

Manual do Engenheiro de Software

Módulo Técnico

*Wilson de Pádua Paula Filho
Janeiro de 2000*

Página em branco

Sumário

Sumário	3
Fundamentos técnicos	9
Engenharia de Software	11
1 Natureza	11
1.1 Engenharia de Software e Ciência da Computação	11
1.2 Sistemas de informática	12
2 Produtos	13
2.1 Problemas	13
2.2 Produção	14
2.3 Requisitos	15
2.4 Prazos e custos	18
2.5 Qualidade	19
Processos	25
1 Visão geral	25
1.1 Processos em geral	25
1.2 Processos de software	25
2 Exemplos de processos	29
2.1 O Processo Pessoal de Software	29
2.2 O Processo de Software para Times	31
2.3 O Processo Orientado a Objetos para Software Extensível	33
2.4 O Processo Unificado	34
3 Praxis	35
3.1 Visão geral	35
3.2 Detalhes das fases	43
3.3 Artefatos	54
3.4 Procedimentos de controle	57
Métodos técnicos	59
Requisitos	61
1 Princípios	61
1.1 Visão geral	61
1.2 A Especificação dos Requisitos do Software	62
1.3 Qualidade dos requisitos	64
2 Atividades	67
2.1 Visão geral	67
2.2 Detalhes das atividades	69
3 Técnicas	89
3.1 Introdução	89
3.2 Protótipos	89
3.3 Oficinas de requisitos	91
3.4 Relacionamento com os clientes	95
Análise	99
1 Princípios	99
1.1 Objetivos	99
1.2 Objetos e classes	99
1.3 Uso de notações alternativas	101
2 Atividades	102
2.1 Visão geral	102
2.2 Detalhes das atividades	103

3	Técnicas.....	127
3.1	Introdução	127
3.2	Oficinas de análise	127
3.3	Documentação do Modelo de Análise.....	129
Desenho	131
1	Princípios	131
1.1	Objetivos	131
1.2	O modelo de desenho	131
1.3	Desenho para a testabilidade	132
2	Atividades.....	133
2.1	Visão geral	133
2.2	Detalhes das atividades	134
3	Técnicas.....	150
3.1	Visão geral	150
3.2	Desenho de interfaces de usuário.....	150
3.3	Desenho para a reutilização.....	153
3.4	Interfaces com bancos de dados relacionais	155
Testes	161
1	Princípios	161
1.1	Visão geral	161
1.2	Objetivos	161
1.3	Métodos.....	161
1.4	Baterias de testes.....	162
2	Atividades.....	163
2.1	Visão geral	163
2.2	Detalhes das atividades	166
3	Técnicas.....	174
3.1	Visão geral	174
3.2	Testes de integração.....	174
3.3	Testes de aceitação.....	177
3.4	Testes de regressão	180
Implementação	181
1	Princípios	181
1.1	Objetivos	181
1.2	Artefatos.....	181
2	Atividades.....	184
2.1	Visão geral	184
2.2	Detalhes das atividades	185
3	Técnicas.....	194
3.1	Introdução	194
3.2	Modularização	195
3.3	Documentação de usuário	200
Padrões técnicos	205
Proposta de Especificação de Software	207
1	Introdução.....	207
2	Preenchimento da Proposta de Especificação de Software	207
2.1	Missão do produto (PESw-1).....	207
2.2	Lista de funções (PESw-2)	207
2.3	Requisitos de qualidade (PESw-3).....	208
2.4	Metas gerenciais (PESw-4)	209
2.5	Outros aspectos (PESw-5).....	209
2.6	Estimativa de custos e prazos para a especificação (PESw-6)	209
Especificação de Requisitos de Software	211
1	Visão geral.....	211
1.1	Introdução	211
1.2	Referências	211
1.3	Convenções de preenchimento	211

1.4	Organização dos requisitos específicos.....	212
2	Preenchimento da Especificação dos Requisitos do Software	213
2.1	Introdução (ERSw-1).....	213
2.2	Descrição geral do produto (ERSw-2)	216
2.3	Requisitos específicos (ERSw-3)	223
2.4	Informação de suporte (ERSw-4)	230
3	Revisão da Especificação dos Requisitos do Software.....	230
3.1	Introdução.....	230
3.2	Revisão da seção <u>Página de Título</u>	231
3.3	Revisão da seção <u>Sumário</u>	231
3.4	Revisão da seção <u>Lista de Ilustrações</u>	231
3.5	Revisão do corpo da Especificação dos Requisitos do Software	231
3.6	Revisão do Modelo de Análise.....	236
	Revisões de Software	239
1	Visão geral.....	239
1.1	Objetivos.....	239
1.2	Alternativas.....	239
2	Reuniões de revisão.....	240
2.1	Participantes.....	240
2.2	Preparação.....	240
2.3	Condução	240
3	Perfil da equipe de revisão.....	241
3.1	Introdução	241
3.2	Perfil do líder	241
3.3	Perfil do relator	242
3.4	Perfil dos revisores em geral	243
4	Resultados da revisão	244
4.1	Relatórios da revisão	244
4.2	Classificação dos defeitos	246
4.3	Listas de tópicos	247
	Desenho de Interfaces de Usuário de Software.....	249
1	Diretrizes	249
1.1	Visão geral	249
1.2	Modelo mental	249
1.3	Consistência e simplicidade	249
1.4	Questões de memorização.....	250
1.5	Questões cognitivas	250
1.6	Realimentação.....	251
1.7	Mensagens do sistema	251
1.8	Modalidade	252
1.9	Reversibilidade.....	253
1.10	Atração da atenção	253
1.11	Exibição.....	254
1.12	Diferenças individuais	254
2	Estilos de interação	255
2.1	Janelas.....	255
2.2	Cardápios	257
2.3	Formulários.....	263
2.4	Caixas	265
2.5	Linguagens de comando	268
2.6	Interfaces pictóricas.....	269
2.7	Outros estilos de interação	270
	Descrição de Desenho de Software	271
1	Visão geral.....	271
1.1	Introdução	271
1.2	Referências.....	271
1.3	Convenções de preenchimento	272
2	Preenchimento da Descrição do Desenho do Software	272
2.1	Introdução (DDSw-1)	272

