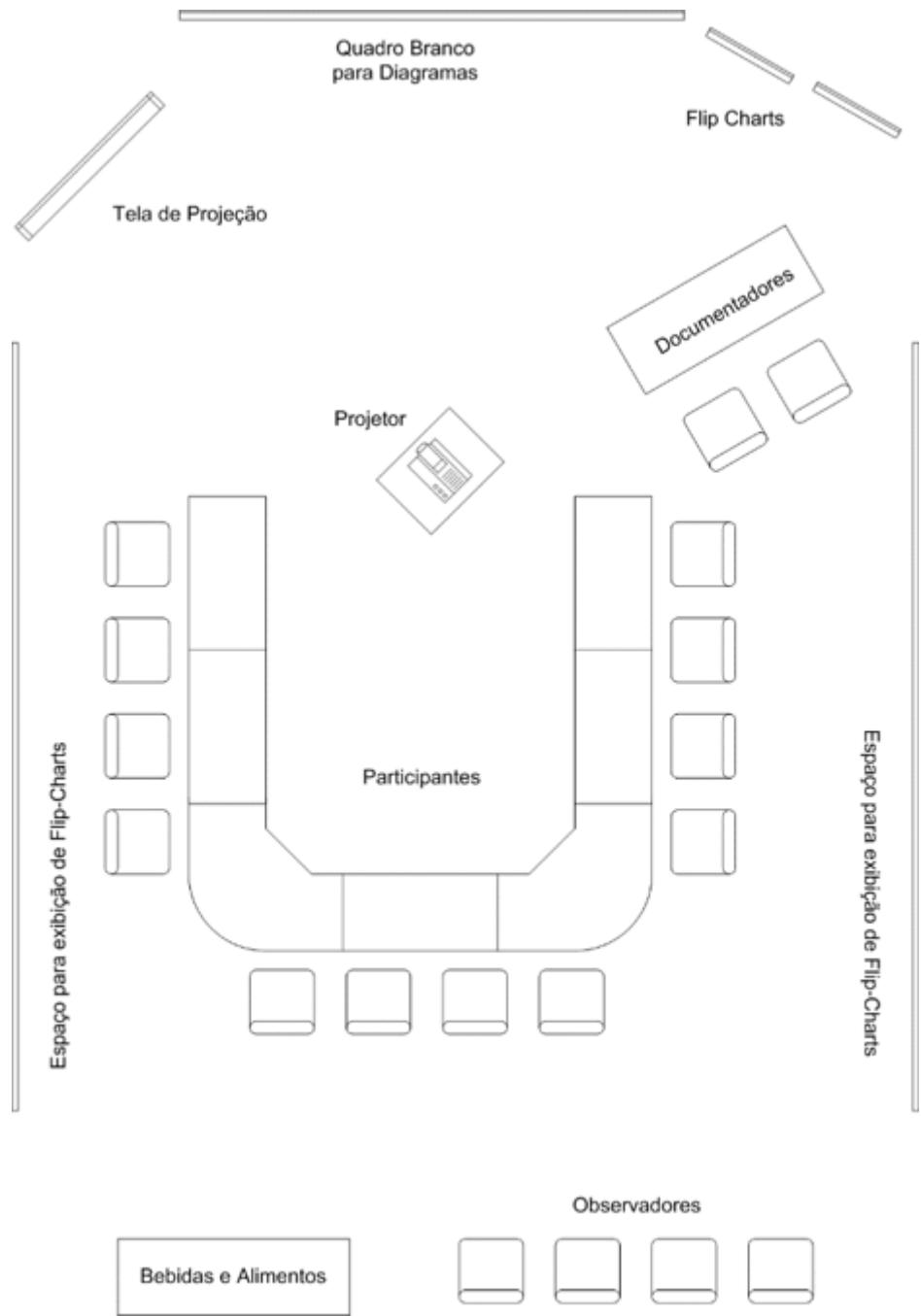


Figura 22.1.: Uma configuração ideal para realização de uma sessão de JAD.

Parte V.

Modelo Estrutural

DRAFT

DRAFT

Modelo de Entidades e Relacionamentos

Conteúdo

23.1.	Modelo Conceitual	309
23.2.	Modelo de Entidades e Relacionamentos	311
23.3.	Diagrama de Entidades e Relacionamentos	314
23.4.	Desenvolvendo um Modelo Conceitual	314
23.5.	Entidades	315
23.6.	Relacionamentos	318
23.7.	Atributos	325
23.8.	Descrição Gráfica do Modelo	326
23.9.	Exemplos de notação da Engenharia de Informação	327
23.10.	Exercícios	328

23.1. Modelo Conceitual

O objetivo da modelagem conceitual é fornecer aos desenvolvedores uma descrição abstrata de alto nível, independente de tecnologia, da informação contida no sistema. Essa descrição é conhecida como o esquema conceitual da base de dados.

A Modelagem Conceitual de Dados pode ser feita de muitas formas, algumas vezes com sutis diferenças. Alguns autores defendem a modelagem do domínio, onde tentamos descrever o domínio de aplicação sendo utilizados, outros tratam diretamente do sistema

23. Modelo de Entidades e Relacionamentos

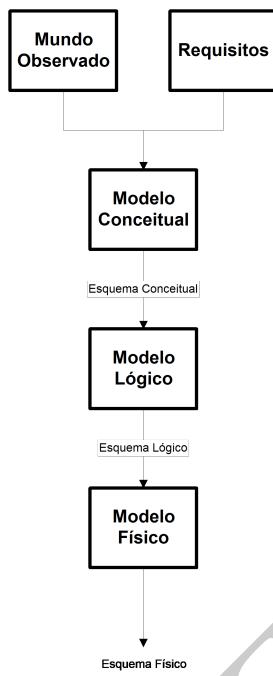


Figura 23.1.: As três camadas de modelo de dados.

sendo desenvolvido.

A principal forma de construir um modelo conceitual de dados, aplicado a dados estruturados e SGDB, é usar o Modelo de Entidades e Relacionamentos (MER)(Bertini, Ceri e Navathe, 1992), na forma do Diagrama de Entidades e Relacionamentos (DER), ambos formando o tema deste capítulo. Outro modelo comum é o Modelo Dimensional (Kimball et al., 2008).

Um dos subsídios mais importantes para a criação do DER é o conjunto de regras de negócio levantadas. Muitas das regras de negócio são representadas diretamente no modelo conceitual. Veremos mais tarde que termos e fatos são candidatos naturais para serem objetos nos nossos modelos conceituais. Não devemos confundir, porém, regras de negócio com modelos de dados. Um relacionamento em uma regra de negócio pode representar uma função do sistema, enquanto um relacionamento no MER representa algo que deve pertencer à memória do sistema.

23.1.1. Modelo Lógico

O modelo lógico descreve a informação contida no sistema de acordo com uma tecnologia adotada, sem utilizar, porém, detalhes de implementação. Ele descreve a estrutura do banco de dados que será processado por um SGDB.

Atualmente, o modelo mais utilizado é o modelo relacional. Além dele, alguns modelos distintos podem ser encontrados em aplicações especiais, como data-warehousing e

sistemas de informação geográfica.

23.1.2. Modelo Físico

No modelo físico devemos levar em conta não só a tecnologia sendo utilizada, mas também os produtos específicos e a interação do sistema com o ambiente de desenvolvimento e operação. É nessa etapa que nos preocupamos com as principais questões de desempenho, como escolha de índices, particionamento, etc.

23.2. Modelo de Entidades e Relacionamentos

O **Modelo de Entidades e Relacionamentos** (MER), segundo Cougo (1999), descreve o mundo como: “...cheio de coisas que possuem características próprias e que se relacionam entre si”. Essas descrições são feitas sobre mundos restritos, e se referem a informação necessária para que um sistema de informações cumpra suas funções. Normalmente a descrição é feita na forma de um diagrama, o diagrama de entidades e relacionamentos.

A Figura 23.2 mostra um diagrama de entidades e relacionamentos simples, usando uma evolução da notação original de Chen (1990) proposta por Bertini, Ceri e Navathe (1992). Esse diagrama, que descreve um mini-mundo sobre novelas, mostra as entidades Novela, Capítulo, Ator, Diretor, Ator Horista e Horas. Ele também apresenta os relacionamentos Dirige, Compõe, Atua, Pode ser e Trabalha, que ligam, cada um, duas dessas entidades. Os números indicam a cardinalidade da participação das entidades nos relacionamentos. Assim, a leitura do modelo da Figura 23.2 é:

- Entidades do modelo: Diretor, Novela, Capítulo, Ator, Ator Horista e Hora
- Um diretor dirige no máximo uma novela, podendo não dirigir nenhuma, e uma novela é dirigida por um e apenas um diretor.
- Um capítulo compõe uma e apenas uma novela e uma novela tem no mínimo um capítulo, podendo ter um número ilimitado deles.
- Um ator atua em uma novela, podendo não atuar em nenhuma e uma novela tem ao menos um ator, podendo ter vários.
- Um ator pode ser um ator horista e um ator horista é obrigatoriamente um ator.
- Um ator horista trabalha de zero a várias quantidade de horas, mas uma quantidade de horas é trabalhada por apenas um ator.

As coisas as quais o MER se refere podem ser pessoas, objetos, conceitos, eventos, etc., existentes ou imaginárias. Elas são classificadas em **entidades**. Alguns autores preferem o termo **tipo de entidade** ao termo entidade. Este texto usa os termos entidade e tipo de entidade indiferentemente, para representar a classe.

A priori, só exigimos de uma entidade que cada um dos seus membros possa ser identificado distintamente, isso é, tenha identidade própria. Cada coisa distintamente

23. Modelo de Entidades e Relacionamentos

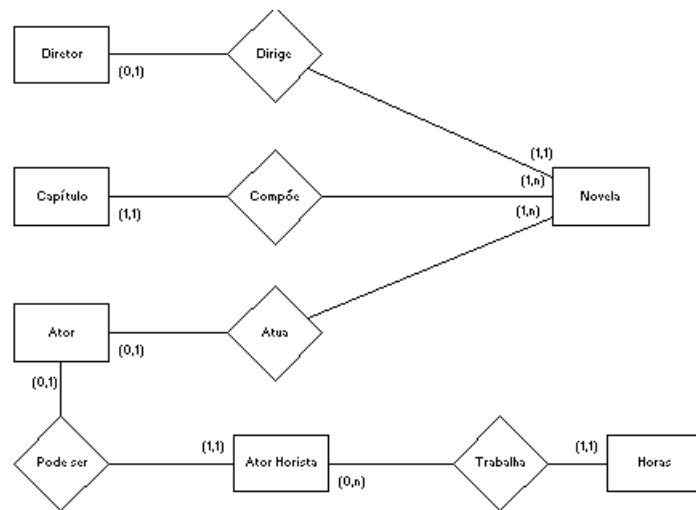


Figura 23.2.: Um diagrama de entidades e relacionamentos simples, sem mostrar atributos

identificada é uma instância.

Uma entidade representa uma classe de objetos do universo de discurso do modelo. Por exemplo, em uma universidade podemos encontrar um funcionário chamado funcionário José e uma aluna chamada Maria. José é uma instância da entidade funcionário, enquanto Maria é uma instância da entidade aluna. Funcionário e aluno são os tipos de entidade. Cada instância, então, deve poder ser identificada unicamente.

A classificação, um tipo de abstração vista no Capítulo 4, consiste em resumir uma quantidade de características comuns de objetos que tem identidade própria por meio da criação de uma classe, ou tipo, que os descreva de alguma forma. Assim, é possível saber que todos os funcionários, por serem instâncias de um mesmo tipo, possuem características comuns (como trabalhar na universidade, ter um salário, etc.).

Para representar os objetos específicos, os membros da classe, é adotado o termo **instância**, ou **instância de entidade**. Porém, no discurso normal, a palavra entidade também é muitas vezes usada no lugar de instância.

A priori, só é exigido de uma entidade que cada um dos seus membros possa ser identificado distintamente, isso é, tenha identidade própria. Cada coisa distintamente identificada é uma instância.

Existem várias propostas de representação do MER. Em quase todas elas, no diagrama de entidade e relacionamentos cada tipo de entidade é representado por um retângulo, identificado pelo nome da entidade.

Apenas algumas entidades do mundo real (ou imaginário) são de interesse para o sistema. Durante a modelagem conceitual nos preocupamos com as “coisas” que o sistema deve lembrar e colocamos essas “coisas” no modelo de entidade e relacionamentos. Uma entidade deve ser relevante para o objetivo do negócio e necessária para a sua operação.

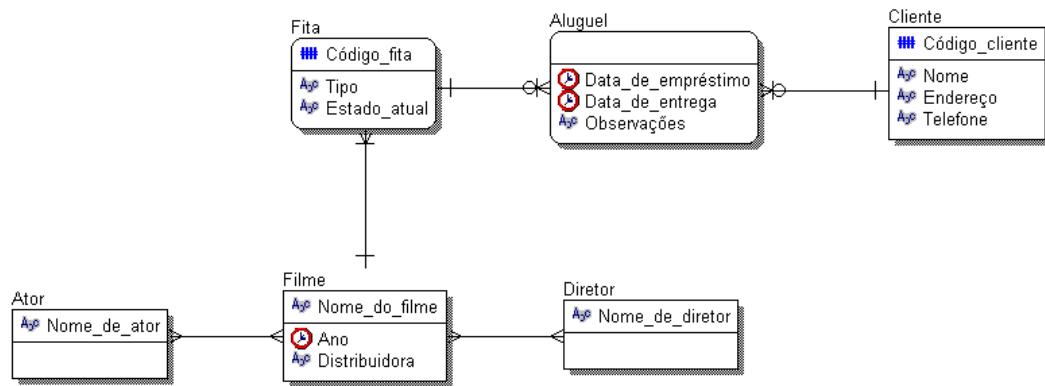


Figura 23.3.: Um modelo de entidades e relacionamentos escrito na notação da Engenharia de Informação

Cada entidade tem dois tipos de características importantes: seus atributos e seus relacionamentos.

Os **atributos** são características que toda a instância de um tipo possui, mas que podem variar entre as instâncias. Uma instância do tipo “aluno” tem os atributos “nome” e “ano de matrícula”, por exemplo.

Atributos caracterizam a informação que deve ser guardada sobre uma entidade. Só devemos colocar como atributos aquelas informações que o sistema precisa lembrar em algum momento. Assim, uma instância de “aluno” não precisa ter o atributo “nome do animal de estimação” em um sistema acadêmico, pois apesar de ser algo importante para o “aluno” propriamente dito, não tem importância alguma para o sistema.

A Figura 23.3 mostra um diagrama de entidades e relacionamentos sobre uma locadora de fitas de vídeo¹.

Cada característica deve possuir um **domínio**. O domínio indica o conjunto de valores válidos para a característica. No caso de “nome”, geralmente aceitamos qualquer sequência de caracteres, enquanto no caso de “altura” podemos aceitar apenas valores reais positivos menores que 2,5.

Atributos eram originalmente descritos por círculos no modelo E-R. As notações mais modernas anotam os atributos dentro dos retângulos da entidade a que pertencem, como na Figura 23.3.

Finalmente, como indica o nome do modelo, entidades podem se relacionar entre si. Essa característica é a principal força do modelo de entidades e relacionamentos, pois permite que, de certa forma, “naveguemos” no modelo.

Podemos indicar relacionamentos apenas pelas entidades envolvidas, como “cliente-

¹O que mostra a idade do autor.

23. Modelo de Entidades e Relacionamentos

pedido”, ou usar um termo como “solicita” que descreva o relacionamento “cliente solicita pedido”.

Modelos de Entidades e Relacionamentos para serem completos exigem também um conjunto de restrições. Algumas dessas restrições, como a cardinalidade dos relacionamentos que veremos a seguir, podem ser descritas em algumas (ou todas) notações. Porém, a maioria das descrições é muito complexa para ser descrita em um diagrama. Nesse caso são necessárias anotações ao diagrama descrevendo as descrições. Isso pode ser feito em linguagem natural ou em alguma notação formal específica, dependendo de escolhas da equipe de projeto ou do método utilizado.

23.3. Diagrama de Entidades e Relacionamentos

Existem muitas notações para Diagrama de Entidades e Relacionamentos. A notação original foi proposta por Chen e é composta de entidades (retângulos), relacionamentos (losangos), atributos (círculos) e linhas de conexão (linhas) que indicam a cardinalidade de uma entidade em um relacionamento. Chen ainda propõe símbolos para entidades fracas e entidades associativas.

As notações modernas abandonaram o uso de símbolos especiais para atributos, incluindo a lista de atributo, de alguma forma, no símbolo da entidade. Consideramos as notações mais interessantes na atualidade:

- IDEF1X, utilizada pela ferramenta ERWIN, bastante difundida no mercado(NIST, 1993);
- Engenharia de Informação, também conhecida como *crow's foot* ou notação de Martin, bastante difundida e também presente como notação alternativa no ERWIN, exemplificada na Figura 23.3;
- Notação de Bertini, Ceri e Navathe (1992), pouco difundida, mas com aspectos teóricos interessantes, ou
- Uso da UML para representar modelos de dados não-orientados a objetos.

Toda a notação moderna tem como característica importante definir a cardinalidade mínima e máxima em uma relação, não utilizar um símbolo especial para relacionamentos, mas sim a linha, e descrever atributos dentro do símbolo de entidade.

23.4. Desenvolvendo um Modelo Conceitual

Desenvolver um modelo conceitual correto para um sistema, completo e consistente, não é uma tarefa fácil. Já desenvolver um modelo conceitual razoavelmente correto, que dê uma idéia do sistema e do negócio e que, de forma evolutiva, resulte em um modelo conceitual correto, é uma tarefa razoavelmente fácil para um analista experiente. Para o analista de sistemas iniciante, porém, parece uma tarefa extremamente difícil.

Isso acontece porque o modelo conceitual de dados exige duas coisas: um alto grau de abstração e a internalização, pelo analista, de conceitos bastante vagos, como entidades e relacionamentos. Assim, o analista iniciante precisa seguir algumas estratégias para entender melhor como desenvolver um modelo conceitual.