2.2	Desenho das interfaces de usuário (DDSw-2)	275
2.3	Desenho interno (DDSw-3)	286
2.4	Desenho das liberações (DDSw-4)	292
2.5	Informação de suporte (DDSw-5)	293
3	Revisão da Descrição do Desenho do Software	297
3.1	Introdução	297
3.2	Revisão da seção <u>Página de Título</u>	297
3.3	Revisão da seção <u>Sumário</u>	298
3.4	Revisão da seção <u>Lista de Ilustrações</u>	298
3.5	Revisão do corpo da Descrição do Desenho do Software	298
Documentação de Testes de Software		307
1	Visão geral	307
2	Descrição dos Testes do Software	307
2.1	Introdução	307
2.2	Referências	307
2.3	Estrutura	307
2.4	Preenchimento da Descrição dos Testes do Software	308
2.5	Revisão da Descrição dos Testes do Software	317
3	Relatórios dos Testes do Software	321
3.1	Introdução	321
3.2	Registros de testes	322
3.3	Relatórios de incidentes dos testes	324
3.4	Relatórios de resumo de testes	324
Desenho Detalhado e Codificação de Software		327
1	Visão geral	327
1.1	Introdução	327
2	Diretrizes genéricas	327
2.1	Aspectos abordados	327
2.2	Dados	327
2.3	Estruturas de controle	333
2.4	Expressões	338
2.5	Leiaute de programa	338
2.6	Comentários	342
3	Diretrizes para C++	345
3.1	Diretrizes aplicáveis a C	345
3.2	Transição de C a C++	347
3.3	Gestão de memória	349
3.4	Desenho e declaração de classes e funções	351
3.5	Herança	354
3.6	Compilação	359
3.7	Tópicos avançados	359
4	Revisões da Implementação	360
4.1	Lista de conferência para o desenho detalhado	360
4.2	Lista de conferência para código	362
Documentação para Usuários de Software		369
1	Visão geral	369
1.1	Introdução	369
1.2	Referências	369
1.3	Padronização de formato	369
1.4	Requisitos de apresentação	369
2	Preenchimento do Manual do Usuário do Software	370
2.1	Página de título	370
2.2	Garantias e obrigações contratuais	370
2.3	Sumário	370
2.4	Índice de ilustrações	371
2.5	Introdução	371
2.6	Corpo do documento	371
2.7	Mensagens de erro, problemas conhecidos e recuperação de erros	373
2.8	Apêndices	373

2.9	Bibliografia	373
2.10	Glossário	373
2.11	Índice remissivo	373
A notação UML		375
1	Introdução.....	375
2	Modelagem funcional.....	375
2.1	Atores	375
2.2	Casos de uso.....	376
2.3	Diagramas de casos de uso.....	376
2.4	Fluxos dos casos de uso	380
2.5	Diagramas de estados	381
2.6	Diagramas de atividade.....	383
3	Modelagem lógica.....	386
3.1	Objetos e classes.....	386
3.2	Organização das classes.....	388
3.3	Relacionamentos	390
3.4	Detalhes das classes.....	394
3.5	Diagramas de interação.....	397
4	Modelagem física.....	404
4.1	Visão de componentes	404
4.2	Visão de desdobramento	406
Glossário		409
Bibliografia.....		419

Página em branco

Fundamentos técnicos

Página em branco

Engenharia de Software

1 Natureza

1.1 Engenharia de Software e Ciência da Computação

O que é Engenharia de Software? É uma das disciplinas da Informática, ou da Ciência da Computação? É um sinônimo de um destes termos? Ou, falando de modo mais prático: um profissional formado em Informática ou Ciência da Computação é automaticamente um engenheiro de software?

O Dicionário Aurélio Eletrônico V.2.0 assim define:

Informática	Ciência que visa ao tratamento da informação através do uso de equipamentos e procedimentos da área de processamento de dados.
Ciência	Conjunto organizado de conhecimentos relativos a um determinado objeto, especialmente os obtidos mediante a observação, a experiência dos fatos e um método próprio.
Processamento de dados	Tratamento dos dados por meio de máquinas, com o fim de obter resultados da informação representada pelos dados.
Engenharia	Arte de aplicar conhecimentos científicos e empíricos e certas habilitações específicas à criação de estruturas, dispositivos e processos que se utilizam para converter recursos naturais em formas adequadas ao atendimento das necessidades humanas.

Tabela 1 – Informática, ciência e engenharia

Nos verbetes acima, fica a Informática definida como uma ciência, cujo assunto é o processamento de informação através de máquinas. A ciência, por sua vez, tem como foco a acumulação do conhecimento através do método científico, geralmente baseado em experimentos e observações.

A definição de Engenharia é conexa, porém distinta. Analisemos cada uma de suas partes, tentando interpretá-las em termos da Engenharia de Software e reordenando-as para fins explicativos.

- **Arte** – Na acepção aqui usada, a “capacidade que tem o homem de pôr em prática uma idéia, valendo-se da faculdade de dominar a matéria”, ou “a utilização de tal capacidade, com vistas a um resultado que pode ser obtido por meios diferentes”. O produto da engenharia é matéria dominada: idéia que se torna material através do emprego das faculdades humanas. Na Engenharia de Software, a matéria dominada consiste em máquinas de processamento da informação configuradas e programadas.
- **Atendimento das necessidades humanas** – O foco da engenharia é a necessidade humana. Nisto, ela tem escopo bem diverso da ciência. O conhecimento é certamente uma necessidade humana, mas uma entre várias outras de uma hierarquia¹: alimentação, moradia, segurança, afeição, auto-estima... Todo produto de engenharia se justifica através da satisfação de uma dessas necessidades; portanto, da geração de algo que tenha valor para alguém. A Engenharia de Software procura gerar valor através dos recursos de processamento de informação.

¹ Os especialistas em desenvolvimento humano usam a escala de necessidades de Maslow [Hitt85].

- **Conhecimentos científicos** – Parte dos métodos da engenharia provém da ciência; parte dos métodos da Engenharia de Software provém da Ciência da Computação.
- **Conhecimentos empíricos** – Outra parte dos métodos da engenharia provém da experiência prática, e não apenas da pesquisa científica. Em muitos casos, a ciência intervém posteriormente para explicar, modelar e generalizar o que a prática descobriu. Na Engenharia de Software, muitas práticas são adotadas porque funcionam, mesmo quando ainda carecem de fundamentação teórica satisfatória.
- **Habilitações específicas** – Toda engenharia é uma atividade realizada por pessoas. Para isto, estas pessoas têm de ter habilitações específicas. A Engenharia de Software possui um conjunto de habilitações específicas, ou disciplinas, que se relaciona com o conjunto das disciplinas da Ciência da Computação, mas não se confunde com elas.
- **Recursos naturais** – Toda engenharia parte de recursos naturais; algumas ciências, por contraste, como a Lógica e a Matemática, têm base inteiramente abstrata. Os recursos naturais da Engenharia de Software são as máquinas de tratamento da informação. A Ciência da Computação se ocupa de abstrações como os algoritmos e as estruturas de dados; a Engenharia de Software usa estas abstrações, desde que sejam realizáveis na prática, através da tecnologia existente em determinado momento.
- **Formas adequadas** – Para satisfazer as necessidades humanas, os recursos naturais devem ser convertidos em formas adequadas. Na Engenharia de Software, estas formas são os programas de computador. Comparado com outros engenheiros, o engenheiro de software tem liberdade extrema na criação de formas. Entretanto, só uma ínfima fração das formas possíveis atende ao critério de utilidade.
- **Dispositivos e estruturas** – O engenheiro reúne dispositivos em estruturas capazes de satisfazer uma necessidade humana. A criação de estruturas é essencial para que se extraia uma função útil do conjunto de dispositivos. O desafio do engenheiro de software é escolher e montar as estruturas de grande complexidade que a programação dos computadores permite realizar.
- **Processos** – A engenharia segue processos, que são “maneiras pelas quais se realiza uma operação, segundo determinadas normas”. O método da engenharia se baseia na ação sistemática, e não na improvisação. A noção de processo será também a espinha dorsal deste livro.

Em suma, a Engenharia de Software não se confunde com a Ciência da Computação, e nem é uma disciplina desta, tal como a Engenharia Metalúrgica não é uma disciplina da Física dos Metais, nem a Engenharia Elétrica é uma disciplina da Física da Eletricidade. Como toda engenharia, a Engenharia de Software usa resultados da ciência, e fornece problemas para estudo desta; mas são vocações profissionais completamente distintas, tão distintas quanto as vocações do engenheiro e do físico, do médico e do biólogo, do político e do cientista político.

1.2 Sistemas de informática

As máquinas de tratamento de informação são organizadas em estruturas úteis, formando os sistemas de informática. Várias definições de sistema são aqui pertinentes.

1. Conjunto de elementos, materiais ou ideais, entre os quais se possa encontrar ou definir alguma relação.
2. Disposição das partes ou dos elementos de um todo, coordenados entre si, e que funcionam como estrutura organizada.

3. Reunião de elementos naturais da mesma espécie que constituem um conjunto intimamente relacionado.

O software é a parte programável de um sistema de informática. Ele é um elemento central: realiza estruturas complexas e flexíveis que trazem funções, utilidade e valor ao sistema. Mas outros componentes são indispensáveis: as plataformas de hardware, os recursos de comunicação de informação, os documentos de diversas naturezas, as bases de dados e até os procedimentos manuais que se integram aos automatizados.

Este livro trata apenas dos componentes de software, por limitação de escopo. O engenheiro de software deverá ter em mente, no entanto, que o valor de um sistema depende da qualidade de cada um de seus componentes. Um sistema pode ter excelentes algoritmos codificados em seu software e ser de péssimo desempenho por defeito de desenho de seu hardware, rede ou banco de dados. Cada um destes elementos pode pôr a perder a confiabilidade e a usabilidade do sistema.

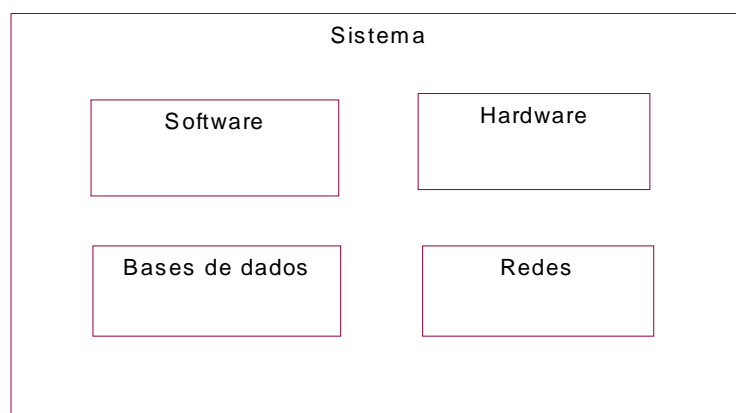


Figura 1 - Sistema de informática e suas partes

Na prática, o engenheiro de software será chamado com frequência a resolver questões pertinentes aos outros componentes do sistema, ou, no mínimo, encontrar quem as resolva. Alguma proficiência nas respectivas disciplinas lhe será necessária. Não trataremos delas neste livro, a não ser tangencialmente, quando necessário.

2 Produtos

2.1 Problemas

Muitas pessoas, inclusive dirigentes de empresa, percebem o computador como problema, e não como solução. Muitos aceitam como fato da vida que os sistemas de informática:

- não façam o que deveriam fazer;
- sejam caros;
- sejam entregues tarde demais;
- sejam de baixa qualidade:
 - sejam cheios de defeitos;
 - sejam difíceis de usar;

- sejam lentos etc.

A tecnologia só resolve problemas quando é usada por pessoas qualificadas, dentro de processos adequados. Os sistemas de informática são os produtos da tecnologia de tratamento da informação. Os problemas que ocorrem com sistemas de informática podem ter várias causas:

- Podem ser fruto de deficiência de qualificação das pessoas que os operam. Isto pode decorrer de falta de treinamento, dificuldade de uso do próprio sistema, ou muitos outros fatores relacionados com pessoas.
- Podem originar-se de processos de negócio inadequados. Por **processo de negócio** entendemos o processo que faz parte da área de aplicação, em que, tipicamente, alguns procedimentos são executados por pessoas e outros são automatizados através do computador. Por exemplo, sacar dinheiro de um banco pode ser feito por dois processos diferentes: diretamente no caixa, ou através do equipamento conhecido como caixa eletrônico. O segundo processo é mais automatizado que o primeiro.
- Podem também ser causados por deficiências de tecnologia, ou seja, do próprio sistema de informática. Neste livro, trataremos apenas dessa classe de problemas.

2.2 Produção

2.2.1 Ciclos de vida

A Engenharia de Software se preocupa com o software como produto. Estão fora de seu escopo programas que são feitos unicamente para diversão do programador. Estão fora de seu escopo também pequenos programas descartáveis, feitos por alguém exclusivamente como meio para resolver um problema, e que não serão utilizados por outros.

Chamaremos de **cliente** a uma pessoa física ou jurídica que contrata a execução de um projeto, ou a um seu representante autorizado, com poder de aceitação de propostas e produtos. A pessoa que efetivamente usará um produto será chamada de **usuário**. Um usuário pode ser o próprio cliente, um funcionário de uma organização cliente, ou mesmo não ser relacionado diretamente com o cliente. Por exemplo, quando se produz software de prateleira, que será vendido no mercado aberto, é útil considerar como cliente, por exemplo, o departamento de marketing da organização produtora.

Como todo produto industrial, o software tem um ciclo de vida:

- ele é concebido a partir da percepção de uma necessidade;
- desenvolvido, transformando-se em um conjunto de itens entregue a um cliente;
- entra em operação, sendo usado dentro de um algum processo de negócio, e sujeito a atividades de manutenção, quando necessário;
- é retirado de operação, ao final de sua vida útil.

Cada fase do ciclo de vida tem divisões e subdivisões, que serão exploradas ao longo deste livro. É interessante observar, na Tabela 1, que a Codificação, que representa a escrita final de um programa em forma inteligível para um computador, é apenas uma pequena parte do ciclo de vida. Para a maioria das pessoas, inclusive muitos profissionais da informática, esta parece ser a única tarefa de um programador, ou seja, um produtor de software.