A primeira estratégia é a estratégia dos exercícios e exemplos. Nada é tão útil ao aprendizado como colocar a mão na massa. Os exemplos, por seu lado, servem não só como orientação geral, mas também como exemplos de pontos específicos da modelagem e de como especialistas resolvem problemas de modelagem, sejam eles simples ou complicados.

A segunda estratégia é desenvolver uma lista de dicas de trabalho. Essas dicas funcionam para o analista como as pistas funcionam para um detetive, mostrando que caminho seguir até encontrar a solução do problema.

23.5. Entidades

Uma **entidade** é uma pessoa, objeto, local, animal, acontecimento, organização ou outra ideia abstrata sobre a qual o sistema deve se lembrar alguma coisa.

Cada instância de uma determinada entidade tem características similares (mas não iguais), o mesmo comportamento e uma identidade própria.

O primeiro passo na determinação das entidades é o levantamento dos candidatos à entidade. Durante as entrevistas e reuniões de análise de sistema, vários objetos e conceitos serão descritos como parte do sistema. Algumas vezes esses objetos são bastante concretos, como um “produto” dentro de um “estoque”, outras vezes são descritos como documentos que guardam alguma informação, como uma “nota fiscal”, outras vezes são abstratos, como um “curso”.

No discurso fluente durante uma entrevista, entidades são geralmente substantivos ocupando o papel de sujeito ou objeto, enquanto relacionamentos geralmente são encontrados na forma de verbo. Muitas vezes uma regra de negócio, como “alunos cursam turmas” ou “clientes fazem pedidos”, nos indica entidades e relacionamentos .

Outro sinal importante da necessidade de uma entidade é o fato de algo que precisa ser lembrado representar um conceito ou ideia completa. Em um sistema acadêmico precisamos nos lembrar do nome do aluno, da data de matrícula do aluno, do curso em que está o aluno, etc. O “aluno” é a nossa ideia completa que aparece várias vezes, algumas vezes caracterizado por seu nome, outras vezes por seu curso. É um bom candidato a entidade.

Alguns autores propõem uma determinação *bottom-up* das entidades, sugerindo que elas sejam construídas pela partição de todos os dados atômicos que o sistema deve lembrar (nome de aluno, data de matrícula do aluno, etc.). Assim, construiríamos uma lista de atributos para depois agrupá-los em entidades. Preferimos, porém, uma abordagem de busca direta das entidades. Um sistema com poucas dezenas de entidades pode ter

23. Modelo de Entidades e Relacionamentos

centenas de atributos, o que torna tudo bem mais confuso.

Aproveitando da cultura da orientação a objetos, podemos adotar a divisão proposta por Shlaer e Mellor (1999), uma entidade pode estar em cinco grandes categorias:

- **objetos tangíveis;**
- **papéis exercidos;**
- **eventos;**
- **interações;**
- **especificações;**

Podemos facilmente ver porque objetos tangíveis são bons candidatos a entidades: normalmente, sistemas de informação falam em algum momento de objetos tangíveis, como produtos e equipamentos. Algumas vezes, porém, um objeto tangível, como uma pessoa, assume uma função ou papel específico, como aluno ou professor.

Eventos, ou interações, acontecem em algum momento do tempo e representam classes importantes de entidades. Um exemplo de evento é uma “reunião” em uma agenda . Normalmente eventos exigem atributos como data e duração.

Exemplos típicos de interações são: “contratação de serviço” ou “venda de produto”. Interações são semelhantes a relacionamentos ou a objetos tangíveis ou eventos, sendo muitas vezes representadas dessa forma.

Finalmente, especificação são tipos especiais de entidades que classificam outra entidade. Um bom exemplo é “fábrica”, que é uma especificação para “automóvel”. Geralmente, especificações também podem ser implementadas como um atributo na entidade especificada, sendo essa uma decisão de análise.

Já Coad, Luca e Lefebvre (1999), ao buscarem uma forma mais objetiva de modelar objetos, sugerem que busquemos inicialmente 4 tipos de objetos (que podemos entender como entidades):

- **momentos ou intervalos:** um momento ou um intervalo representa qualquer coisa que precisa ser acompanhada, por motivos de negócio ou legais, e que acontecem em um instante de tempo ou por um período de tempo. Muitas vezes pode ser mais fácil começar nossa análise por esse tipo de entidade, pois estamos tratando de atividades de negócio que devem exigir a participação das outras entidades. Exemplos são: aulas, consultas, contratação, etc.
- **papéis:** representam papéis assumidos pelas pessoas que estão envolvidas com o sistema sendo analisado. Cuidado, pois não são apenas os usuários, nem representam os cargos que as pessoas ocupam nas empresas necessariamente. Exemplos são: aluno, professor.
- **pessoas, locais ou coisas:** representam os objetos tangíveis e localidades. Exemplos são: sala, automóvel.
- **descrições:** são basicamente as especificações propostas por **DESCOBRIR**. Modelos de um produto é um bom exemplo.

J. Robertson e S. Robertson (1998) sugerem algumas regras para que verifiquemos se

um conceito deve ser realmente escolhido como uma entidade:

- toda entidade deve ter um papel único e definido no negócio, se você não pode explicá-la, provavelmente não precisa se lembrar dela;
- entidades devem ter ao menos um atributo que as descrevam, e é preferível que tenham vários;
- entidades devem ter mais de uma instância. Se a instância é única, então não deve ser uma entidade, mas uma informação constante, que é parte do negócio da empresa (uma regra de negócio?);
- entidades devem possuir instâncias unicamente identificáveis;
- entidades não possuem valores, apenas atributos possuem valores;
- pessoas e organizações que interagem com o sistema são candidatos a entidade quando precisamos nos lembrar alguma coisa específica sobre elas, para gerar relatórios ou processar dados entrados. Isso não se aplica a logons ou passwords utilizados para a segurança do sistema, pois segurança é um problema tratado no projeto físico. Devemos aplicar essa regra em relação à necessidade de identificação e endereçamento, por exemplo;
- relatórios raramente são entidades. Normalmente eles são apenas os resultados de um processo que acessa várias entidades;
- linhas de relatório geralmente são entidades;
- nomes de colunas indicam entidades ou seus atributos;
- nenhum valor calculado ou derivado é atributo ou entidade;
- substantivos em regras de negócio são normalmente entidades;
- produtos, quando não são únicos, são normalmente entidades;
- papéis, como funcionário, atendente, apostador, etc., são bons candidatos para entidade, e
- um grupo de dados que se repete em uma entrada ou saída de dados é normalmente uma entidade (ou mais).

23.5.1. Onde encontrá-las

Além de encontrá-las em entrevistas e em regras de negócio, podemos utilizar alguns documentos para encontrar entidades:

- relatórios;
- formulários de entrada de dados;
- arquivos, tanto de papel quanto no computador;
- fichas, como fichas de cadastro, de empréstimo, etc;
- pedidos, requisições e documentos do gênero;
- documentos contábeis e fiscais, como nota fiscal;
- planilhas de dados, em papel ou eletrônicas;
- listagens, registros, agendas, protocolos e outros documentos de trabalho;
- sistemas já existentes,e
- bancos de dados já existentes.

23. Modelo de Entidades e Relacionamentos

Outra forma de encontrar entidades é buscar sistemas semelhantes já resolvidos e padrões de projeto ou padrões internacionais sobre o assunto sendo tratado.

23.5.2. Descrevendo Entidades

É extremamente importante a descrição precisa de cada entidade, pois sua descrição não serve só de documentação, mas também de teste para verificar se entendemos realmente sua presença no sistema. Uma boa descrição de entidade conter os seguintes itens:

- nome, incluindo uma listagem de sinônimos e homônimos;
- definição;
- exemplos;
- atributos (veremos a seguir);
- relacionamentos (veremos a seguir);
- eventos que a utilizam (veremos no próximo capítulo);
- correlação, descrevendo outras partes da análise que se referem a ela;
- regras e exceções relacionadas a essa entidade, incluindo regras de negócio;
- outros comentários e observações, e
- uma idéia da quantidade esperada de instâncias no sistema.

Durante a definição devemos tentar responder várias perguntas, procurando deixar claro o porquê dessa entidade fazer parte do sistema. Assim devemos nos preocupar em dizer o que é essa entidade, o que faz e para que está no sistema, quando algo é ou não é uma dessas entidades, quando passa a ser ou deixa de ser, ou se é permanentemente.

Quando algum elemento passa de uma entidade para outra devemos tomar bastante cuidado para descrever as ações necessárias para tal fato.

23.6. Relacionamentos

A principal característica das entidades que compõe um sistema é se relacionarem umas com as outras. É impossível imaginar uma entidade isolada em um sistema de informação. Toda entidade deve possuir ao menos um relacionamento que a coloque em contato com as outras entidades do sistema.

Relacionamentos representam que existe alguma conexão entre entidades dentro do negócio sendo analisado [B34]. Cada relacionamento deve ser também uma regra de negócio e é utilizado em pelo menos um processo que lida com as entidades envolvidas.

Relacionamentos indicam a possibilidade de buscar um grupo de entidades a partir de outra entidade. Assim, permite encontrar os visitantes que emprestaram um livro específico (navegando de livro para clientes), ou descobrir que produtos um cliente pediu (navegando de clientes para livros). Indicam também que precisamos nos lembrar de algo

que envolve, simultaneamente, duas ou mais entidades do sistema, e que essa lembrança só faz sentido quando todas as instâncias envolvidas são recuperáveis simultaneamente ou seqüencialmente.

Existem muitos relacionamentos comuns, encontrados em muitos sistemas, como compõe (peças compõe máquinas), é um (bicicleta é um meio de transporte), faz ou gera (cliente faz ou gera pedido), atende (visita atende solicitação de reparo), usa (cliente usa produto), etc.

O relacionamento é um tão comum, e tão útil, que foi escolhido como um relacionamento especial em muitos métodos derivados da Modelagem Entidade e Relacionamento original. É a relação de herança, onde dizemos que uma entidade herda todas as características de outra entidade. A herança equivale à abstração de generalização/ especificação e será discutida mais adiante.

Outro relacionamento tão comum que mereceu um tratamento especial em muitos métodos é o relacionamento é parte de. Esse relacionamento equivale à abstração de composição. É normal que utilizemos esse relacionamento apenas quando a parte só existe em função do todo, porém não é uma exigência muito forte.

Relacionamentos podem unir indiferentemente entidades do mesmo tipo ou entidades de tipos diferentes. Quando relaciona entidades do mesmo tipo, por exemplo, pessoas com pessoas, dizemos que é um auto-relacionamento. Ao especificar um auto-relacionamento devemos ter mais cuidado em declarar os papéis das entidades no relacionamento, além de atentar para não produzir um loop infinito no relacionamento. Para isso, algum lado do relacionamento tem que ser opcional.

Apesar de ser possível trabalhar com relacionamentos múltiplos, isto é, relacionamentos contendo mais de duas entidades, normalmente trabalhamos apenas com relacionamentos entre duas entidades. É comum transformar um relacionamento múltiplo em alguma entidade que o represente.

O mesmo acontece com o uso de atributos no relacionamento. Apesar do método original permitir, atualmente criamos uma entidade para representar esse relacionamento. Bertini, Ceri e Navathe (1992) mostram algumas operações que, aplicadas a um modelo e-r, criam diagramas diferentes que podem representar uma mesma realidade. Assim, algo que foi representado como uma entidade em um modelo pode ser representado como duas em outro, ou um relacionamento em um modelo pode ser transformado em uma entidade que se relaciona com as entidades originais (ou vice-versa) sem que haja uma representação falsa da realidade. Pode acontecer de uma ou outra representação ser mais interessante em um contexto.

Relacionamentos podem ser condicionais ou incondicionais, isto é, uma entidade pode ser obrigada a ter um relacionamento com outra ou não. Por exemplo: um automóvel é obrigatoriamente fabricado em uma fábrica, mas nem todos os livros em uma livraria já foram vendidos. Como veremos adiante, o fato de um relacionamento ser opcional é representado pela definição da cardinalidade mínima do relacionamento, que pode ser 0 ou 1.

23. Modelo de Entidades e Relacionamentos

Também é importante notar que existem também relações que ocorrem entre relacionamentos. Dois relacionamentos podem ocorrer sempre juntos (contingentes) ou nunca ocorrer juntos (mutuamente exclusivos). Existem métodos que permitem anotar diretamente no diagrama essas características, porém são pouco utilizados.

Tudo que não puder ser anotado no diagrama deverá ser anotado em um documento associado. O principal tipo de anotações são as regras de negócio que funcionam como restrições, como um professor só pode dar aulas para alunos da escola em que trabalha. Restrições são geralmente impossíveis de desenhar diretamente no diagrama .

Normalmente associamos restrições a ciclos no diagrama. Por exemplo, se temos que fazer pedidos de livros para uma editora, então temos um relacionamento entre livro e pedido, um entre livro e editora e um entre editora e pedido, formando um ciclo. A restrição é que um pedido só pode conter livros da editora indicada no pedido. É possível desenhar o diagrama sem ciclos, eliminado, por exemplo, a ligação entre pedido e editora, porém aconteceriam duas coisas: primeiro teríamos que escrever uma restrição que é possivelmente mais complexa (um pedido só pode conter livros da mesma editora), segundo não teríamos nenhuma indicação no diagrama que o pedido é feito para a editora, exigindo uma nova regra. Finalmente, a falta do ciclo funciona também como falta de indicação que existe uma restrição, pois todo ciclo é um aviso de restrição .

23.6.1. O Relacionamento de Herança

Existem duas formas básicas de herança. Na herança exclusiva dividimos uma classe em categorias. Essa forma de herança traz poucas dificuldades na modelagem e é conhecida como separação em categorias. Uma pessoa, por exemplo, pode ser dividida em duas categorias, a dos homens e a das mulheres. Quando a divisão não é exclusiva, ou seja, quando é possível que uma instância de uma entidade específica seja classificada em duas (ou mais) de suas subclasses, temos alguns problemas que devem ser resolvidos na modelagem lógica. Por exemplo, uma pessoa pode ser aluno e professor simultaneamente em uma faculdade.

Também é possível que existam instâncias que não fazem parte de nenhum dos tipos de entidade especializados, mas fazem parte do tipo geral. Isso também exige um tratamento especial durante a modelagem lógica.

Nós dizemos então que:

- A cobertura é total, se cada elemento da classe genérica é mapeado em ao menos um elemento das subclasses.
- A cobertura é parcial, se existe ao menos um elemento da classe genérica não mapeado nas estruturas das subclasses.
- A cobertura é exclusiva, se cada elemento da superclasse é mapeado em no máximo um elemento das subclasses.
- A cobertura é sobreposta, se existe um elemento da superclasse mapeado em mais de um elemento das subclasses.

Devemos tentar obter apenas heranças totais e exclusivas, pois são mais fáceis de serem tratadas. Dado um grupo de entidades candidatas a construir um relacionamento de herança, devemos analisar se existe um atributo ou relacionamento que é aplicável a apenas um subconjunto dessas entidades, se simplificamos o modelo e se aumentamos sua compreensão. Ou seja, devemos usar a herança para aumentar a semântica do modelo sem causar excesso de informação.

23.6.2. Cardinalidade

Para bem representar um relacionamento, devemos indicar a cardinalidade desse relacionamento, isto é, quantas vezes uma instância da entidade pode se relacionar com instâncias de outras entidades.

Veja por exemplo o relacionamento mãe-filha. Uma filha só pode ter uma mãe, mas uma mãe pode ter várias filhas. Existem três tipos básicos de relacionamentos: o 1:1, um para um, o 1:N, um para muitos, e o N:M, muitos para muitos. Nesse caso só estamos falando da cardinalidade máxima. A cardinalidade máxima indica quantas vezes uma entidade pode aparecer em um relacionamento.

No relacionamento 1:1 cada entidade só pode se relacionar com uma entidade do outro conjunto. Geralmente indica semelhança, igualdade, utilização conjunta, etc.

No relacionamento 1:N cada entidade de um conjunto pode ser relacionar com várias entidades do outro conjunto, mas as entidades do segundo conjunto só podem se relacionar com uma entidade do primeiro conjunto. Geralmente indicam relações de posse, hierarquia ou de composição.

No relacionamento N:M qualquer número de relacionamentos é válido. Podem indicar várias coisas, como eventos, contratos, acordos, ligações temporárias como empréstimos e aluguéis, etc. É normal aparecerem também quando o relacionamento é do tipo 1:1 ou 1:N em certo momento ou período (como o aluguel de uma fita de vídeo), mas se deseja manter a história de todos os relacionamentos. Quando falamos também da cardinalidade mínima usamos notação de par ordenado, (0,1):(1,N) por exemplo, onde o primeiro número do par indica a cardinalidade mínima e o segundo a máxima. A cardinalidade mínima indica uma exigência da participação de uma instância da entidade em relacionamentos. A cardinalidade mínima 0 em ambos os lados indica a existência própria de ambos os objetos. A cardinalidade mínima 1 pode indicar a necessidade de um objeto pertencer ou ser criado por outro.

É comum evitarmos relacionamentos onde ambos os lados exijam como cardinalidade mínima 1. O motivo é que um par de entidades só pode ser colocado na memória do sistema em uma mesma transação, não permitindo que primeiro coloquemos a instância de uma entidade na memória e depois uma instância relacionada da outra entidade. Temos então os seguintes tipos de relacionamentos:

- Relacionamentos um para um.