Ciclo de vida	Percepção da necessidade			
	Desenvolvimento	Concepção		
		Elaboração		
		Construção	Desenho inicial	
			Liberação	Desenho detalhado
				Codificação
				Testes de unidade
		Testes alfa		
	Transição			
	Operação			
Retirada				

Tabela 2 - Esquema simplificado do ciclo de vida do software

2.2.2 Projetos

Normalmente, o desenvolvimento de software é feito dentro de um **projeto**. Todo projeto tem uma data de início, uma data de fim, uma equipe (da qual faz parte um responsável, a que chamaremos de **gerente do projeto**) e outros recursos. Um projeto representa a execução de um **processo**.

Quando um processo é bem definido, ele terá subdivisões que permitam avaliar o progresso de um projeto e corrigir seus rumos quando ocorrerem problemas. Essas subdivisões são chamadas de fases, atividades ou iterações; posteriormente, usaremos estas palavras com significados técnicos específicos.

As subdivisões devem ser terminadas por **marcos**, isto é, pontos que representam estados significativos do projeto. Geralmente os marcos são associados a **resultados** concretos: documentos, modelos ou porções do produto, que podem fazer parte do conjunto prometido aos clientes, ou ter apenas utilização interna ao projeto. O próprio produto é um resultado associado ao marco de conclusão do projeto.

2.3 Requisitos

2.3.1 Características

O valor de um produto vem de suas **características**. Tratando-se de software, costuma-se dividir as características em:

- características **funcionais**, que representam os comportamentos que um programa ou sistema deve apresentar diante de certas ações de seus usuários;
- características **não funcionais**, que quantificam determinados aspectos do comportamento.

Por exemplo, em um terminal de caixa automático, os tipos de transações bancárias suportadas são características funcionais. A facilidade de uso, o tempo de resposta e o tempo médio entre falhas são características não-funcionais.

Os **requisitos** são as características que definem os critérios de aceitação de um produto. A engenharia tem por objetivo colocar nos produtos as características que são requisitos. Outras características podem aparecer acidentalmente, mas os produtos não devem ser desenhados para incluí-las, já que, normalmente, toda característica extra significa um custo adicional de desenho ou de fabricação.

2.3.2 *Especificação dos requisitos*

Os requisitos podem ser dos seguintes tipos:

- Os requisitos **explícitos** são aqueles descritos em um documento que arrola os requisitos de um produto, ou seja, um documento de **especificação de requisitos**.
- Os requisitos **normativos** são aqueles que decorrem de leis, regulamentos, padrões e outros tipos de normas a que o tipo de produto deve obedecer.
- Os requisitos **implícitos** são expectativas dos clientes e usuários, que são cobradas por estes, embora não documentadas.

Requisitos implícitos são indesejáveis porque, não sendo documentados, provavelmente não serão considerados no desenho do produto. O resultado será um produto que, embora satisfazendo os compromissos formais, que são os requisitos explícitos e normativos, não atenderá às necessidades do consumidor.

Mesmo requisitos documentados podem apresentar problemas. Uma especificação de requisitos pode conter requisitos incompletos, inconsistentes ou ambíguos. Alguns desses problemas decorrem da natureza da própria linguagem natural, que normalmente é usada para expressá-los. Outros decorrem de técnicas deficientes de elaboração dos requisitos.

2.3.3 *Engenharia dos requisitos*

Um dos problemas básicos da engenharia de software é o levantamento e documentação dos requisitos dos produtos de software. Quando este levantamento é bem feito, os requisitos implícitos são minimizados. Quando a documentação é bem feita, os requisitos documentados têm maiores chances de serem corretamente entendidos pelos desenvolvedores. Algumas técnicas de análise dos requisitos ajudam a produzir especificações mais precisas e inteligíveis. O conjunto das técnicas de levantamento, documentação e análise forma a **engenharia dos requisitos**, que é uma das disciplinas da Engenharia de Software.

Infelizmente, muitos clientes não entendem a necessidade de especificações de requisitos. Pior ainda, muitos desenvolvedores de software e, pior de tudo, muitos gerentes também não. É uma situação tão absurda quanto querer resolver um problema sem escrever o respectivo enunciado: existe grande risco de resolver-se o problema errado. Por outro lado, é possível também a existência de requisitos que não correspondam a necessidades reais dos clientes e usuários. Essa falha de engenharia de requisitos indica que não foi feita uma análise do valor de cada requisito, do ponto de vista da missão que o produto deve cumprir.

Cabe aos engenheiros de software insistirem sempre na elaboração de uma boa especificação de requisitos. Faz parte do seu trabalho convencer clientes e usuários de que:

- boas especificações de requisitos são indispensáveis;
- elas não representam custos supérfluos, mas investimentos necessários, que se pagam com altos juros;
- a participação dos usuários na engenharia de requisitos é fundamental para que as suas necessidades sejam corretamente atendidas pelo produto;
- uma boa especificação de requisitos custa tempo e dinheiro;
- a ausência de uma boa especificação de requisitos custa muito mais tempo e dinheiro.