23. Modelo de Entidades e Relacionamentos

- (0,1):(0,1)
 - ◊ Esse relacionamento significa que uma instância do primeiro conjunto pode ou não se relacionar com uma instância do segundo conjunto, porém pode ter apenas um relacionamento. O mesmo vale do segundo conjunto para o primeiro.
 - ◊ Esse relacionamento é encontrado quando é possível formar pares entre duas entidades, mas esses pares são opcionais.
 - ◊ Um exemplo seria o caso da alocação cabines e reboques de caminhões em uma empresa de aluguel de viaturas. Cabines e reboques podem ser trocados arbitrariamente. Em certo momento, cada cabine só pode ter um reboque e vice-versa. Além disso, algumas vezes uma das partes fica guardada na garagem enquanto a outra é utilizada.
 - ◊ Outro caso que podem mostrar são auto-relacionamentos desse tipo. Uma igreja ou um templo, por exemplo, pode ter um catálogo de freqüentadores e querer saber quem é casado com quem (e quem é solteiro, ou seja, não tem nenhum relacionamento).
- (1,1) :(0,1)
 - ◊ Esse relacionamento significa que a primeira entidade obrigatoriamente tem um relacionamento, mas ele é opcional para a segunda entidade. Em ambos os casos apenas um relacionamento é permitido.
 - ◊ Esse relacionamento é encontrado quando uma entidade possui ou controla de alguma forma outra. Em alguns casos as duas entidades podem ser unidas em uma só.
 - ◊ Ele é menos comum que o relacionamento (1,1):(0,N).
 - ◊ Um exemplo seria uma distribuição de papéis de uma peça de teatro em uma companhia de atores. Cada ator só pode fazer um papel, alguns atores podem não ter papel, mas todos os papéis têm um ator, e apenas um ator.
- (0,1):(1,1), similar ao anterior
- (1,1):(1,1)
 - ◊ Esse relacionamento é pouco comum, pois indica que uma entidade não pode existir sem estar relacionada com outra, e tudo isso apenas uma vez. Normalmente pode ser substituído pela unificação das duas entidades em uma só.
 - ◊ Algumas vezes é utilizado para diferenciar aspectos diferentes da mesma entidade. Por exemplo, um avião é uma entidade que tem visões comerciais, de mecânica, de operação, etc. Fica muito complicado, em um modelo ER, colocar todos os atributos, que chegam a centenas, em uma só entidade, assim podem ser criados relacionamentos (1,1):(1,1) para tratar essa modelagem.
 - ◊ Esse relacionamento não é recomendado, pois exige que ambas as entidades sejam sempre criadas juntas.
- Relacionamentos 1 para N
 - (0,1):(0,N)

- ◊ A primeira entidade pode ou não participar do relacionamento, mas apenas uma vez. A segunda entidade pode ou não participar do relacionamento, e ainda pode fazê-lo várias vezes.
- ◊ Esse é um relacionamento muito comum. Normalmente significa que dois objetos que não possuem nenhum relacionamento de propriedade ou restrição de existência podem ser colocados em uma relação hierárquica.
- ◊ Exemplo: esse tipo de relacionamento pode ser encontrado em locais onde temos um estoque de objetos que são alocados a departamentos da empresa, por exemplo, computadores. Um computador só pode estar alocado em um departamento ou pode estar no estoque (sem alocação). Um departamento pode ter 0, 1 ou vários computadores alocados para si.
- (0,N):(0,1), similar ao anterior
- (0,1):(1,N)
 - ◊ Esse relacionamento normalmente também indica uma relação hierárquica.. O primeiro objeto pode opcionalmente pertencer a essa relação, enquanto o segundo objeto obrigatoriamente pertence a relação.
 - ◊ Não é muito comum, pois exige que uma instância tenha no mínimo uma filha, mas as filhas podem existir de forma independente.
 - ◊ Pode ser usado quando algo para existir deve ter ao menos uma parte, mas estas partes tem vida própria, mesmo só podendo ser usadas em um lugar.
 - ◊ Isso pode ser encontrado, por exemplo, no relacionamento entre uma venda e os itens (quando únicos) vendidos. Uma loja de carros novos, por exemplo, pode em uma mesma venda negociar vários carros, mas necessariamente a venda contém um carro. Já o carro pode ter sido vendido ou não.
- (1,N): (0,1), similar ao anterior
- (1,1):(0,N)
 - ◊ Indica uma maternidade da segunda entidade em relação à primeira, ou seja, cada instância da primeira entidade é obrigada a possuir uma mãe, e apenas uma, que seja instância da segunda entidade.
 - ◊ É um dos relacionamentos mais comuns.
 - ◊ Pode ser encontrado, por exemplo, na relação entre automóveis de uma empresa e multas recebidas. Cada multa é de apenas um automóvel, mas podem existir automóveis com 0, 1 ou mais multas.
- (0,N) :(1,1), similar ao anterior
- (1,1):(1,N)
 - ◊ Indica uma maternidade da segunda entidade em relação à primeira, ou seja, cada instância da primeira entidade é obrigada a possuir uma mãe, e apenas uma, que seja instância da segunda entidade. Além disso, obrigatoriamente a mãe deve possuir uma filha.
 - ◊ Esse relacionamento apresenta o inconveniente de exigir criar uma entidade filha para criar a entidade mãe.
 - ◊ Pode ser encontrado, por exemplo, em um cadastro de pessoas jurídicas,

23. Modelo de Entidades e Relacionamentos

- que devem possuir um endereço, mas podem possuir mais de um.
- ◊ Também é encontrado na modelagem normatizada de objetos que contém listas que obrigatoriamente possuem um item, como uma nota fiscal.
 - (1,N):(1,1) , similar ao anterior
 - Relacionamentos N para M
 - (0,N):(0,N)
 - ◊ Esse relacionamento é muito comum. Representa a forma mais geral de relacionamento, opcional e com todas as possibilidades para ambos os lados.
 - ◊ Pode ser encontrado, por exemplo, na relação entre alunos e cursos oferecidos em um semestre em uma universidade. Alguns cursos não recebem inscrição, alguns alunos não fazem inscrição em nenhum curso.
 - (0,N):(1,N)
 - ◊ Semelhante ao (0,N):(0,N). Também muito comum, porém agora exigimos que haja pelo menos um relacionamento na segunda entidade.
 - ◊ Pode ser encontrado, por exemplo, na relação entre músicas e CDs onde estão gravadas, para controle de uma discoteca. Uma mesma música pode estar em vários CDs, mas não é possível registrar um CD sem músicas (deve existir pelo menos uma). Porém uma música pode nunca ter sido gravada.
 - (1,N):(0,N), similar ao anterior
 - (1,N):(1,N)
 - ◊ Aqui temos um relacionamento múltiplo que deve existir pelo menos uma vez.
 - ◊ Um exemplo é o relacionamento entre salas de uma empresa e móveis colocados nessa sala.
 - ◊ Essa representação muitas vezes é verdadeira, mas é evitada, sendo trocada pelo relacionamento (0,N):(1,N), pois exige que ambas as entidades, quando estão sendo criadas, sejam sempre criadas juntas, ou que existam algumas entidades na base como semente.

23.6.3. Descrevendo Relacionamentos

Dependendo da notação, relacionamentos podem ser descritos por linhas ligando duas entidades ou por um losango ligado por linhas às entidades. Em ambos os casos é possível anotar os relacionamentos com nomes e com a sua cardinalidade (ver exemplos mais a frente).

O nome escolhido para o relacionamento pode estar na voz ativa (mãe gera filho) ou na voz passiva (filho é gerado por mãe). Algumas notações permitem que se usem os dois nomes (um por cima e um por baixo da linha de relacionamento). Geralmente se usa o nome que permite a leitura do relacionamento da esquerda para a direta na parte de cima da linha (ou se dá preferência a esse nome quando apenas um pode ser utilizado).

Relacionamentos também devem ser descritos e comentados, sendo importante responder qual sua função no sistema, o que eles representam, como e quando são estabelecidos ou destruídos.

23.7. Atributos

Todo atributo descreve de alguma forma a instância da entidade. Alguns atributos são especiais e definem a entidade, mesmo que não de forma unívoca. Esses são os atributos nominativos. Outros atributos permitem definir outro objeto que não é o sendo tratado, são os atributos referenciais. Um exemplo de atributo referencial é fábrica para automóvel, referenciando a fábrica onde foi construído. É uma opção do analista criar ou não entidades que permitem a substituição de um atributo referencial por um relacionamento.

23.7.1. Descrevendo Atributos

Devemos definir as seguintes características:

- Nome
- Descrição
- Domínio (valores válidos, como inteiro, real, string ou uma lista de valores, ou ainda tipos criados pelo projetista).
- Tipos de nulos aceitos Exemplo

Na descrição devemos nos preocupar em explicar qual a finalidade do atributo, como são atribuídos os valores, o que significa cada valor, quem define a escolha do valor, quando, por que e por quem o valor é atribuído ou alterado, etc. Atributos são atualmente denotados no mesmo retângulo da entidade, como mostrado nos exemplos a seguir.

23.7.2. Atributos Identificadores (Chaves Candidatas e Chaves Primárias)

Alguns atributos têm o poder de distinguir as ocorrências das entidades, isto é, servem para identificar univocamente uma instância de entidade à instância do mundo real. Definido o valor desse atributo, os outros valores são dependentes e não podem ser escolhidos, mas sim devem possuir um valor exato seguindo a realidade.

Um atributo identificador típico em sistemas financeiros é o CPF de uma pessoa física ou o CNPJ de uma pessoa jurídica. Definido o CNPJ, a empresa, e todos os seus dados, estão univocamente definidos (nome fantasia, endereço, etc.) no mundo real, e assim deve seguir o sistema que estamos construindo.

Muitas vezes precisamos de mais de um atributo identificador para realmente identificar

23. Modelo de Entidades e Relacionamentos

uma instância. Dizemos então que a chave primária é composta. Se usarmos apenas um atributo como identificador, então dizemos que a chave primária é simples.

23.7.3. Relacionamentos Identificadores

Algumas instâncias são identificadas também, ou até mesmo unicamente, por seus relacionamentos. Uma forma de denotar isso é utilizar uma linha mais grossa no relacionamento ou algum símbolo específico. Alguns autores chamam as entidades que são identificadas por seu relacionamento com outras entidades de entidades fracas ou entidades dependentes. Atualmente esses nomes são considerados derogatórios para entidades que podem ser muito importantes em um modelo. Os alunos também, muitas vezes, tendem a achar que não devemos modelar entidades fracas, conclusão que está absolutamente errada.

Chaves Estrangeiras

No modelo conceitual não existe o conceito de chave estrangeira, que é uma característica do modelo relacional. Uma chave estrangeira é uma chave de outra tabela que usamos em uma tabela para indicar o relacionamento. Porém, é comum que as ferramentas de modelagem copiem as chaves estrangeiras automaticamente. Em benefício da prática atual, e em detrimento da pureza teórica, mostramos a seguir algumas possibilidades da notação de relacionamento.

23.8. Descrição Gráfica do Modelo

Várias são as notações existentes para o modelo de entidade e relacionamento. Usaremos nesse texto a notação de Martin, também conhecida como Information Engineering, fornecida pelo software Erwin.

Nessa notação não temos um símbolo para relacionamentos, apenas um retângulo para entidades. Os relacionamentos são indicados por linhas. Uma linha cheia indica um relacionamento identificador. Uma linha tracejada indica um relacionamento não identificador. Por isso, não podemos usar relacionamentos com atributos, necessitando de uma nova entidade nesse caso. Também não podemos criar relacionamentos múltiplos, necessitando de criar entidades para representá-los.

Apesar de parecer que temos um modelo menos poderoso, temos na verdade apenas uma sintaxe mais simples, com o mesmo poder de modelagem. Algumas decisões também ficam tomadas automaticamente também. Por exemplo, não precisamos decidir se um objeto com atributo é um relacionamento ou uma entidade, pois relacionamentos não têm atributos em nosso modelo. Acreditamos que a modelagem segundo as regras do IDEF1X ou da Engenharia de Informação possibilita encontrar mais facilmente um

23.9. Exemplos de notação da Engenharia de Informação

modelo essencial do sistema que as regras tradicionais de Chen ou ainda extensões as mesmas.

A cardinalidade é indicada por três símbolos usados na ponta da linha que indica o relacionamento: uma linha indica 1, um círculo indica 0 (zero), e um pé-de-galinha indica n. Dessa forma podemos anotar o mínimo e o máximo da cardinalidade usando dois símbolos em cada ponta.

O nome do relacionamento é colocado acima (à esquerda) da linha que o indica, sendo o nome do relacionamento inverso colocado abaixo (à direita). Normalmente se lê a notação partindo de uma entidade, lendo o nome do relacionamento, lendo a cardinalidade da ponta oposta e finalmente o nome da segunda entidade.

Nessa notação os atributos podem ser colocados dentro da caixa que representa as entidades, como apresentado na próxima seção.

23.9. Exemplos de notação da Engenharia de Informação

Vamos descrever um modelo na notação da Engenharia da Informação, também conhecida no Brasil como “pés de galinha”.

Apresentaremos o modelo de uma locadora de vídeo. A locadora trabalha com fitas de vídeo. Cada fita de vídeo contém um filme, porém cada fita deve ser identificada unicamente, pois elas podem ser dubladas ou legendadas. As fitas são emprestadas para clientes em um dia e hora específico. Um cliente pode ficar com várias fitas, ou nenhuma. Uma fita pode estar com apenas um cliente, ou estar na loja e não estar com cliente nenhum. É importante saber para quem cada fita específica foi emprestada, para auditar clientes que estragam fitas, por isso todas as fitas são numeradas com um código único. Os filmes são dirigidos por diretores e contém atores.

Observamos que o cliente é um papel assumido por uma pessoa, a fita de vídeo é um objeto físico, existente, o filme é uma obra de arte que está representada na fita (um conceito), diretor e ator são também papéis assumidos por pessoas dentro da idéia de filme e que um aluguel é um contrato entre o cliente e a locadora.

Atenção para outro detalhe: não existe a entidade locadora, pois este sistema é destinado a uma só locadora. Seria uma entidade única, que claramente é uma constante do sistema. A presença de entidades desse tipo é um erro comum nos modelos feitos por principiantes. Porém, se tivéssemos um software para uma rede de locadoras, seria interessante guardar em que locadora está cada fita, o que exigiria essa entidade.

23. Modelo de Entidades e Relacionamentos

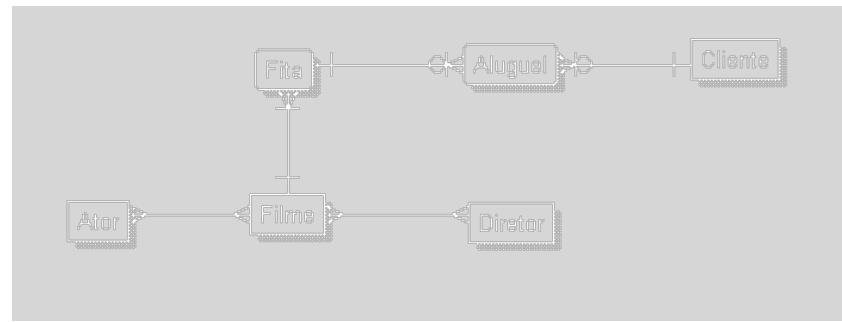


Figura 23.4.: Modelo ER só com entidades e relacionamentos, notação da Engenharia da Informação

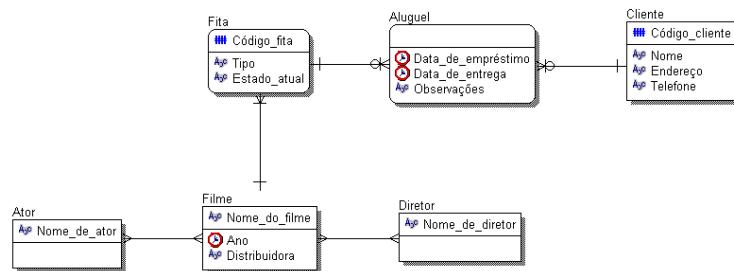


Figura 23.5.: Modelo ER com atributos, notação Engenharia da Informação

23.10. Exercícios

Exercício 23.10-1: Escreva um modelo de entidades e relacionamentos para um aplicativo de celular que sirva como relógio e despertador, permitindo vários alarmes.

Exercício 23.10-2: Vá para o site <http://jogodeanalisedesistemas.xexeo.net/> e visite a Livraria Resolve. A partir da sua visita faça um modelo de entidades e relacionamentos para o sistema proposto.

24

Modelo Orientado a Objeto

Conteúdo

24.1. Exercícios	329
----------------------------	-----

Aqui entrará o modelo de classes.

24.1. Exercícios

Exercício 24.1-1: Escreva um modelo de classes para um aplicativo de celular que sirva como relógio e despertador, permitindo vários alarmes.

Exercício 24.1-2: Vá para o site <http://jogodeanalisedesistemas.xexeo.net/> e visite a Livraria Resolve. A partir da sua visita faça um modelo de classes para o sistema proposto.

DRAFT

Parte VI.

Estimativas

DRAFT

DRAFT

25

Medidas e Estimativas do Tamanho do Software

Measurement is the first step that leads to control and eventually to improvement. If you cant measure something, you cant understand it. If you cant understand it, you cant control it. If you cant control it, you cant improve it.

(H. James Harrington)

Conteúdo

25.1.	Esfogo	335
25.2.	Algumas Medidas Conceitualmente Simples	336
25.3.	Visão Geral dos Métodos de Estimativa	340
25.4.	Métodos de Estimativa Baseado em Opinião	341

Por que estimativas de tamanho de software?

Ao final do processo de análise é necessário que seja possível avaliar, com baixo risco, o custo de desenvolvimento do mesmo, em termos como esforço, tempo e mão de obra necessária. Este capítulo fornece as ferramentas básicas para um analista de sistemas levantar essas informações a partir do seu modelo.

Ao iniciar um projeto de software, ou qualquer outro projeto, é necessário ter uma

25. Medidas e Estimativas do Tamanho do Software

estimativa da quantidade de tempo e recursos que serão usadas para desenvolvê-lo. Essa estimativa é feita a partir de outra: a estimativa de tamanho do software. Para essa estimativa ser feita é preciso, antes, entender como esse tamanho pode ser medido. Isso foi visto parcialmente no Capítulo 14, mas existe muito material sobre o assunto tanto na área de Gestão de Projetos (PMI, 2017) quanto na área de Engenharia de Software (IEEE Computer Society, 2014a; Pressman e Maxim, 2014).