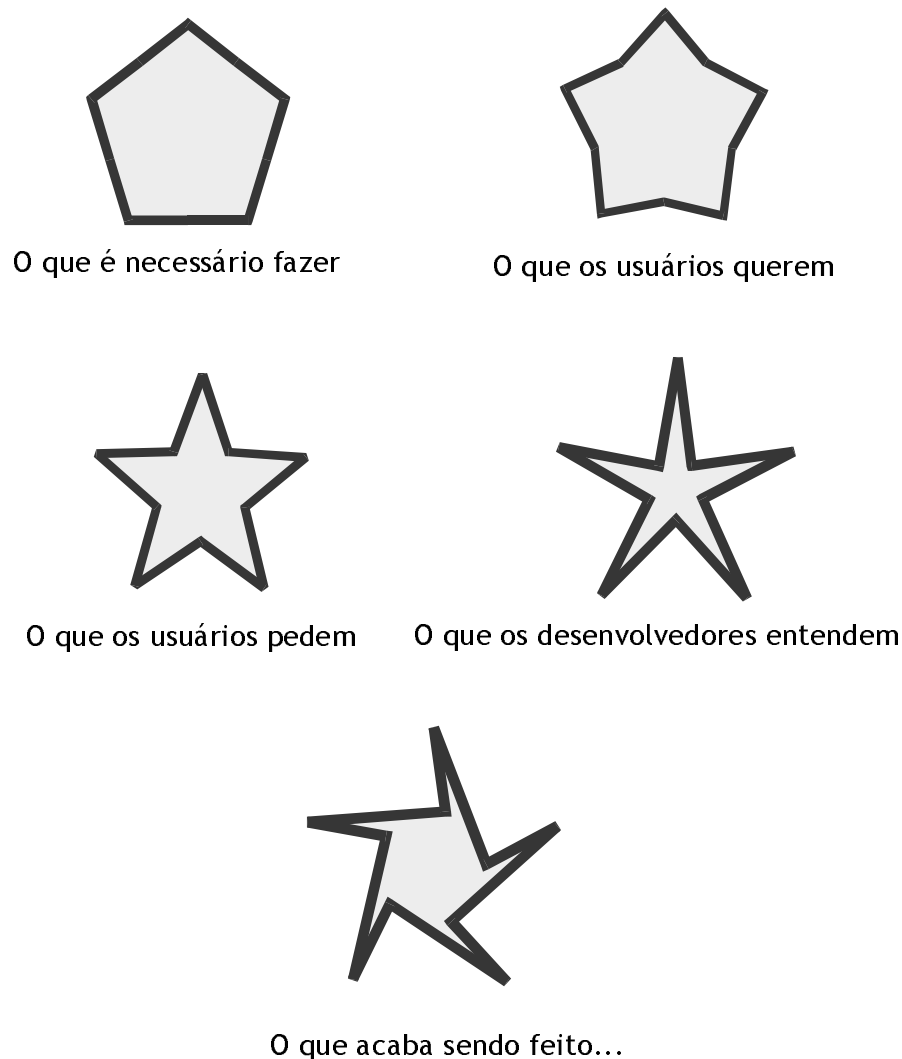


Figura 2 - Como os requisitos evoluem

2.3.4 Gestão dos requisitos

Um problema comum no desenvolvimento de software é a **instabilidade dos requisitos**, que ocorre quando clientes e usuários trazem novos requisitos, ou alterações de requisitos, quando o desenvolvimento já está em fase adiantada. A instabilidade dos requisitos costuma ter um custo muito alto; geralmente significa perder trabalho já feito, desfazer algumas coisas e remendar outras. Na Engenharia de Software, a instabilidade dos requisitos é tão danosa quanto nas outras engenharias. Quando se muda a planta de um edifício durante a construção, geralmente é preciso desfazer parte do que já foi construído, e o remendo raramente é satisfatório.

A boa engenharia de requisitos reduz a instabilidade destes, ajudando a obter os requisitos corretos em um estágio anterior ao desenvolvimento. Entretanto, alterações dos requisitos são às vezes inevitáveis. A engenharia de requisitos é sujeita a limitações humanas, e, mesmo que o levantamento seja perfeito, podem ocorrer alterações de requisitos por causas externas aos projetos. Por exemplo, a legislação pode mudar no meio do projeto, requerendo alterações nos relatórios que o produto deve emitir. A **gestão dos requisitos** é a disciplina da engenharia de software que procura manter sob controle o conjunto dos requisitos de um produto, mesmo diante de algumas alterações inevitáveis.

2.4 Prazos e custos

2.4.1 *Realismo de prazos e custos*

Por que tantos sistemas informatizados são entregues com atraso e custam mais do que o previsto? Estourar cronogramas e orçamentos é parte da rotina da maioria dos profissionais de software. Clientes e gerentes se desesperam com os atrasos dos projetos de software, e às vezes sofrem enormes prejuízos com eles. Entretanto, no contrato seguinte, eles provavelmente escolherão o ofertante que prometer menor prazo e/ou menor custo. Se for um projeto interno da organização, farão todo tipo de pressão para conseguir que os desenvolvedores prometam prazos politicamente agradáveis, embora irreais.

Estimar prazos e custos faz parte da rotina de qualquer ramo da engenharia. Para um produto ser viável, não basta que atenda aos requisitos desejados; tem de ser produzido dentro de certos parâmetros de prazo e custo. Se isso não for possível, o produto pode não ser viável do ponto de vista de mercado, ou pode ser preferível adquirir outro produto, ainda que sacrificando alguns dos requisitos. Ter estimativas de prazos e custos, portanto, é uma expectativa mais que razoável de clientes e gerentes.

O problema é que existem alguns desenvolvedores pouco escrupulosos. E existem muitos que, mesmo sendo honestos, não conhecem métodos técnicos de estimativa de prazos e custos do desenvolvimento de software. E existem ainda os que, mesmo sabendo fazer boas estimativas, trabalham em organizações onde não existe clima para que os desenvolvedores possam apresentar avaliações francas das perspectivas dos projetos. Nestas organizações, existe a política de "matar os mensageiros de más notícias". Esta política foi usada por muitos reis da antiguidade, com resultados geralmente desastrosos.

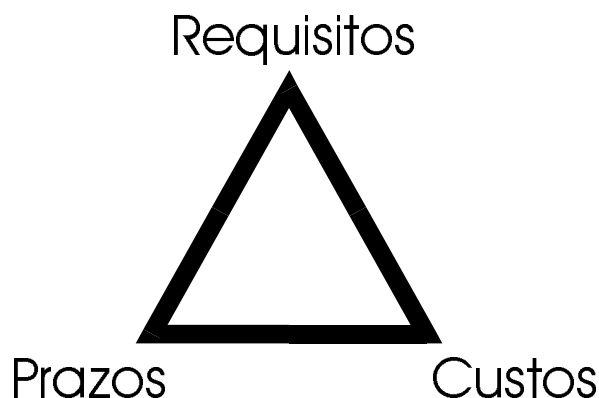


Figura 3 - Um triângulo crítico da Engenharia de Software

Requisitos, prazos e custos formam os vértices de um triângulo crítico. Aumentos de requisitos levam a aumentos de prazos ou de custos, ou de ambos. Reduções de requisitos podem levar a reduções de prazos ou de custos (mas nem sempre).

2.4.2 *Planejamento de projetos*

Uma coisa é exigir dos engenheiros de software estimativas de prazos e cobrar o cumprimento dos prazos prometidos. Clientes e gerentes podem e devem fazê-lo. Outra coisa é pressioná-los para que façam promessas que não podem ser cumpridas. Uma frase comum dessa cultura é: "Não me interessa como você vai fazer, desde que entregue no prazo!". Na realidade, o cliente ou gerente deve, no seu próprio interesse, ter algum meio de checar se o cronograma e o orçamento propostos são realistas; e, se preciso, recorrer aos serviços de uma terceira parte para fazer esta verificação.

A cultura do prazo político é ruim para todos. Para os desenvolvedores, ela significa estresse e má qualidade de vida. Para os gerentes, perda de credibilidade e prejuízos. E, para os clientes, produtos de má qualidade e mais caros do que deveriam. Ainda por cima, entregues fora do prazo.

Para cumprir compromissos de prazos e custos, esses compromissos têm de ser assumidos com base em requisitos bem levantados, analisados e documentados. E os planos dos projetos têm de ser feitos com boas técnicas de estimativa e análise de tamanho, esforços, prazos e riscos. Estas técnicas pertencem à disciplina de **planejamento de projetos**, que faz parte da Engenharia de Software.

2.4.3 *Controle de projetos*

Todo plano comporta incertezas. Por exemplo, o tamanho de certas partes do produto pode ser estimado grosseiramente a partir dos requisitos, mas o desenho detalhado das partes do produto permite refinar as estimativas, e o tamanho correto só é exatamente conhecido no final dos projetos. A produtividade dos desenvolvedores pode ser estimada com base em projetos anteriores da organização, mas é afetada por muitas variações, que dependem de pessoas, processos e tecnologia. E riscos previstos e imprevistos podem se materializar.

Ao longo de um projeto, os gerentes têm de enfrentar problemas e tentar controlar variáveis que afetem o cumprimento de seus compromissos. Algumas vezes, os problemas podem ser resolvidos por meio de contratação e remanejamento de pessoal, ou de uma melhoria de ferramentas. Outras vezes não existe maneira viável de contornar os problemas, e é necessário renegociar requisitos, prazos ou custos. Para renegociar, é preciso replanejar, atualizando as estimativas para levar em conta os novos dados.

A disciplina complementar do planejamento de projetos é o **controle dos projetos**, que compreende:

- o acompanhamento do progresso dos projetos, comparando-se o planejado com o realizado;
- a busca de alternativas para contornar problemas surgidos na execução dos projetos;
- o replanejamento dos projetos, quando não é possível manter os planos anteriores dentro de um grau razoável de variação;
- a renegociação dos compromissos assumidos, envolvendo todas as partes interessadas.

2.5 **Qualidade**

2.5.1 *Conformidade com requisitos*

Entenderemos como **qualidade** de um produto o seu grau de conformidade com os respectivos requisitos. De acordo com essa definição de qualidade, por exemplo, um carro popular pode ser de boa qualidade, e um carro de luxo pode ser de má qualidade. O que decide a qualidade é comparação com os respectivos requisitos: o confronto entre a promessa e a realização de cada produto.

Geralmente a qualidade de um produto decorre diretamente da qualidade do processo utilizado na produção dele. Note-se que importa aqui a qualidade do processo efetivamente utilizado, não do "processo oficial", que pode eventualmente estar descrito nos manuais da organização. Muitas vezes os processos oficiais não são seguidos na prática, por deficiência de ferramentas, por falta de qualificação das pessoas, ou porque pressões de prazo levam os gerentes dos projetos a eliminar etapas relacionadas com controle da qualidade.

Em um produto de software de má qualidade, muitos requisitos não são atendidos completamente. As deficiências de conformidade com os requisitos se manifestam de várias maneiras. Em alguns casos, certas funções não são executadas corretamente sob certas condições, ou para certos valores de entradas. Em outros casos, o produto tem desempenho insuficiente, ou é difícil de usar.

Cada requisito não atendido é um **defeito**. No mundo da informática, criou-se a usança de chamar de "bugs" os defeitos de software. Assim, erros técnicos adquirem conotação menos negativa, que lembra simpáticos insetos de desenho animado. E o nome ajuda a esquecer que esses defeitos foram causados por erro de uma pessoa falível, e que cada defeito tem responsáveis bem precisos.

Note-se que defeitos incluem situações de falta de conformidade com requisitos explícitos, normativos e implícitos. Os defeitos associados a requisitos implícitos são os mais difíceis de tratar. Eles levam a desentendimentos sem solução entre o fornecedor e o cliente do produto. Além disso, como esses requisitos, por definição, não são documentados, é bastante provável que eles não tenham sido levados em conta no desenho do produto, o que tornará a correção dos defeitos particularmente trabalhosa.

2.5.2 *Garantia da qualidade*

Um erro conceitual comum é imaginar que é possível trocar prazo, e talvez custo, por qualidade. Na realidade, é possível, em muitos casos, reduzir prazos e custos através da redução dos requisitos de um produto. A qualidade, por outro lado, é consequência dos processos, das pessoas e da tecnologia. A relação entre a qualidade do produto e cada um desses fatores é complexa. Por isto, é muito mais difícil controlar o grau de qualidade do produto do que controlar os requisitos.

Em todas as fases do desenvolvimento de software, as pessoas introduzem defeitos. Eles decorrem de limitações humanas: erros lógicos, erros de interpretação, desconhecimento de técnicas, falta de atenção, ou falta de motivação. Em todo bom processo, existem atividades de garantia da qualidade, tais como revisões, testes e auditorias. Essas atividades removem parte dos defeitos introduzidos. Quando atividades de controle da qualidade são cortadas, parte dos defeitos deixa de ser removida em um ponto do projeto.

Defeitos que não são removidos precocemente acabam sendo detectados depois. Quanto mais tarde um defeito é corrigido, mais cara é a sua correção, por várias razões que serão discutidas posteriormente. O pior caso acontece quando o defeito chega ao produto final. Neste caso, ele só será removido através de uma operação de **manutenção**. Esta é a forma mais cara de remoção de defeitos. Em certos casos, como acontece em sistemas de missão crítica, defeitos de software podem trazer prejuízos irreparáveis.

A Figura 4 mostra que o tempo de desenvolvimento é geralmente reduzido com o aumento da qualidade do processo. Isso acontece porque um processo melhor é mais eficiente na detecção e eliminação precoce dos defeitos. Em geral, o tempo gasto com a correção precoce é mais do que compensado pela eliminação do tempo que seria gasto com a correção tardia. O prazo aumenta apenas quando se quer reduzir o nível de defeitos do produto final a um parâmetro mais rigoroso em relação ao estado-da-arte. Em certos casos, isso se justifica pelo caráter crítico do sistema: por exemplo, quando defeitos podem colocar pessoas em perigo, ou causar prejuízos materiais vultosos.