Apesar de muito estudada, a estimativa de projetos de software ainda é um problema em aberto. Isso acontece não só pela dificuldade inerente de prever alguma coisa, mas também por alguns motivos adicionais. O primeiro motivo importante é a falta de maturidade do processo de muitas organizações, que resulta na falta de dados que ajudem a fazer previsões com baixa margem de erro. Em muitos lugares ainda são praticadas estimativas *ad-hoc*, baseadas na experiência dos desenvolvedores ou gerentes, e ainda sujeitas a pressões de clientes e da alta gerência. Em outros, mesmo com processos mais definidos, não há dados suficientes sobre o desempenho da equipe em projetos anteriores e similares para fazer boas previsões. Finalmente, muitos projetos de software começam com uma definição incerta do que vai ser feito, tendo um foco inicial no problema a ser resolvido, o que dificulta também a previsão. Os métodos ágeis atacam alguns desses problemas evitando fazer grandes estimativas do trabalho total do projeto no seu início, porém isso impossibilita certas formas de contrato.

Quanto mais tradicional for o desenvolvimento, seja por cultura da empresa, seja por imposição do cliente, mais será necessário fazer e reduzir os erros dessa estimativa, inclusive por motivos contratuais. Além disso, ela influencia todas as expectativas das partes interessadas, incluindo, principalmente: o orçamento, a funcionalidade entregue e o prazo do projeto. Já nos projetos ágeis, o foco é na entrega de valor em prazos pré-delimitados e o projeto é visto de outra forma.

Logo, o objetivo da estimativa de um projeto de software exige, em um certo momento no tempo, no início ou ao longo do projeto, prever os recursos necessários, cujo custo é basicamente dominado pelos recursos humanos¹, e o tempo necessário para terminar o projeto, dando ferramentas ao gestor para planejar as próximas fases. Isso é feito com um grau de incerteza, que é maior no início, por ser uma previsão baseada em informações limitadas, mas é importante lutar para que essa incerteza seja pequena e conhecida. Ao longo do projeto as estimativas continuam e são constantemente comparadas e revisadas. Como há um aumento da quantidade de informações, normalmente se tornam mais precisas. No limite, ao final do projeto, a previsão dos recursos e tempo necessário para o projeto é perfeita, apesar de ser inútil.

Este livro é basicamente dedicado a responder a pergunta **o que** devemos desenvolver, de algumas formas, mas não seria completo se não apresentasse também alguma maneira de avaliar o tamanho do software que está sendo proposto, principalmente após o processo de análise. Este capítulo e os próximos tentam mostrar como essas outras perguntas podem ser respondidas. Para isso trataremos de algumas medidas de tamanho possíveis

¹Ou proporcional a eles

para o software, respondendo a pergunta 2, e discutiremos como essas medidas podem ser previstas, respondendo a pergunta 3. Finalmente mostraremos, com alguma simplificação, como o método COCOMO II(Barry W. Boehm et al., 2000) pode ser usado para calcular o esforço, a equipe média e o tempo de desenvolvimento.

25.1. Esforço

Tendo em vista o forte domínio do valor dos recursos humanos em projeto de software, a principal forma de entender a demanda de recursos e tempo para um projeto de software é a quantidade de trabalho que será necessária para construí-lo. Em Engenharia de Software, o trabalho necessário para desenvolver um produto é conhecido como **esforço**. O esforço é medido em pessoas-hora ou pessoas-mês, isto é, a quantidade total de horas, ou meses, trabalhados por uma quantidade de pessoas para fazer o produto.

Assim, um software pequeno pode precisar de 4 pessoas-mês, dando a ideia que 2 pessoas podem fazê-lo em 2 meses, ou 1 pessoa pode fazê-lo em quatro meses. Porém, é difícil imaginar que 8 pessoas possam fazê-lo em 15 dias. Acontece que sistemas de informação são um pouco como bebês: não podemos ter a gestação de um bebê com nove mães em um mês, também não podemos colocar milhares de pessoas para fazer um software em um dia. Ao contrário, se colocarmos pessoas demais é possível que a necessidade de gerenciar atrapalhe a produção(Brooks, 1995). Na verdade, Barry W. Boehm et al. (2000) encontraram uma relação entre o esforço necessário e o tempo necessário para fazer um sistema, e consequentemente o tamanho médio da equipe, incluída no método COCOMO II.

A medição do esforço de um projeto em andamento ou terminado é ao mesmo tempo uma prática fácil e difícil. Em um projeto onde toda a equipe está dedicada, o esforço pode ser medido simplesmente considerando o tempo e a quantidade de pessoas envolvidas no projeto. Já em casos onde há vários colaboradores parciais, ou se deseja uma contagem mais exata, é necessário que todo o trabalho seja registrado e contado detalhadamente.

A pergunta principal do processo de estimativa de software, então, é o esforço necessário para desenvolver uma certa funcionalidade, dentro de um contexto. A expectativa de um gerente de projeto é que esse esforço seja estimado diretamente a partir de uma especificação, de preferência o mais cedo possível. O que se espera, e há uma confirmação pelo método COCOMO (Barry W. Boehm et al., 2000), é que o esforço necessário seja função, pelo menos em maior parte, do tamanho do software.

Disso podemos concluir que uma das mais importantes informações que podemos ter sobre um produto de software é seu **tamanho**. O problema é que tamanho, em software, é um conceito com muitos significados. Por exemplo, você pode simplesmente medir o tamanho de um programa na forma compilada, e isso tem impacto no processo de distribuição do produto, como tamanho de mídia ou ocupação da rede. Porém, esse tamanho tem pouco significado em relação a questões muito mais importantes, como é o esforço necessário para construí-lo. Grande parte do tamanho do programa a ser

25. Medidas e Estimativas do Tamanho do Software

distribuído, por exemplo, pode vir de bibliotecas anexadas a ele. Qual o tamanho de um programa “Hello World!” para um sistema operacional de janelas e para um sistema de linha de comando²? Assim, entre as medidas propostas, existem algumas mais úteis do que outras, em cada contexto.

Dessa forma, temos uma sequência de questões a serem respondidas:

1. O que o software fará?
2. Como medir o tamanho de um software?
3. Como prever o tamanho do software que faz o que foi pedido?
4. Qual o **esforço** necessário para desenvolver um software desse tamanho?
5. Qual a equipe média e o tempo de desenvolvimento necessário para realizar esse **esforço**?

25.2. Algumas Medidas Conceitualmente Simples

Algumas medidas de software são razoavelmente simples de entender, o que permite que elas sejam usadas e compreendidas por pessoas com pouco ou nenhum treinamento, como por exemplo:

- linhas de código fonte;
- pontos de história, e
- tamanhos de camisa.

Essas medidas, apresentadas a seguir, podem ser usadas em métodos de estimativas baseados em especialistas, que serão o tema da próxima seção.

25.2.1. A Medida Mais Simples: Linhas de Código

A forma mais conhecida de medir o tamanho de software é a contagem de (milhares) linhas de código fonte, conhecida como LOCs, KLOCs, SLOCS ou KSLOCs, do inglês (*Kilo*) (*Source*) *Lines of Code*. A ideia de contar as linhas de código do produto de software e ter uma medida direta e facilmente verificada.

Só que a questão não é tão fácil.

Primeiro, linguagens diferentes vão apresentar uma quantidade de linhas de código diferentes para a mesma funcionalidade, como mostram as Listagens 25.1 e 25.2. Como comparar o tamanho de dois programas em linguagens diferentes então?

Lista de Programas 25.1: Programa Hello World para Win32(Rösler, 1994)

```
; Hello world em Assembler para arquitetura Win32
TITLE Hello world in win32. Tasm
```

²Em um pequeno experimento, a diferença foi de cinco vezes

```

VERSION T310
Model use32 Flat,StdCall

start_code segment byte public 'code' use32
begin:
    Call MessageBox, 0, offset sHallo, offset caption, 0
    Call ExitProcess, 0
start_code Ends

start_data segment byte public 'data' use32

sHallo db 'Hello world',0
caption db "Hi",0

start_data

```

Lista de Programas 25.2: Programa Hello World em Python 3(Rösler, 1994)

```
# Hello world em Python 3
print("Hello World")
```

Segundo, o que contar como uma linha de código? Por que uma sentença com o mesmo significado pode ser escrita em um número diferente de linhas. Além disso, contam-se os comentários ou não? Contam-se os espaços em brancos destinados a facilitar a leitura? Esse problema, ao menos, foi resolvido com uma definição do que deve, e do que não deve, ser contado, por meio de padrões razoavelmente complicados. porém esse trabalho é difícil, também dependente de linguagem, e pode gerar dúvidas em função do estilo de programação.

Linhos de código tem como vantagens ser intuitivas, mesmo considerando os complicados padrões que dizem o que pode e o que não pode ser contado, e permitem a automação da contagem. A automatização acaba sendo importante na análise estatística de projetos, que levou ao método COCOMO II, por exemplo, e pode gerar adaptações locais aos valores das constantes desse método e de suas variantes(Barry W. Boehm et al., 2000).

Seus problemas, porém, são muitos:

- não têm relação com o esforço de análise e projeto, sendo ligadas apenas ao esforço de programação;
- não tem relação real com a funcionalidade entregue;
- variam de acordo com linguagem, estilo de programação, experiência do programador, uso de bibliotecas, etc.;
- não atendem as tecnologias mais modernas, com uso de GUI e geração de código;
- quando usadas como parâmetro de remuneração, incentivam soluções mais longas, e
- são difíceis de usar como medida inicial de estimativa.

Provavelmente a maneira mais difundida de contar linhas de código é a fornecida pelo Software Engineering Institute em um manual de 242 páginas, que inclui uma *check-list* de cinco páginas para exemplificar como isso pode ser feito. Para linhas de código fonte lógicas (LSC) as instruções básicas sugerem contar instruções executáveis ou não e declarações, e não contar comentários e linhas de código. Além disso, devem ser contadas linhas programadas, geradas, convertidas, copiadas e modificadas, mas não linhas removidas (Park, 1992).

Tendo em vista o quadro de problemas da medida de linha de código, é possível pensar que uma medida realmente interessante sobre software não fala do artefato criado, mas sim do resultado fornecido pelo artefato. Assim, outras medidas podem fornecer uma ideia melhor do software.

Isso significa entender que o verdadeiro valor do software não está nas medidas diretas que podemos fazer, mas sim no que ele fornece ao usuário, e que esse tamanho que é interessante.

Essa foi a motivação da criação da medida **Pontos de Função**, que é tratada no Capítulo 26.

25.2.2. Pontos de História

“**Pontos de história**” é uma medida ágil usada como estimativa do esforço necessário para desenvolver uma história do usuário. Pontos de História são bastante informais e, devido a forma como são definidos e usados, não podem ser comparados entre projetos.

Segundo Cohn (2005) “pontos de histórias são uma unidade de medida para expressar o tamanho geral de uma história do usuário, funcionalidade ou outra peça de trabalho”. Além disso, Cohn (2005) avisa que “O valor bruto que damos não é importante. O que importa são os valores relativos.”

Pontos de história podem ser medidos em uma escala qualquer, porém o mercado praticamente estabeleceu o uso de um conjunto de valores que aproxima uma série de Fibonacci: 0, $\frac{1}{2}$, 1, 2, 3, 5, 8, 13, 20, 40 e 100. Esses números devem ser vistos como uma referência de comparação.

Essa sequência, em vez de usar qualquer número, tem uma função: evitar discussões infrutíferas e aceitar uma margem de incerteza em estimativas. Dessa forma se evita que em uma discussão para estimar o tamanho de uma história se perca tempo discutindo se é 12 ou 13, por exemplo. Os números dessa série, então, tem a propriedade de mostrarem não só tamanhos claramente distintos, mas também com uma importância nessa distinção: cada número é, aproximadamente, a soma dos dois anteriores, assim a comparação entre tamanho histórias pode ficar mais clara.

Para fazer a estimativa, em cada início de projeto, a história do usuário mais simples, para aquele projeto específico, recebe o valor 2 como estimativa do seu esforço. A partir daí, os outros valores são usados no processo de estimativa escolhido, tentando mostrar a

relação proporcional de esforço necessário em uma tarefa em relação a tarefa de peso 2. Claro que nenhuma história no início do projeto vai ser pontuada com um valor menor que 2, mas é comum aparecerem histórias menores com o tempo.

A forma normalmente escolhida para fazer a estimativa é o *Planning Poker*, principalmente em projetos ágeis.

A estimativa de pontos de história é calibrada ao longo do projeto, pois a equipe vai aprendendo a estimar melhor ao realizar as histórias.

Alguns autores, como ISBS (2010), propõem que a equipe analise a relação entre os pontos de histórias atribuídos a uma história de usuário e as horas efetivamente gastas para implementá-las. Isso pode trazer um visão mais clara aos membros da equipe, porém não é necessariamente vantajoso frente a experiência real da equipe no projeto e depende, para ser uma ação prática, do método de gestão levantar essas informações com facilidade, como acontece com o uso de algumas ferramentas de gestão de projetos ágeis.

Velocidade da Equipe

A velocidade de uma equipe ágil é o número de pontos de história que ela faz, em média, em cada ciclo (ou *sprint*). No início do projeto, como a equipe não sabe ainda estimar muito bem os tamanhos das histórias, essa velocidade tende a variar. Ao longo do projeto ela costuma se tornar estável e até mesmo diminuir com o aumento da produtividade da equipe. Vigiar a velocidade é uma tarefa dos gestores ágeis, porém ela é apenas um sinalizador, não é razoável utilizá-la como um número absoluto de produtividade, porque as equipes vão adaptando suas estimativas ao longo do projeto.

25.2.3. Tamanhos de Camisa

Uma medida curiosa relatada por Dimitrov (2020), Rubin (2013) e vários outros autores é medir o tamanho de software fazendo uma analogia com tamanhos de camisa. Nesse caso, usado em métodos ágeis, cada história do usuário recebe um valor entre os tamanhos que encontramos para camisetas. Uma seleção possível de valores seria: XP, P, M, G, XG, XXG.

T-Shirt Sizing

Para isso funcionar, é possível manter uma tabela que associe cada tamanho com uma margem de variação, possivelmente em horas de trabalho, para estimativas de artefatos pequenos, mas talvez de custo se são estimados softwares completos, por exemplo.

Essa escala estimula a comparação e a abstração, porém dificulta a determinação de valores como a velocidade da equipe e o tamanho global de um projeto ou de uma fase de projeto.

Rubin (2013) considera que medidas de tamanho de camisa associadas a faixas de valores de custo são boas para o planejamento de portfolio.

25.3. Visão Geral dos Métodos de Estimativa

Como visto, os motivos principais para se medir software têm relação com a tarefas de gerenciar projetos de software. Precisamos saber o tamanho para entender quanto trabalho fizemos ou vamos fazer, qual a dificuldade de fazê-lo, ou qual será o custo. Entender o tamanho de um produto permite responder perguntas como:

- Qual o custo de desenvolver o sistema?
- Qual o esforço para desenvolver o sistema?
- Quantas pessoas serão necessárias para fazer esse software?
- Em quanto tempo o sistema ficará pronto?
- Que recursos são necessários para o sistema executar?
- Qual o valor da nossa carteira de software?

Métodos de estimativa de projeto de software podem ser divididos em três grandes grupos (ISBS, 2010):

- métodos macro, que buscam estimativas de alto nível;
 - uso de equações;
 - comparação, com projetos similares, e
 - analogia, que busca um projeto completo muito similar;
- métodos micro, que buscam estimativas a partir dos detalhes, baseados na estrutura analítica do projeto ou em outra forma de quebrar sucessivamente o projeto em tarefas menores que podem ser estimadas com mais facilidade;
- métodos usando IA, que não serão tratadas neste texto.

Estimar software sem o uso de IA sempre implica em ter, em algum momento, a opinião de pessoas. Algumas práticas se baseiam fortemente nas opinião de especialistas, talvez um único especialista. Outros, principalmente os aplicados em processos ágeis, se utilizam da equipe e tendem a cultivar o aprendizado e praticar revisões nas estimativas.

A questão aqui é realmente ligada ao tipo de contrato de desenvolvimento que está sendo feito. Um contrato tradicional não tem como escapar de fazer uma estimativa mais formal, considerar riscos e cobrar valores que vão cobrir o custo e gerar lucro. Um contrato por tempo de serviço, onde a funcionalidade é determinada de tempos em tempos pelo cliente, facilita o processo de estimar, principalmente se este intervalo de tempo for curto.

É importante deixar claro que o problema de estimar software não é complicado para pequenas tarefas, mas extremamente difícil para grandes sistemas. Mesmo dividindo um software com uma metodologia como a Estrutura Analítica de Projeto, há sempre o problema de integrar as partes e a complexidade do sistema não é o somatório das complexidades das partes. Pelo contrário, quanto maior o sistema, mais complexo é construí-lo. Essa afirmação é corroborada pelos resultados da pesquisa que levou ao método COCOMO II (Barry W. Boehm et al., 2000).

25.4. Métodos de Estimativa Baseado em Opinião

Alguns métodos simples para estimar baseado em opiniões são:

- Método de Estimativa do PERT;
- Método de Delphi, e
- *Planning Poker*.

25.4.1. O Método de Estimativa do PERT

Uma maneira simples simples de fazer uma estimativa através da opinião de especialistas é a usada pelo método PERT (ISBS, 2010; PMI, 2017), que estima um valor a partir da fórmula:

$$T_e = \frac{T_o + 4 \times T_\mu + T_p}{6} \quad (25.1)$$

onde T_p é a estimativa mais pessimista, T_o a estimativa mais otimista e T_μ a estimativa média. Essas três estimativas são idealmente calculadas a partir de várias opiniões dadas por pessoas diferentes, mas podem ser até mesmo fornecidas por um especialista único.

25.4.2. O Método de Delphi

Outra forma conhecida de fazer estimativas é o método de Delphi (Dalkey, 1966; Helmer-Hirschberg, 1967; ISBS, 2010), cuja prática original é um ciclo iterativo onde:

1. especialistas recebem uma ou mais perguntas para fazer sua estimativa;
2. as estimativas são coletadas e relatadas a todos os especialistas de forma anônima;
3. os especialistas podem revisar e dar justificativas para suas novas estimativas, e
4. o ciclo é repetido até as opiniões convergirem.

Esse método, proposto em 1966, recebe o nome do Oráculo de Delphi, ou Delfos, e vem sendo reconstruído de várias formas (ISBS, 2010), como em reuniões onde não há anonimato e em aplicativos na rede, porém a que tem mais feito sucesso nos últimos anos é o *Planning Poker*.

O Oráculo de Delfos foi um templo na Grécia Antiga onde sacerdotisas faziam previsões do futuro, normalmente ambíguas.