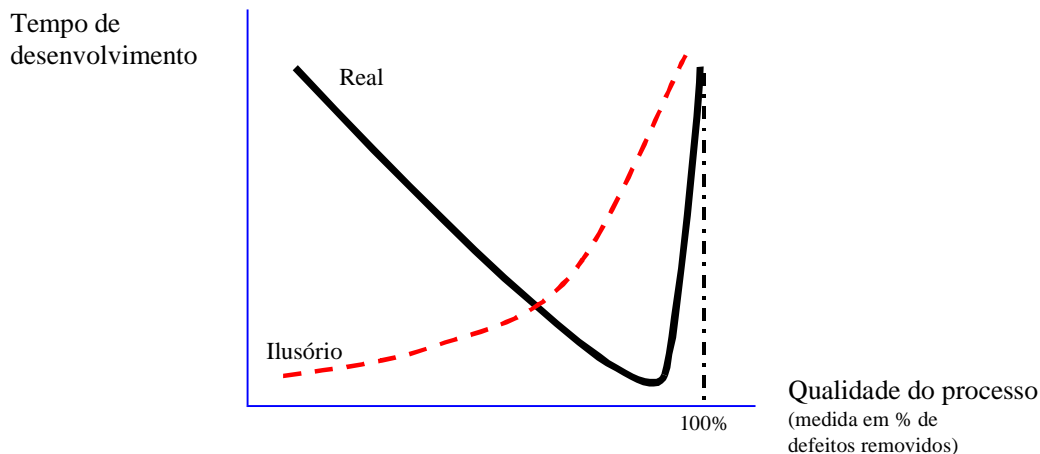


Figura 4 - A curva prazo \times qualidade

Vários métodos de garantia da qualidade levam em conta uma limitação humana: somos mais eficazes para achar os defeitos dos outros do que os nossos próprios defeitos. Por isso, os tipos mais eficazes de revisão são feitos por revisores distintos dos autores. Testes de aceitação de um produto devem ser desenhados e realizados de preferência por testadores independentes. E o controle da qualidade, como

um todo, funciona melhor quando é coordenado por um grupo da organização que é independente dos projetos controlados e que tem acesso direto à alta gerência da organização.

2.5.3 *Gestão de configurações*

Um produto de software é composto por muitos **artefatos**: códigos executáveis, códigos fontes, modelos, relatórios e outros documentos. Alguns desses artefatos são resultados oficiais do projeto; a aprovação dos resultados assinala que um marco do projeto foi cumprido. Outros artefatos têm caráter mais informal; por exemplo, documentos e modelos temporários de trabalho dos desenvolvedores.

A maioria dos artefatos evolui ao longo de um projeto, e até ao longo de toda a vida de um produto. Mesmo depois de terminado um projeto, é importante que os resultados sejam guardados e controlados. Pode ser necessário atualizá-los em caso de manutenção. Documentos e modelos consistentes são indispensáveis para facilitar a manutenção e evitar que esta introduza novos defeitos. A guarda e a atualização de documentos e modelos são rotineiras em todos os ramos da engenharia, e a Engenharia de Software não deveria ser exceção.

Mesmo um pequeno projeto pode gerar mais de uma dezena de resultados diferentes, cada um com mais de uma dezena de versões. Organizar e controlar a proliferação dos artefatos é o objetivo da disciplina de **gestão de configurações**. Sem gestão de configurações é impossível atingir sequer níveis razoáveis de qualidade. Versões corrigidas de artefatos serão perdidas, e versões defeituosas acabarão reaparecendo. Com efeito, o número de itens diferentes produzidos em projetos de software, mesmo pequenos, ultrapassa facilmente os limites da memória e da atenção humanas.

2.5.4 *Gestão de contratos*

Muitas organizações atuais procuram, justificadamente ou não, reduzir sua força de trabalho permanente. Muitas organizações para as quais a produção de software não é uma atividade fim têm preferido contratar externamente o desenvolvimento dos sistemas de que necessitam. E muitos profissionais de informática preferem trabalhar como empresários ou profissionais liberais do que como assalariados. Por causa dessas forças, muitas organizações optam por encomendar a produtores externos o desenvolvimento de seus sistemas informatizados, ou de parte deles.

Terceirizar o desenvolvimento de software é uma boa solução, em muitos casos. Mas não é panacéia. O esforço de uma organização para melhorar a qualidade de seus sistemas informatizados pode ser perdido por causa de falhas dos contratados. Para que isso não aconteça, a organização contratante deve estar capacitada em **gestão de contratos**. Para isso, ela tem de ser capaz, no mínimo, de:

- especificar correta e completamente o produto a ser desenvolvido;
- fazer uma boa seleção entre os candidatos a subcontratado, avaliando o grau de realismo das propostas destes;
- acompanhar o desempenho do subcontratado sem interferir no trabalho destes, mas detectando precocemente sintomas de problemas;
- planejar e executar os procedimentos de aceitação do produto.

2.5.5 *Desenho*

Os defeitos mais grosseiros de um produto de software são os defeitos de requisitos. Felizmente, estes defeitos são relativamente raros, desde que a engenharia de requisitos tenha sido levada a sério tanto por desenvolvedores como por usuários. Defeitos de implementação, por outro lado, são mais comuns. Programadores experientes em uma linguagem de programação, entretanto, não erram com frequência, principalmente quando fazem revisões cuidadosas de seu código.

Entre os requisitos e o código final existe sempre um **desenho**². Ele pode ser explícito, documentado e feito de acordo com determinadas técnicas. Ou pode existir apenas na cabeça do programador, de maneira informal e semiconsciente. Neste último caso, pesam mais as limitações humanas: de raciocínio, de memória e de capacidade de visualização. Por isso, geralmente um desenho de boa qualidade é explícito e documentado.

Os defeitos de desenho geralmente são quase tão graves quanto os de requisitos. Quando os programadores não são competentes em desenho, são quase tão frequentes quanto os defeitos de implementação. E muitos programadores que têm excelente domínio de uma linguagem de programação nunca tiveram formação em técnicas de desenho. Estas técnicas formam uma das disciplinas mais importantes da Engenharia de Software.

Defeitos de desenho têm geralmente consequências graves em todos os ramos da engenharia. Em construções, por exemplo, erros de desenho podem levar a vazamentos, perigo de incêndios, rachaduras e até desabamentos. As consequências nos produtos de software são igualmente sérias. Algumas resultados típicos de defeitos de desenho são:

- dificuldade de uso;
- lentidão;
- problemas imprevisíveis e irreprodutíveis;
- perda de dados;
- dificuldade de manutenção;
- dificuldade de adaptação e de expansão.

2.5.6 Modelos de maturidade

A produção industrial de software é quase sempre uma atividade coletiva. Alguns produtos são construídos inicialmente por indivíduos ou por pequenas equipes. Na medida em que se tornam sucesso de mercado, passam a evoluir. A partir daí, um número cada vez maior de pessoas passa a cuidar da sua manutenção e evolução. Por isso, quase todas as atividades de Engenharia de Software são empreendidas por organizações.

A maturidade de uma organização em Engenharia de Software mede o grau de competência, técnica e gerencial, que essa organização possui para produzir software de boa qualidade, dentro de prazos e custos razoáveis e previsíveis. Em organizações com baixa maturidade em software, os processos geralmente são informais. Processos informais são aqueles que existem apenas na cabeça de seus praticantes.

A existência de processos definidos é necessária para a maturidade das organizações produtoras de software. Os processos definidos permitem que a organização tenha um *modus operandi* padronizado e reproduzível. Isso facilita a capacitação das pessoas e torna o funcionamento da organização menos dependente de determinados indivíduos. Entretanto, não é suficiente que os processos sejam definidos. Processos rigorosamente definidos, mas não alinhados com os objetivos da organização são entaves burocráticos, e não fatores de produção.