25.4.3. O Planning Poker

No *Planning Poker* (Cohn, 2005, p 56) cada membro da equipe deve fazer uma estimativa do esforço necessário para desenvolver um história do usuário. Geralmente esse esforço é medido em valores arbitrários, como os **pontos de história**, descritos na seção 25.2.2.

Para o *Planning Poker* é necessário um baralho especial, ou um aplicativo que simule esse baralho, e que possua a sequência de pontos de história usada pela sua equipe e algumas cartas adicionais, como uma carta com uma interrogação, que indica que

25. Medidas e Estimativas do Tamanho do Software

o estimador está sem ideia de como estimar, e uma xícara de café, que indica que o estimador está cansado e precisa parar um pouco.

As rodadas, inspiradas no Método de Delphi, são iniciadas com os membros da equipe ouvindo e questionando um *product owner* ou um analista sobre como deve ser o comportamento da história do usuário sendo estimada (Figura 25.1a). A cada rodada, cada membro do time escolhe uma carta em segredo com um número desejado(Figura 25.1b). As cartas são todas mostradas simultaneamente (Figura 25.1c) e, então, se as estimativas forem diferentes, o mais otimista e o mais pessimista devem justificar seu voto (Figura 25.1d)e o processo é refeito até que uma condição de parada seja aceita (Figura 25.1e e 25.1f).

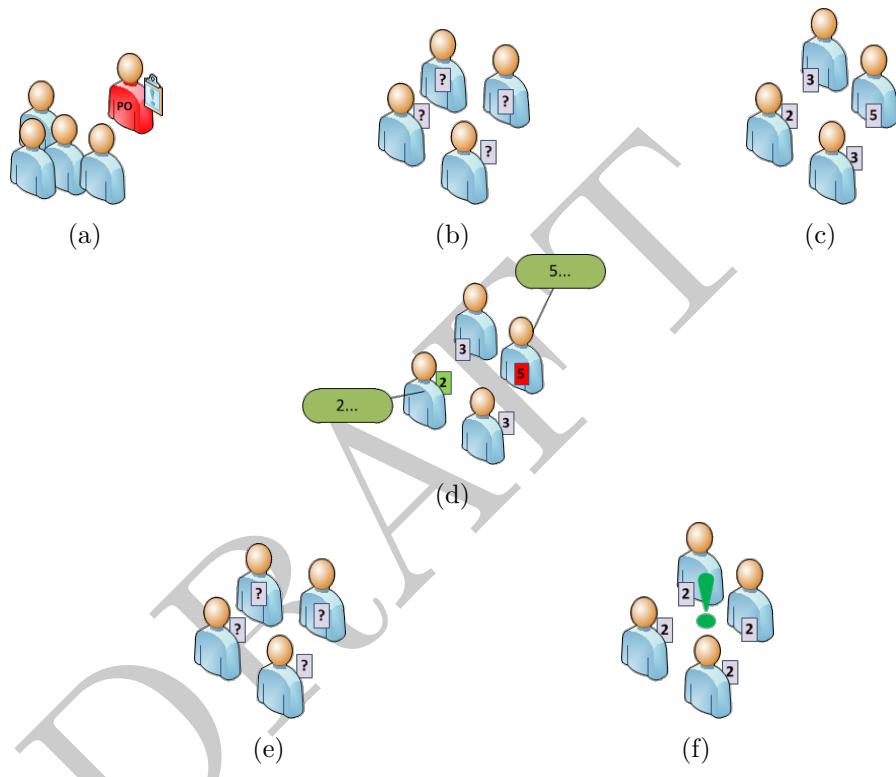


Figura 25.1.: Ilustração da sequência do Planning Poker com duas rodadas

A questão da condição de parada é um pouco controversa. Normalmente o número de rodadas é limitado a 3, com a expectativa que **um número comum a todos seja alcançado**. Caso isso não aconteça, existem várias regras de como agir: tentar mais algumas vezes, adiar a avaliação para outra ocasião, propor a quebra da história em histórias menores, usar o valor mais frequente(Cohn, 2005), usar a previsão mais pessimista(Maximini, 2018), usar a previsão mais otimista(ISBS, 2010), ou até mesmo gerar uma solução *ad-hoc* de consenso naquele instante. Essa decisão também depende da variação das estimativas. Um caso onde todas são iguais menos uma é diferente de um caso onde as estimativas estão espalhadas entre 3 números.

26

Análise de Pontos de Função

Conteúdo

26.1.	Visão Geral de Pontos de Função	344
26.2.	Procedimento de Contagem	344
26.3.	Funções Transacionais	345
26.4.	Funções de Dados	346
26.5.	Identificando Funções de Negócio	346
26.6.	Identificando Entradas	347
26.7.	Identificando Arquivos Internos	349
26.8.	As Perguntas	351
26.9.	Cálculo dos Pfs Finais	352
26.10.	Conclusão	353

Por que Pontos de Função?

Ao fazer uma modelagem de sistema, é importante ter alguma ideia do esforço necessário para desenvolvê-lo. A Análise de Pontos de Função é uma prática aceita em todo mundo que permite avaliar o tamanho previsto de um software.

Até 1979 não tínhamos uma boa medida do tamanho do software em função da sua funcionalidade como vista pelo usuário. Apenas nesse ano, (Albrecht, 1979) apresentou uma medida conhecida como Pontos de Função, cujo objetivo era servir como avaliador e preditor do tamanho de um sistema. Atualmente Pontos de Função são uma medida

26. Análise de Pontos de Função

internacionalmente aceita e uma das formas possíveis de se fazer um contrato com o governo no Brasil (Brasil, 2018)¹.

26.1. Visão Geral de Pontos de Função

Um Ponto de Função (PF) é uma medida abstrata e relativa que conta “o número de funções de negócio entregues ao usuário”. Um relatório simples, por exemplo, pode medir “4 Pontos de Função”. Da mesma forma que um metro ou um litro, Pontos de Função só fazem sentido quando comparados com um padrão. Assim, um sistema com 1.000 PF entrega o dobro de funcionalidade de que um sistema com 500 PF. Com o tempo, aprendemos a ter uma ideia absoluta do tamanho de um sistema a partir da contagem de seus PFs (IFPUG, 2012).

Pontos de Função avaliam o tamanho do software a partir de seis características principais (IFPUG, 2012):

- entradas;
- saídas;
- consultas;
- arquivos lógicos internos;
- interfaces lógicas externas, e
- ajustes de complexidade.

A maneira padronizada de contar pontos de função é fornecida pelo *International Function Points User Group (IFPUG)*(IFPUG, 2012), que fornece aos seus associados um manual contendo detalhes do que deve e do que não deve ser contado. Esse capítulo é apenas uma introdução ao método, servindo para fazer cálculos aproximados do número de pontos de função.

26.2. Procedimento de Contagem

O procedimento genérico de contagem de Pontos de Função é representado na Figura 26.1.

Ele se inicia com a determinação do **propósito da contagem**, isto é, a explicitação do motivo da contagem estar sendo realizada. Normalmente esse propósito estará relacionado a fornecer uma resposta a um problema de negócio existente, como a contratação de um serviço.

A partir do propósito podemos determinar o tipo de contagem. São considerados três tipos de contagem:

¹A especificação brasileira de Pontos de Função para o governo é mais complexa na sua forma de usar, mas parte dos mesmos princípios.

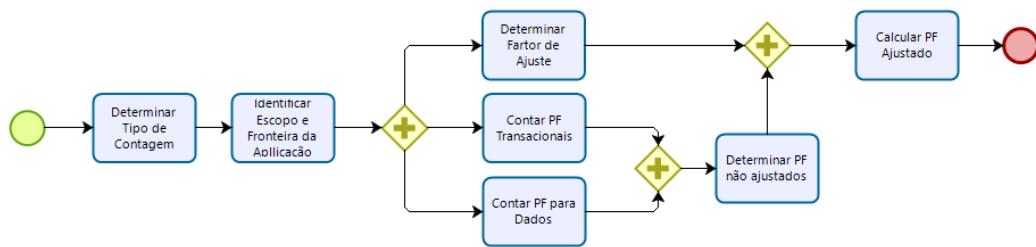


Figura 26.1.: O processo geral de contagem de Pontos de Função. Fonte: do autor

1. **projeto de desenvolvimento**, onde medimos as funcionalidades entregues ao usuário em uma versão onde o software é desenvolvido desde o início. Inclui as funcionalidades necessárias para a conversão de dados, mesmo que usadas uma única vez;
2. **projeto de melhoria**, onde medimos as modificações que alteram, adicionam ou removem funcionalidades em uma aplicação existente. Inclui as funcionalidades necessárias para a conversão de dados, mesmo que usadas uma única vez, e
3. **aplicação**, onde medimos uma aplicação já instalada. Também é chamada de *baseline* e é normalmente realizada no fim de um projeto de desenvolvimento e mantida atualizada nos projetos de melhoria.

O escopo da contagem define um conjunto ou um subconjunto do sistema, e permite dizer se uma funcionalidade deve ou não ser contada.

A fronteira da aplicação delimita o software medido, definindo sua interface com o mundo exterior. Ela servirá não só para considerarmos se uma função deve ou não ser contada, mas também para considerar se um arquivo lógico deve ser contado como interno ou externo a aplicação.

Definido o tipo de contagem, a fronteira e o escopo da contagem, que influenciarão a forma final de contagem, identificamos as funções de negócio como percebidas pelo usuário, dividindo-as em 5 grupos, agrupados em 2 tipos: funções transacionais e funções de dados.

26.3. Funções Transacionais

Um processo elementar é a menor atividade significativa para o usuário de forma indivisível. Isso significa que o usuário vê como um processo único, completo e independente de outros, que se inicia com o sistema em um estado consistente e termina com o sistema em um estado consistente.

Um processo elementar será contado como uma de três possíveis formas de função transacional:

- **saídas externas** (SE ou EO), são informações de negócio que o usuário final pode

26. Análise de Pontos de Função

receber, representando relatórios, telas e mensagens de erro como um todo e não em suas partes individuais;

- **consultas externas** (CE ou EQ), que são saídas simples e imediatas, sem alteração na base, usualmente caracterizáveis por chaves simples de consulta, e
- **entradas externas** (EE ou EI), que são processos elementares que processam informações de negócio recebidas pelo sistema de fora da fronteira da aplicação e cuja finalidade principal é manter um Arquivo Lógico Interno .

26.4. Funções de Dados

As funções de dados estão divididas em dois tipos:

- **arquivos lógicos internos** (ALI ou ILF), que contém os dados permanentes, relevantes para o usuário e mantidos e utilizados pelo sistema. O sistema cria, altera e apaga esses dados, e
- **arquivos de interface externos** (AIE ou EIF), que contém dados permanentes e relevantes para os usuários, guardados em algum lugar por outra aplicação, mas referenciados pela aplicação em questão, sendo que outro sistema mantém esses dados.

O segundo passo é determinar a complexidade de cada função de negócio. A complexidade fornece um peso para cada função de negócio encontrada. O somatório do número de funções multiplicadas pelo seu peso fornece o número básico de pontos de função. Esse número é um indicador preliminar do tamanho do sistema.

Finalmente, no terceiro passo, o número básico de pontos de funções é corrigido em função de fatores que aumentam ou diminuem a complexidade do sistema.

26.5. Identificando Funções de Negócio

Para identificar as funções de negócio devemos partir de algum documento que aponte as funções **aprovadas** e pelo usuário e **úteis** para o negócio. Não devem ser contadas funções necessárias por causa da tecnologia aplicada. Basicamente, só é cobrado o que o usuário pode ver e está disposto a pagar. Também é importante que as funções de negócio sejam cobradas **como o usuário as percebem**. Isto significa que não interessa se estamos usando um ou vinte arquivos para guardar uma informação, mas sim de quantas formas o usuário pode acessar essa informação.

Além disso, devemos identificar as funções seguindo certa ordem. A ordem é importante porque encontrar um tipo de função de negócio ajuda a encontrar as funções de outro tipo. Assim, em um sistema novo devemos usar a ordem: saídas, consultas, entradas, arquivos e interfaces. Por outro lado, em um sistema já existente devemos usar a ordem arquivos, interfaces, saídas, consultas e entradas.

Para serem contados as funções devem:

- beneficiar claramente o usuário;
- ser especificamente aprovadas pelo usuário, e
- influenciar em algum grau mensurável o projeto, desenvolvimento, implementação e suporte à aplicação.

No manual da IFPUG podemos encontrar vários exemplos que nos permitem dirimir as dúvidas de como realizar a contagem. A técnica é simples, porém difícil de dominar.

26.5.1. Identificando Saídas

Para contar as saídas é necessário contar todas as informações que o sistema gera para o ambiente de forma procedural. Tipicamente, saídas são relatórios impressos, telas, janelas e arquivos gerados para outras aplicações.

Devem ser contadas como saídas distintas cada formato utilizado. Basicamente, se for necessário fazer outro procedimento para produzir a informação, contamos como uma saída distinta. Também contamos cada tipo de lógica utilizada para fazer gerar a informação. Assim, se um relatório de vendas contém as vendas por vendedor e por loja, contaremos como duas saídas, pois são necessários procedimentos lógicos distintos para calcular cada um desses valores. Linhas de sumário e total, porém, não são contadas como novas saídas.

Uma saída externa pode ter uma parte de entrada, para, por exemplo, selecionar os registros necessários em um relatório, usando alguns campos como filtro. Essa parte entrada não é contada a parte, já está considerada nessa contagem.

26.5.2. Identificando Consultas

Consultas são, na prática, saídas simplificadas. Normalmente utilizadas para achar informações para modificá-las ou apagá-las. São sempre no monitor, não existe uma consulta em relatório de papel. Uma consulta não pode calcular nenhum valor. Em caso de cálculo de qualquer valor, temos uma saída.

26.6. Identificando Entradas

Entradas permitem adicionar, modificar e apagar informações. Se uma tela permite estas 3 funções, são contadas 3 entradas. Normalmente as funções de modificar e apagar ainda exigem consultas correspondentes para achar a informação que será alterada.

Um comando específico para o sistema executar algo é uma entrada.

Mensagens de erro que fazem parte de um processo não são contadas isoladamente,

26. Análise de Pontos de Função

mas sim como um DET. Por exemplo, se você esquecer de colocar um campo obrigatório e receber uma mensagem de erro, não deve contar uma saída a mais, e sim essa mensagem como um DET da entrada ou consulta.

Mensagens de notificação, por outro lado, são processos elementares e devem ser contados como uma saída a parte. Por exemplo, ao tentar comprar um produto que não existe e receber uma mensagem com essa notificação foi feito um processamento, que é contado como uma saída externa adicional.

26.6.1. Entendendo as Lógicas de Processamento

A lógicas de processamento, ou formas de processamento lógico, são as características que uma transação tem de acordo com os requisitos solicitados especificamente pelo usuário.

A partir dessas lógicas de processamento podemos determinar, como indicado na Tabela 34, qual o tipo de função transacional que deve ser contado.

Tabela 26.1.: Tabela de apoio a determinação do tipo de função transacional.
P=possível, N=Não permitida, O=obrigatório, O^{*} = obrigatório que pelo menos uma das condições aconteça.

Forma de Processamento Lógico	Tipo de Função Transacional		
	EE	SE	CE
Realiza validação dos dados	P	P	P
Realiza cálculos ou fórmulas matemáticas	P	O [*]	N
Converte valores equivalentes	P	P	P
Filtre dados e seleciona usando critérios específicos para comparar múltiplos conjuntos de dados	P	P	P
Analisa condições para determinar qual é aplicável	P	P	P
Altera ou inclui ao menos um ILF	O [*]	O [*]	N
Referencia ao menos um ILF ou EIF	P	P	O
Recupera dados ou informações de controle	P	P	O
Cria dados derivados	P	O [*]	N
Altera o comportamento do sistema	O [*]	O [*]	N
Prepara e apresenta informações fora das fronteiras do sistema	P	O	O
É capaz de aceitar dados ou informação de controle que entra pela fronteira da aplicação	O	P	P
Reordena ou reorganiza um conjunto de dados	P	P	P

26.7. Identificando Arquivos Internos

Arquivos representam um agrupamento lógico requerido pelo usuário. Podem incluir uma ou mais tabelas ou entidades.

Esse é uma das partes mais difíceis da contagem de pontos de função, pois devemos separar o que o usuário pensa do modelo que criamos. Nossa modelo muitas vezes usa vários grupos de dados, ou tabelas, ou entidades, para modelar algo que o usuário vê como um conceito único. Mesmo na modelagem conceitual, a tendência do analista é incluir entidades que o usuário não vê naturalmente.

Um Tipo de Elemento de Registro (RET, Record Element Type) é um subgrupo de elementos de dados dentro de um arquivo ou interface. Na prática, em um modelo de dados, um arquivo do usuário (ILF) é composto de um ou mais objetos do modelo.

Outra característica difícil de contar é que cada forma de acesso a um arquivo lógico conta novamente. Assim, por exemplo, se o usuário exige acessar um automóvel tanto por sua placa quanto por seu número do chassi, temos 2 arquivos lógicos para contar.

Exemplos: Uma nota fiscal é um arquivo lógico, com dois RETs: dados da nota, item de nota.

26.7.1. Identificando Arquivos Externos

Arquivos Externos representam um agrupamento lógico requerido pelo usuário. Podem incluir uma ou mais tabelas ou entidades.

Arquivos Externos são mantidos por outras aplicações. Arquivos importados contam também como Entrada Externa, arquivos exportados contam também como Consulta Externa ou Saída Externa.

26.7.2. Identificando Itens de Dados (DETs)

Itens de dados ou elementos de dados (DETs) campos únicos, reconhecidos pelos usuários, desconsiderando-se recursão e repetição. DETs também podem invocar ações.

Exemplos de DETs em entradas externas são:

- campos de entrada de dados;
- mensagens de erro;
- valores calculados que são guardados;
- botões;
- mensagens de confirmação, e
- campos repetidos contam apenas como um DET.

Exemplos de DETs em saídas externas são:

26. Análise de Pontos de Função

- campos em relatório;
- valores calculados
- mensagens de erro;
- cabeçalhos de coluna que são gerados, e dinamicamente em um relatório

Exemplos de DETs em consultas externas (além dos anteriores que se enquadrem) são:

- campos usados em filtros de procura, e
- o clique do mouse.

Em GUIs, botões onde só se pode fazer uma seleção entre muitas (normalmente radio buttons) devem ser contados como um DET apenas. Já *check boxes* são normalmente contadas uma a uma. Botões de comando devem ser contados como um elemento de dados levando em conta o fato de executarem uma função.

Tabela 26.2.: Cálculo da complexidade das saídas externas.

Arquivos Referenciados	Itens de dados referenciados		
	1-5	6-19	20+
0-1	Simples(4)	Simples(4)	Média(5)
2-3	Simples(4)	Média(5)	Complexa(7)
4+	Média(5)	Complexa(7)	Complexa(7)

Tabela 26.3.: Cálculo da complexidade das entradas externas.

Arquivos Referenciados	Itens de dados referenciados		
	1-5	6-19	20+
0-1	Simples(3)	Simples(3)	Média(4)
2	Simples(3)	Média(4)	Complexa(6)
3+	Média(4)	Complexa(6)	Complexa(6)

Tabela 26.4.: Cálculo da complexidade das consultas externas.

Arquivos Referenciados	Itens de dados referenciados		
	1-5	6-19	20+
0-1	Simples(3)	Simples(3)	Média(4)
2-3	Simples(3)	Média(4)	Complexa(6)
4+	Média(4)	Complexa(6)	Complexa(6)

Tabela 26.5.: Cálculo da complexidade dos arquivos lógicos internos.

RETs	Itens de dados referenciados		
	1-19	20-50	51+
1	Simples(7)	Simples(7)	Média(10)
2-5	Simples(7)	Média(10)	Complexa(15)
6+	Média(10)	Complexa(15)	Complexa(15)

Tabela 26.6.: Cálculo da complexidade das interfaces lógicas externas.

RETs	Itens de dados referenciados		
	1-19	20-50	51+
1	Simples(5)	Simples(5)	Média(7)
2-5	Simples(5)	Média(7)	Complexa(10)
6+	Média(7)	Complexa(10)	Complexa(10)

26.8. As Perguntas

São 14 as perguntas que devem ser feitas e ajudaram a determinar a quantidade de PF relativa a um sistema. Cada uma deve ser respondida com um número, de 0 a 5, indicando a importância da característica que se pergunta sobre o sistema, da seguinte forma:

- 0 - Não tem influência
- 1 - Influência incidental
- 2 - Influência moderada
- 3 - Influência média
- 4 - Influência significativa
- 5 - Influência essencial em todo o sistema

Para cada pergunta, o padrão IFPUG determina tipos de respostas padronizadas que nos permitem dar a resposta (entre 0 e 5) mais facilmente, como é exemplificado no item 1 (Comunicação de Dados). Foge ao objetivo desse texto fornecer um detalhamento completo do padrão de contagem, que pode ser obtido em (IFPUG, 2010).

- Quantas facilidades de comunicação existem para facilitar a transferência ou troca de informação com a aplicação ou sistema?
- Como será tratada a distribuição de dados e processamento?
- O usuário exige tempos de resposta ou throughput, ou seja, o desempenho é crítico?
- Quão fortemente é utilizada a plataforma (hardware) onde a aplicação vai ser executada?
- Qual a frequência das transações (diárias, semanais, altas o suficiente para exigir um estudo de desempenho)?
- Que percentagem das informações é inserida on-line? o Se mais de 30

26. Análise de Pontos de Função

- A aplicação é projetada para eficiência para o usuário final?
- Quantas ILFs são alteradas por transações on-line?
- A aplicação tem processamento lógico ou matemático extensivo?
- A aplicação é desenvolvida para atender um ou muitos tipos de usuários diferentes?
- Qual a dificuldade de conversão e instalação?
- Qual a eficiência e grau de automação de inicialização, backup e recuperação?
- A aplicação foi especialmente projetada, desenvolvida e suportada para funcionar em locais diferentes em diferentes organizações?
- A aplicação foi especialmente projetada, desenvolvida e suportada para facilitar mudanças?

26.9. Cálculo dos Pfs Finais

Os PF são calculados em etapas: contam-se os números de entradas, saídas, consultas, arquivos e interfaces do sistema; Para cada entrada se determina um grau de complexidade, de acordo com as tabelas complexidade da função , e somando os resultados (Tabela 40) se obtém a contagem básica de pontos de função; responde-se a uma série de perguntas, as quais fornecem, cada uma, um valor de 0 a 5 ($p_i, 0 \leq i \leq 5$), calcula-se o número de pontos de função com a equação:

$$PF = \text{total-de-2} \times (0,65 + 0,01 \times \sum(p_i)). \quad (26.1)$$

Devemos notar que se o cálculo de PF for usado para fazer previsões seguindo o método COCOMO, apenas a contagem básica é necessária (só devem ser feitos os passos 1 e 2).

Função	Contagem Total	Complexidade					Total
		Simples	x	Média	x	Complexa	
Saídas Externas		4		5		7	=
Consultas Externas		3		4		6	=
Entradas Externas		3		4		6	=
Arquivos Lógicos Internos		7		10		15	=
Arquivos de Interface Externos		5		7		10	=
		Total					

Figura 26.2.: Planilha de registro para ajudar a contar os Pontos de Função

26.10. Conclusão

Este capítulo é apenas uma introdução aos Pontos de Função. Este método é bastante padronizado e tem muitas dicas de como contar, mas para nossa necessidade, ter uma ideia do tamanho do sistema, a forma de contagem aqui apresentada é razoável. Não foram feitas simplificações no método básico, mas nos padrões são tratados detalhes que podem levar a diferenças na contagem entre um profissional e a realizada por quem usou esse capítulo. Isso não é muito grave, já que o autor conhece casos onde empresas diferentes contam Pontos de Função de um mesmo sistema de forma diferente, com disparidades de 100%.

O melhor também é que seja feita apenas a contagem básica, por alguns motivos. O primeiro é que ela é mais aceita no mundo todo, sendo que alguns países só normatizaram ela. O segundo é que, para dar as respostas as perguntas existem definições bem mais precisas no padrão da IFPUG, e só demos uma leitura geral. O terceiro é que algumas dessas perguntas são muito antigas em relação a tecnologia.

DRAFT

Uma Visão Reduzida do Modelo COCOMO II

Conteúdo

27.1.	Dois Modelos em Um	356
27.2.	Esfoco em Função do Tamanho	356
27.3.	Existe uma Relação Entre Esfoco e Tempo	357

Por que COCOMO II?

Conhecer as relações entre o tamanho do projeto e o esfoco necessário para desenvolvê-lo, e ainda o tempo necessário para esse esfoco, é um passo importante na finalização da análise de sistemas e ainda para permitir escolhas de escopo.

COCOMO II (Constructive Cost Model II) é um método de previsão de custos de software (Barry W. Boehm et al., 2000), focado no esfoco em pessoas-mês necessários para terminar um projeto. É um método baseado em estatísticas recolhidas em centenas de projetos de software. Por ser um método totalmente aberto, uma empresa pode calibrar o método para refletir o seu ambiente de desenvolvimento.

Atualmente é possível conseguir gratuitamente um software COCOMO, o que facilita o cálculo de custos de projeto de sistemas de informação¹.

Neste texto serão usadas as fórmulas e constantes do *COCOMO II Model Definition Manual version 1.4* (B. Boehm, Abts e Brad Clark, 1997).

¹Uma lista de softwares na rede pode ser encontrada em <https://sites.google.com/cos.ufrj.br/livroanalisedesistemas>

27.1. Dois Modelos em Um

O método COCOMO II possui dois modelos, um chamado *Early Design Model*, adequado ao momento inicial de estimativa do projeto, e outro chamado *Post Architecture Model*, adequado ao momento onde a arquitetura do ciclo de vida de software está desenvolvida.

O *Early Design Model* usa como métrica de entrada a contagem de pontos de função não ajustada, e produz como métrica de saída o esforço necessário para completar o projeto.

27.2. Esforço em Função do Tamanho

O modelo básico do COCOMO II calcula o esforço em Pessoas-Mês em função do tamanho do software. As fórmulas, porém, se baseiam no tamanho do software em SLOCs, logo deve ser usada uma tabela de conversão de Pontos de Função para linhas de código, como a mostrada na Tabela 27.1.

$$PM_{\text{NOMINAL}} = A \times (\text{Size})^B \quad (27.1)$$

Onde A captura o esforço multiplicativo do aumento de tamanho e o expoente B captura as economias, e deseconomias, de escala que são encontradas em projetos de tamanhos diferentes(B. Boehm, Abts e Brad Clark, 1997). A Figura 27.1 exemplifica o efeito dessas constantes.

Linguagem	SLOC/PF
Assembly (com macros)	213
C	128
C++	128
Pascal	91
Planilha Eletrônica	6

Tabela 27.1.: Conversão de Pontos de Função não ajustados para linhas de código.

Fonte: (B. Boehm, Abts e Brad Clark, 1997)

27.3. Existe uma Relação Entre Esforço e Tempo

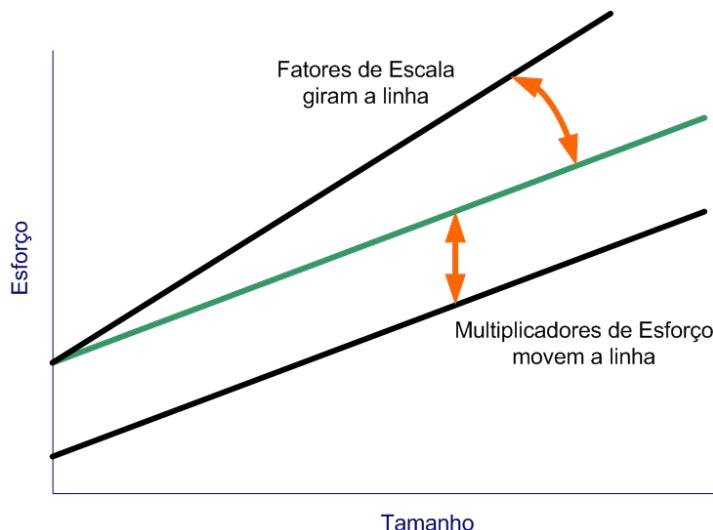


Figura 27.1.: Efeitos das constantes A e B na relação entre tamanho e esforço

27.3. Existe uma Relação Entre Esforço e Tempo

Um contribuição importante do método COCOMO II é determinar que existe uma relação entre o esforço necessário para desenvolver software e o tempo necessário para completar esse esforço. Antes mesmo de conhecer esse método, é importante deixar claro essa relação, pois muitos gerentes acreditam que podem espremer o tempo necessário para o desenvolvimento de um produto de software, o que leva as bastante conhecidas “Marchas da Morte” e, é claro, projetos atrasados, porque foram planejados de forma errada (Edward Yourdon, 1994; Edward Yourdon e Becker, 1997).

A suposição de que basta colocar mais gente, ou trabalhar mais horas, é suficiente para acelerar o tempo de desenvolvimento de um projeto, pode ser facilmente contestada com exemplos absurdos. Se esperamos que um software pode ser feito por 1 pessoa em 8 meses, é razoável imaginar, apesar de estar tecnicamente errado, que 2 pessoas poderiam fazê-lo em 4 meses. Porém, muito difficilmente 16 pessoas poderiam fazê-lo em 15 dias. Esse tipo de conta foi chamado de “Mito do Homem-Mês” (Brooks, 1995).

A relação entre o tempo de desenvolvimento e o esforço necessário, que apresentamos em sua forma completa a seguir, é parte importante do modelo COCOMO

$$T_{dev} = [C \times PM_{NS})^{D+0,2\times(E-B)}] \times \frac{SCED\%}{100} \quad (27.2)$$

Nessa fórmula PM_{NS} é a quantidade nominal de pessoas-mês , SCED% é a compressão necessária no tempo de desenvolvimento, B,C e D são constantes calibráveis e E é um coeficiente calculado a partir de fatores de escala.

Os valores das constantes estão na tabela

Tabela 27.2.: Valores das constantes no modelo COCOMO II padrão

A	2.94
B	0.91
C	3.67
D	0.28

O principal significado dessa fórmula pode ser associado ao fato que um projeto de desenvolvimento não pode ser comprimido, no tempo, além de certos limites. Coloquialmente, dizemos que software, como um bebê em gestação, pode até ficar bem feito se apressarmos o processo, porém não adianta pegar nove mães para fazer um bebê em um mês.

27.3.1. Cálculo dos Fatores de Escala

O valor do coeficiente E é calculado pela equação:

$$E = B + 0.01 \times \sum_{j=1}^N SF_j \quad (27.3)$$

Na fase inicial do projeto, os fatores de escala são 5 ($N = 5$), e representam a precedência do sistema (PREC), a flexibilidade do projeto (FLEX), o risco do projeto e da arquitetura (RESL), a coesão da equipe (TEAM) e a maturidade do processo (PMAT). Cada um desses fatores de escala pode variar entre os conceitos:

- *Very Low* (muito baixa);
- *Low* (baixa);
- *Nominal* (nominal ou média);
- *High* (alta);
- *Very High* (muito alta), e
- *Extra High* (extremamente alta).

Originalmente, esses valores deveriam variar entre 5 (VL) e 0 (EH), porém foram corrigidos estatisticamente. Em situação nominal para todos os casos, o valor de E seria a 1,06, porém o software COCOMO usa outros valores, calibrados, o valor de E acaba sendo 1,10.

$$PM = A \times \prod_{j=1}^{17} c_i \times \left(\left(1 + \frac{BRAK}{100} \right) \times PM_{NS} \right)^E \quad (27.4)$$

$$T_{DEV} = 3,67 \times (PM_{NS})^{0,32} \quad (27.5)$$

$$PM = 2,94 \times MLDC^{1,1} \quad (27.6)$$

Índice Remissivo

- RFP*, 231
Request for Proposal, 231
future value, 15
present value, 15
100-pontos, 20
5W2H, 28
5w2h, 27
abstração, 35, 36
abstração de caixa-preta, 39
alpha, 94
Análise de Kano, 20
análise de sistemas, 105
ARIS, 183
arquivos de interface externos, 346
arquivos lógicos internos, 346
atividades, 186
ator, 271
ator principal, 270, 271
atributos, 313
autorização, 215
bem sucedidos, 87
benefício, 22, 154
BPMN, 201
 atividade, 203
 Gateway, 209
 gateway
 baseado em eventos, 210
complexo, 211
exclusivo, 209
inclusivo, 210
paralelo, 210
paralelos baseado em eventos, 210
participante, 203
processo
 público, 202
processos
 privados, 202
subprocesso, 203
tarefa, 203
caixa-branca, 39
caso de uso, 269, 270
 diagrama, 286
caso de uso 2.0, 270
casos de uso, 267
causa raiz, 131
cenário, 271, 273
 alternativo, 271
 principal, 271
cenário alternativo, 274
cenário principal, 273
classe, 39
classificar, 39
classificação, 39
coisas, 316

ÍNDICE REMISSIVO

- colaboração, 202
com problemas, 87
Commercial Of The Shelf, 61
composição, 39, 41
condição, 43, 215, 271
consenso, 305
consultas externas, 346
coreografia, 202
COTS, 61
custo de oportunidade, 8
- dados, 46
deadlock, 196
declaração estrutural, 216
declarações de ação, 215
declarações estruturais, 215
decomposição, 42
demanda, 9, 161
estados, 9
derivações, 215
DESCOBRIR, 316
descrições, 316
desejo, 9
diagrama de caso de uso, 286
Diagrama de Causa Raiz, 137
Diagrama de Espinha de Peixe, 137
dinheiro de brinquedo, 20
do nothing branch, 198
documentador, 305
domínio, 313
- elicitação, 237
Engenharia de Software, 77, 89
entidade, 315
entidades, 311
entradas externas, 346
entrega intermediária, 100
entregas, 167
Entreprise Resource Planning, 61
EPC, 183
 Loops, 195
 Padrões, 197
 paralelismo, 197
equipe, 164, 305
- ER, 309
ERP, 61, 71
esforço, 335
especialização, 41
especificação de requisitos, 230
especificações, 316
estimativa de tamanho software, 333
estruturas de dados, 78
evento, 187
eventos, 187, 316
- fato, 217
fatos, 215
fluxo condicional, 211
fluxo de controle, 185
fluxo de exceção, 211
fluxo de execução, 185
fluxo de mensagem, 211
fluxo de sequência normal, 211
fluxo default, 211
fluxo não controlado, 211
foco, 43
foco/inibição, 39
fracassos, 87
Framework de Zachman, 30
funcionalidade, 87
função utilidade, 7
- garantias, 282
generalização, 39, 41
- hardware, 78
hiperonímia, 41
hiponímia, 41
história do usuário, 257, 258
histórias complexas, 265
histórias compostas, 265
holónímia, 42
holônimo, 42
- identificação, 39, 42
informação, 45, 78
inibição, 43
instanciação, 40
instrução, 78

- instância, 39, 312
instância de entidade, 312
interativo, 71
interações, 316
interesse, 117
interface de processo, 189
interfaces de processos, 189
intervalos, 316
IRACIS, 22, 157

JAD, 305
Joint Application Design, 305
juro, 15
justificativa, 152

Kano, 20
kernel, 94

legitimidade, 121
lista de atores, 286
locais, 316
líder de sessão, 305

markup, 7
Matriz de Engajamento, 119
menor produto viável, 264
MER, 309
meronímia, 42
merônimo, 42
meta, 155
metas, 59
missão, 59
Modelagem de negócio, 177
modelagem de processos, 178
Modelagem de Processos de Negócio, 179
modelo, 35, 36
Modelo de Entidades e Relacionamentos, 309, 311
modo de trabalho, 94
moeda virtual, 247
momentos, 316
montante, 15
MoSCoW, 20, 247
MTBF, 235
mundo claro, 215

mundo escuro, 215
método, 94

narrativa, 273
numerada, 273
necessidade, 9

objetivo, 117, 271
objetivos, 59
objetivos SMART, 152
objetos tangíveis, 316
ocultação da informação, 39
oportunidade, 94, 129, 130
 tecnológica, 143
oportunidades
 de negócio, 143
organização, 56
organograma, 62
orçamento, 87

pagamento, 15
papéis, 316
papéis exercidos, 316
parte interessada, 162, 281
partes interessadas, 94, 101, 110
patrocinador, 305
pessoas, 316
pitch, 150
planning poker, 339, 341
poder, 121
políticas de negócio, 214
Pontos de função, 343
Pontos de história, 338
Pontos de história, 341
prazo, 15, 87
premissa, 166
prestação, 15
principal, 15
princípio da diminuição da utilidade
 marginal, 8
Princípio de Pareto, 133
problema, 129
problemas de negócio, 132
problemas funcionais, 132
problemas operacionais, 132

ÍNDICE REMISSIVO

- processo, 178
processo de negócio, 178
Project Model Canvas, 145
projeto, 99
prática, 94
pré-condição, 281
pós-condição, 282
reativo, 71
refinamento sucessivo, 39, 43
regra de negócio, 214
regras, 188
remuneração, 15
requisito, 94, 160, 230
 arbitrário, 242, 243
 do negócio, 242
 do sistema, 242
 do software, 242
 do usuário, 242
 elicitação, 237
 especificação, 230
 falso, 242
 tecnológico, 242
requisito de software, 257
requisitos de desempenho, 236
requisitos de interface, 236
requisitos funcionais, 234
requisitos não funcionais, 234
restrição, 215
restrições, 234
risco, 170
saldo, 15
saliência, 121
satisfação, 10
saídas externas, 345
separação de interesses, 39, 44
simplificação pelo Caso Normal, 39
simplificação pelo caso normal, 43
sistema, 49, 70
sistema de software, 94
Sistemas de Informação, 57
sistemas de informação, 69
sistemas de respostas planejadas, 71
SMART, 152
sobreposição, 263
software, 78
stakeholders, 110
tamanho do software, 333, 335
taxa, 15
taxa de juros, 15
termo, 216
termos atômicos, 215
time, 94
tipo de entidade, 311
trabalho, 94
uma aluno pertence a um curso, 217
urgência, 121
user story, 257
usuário, 113
usuário externo, 114
usuário final, 113
usuário interno, 114
usuário onisciente, 113
utilidade, 7
valor, 5, 6, 10, 21, 22, 79
valor atual, 15
valor de resgate, 15
valor descontado, 15
valor futuro, 15
valor presente, 15
visão, 59
Volere, 247
wicked problems, 85

Bibliografia

- Adzic, Gojko e David Evans (2014). *Fifty Quick Ideas to Improve your User Stories*. Neuri Consulting.
- Albrecht, A.J (1979). “Measuring Application Development Productivity”. Em: *Proc. IBM Aplic. Dev. Symposium*. Monterey, CA, pp. 89–92.
- Almquist, Eric, Jamie Cleghorn e Lori Sherer (2018). “The B2B Elements of Value”. *Harvard Business Review*, pp. 72–81. URL: <https://hbr.org/2018/03/the-b2b-elements-of-value> (acesso em 09/02/2020).
- Almquist, Eric, John Senior e Nicolas Bloch (1 de set. de 2016). “The Elements of Value”. *Harvard Business Review*, pp. 46–53. URL: <https://hbr.org/2016/09/the-elements-of-value> (acesso em 09/02/2020).
- American Heritage (2019). *The American Heritage Dictionary of the English Language*. URL: <https://www.ahdictionary.com/> (acesso em 25/12/2019).
- Andersen, Erling S, Kristoffer V Grude e Tor Haug (2009). *Goal Directed Project Management: Effective techniques and strategies*. 4^a ed. London: Kogan Page.
- Anklesaria, Jimmy (2008). *Supply Chain Cost Management. The AIM & DRIVE Process for Achieving Extraordinary Results*. New York: Amacom - American Management Association.
- Beck, Kent (29 de set. de 1999). *eXtreme Programming eXplained: Embrace Change*. 1^a ed. USA: Addison-Wesley Longman Publishing Co., Inc.
- Beck, Kent, Alistair Cockburn et al. (10 de nov. de 2014). *User Story And Use Case Comparison*. URL: <http://wiki.c2.com/?UserStoryAndUseCaseComparison> (acesso em 13/01/2020).
- Beck, Kent, Martha Roden et al. (8 de jan. de 2014). *User Story*. URL: <http://wiki.c2.com/?UserStory> (acesso em 13/01/2020).
- Bellovin, Steve (11 de ago. de 1997). *Software error may have contributed to Guam crash*. Ed. por Peter G. Neumann. The RISKS Digest Forum on Risks to the Public in Computers, Related Systems ACM Committee on Computers e Public Policy.

Bibliografia

- URL: %5Curl%7Bhttp://catless.ncl.ac.uk/Risks/19/29#subj1%7D (acesso em 20/12/2019).
- Bertalanffy, Ludwig von (1969). *General System Theory. Foundations Development Applications*. revised. George Braziller, Inc.
- Bertini, C., S. Ceri e Shamkant B. Navathe (1992). *Conceptual Database Design*. The Benjamin/Cummings Publishing Company.
- BeSeen (17 de ago. de 2015). *The value triangle managing customers expectations*. BeSeen. URL: <https://www.beseen-marketing.co.uk/marketing-blog/the-value-triangle> (acesso em 09/02/2020).
- Biffl, Stefan et al., ed. (2006). *Value-Based Software Engineering*. Berlin, Heidelberg: Springer.
- Blanchard, Kenneth e Spencer Johnson (1983). *The One Minute Manager*. 10th anniversary. Berkley Trade.
- Boehm, B. W. (mai. de 1988). “A spiral model of software development and enhancement”. *Computer* 21.5, pp. 61–72. ISSN: 1558-0814. DOI: 10.1109/2.59.
- Boehm, Barry, Chris Abts e and Sunita Devnani-Chulani Brad Clark (1997). *COCOMO II Model Definition Manual version 1.4*.
- Boehm, Barry e Hasan Kitapci (2006). “The WinWin Approach: Using a Requirements Negotiation Tool for Rationale Capture and Use”. Em: *Rationale Management in Software Engineering*. Ed. por Raymond Dutoit Allen H. and McCall, Ivan Mistrík e Barbara Paech. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 173–190. ISBN: 978-3-540-30998-7. DOI: 10.1007/978-3-540-30998-7_8. URL: https://doi.org/10.1007/978-3-540-30998-7_8.
- Boehm, Barry W. et al. (2000). *Software Cost Estimation with Cocomo II with Cdrom*. 1st. USA: Prentice Hall PTR. ISBN: 0130266922.
- Brasil (2018). *Roteiro de Métricas de Software do SISP: versão 2.3*. Roteiro. Versão 2.3. Ministério do Planejamento, Desenvolvimento e Gestão. Secretaria de Tecnologia da Informação e Comunicação - Setic.
- Brooks, Frederick P. (1978). *The Mythical Man-Month. Essays on Software Engineering*. 1^a ed. USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0201006502.
- (1995). *The Mythical Man-Month (Anniversary Ed.) Essays on Software Engineering*. 2^a ed. USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0201835959.
- Bunge, Mario (1979). *Treatise on Basic Philosophy - Ontology II: A World of Systems*. Vol. 4. Treatise on Basic Philosophy. Springer Netherlands. 314 pp. ISBN: 978-90-277-0944-8.
- Cairncross, A. (1951). *Introduction to Economics*. Butterworth.
- Carr, Nicholas G. (22 de jan. de 2005). “Does Not Compute”. *The New York Times*. ISSN: 0362-4331. URL: <http://www.nytimes.com/2005/01/22/opinion/does-not-compute.html> (acesso em 04/03/2017).
- Chaitin, Gregore (mar. de 2006). “The Limits of Reason”. *Scientific American*.
- Chen, Peter (1990). *Modelagem de Dados: A abordagem entidade-relacionamento para projeto lógico*. São Paulo: Makron Books.
- Chiavenato, Idalberto (2014). *Introdução a Teoria Geral da Administração*. 9^a ed. Barueri, SP: Editora Manole.

- Christel, M. e K. Kand (set. de 1992). *Issues in Requirements Elicitation*. Technical Report CMU/SEI-92-TR-012. Software Engineering Institute / CMU, p. 80. URL: https://resources.sei.cmu.edu/asset_files/TechnicalReport/1992_005_001_16478.pdf (acesso em 24/02/2020).
- Claude E Shannon, Warren Weaver (1963). *The Mathematical Theory of Communication*. University of Illinois Press.
- Coad, Peter, Jeff de Luca e Eric Lefebvre (1999). *Java Modeling Color with Uml: Enterprise Components and Process with Cdrom*. 1st. USA: Prentice Hall PTR. ISBN: 013011510X.
- Cockburn, Alistair (jan. de 2000). *Writing Effective Use Cases*. Addison-Wesley.
- Cohn, Mike (2004). *User Stories Applied: For Agile Software Development*. USA: Addison Wesley Longman Publishing Co., Inc. ISBN: 0321205685.
- (2005). *Agile Estimating and Planning*. Prentice Hall.
- CompTIA (nov. de 2019). *CompTIA IT Industry Outlook 2020: Taking the Next Step*. Report. Computing Technology Industry Association (CompTIA). URL: <https://comptiacdn.azureedge.net/webcontent/docs/default-source/research-reports/comptia-it-industry-outlook-2020.pdf>.
- Conklin, Jeff (18 de nov. de 2005). *Dialogue Mapping: Building Shared Understanding of Wicked Problems*. 1 edition. Chichester, England ; Hoboken, NJ: Wiley. 266 pp. ISBN: 978-0-470-01768-5.
- Cougo, Paulo (1999). *Modelagem Conceitual e Projeto de Banco de Dados*. Rio de Janeiro: Campus.
- Craigie, Dan, Susan Gerhart e Ted Ralston (1993). “An International Survey of Industrial Applications of Formal Methods”. Em: *Z User Workshop, London 1992*. Ed. por J. P. Bowen e J. E. Nicholls. London: Springer London, pp. 1–5.
- Czarnacka-Chrobot, Beata (jul. de 2012). “What Is the Cost of One IFPUG Method Function Point? Case Study”. Em: *The 11th International Conference on Software Engineering Research and Practice (SERP12), The 2012 World Congress in Computer Science, Computer Engineering & Applied Computing (WORLDCOMP12)*. The 11th International Conference on Software Engineering Research and Practice (SERP12), The 2012 World Congress in Computer Science, Computer Engineering & Applied Computing (WORLDCOMP12), Las Vegas, Nevada, USA: CSREA Press.
- Dal Zot, Wili e Manuela Longoni de Castro (2015). *Matemática Financeira: fundamentos e aplicações*. Porto Alegre: Bookman, p. 151.
- Dalkey, Norman Crolee (out. de 1966). *Delphi*. Technical Report P-3704. Santa Monica, California: RAND Corporation. URL: <https://www.rand.org/pubs/papers/P3704.html> (acesso em 16/01/2020).
- Dijkstra, Edsger W. (out. de 1972). “The Humble Programmer”. *Commun. ACM* 15.10, pp. 859–866. ISSN: 0001-0782. DOI: 10.1145/355604.361591. URL: <http://doi.acm.org/10.1145/355604.361591>.
- Dimitrov, Dimitri (2020). *Software Project Estimation. Intelligent Forecasting, Project Control, and Cliente Relationship Management*. Apress.

Bibliografia

- Erdogmus, Hakan, John Favaro e Michael Halling (2006). "Valuation of Software Initiatives Under Uncertainty: Concepts, Issues, and Techniques". Em: *Value-Based Software Engineering*. Ed. por Stefan Biffl et al. Berlin, Heidelberg: Springer.
- Extra (18 de jun. de 2019). *Se quer levar mais de 10 quilos, pague, sem problema nenhum', diz Bolsonaro sobre fim do despacho gratuito*. URL: <https://extra.globo.com/noticias/economia/se-quer-levar-mais-de-10-quilos-pague-sem-problema-nenhum-diz-bolsonaro-sobre-fim-do-despacho-gratuito-23747656.html> (acesso em 30/01/2020).
- Facetation (12 de mai. de 2015). *What is the history of the RACI chart?* URL: <http://facetation.blogspot.com/2015/05/what-is-history-of-raci-chart.html> (acesso em 06/10/2020).
- FATTO (2020). *Quanto pagar por um ponto de função?* URL: <https://docplayer.com.br/13499465-Quanto-pagar-por-um-ponto-de-funcao.html> (acesso em 02/02/2020).
- Fedotov, Alex (22 de fev. de 2019). *Septem Circumstantiae, five W's and H or 'six serving-men'*. URL: <https://alxfed.github.io/blog/posts/2019/02/22/Septem-Circumstantiae.html> (acesso em 08/01/2020).
- Finocchio Jr., José (2013). *Project model Canvas*. Rio de Janeiro: Elsevier.
- Folha Online (29 de mai. de 2006). *Volkswagen anuncia recall de 123 mil veículos Gol, Fox e Kombi*. Folha Online - Dinheiro. URL: <http://www1.folha.uol.com.br/folha/dinheiro/ult91u108087.shtml> (acesso em 03/03/2017).
- Franceschini, Fiorenzo (2016). *Advanced Quality Function Deployment*. CRC Press, p. 208. ISBN: 9781420025439.
- Gane, Chris P. e Trish Sarson (1979). *Structured Systems Analysis: Tools and Techniques*. 1^a ed. Prentice Hall Professional Technical Reference. ISBN: 0138545472.
- Gibbs, Wayt W. (set. de 1994). "Software's Chronic Crisis". *Scientific American*, pp. 86–100.
- Gillenson, Mark L. e Robert Goldberg (1984). *Strategic Planning, Systems Analysis, and Database Design: The Continuous Flow Approach*. John Wiley & Sons.
- Gladys S. W. Lam, Ronald G. Ross and (out. de 2015). *Building Business Solutions: Business Analysis with Business Rules*. 2^a ed. sponsored by IIBA: International Institute of Business Analysis. Business Rule Solutions LLC., p. 304.
- Greenlaw, Steven A., David Shapiro e Timothy Taylor (2017). *Principles of Microeconomics for AP Courses*. 2^a ed. OpenStax, Rice University.
- Group, Gartner (21 de mar. de 2014). *Gartner Says Worldwide Software Market Grew 4.8 Percent in 2013*. URL: [%5Curl%7Bhttp://www.gartner.com/newsroom/id/2696317%7D](http://www.gartner.com/newsroom/id/2696317%7D).
- Hastie, Shane e Stéphane Wojewoda (4 de out. de 2015). *Standish Group 2015 Chaos Report - Q&A with Jennifer Lynch*. URL: <https://www.infoq.com/articles/standish-chaos-2015/> (acesso em 15/12/2019).
- Hay, D. et al. (jul. de 2000). *Defining Business Rules: What they are really?* Final Report. The Business Rule Group.
- Hayes, Adam (25 de jun. de 2019). *Internal Rate of Return IRR*. Investopedia. URL: <https://www.investopedia.com/terms/i/IRR.asp> (acesso em 08/02/2020).

- Head, Robert V. (out. de 2002). "Getting Sabre off the Ground". *IEEE Annals of the History of Computing* 24 (4), pp. 32–39. DOI: 10.1109/MAHC.2002.1114868.
- Helmer-Hirschberg, Olaf (mar. de 1967). *Analysis of the Future: The Delphi Method*. Technical Report P-3558. Santa Monica, California: RAND Corporation. URL: <https://www.rand.org/pubs/papers/P3558.html> (acesso em 16/01/2020).
- Higgins, J.M. (1994). *101 Creative Problem Solving Techniques: The Handbook of New Ideas for Business*. New Management Publishing Company. ISBN: 9781883629007. URL: https://books.google.com.br/books?id=Q1%5C_9LcYV03kC.
- Holanda Ferreira, Aurélio Buarque de (1986). *Novo Dicionário da Língua Portuguesa*. 2^a ed. Nova Fronteira.
- Houaiss, Antônio, Mauro Villar e Francisco Manoel de Mello Franco (2009). *Dicionário Houaiss da língua portuguesa*. Objetiva. ISBN: 978-972-759-664-5.
- IEEE (25 de jun. de 1998). *Recommended Practice for Software Requirements Specifications*. Standard ISO/IEC/IEEE 830:1998. IEEE.
- (dez. de 2010). *ISO/IEC/IEEE International Standard - Systems and software engineering – Vocabulary*. Standard 24765:2010(E). IEEE, pp. 1–418. DOI: 10.1109/IEEESTD.2010.5733835.
 - (2017). *ISO/IEC/IEEE International Standard - Systems and software engineering– Vocabulary*. Standard ISO/IEC/IEEE 24765:2017(E). ISO/IEC/IEEE, pp. 1–541. DOI: 10.1109/IEEESTD.2017.8016712.
 - (nov. de 2018). *ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes – Requirements engineering*. Standard 29148:2018. IEEE, pp. 1–104. DOI: 10.1109/IEEESTD.2018.8559686.
- IEEE Computer Society (17 de jan. de 2014a). *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. Ed. por Pierre Bourque e Richard E. Fairley. 3 edition. IEEE Computer Society Press. 346 pp. ISBN: 978-0-7695-5166-1.
- (2014b). *Software Engineering Competency Model Version 1.0 SWECOM*. New York: IEEE Computer Society.
- IFPUG (2010). *Function Point Counting Practices Manual Release 4.3.1*. Standard. International Function Point Users Group.
- (2012). *The IFPUG Guide to IT and Software Measurement*. Auerbach Publications.
- IIBA (2011). *Um guia para o Corpo de Conhecimento de Análise de Negócios (Guia BABOK) Version 2.0*. Ontário, Canadá: International Institute of Business Analysis.
- ISBS, International Software Benchmarking Standards Group - (2010). *Practical Software Project Estimation: A Toolkit for Estimating Software Development Effort & Duration*. Ed. por Peter Hill. McGraw Hill.
- Jacobson, Ivar (dez. de 1987). "Object-Oriented Development in an Industrial Environment". *SIGPLAN Not.* 22.12, pp. 183–191. ISSN: 0362-1340. DOI: 10.1145/38807.38824. URL: <https://doi.org/10.1145/38807.38824>.
- (1 de ago. de 2004). "Use cases Yesterday, today, and tomorrow". *Software & Systems Modeling* 3.3, pp. 210–220. ISSN: 1619-1374. DOI: 10.1007/s10270-004-0060-3. URL: <https://doi.org/10.1007/s10270-004-0060-3>.

Bibliografia

- Jacobson, Ivar, Magnus Christerson et al. (1992). *Object-oriented software engineering: a use case driven approach*. ACM Press Series. Reading: Addison-Wesley. ISBN: 9780201544350.
- Jacobson, Ivar, Harold "Bud" Lawson e Pan-Wei Ng (19 de jul. de 2019). *The Essentials of Modern Software Engineering: Free the Practices from the Method Prisons!* ACM Books. 400 pp. ISBN: 978-1-947487-24-6.
- Jacobson, Ivar, Paul E. McMahon e Roland Racko (2015). *24 Questions SEMAT and Essence: The Whys, Whats and Hows to See the Difference*. Rel. técn. Ivar Jacobson International.
- Jacobson, Ivar, Pan-Wei Ng et al. (26 de jan. de 2013). *The Essence of Software Engineering: Applying the SEMAT Kernel*. 1 edition. Upper Saddle River, NJ: Addison-Wesley Professional. 352 pp. ISBN: 978-0-321-88595-1.
- Jacobson, Ivar, Ian Spence e Kurt Bittner (dez. de 2011). *Use-Case 2.0. The Guide to Succeeding with Use Cases*. White Paper. Ivar Jacobson International S.A.
- Jeffries, Ron (30 de ago. de 2001). *Essential XP: Card, Conversation, Confirmation*. URL: <https://ronjeffries.com/xprog/articles/expcardconversationconfirmation/> (acesso em 13/01/2020).
- Johnson, Phil (8 de ago. de 2012). *Curiosity about lines of code*. ITworld. URL: <http://www.itworld.com/article/2725085/big-data/curiosity-about-lines-of-code.html> (acesso em 04/03/2017).
- Jones, Caper (abr. de 2005). "Software Cost Estimating Methods for Large Projects". *Crosstalk The Journal of Defense Software Engineering* (April 2005). Disponível via WayBack Machine. URL: <http://www.stsc.hill.af.mil/crosstalk/2005/04/0504Jones.html>.
- Kendall, Kenneth E. e Julie E. Kendall (2013). *Systems Analysis and Design*. 9^a ed. Pearson.
- Kenton, Will (25 de jun. de 2019). *Net Present Value (NPV)*. Investopedia. URL: <https://www.investopedia.com/terms/n/npv.asp> (acesso em 08/02/2020).
- Kerzner, Harold (2017). *Project Management. A system approach to planning, scheduling and controlling*. 12^a ed. Hoboken, New Jersey: John Wiley & Sons.
- Kevin L (2016). *SMART Killer Project Management in just Five Steps*. URL: <http://trustteck.com/smarter-killer-project-management/> (acesso em 30/12/2019).
- Kim, W.C. e R. Mauborgne (2005). *A Estratégia Do Oceano Azul*. Elsevier. ISBN: 9788535215243.
- Kimball, Ralph et al. (2008). *The Data Warehouse Lifecycle Toolkit: Practical Techniques for Building Data Warehouse and Business Intelligence Systems*. 2nd. Wiley.
- King, J. E. e Michael McLure (2014). *History of the Concept of Value*. Discussion Paper 14.06. The University of Western Australia, Department of Economics. URL: <https://ideas.repec.org/p/uwa/wpaper/14-06.html>.
- Kotler, Philip, Gary Armstrong et al. (2017). *Principles of Marketing*. 7th European Edition. Pearsmpm.
- Kotler, Philip e Kevin Lana Keller (2012). *Marketing Management*. 14^a ed. Boston: Prentice Hall.
- (2013). *Administração de Marketing*. 14^a ed. São Paulo: Pearson.

- Krasner, Herb (1 de jan. de 2021). *The Cost of Poor Software Quality in the US: A 2020 Report*. type. CISQ Consortium for Information & Software Quality. 46 pp. URL: <https://www.it-cisq.org/pdf/CPSQ-2020-report.pdf> (acesso em 26/06/2021).
- Krugman, Paul e Robin Wells (2013). *Microeconomics*. 3^a ed. New York: Worth Publishers.
- Ladkin, Peter (7 de mai. de 1996). *The Cali and Puerto Plata B757 Crashes*. Ed. por Peter G. Neumann. Volume 18 Issue 10. The RISKS Digest Forum on Risks to the Public in Computers, Related Systems ACM Committee on Computers e Public Policy. URL: %5Curl%7Bhttp://catless.ncl.ac.uk/Risks/18/10#subj1%7D (acesso em 20/12/2019).
- Laudon, Kenneth e Jane Laudon (2011). *Sistemas de Informação Gerenciais*. 9^a edição. São Paulo: Pearson.
- Leffingwell, Dean e Don Widrig (1999). *Managing Software Requirements: A Unified Approach*. USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0201615932.
- Levenson, Nancy G. e Clark S. Turner (jul. de 1993). “An Investigation of the Therac-25 Accidents”. *Computer* 26.7, pp. 18–41.
- Levere, Jane L. (16 de mar. de 2001). “TECHNOLOGY; Electronic Data Systems to Buy Sabre Airline Computer Unit”. *The New York Times*, Section C, Page 8. URL: <https://www.nytimes.com/2001/03/16/business/technology-electronic-data-systems-to-buy-sabre-airline-computer-unit.html> (acesso em 02/02/2020).
- Lions, J. L. (19 de jul. de 1996). *ARIANE 5 Flight 501 Failure Report by the Inquiry Board*. Report. Paris: European Space Agency.
- Maximini, Dominik (2018). *The Scrum Culture. Introducing Agile Methos in Organizations*. 2^a ed. Management for Professionals. Springer.
- McMenamin, Stephen M. e John F. Palmer (1984). *Essential Systems Analysis*. USA: Yourdon Press. ISBN: 0917072308.
- Mitchell, Ronald K., Bradley R. Agle e Donna J. Wood (1997). “Toward a Theory of Stakeholder Identification and Salience: Defining the Principle of Who and What Really Counts”. *The Academy of Management Review* 22.4, pp. 853–886. ISSN: 0363-7425. DOI: 10.2307/259247. URL: %5Curl%7Bhttp://www.jstor.org/stable/259247%7D (acesso em 15/01/2017).
- Moorman, Jan (9 de out. de 2012). *Leveraging the Kano Model for Optimal Results*. Article No :882. UX Magazine. URL: <https://uxmag.com/articles/leveraging-the-kano-model-for-optimal-results> (acesso em 08/02/2020).
- Naur, Peter e Brian Randell, ed. (jan. de 1969). *Software Engineering. Report on a conference sponsored by the NATO SCIENCE COMMITTEE Garmisch, Germany, 7th to 11th October 1968*.
- Niquette, Paul (2006). *Softword: Provenance for the Word 'Software'*. URL: <http://niquette.com/books/softword/tocsoft.html> (acesso em 02/02/2017).
- NIST (1993). *Integration Definition for Information Modeling (IDEF1X)*. Federal Information Processing Standards Publication FIPS PUB 184. National Institute of Standards e Technology.
- Nonaka, Ikujiro e Hirotaka Takeuchi (1995). *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*. New York: Oxford University Press, xii, 284 p.

Bibliografia

- North, Dan (16 de nov. de 2013). *User Story Template*. URL: <http://wiki.c2.com/?UserStoryTemplate> (acesso em 13/01/2020).
- OMG (jan. de 2013). *Business Process Model and Notation (BPMN) Version 2.0.2*. Standard. Object Management Group. URL: <http://www.omg.org/spec/BPMN/2.0>.
- (19 de mai. de 2015). *OMG Business Motivation Model. version 1.3*. specification formal/2015-05-19. Object Management Group.
 - (5 de dez. de 2017). *OMG Unified Modeling Language (OMG UML)*. version 2.5.1. specification formal/2017-12-05. Object Management Group.
 - (out. de 2018). *Essence Kernel and Language for Software Engineering Methods: Version 1.2*. specification. Object Management Group. URL: [%5Curl%7Bhttps://www.omg.org/spec/Essence/%7D](#).
 - (out. de 2019). *Semantics of Business Vocabulary and Business Rules*. specification formal/2019-10-02. Versão 1.5. Object Management Group - OMG.
- Osterwalder, A., Y. Pigneur e T. Clark (2010). *Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers*. Strategyzer series. Wiley.
- Park, Robert E. (set. de 1992). *Software Size Measurement: A Framework for Counting Source Statements*. Technical Report CMU/SEI-92-TR-020 ESC-TR-92-020. Software Engineering Institute, Carnegie Mellon University. URL: https://resources.sei.cmu.edu/asset_files/TechnicalReport/1992_005_001_16082.pdf (acesso em 16/01/2020).
- Parrochia, Daniel (2020). *Classification*. URL: <https://www.iep.utm.edu/classifi/> (acesso em 21/01/2020).
- Patton, Jeff e Peter Economy (2014). *User Story Mapping: Discover the Whole Story, Build the Right Product*. 1st. O'Reilly Media, Inc. ISBN: 1491904909.
- PMI (2017). *Um Guia do Conhecimento em Gerenciamento de Projetos (Guia PM-BOK)*. Sexta Edição. Newtown Square, PA: Project Management Institute. ISBN: 9788502223745.
- Pressman, Roger S. e Bruce Maxim (23 de jan. de 2014). *Software Engineering: A Practitioner's Approach*. 8 edition. New York, NY: McGraw-Hill Education. 976 pp. ISBN: 978-0-07-802212-8.
- Ricardo, David (1996). *Princípios de Economia Política e Tributação*. Os Economistas. texto original de 1821. São Paulo: Editora Nova Cultural Ltda.
- Robertson, James e Suzanne Robertson (1998). *Complete Systems Analysis*. New York: Dorser House.
- (2006). *Mastering the Requirements Process*. 2^a ed. Addison-Wesley Professional. ISBN: 0321419499.
- Rosen, Gideon (2018). “Abstract Objects”. Em: *The Stanford Encyclopedia of Philosophy*. Ed. por Edward N. Zalta. Winter 2018. Metaphysics Research Lab, Stanford University.
- Rösler, Wolfram (1994). *The Hello World Collection*. URL: <http://helloworldcollection.de/> (acesso em 15/01/2020).
- Ross, R.G. (1998). *Business Rule Concepts: The New Mechanics of Business Information Systems*. Business Rule Solutions. ISBN: 9780941049047.
- Ross, Ronald G. (1999). *The Business Rule Book: Classifying, Defining and Modeling Rules*. Vol. Second. Boston, Massachusetts: Database Research Group.

- (ago. de 2004). “The Light World vs. the Dark World ~Business Rules for Authorization”. *Business Rules Journal* 5.8. URL: <http://www.brcommunity.com/a2004/b201.html> (acesso em 08/02/2020).
- (2017). *How to Define Business Terms in Plain English: A Primer. Creating definitions using ConceptSpeak*. White Paper. Business Rule Solutions.
- Rowley, Jennifer (2006). “Where is the wisdom that we have lost in knowledge?” *Journal of Documentation* 62 (2), pp. 251–270.
- (2007). “The wisdom hierarchy: representations of the DIKW hierarchy.” *Journal of Information Science* 33.2, pp. 163–180.
- Rubin, Kenneth S. (2013). *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Pearson Education.
- Ruble, David A. (1997). *Practical Analysis & Design for Client/Server and GUI Systems*. Upper Saddle River: Yourdon Press.
- Satpathy, Tridibesh et al. (2016). *A Guide to the SCRUM BODY OF KNOWLEDGE (SBOK Guide). A Comprehensive Guide to Deliver Projects using Scrum*. 2016^a ed. SCRUMStudy, VMEdú, Inc.
- Savage, Neil (jun. de 2020). “An Animating Spirit”. *Communications of the ACM* 63.6, pp. 10–12. DOI: 10.1145/3392520.
- Scheel, Henrik von et al. (2015). “Phase 1: Process Concept Evolution”. Em: *The Complete Business Process Handbook*. Elsevier, pp. 1–9. DOI: 10.1016/b978-0-12-799959-3.00001-x.
- SEMAT Inc. (2019). *ESSENCE User Guide*. URL: %5Curl%7Bhttp://semat.org/essence-user-guide%7D (acesso em 20/12/2019).
- Serrat, Olivier (2017). “The Five Whys Technique”. Em: *Knowledge Solutions: Tools, Methods, and Approaches to Drive Organizational Performance*. Singapore: Springer Singapore, pp. 307–310. ISBN: 978-981-10-0983-9. DOI: 10.1007/978-981-10-0983-9_32. URL: https://doi.org/10.1007/978-981-10-0983-9_32 (acesso em 10/02/2020).
- Shlaer, Sally e Stephen J. Mellor (1999). *Object-Oriented Systems Analysis, Modelling the World in Data*.
- Sloan, Michael C. (2010). “Aristotles Nicomachean Ethics as the Original Locus for the Septem Circumstantiae”. *Classical Philology* 105.3, pp. 236–251. ISSN: 0009837X, 1546072X. URL: <http://www.jstor.org/stable/10.1086/656196>.
- Smith, Adam (1996). *A Riqueza das Nações. Investigação sobre sua natureza e suas causas*. Os Economistas. texto original de 1776. São Paulo: Editora Nova Cultural Ltda.
- (2003). *The Wealth of Nations*. texto original de 1854. Bantam Classics. ISBN: 0553585975.
- Software AG (2019a). *Event*. Help > ARIS Express > Glossary > Event. acessado em 2019-11-01. Aris Community by Software AG. URL: <https://www.ariscommunity.com/help/arlis-express/35908> (acesso em 01/11/2019).
- (2019b). *Rule*. Help > ARIS Express > Glossary > Rule. Help > ARIS Express > Glossary > Rule. acessado em 1/11/2019. Softwre AG. URL: <https://www.ariscommunity.com/help/arlis-express/35909> (acesso em 01/11/2019).

Bibliografia

- Sommerville, Ian (3 de abr. de 2015). *Software Engineering*. 10 edition. Boston: Pearson. 816 pp. ISBN: 978-0-13-394303-0.
- Sommerville, Ian e Pete Sawyer (1997). *Requirements Engineering: A Good Practice Guide*. 1st. USA: John Wiley & Sons, Inc. ISBN: 0471974447.
- Sowa, J.F. e J.A. Zachman (1992). “Extending and Formalizing the Framework for Information Systems Arquitecture”. *IBM Systems Journal* 31.3, p. 590.
- Stapenhurst, Tim (2009). *The Benchmarking Book*. Butterworth. ISBN: 0750677775.
- Strathern, Paul (2003). *Uma Breve História da Economia*. Zahar.
- The Standish Group (1994). *The Chaos Report 1994*. Report. The Standish Group.
- (2015). *Chaos Report 2015*. Report. Standish Group.
- Tricentis (2019). *Software FAILS Watch*. White Paper. Versão 5th Edition. Tricentis.
- Tukey, John W. (1958). “The Teaching of Concrete Mathematics”. *The American Mathematical Monthly* 65.1, pp. 1–9. ISSN: 0002-9890. DOI: 10.2307/2310294. URL: www.jstor.org/stable/2310294 (acesso em 20/12/2019).
- Vaughan-Nichols, Steven J. (7 de nov. de 2017). *MINIX: Intel's hidden in-chip operating system*. URL: <https://www.zdnet.com/article/minix-intels-hidden-in-chip-operating-system/> (acesso em 02/02/2020).
- Volare (2020). *Prioritisation Analysis*. URL: <https://www.volere.org/prioritisation-analysis/> (acesso em 24/02/2020).
- Von Halle, Barbara (2002). *Business Rules Applied: Building Better Systems Using the Business Rule Approach*. New York: John Wiley & Sons. ISBN: 9780471412939.
- Wake, Bill (17 de ago. de 2003). *INVEST in Good Stories, and SMART Tasks*. URL: <https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/> (acesso em 14/01/2020).
- (8 de fev. de 2012). *Independent Stories in the INVEST Model*. URL: <https://xp123.com/articles/independent-stories-in-the-invest-model/> (acesso em 14/01/2020).
- Wiegers, Karl (mai. de 1999). “Writing Quality Requirements”. *Software Development* 7.5, pp. 44–48. ISSN: 1070-8588.
- Wiegers, Karl E. e Joy Beatty (2013). *Software Requirements*. 3^a ed. Redmond, WA, USA: Microsoft Press.
- Woods-Wilson, Pauline (2003). *S930 Business Engineering with ERP Solutions: Lecture 2 Process Modeling for ERPs*. Disponível na Wayback Machine. London. URL: <https://web.archive.org/web/20040504111417/http://www.soi.city.ac.uk/~pauline/S930MScLecture3-implementation.ppt> (acesso em 01/12/2003).
- Wren, Alan (2003). *Project Management A-Z. A Compendium of Project Management Techniques and How to Use Them*. Routledge.
- Yourdon, Ed (2006). *Just Enough Structured Analysis*. Ed Yourdon.
- Yourdon, Edward (1994). *Decline and Fall of the American Programmer*. 1st. USA: Prentice Hall PTR. ISBN: 013191958X.
- Yourdon, Edward e Paul D. Becker (1997). *Death March: The Complete Software Developers Guide to Surviving Mission Impossible Projects*. USA: Prentice Hall PTR. ISBN: 0137483104.

- Zachman, J.A. (1987). "A Framework for Information Systems Architecture". *IBM Systems Journal* 26.3, p. 276.
- Zins, Chaim (fev. de 2007). "Conceptual Approaches for Defining Data, Information, and Knowledge: Research Articles". *J. Am. Soc. Inf. Sci. Technol.* 58.4, pp. 479–493. ISSN: 1532-2882.
- (2012). *Critical Delphi Research Methodology*. URL: <http://www.success.co.il/critical-delphi.html> (acesso em 31/12/2019).

DRAFT

DRAFT

A

Agradecimentos

Agradecimentos aos alunos, alunas e amigos que deram sugestões e encontraram erros.

Essa lista só foi iniciada em 18/3/2020.

- Letícia Freire Carvalho de Sousa.

DRAFT

DRAFT