Para tornar uma organização mais madura e capacitada, é realmente preciso melhorar a qualidade dos seus processos. Processos não melhoram simplesmente por estarem de acordo com um padrão externo.

² Esta palavra é aqui usada como sinônimo de design, e não na acepção de desenho pictórico (que equivaleria a “drawing” ou “drafting”). Muitas vezes o desenho é chamado de projeto. Usaremos o termo projeto apenas na acepção de unidade gerencial, conforme discutido no capítulo referente à Gestão de Projetos (correspondente a “project”).

O critério de verdadeiro êxito dos processos é a medida de quanto eles contribuem para que os produtos sejam entregues aos clientes e usuários com melhor qualidade, por menor custo e em prazo mais curto.

Diversas organizações do mundo propuseram paradigmas para a melhoria dos processos dos setores produtivos; em particular, algumas desenvolveram paradigmas para a melhoria dos processos de software. Estes paradigmas podem assumir diversas formas. Interessam aqui, especialmente, os paradigmas do tipo **modelos de capacitação**. Um modelo de capacitação serve para avaliar a maturidade dos processos de uma organização. Ele serve de referência para avaliar-se a maturidade destes processos.

Um modelo de capacitação particularmente importante para a área de software é o **CMM** (Capability Maturity Model), do Software Engineering Institute. O CMM é patrocinado pelo Departamento de Defesa americano, que o utiliza para avaliação da capacidade de seus fornecedores de software. Esse modelo teve grande aceitação da indústria americana de software e considerável influência no resto do mundo.

O CMM foi baseado em algumas das idéias mais importantes dos movimentos de qualidade industrial das últimas décadas. Destacam-se entre elas os conceitos de W. E. Deming, que também teve grande influência na filosofia japonesa de qualidade industrial. Esses conceitos foram adaptados para a área de software por Watts Humphrey [Humphrey90]. A primeira versão oficial do CMM foi divulgada no final dos anos 1980, e é descrita em relatórios técnicos do SEI ([Paulk+93], [Paulk+93a]) e um livro ([Paulk+95]).

O CMM focaliza os processos, que considera o fator de produção com maior potencial de melhoria a prazo mais curto. Outros fatores, como tecnologia e pessoas, só são tratados pelo CMM na medida em que interagem com os processos. Para enfatizar que o escopo do CMM se limita aos processos de software, o SEI passou a denominá-lo de SW-CMM, para distingui-lo de outros modelos de capacitação aplicáveis a áreas como desenvolvimento humano, engenharia de sistemas, definição de produtos e aquisição de software. Neste texto, fica entendido que CMM se refere sempre ao SW-CMM. A Tabela 3 resume os níveis do CMM, destacando as características mais marcantes de cada nível.

Número do nível	Nome do nível	Característica da organização	Característica dos processos
Nível 1	Inicial	Não segue rotinas	Processos caóticos
Nível 2	Repetitivo	Segue rotinas	Processos disciplinados
Nível 3	Definido	Escolhe rotinas	Processos padronizados
Nível 4	Gerido	Cria e aperfeiçoa rotinas	Processos previsíveis
Nível 5	Otimizante	Otimiza rotinas	Processos em melhoria contínua

Tabela 3 - Níveis do CMM

Página em branco

Processos

1 Visão geral

1.1 Processos em geral

Este capítulo trata de processos de desenvolvimento de software. Um processo é um conjunto de passos parcialmente ordenados, constituídos por atividades, métodos, práticas e transformações, usado para atingir uma meta. Esta meta geralmente está associada a um ou mais resultados concretos finais, que são os produtos da execução do processo.

Um processo é uma receita que é seguida por um projeto; o projeto concretiza uma abstração, que é o processo. Não se deve confundir um processo (digamos, uma receita de risoto de camarão) com o respectivo produto (risoto de camarão) ou com a execução do processo através de um projeto (a confecção de um risoto do camarão por determinado cozinheiro, em determinado dia).

Um processo é definido quando tem documentação que detalha: o que é feito (produto), quando (passos), por quem (agentes), as coisas que usa (insumos) e as coisas que produz (resultados). Processos podem ser definidos com mais ou menos detalhes, como acontece com qualquer receita. Os passos de um processo podem ter ordenação apenas parcial, o que pode permitir paralelismo entre alguns passos.

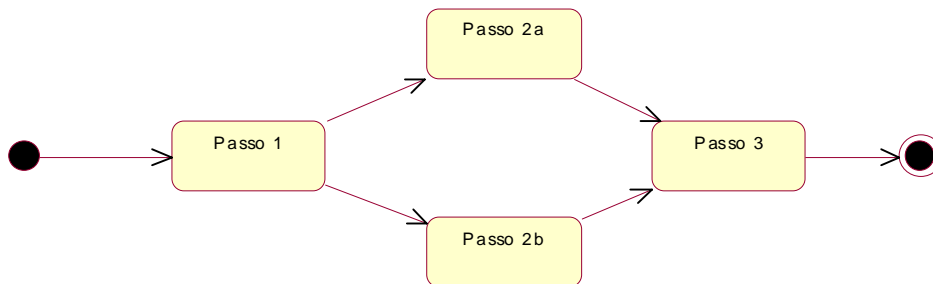


Figura 5 – Passos de um processo

Um subconjunto de passos pode ser definido como um subprocesso. Por exemplo, na Figura 5, um processo é representado na forma de um grafo. Existe paralelismo entre os passos 2a e 2b; os subconjuntos 1-2a e 2b-3 poderiam ser considerados subprocessos. Passos, subprocessos, agentes, insumos e resultados estão entre os **elementos** de um processo. A **arquitetura** de um processo define um arcabouço conceitual para a organização dos elementos de um processo. Uma discussão aprofundada sobre definição de processos pode ser encontrada em [Humphrey95].

1.2 Processos de software

Em engenharia de software, processos podem ser definidos para atividades como desenvolvimento, manutenção, aquisição e contratação de software. Pode-se também definir subprocessos para cada um destes; por exemplo, um processo de desenvolvimento abrange subprocessos de determinação dos requisitos, análise, desenho, implementação e testes. Em um processo de desenvolvimento de software, o ponto de partida para a arquitetura de um processo é a escolha de um modelo de ciclo de vida.

O ciclo de vida mais caótico é aquele que pode ser chamado de “**Codifica-remenda**” (Figura 6). Partindo apenas de uma especificação (ou nem isso), os desenvolvedores começam imediatamente a codificar, remendando à medida que os erros vão sendo descobertos. Nenhum processo definido é seguido. Infelizmente, é provavelmente o ciclo de vida mais usado. Para alguns desenvolvedores, esse modelo é atraente porque não exige nenhuma sofisticação técnica ou gerencial. Por outro lado, é um modelo de alto risco, impossível de gerir e que não permite assumir compromissos confiáveis.

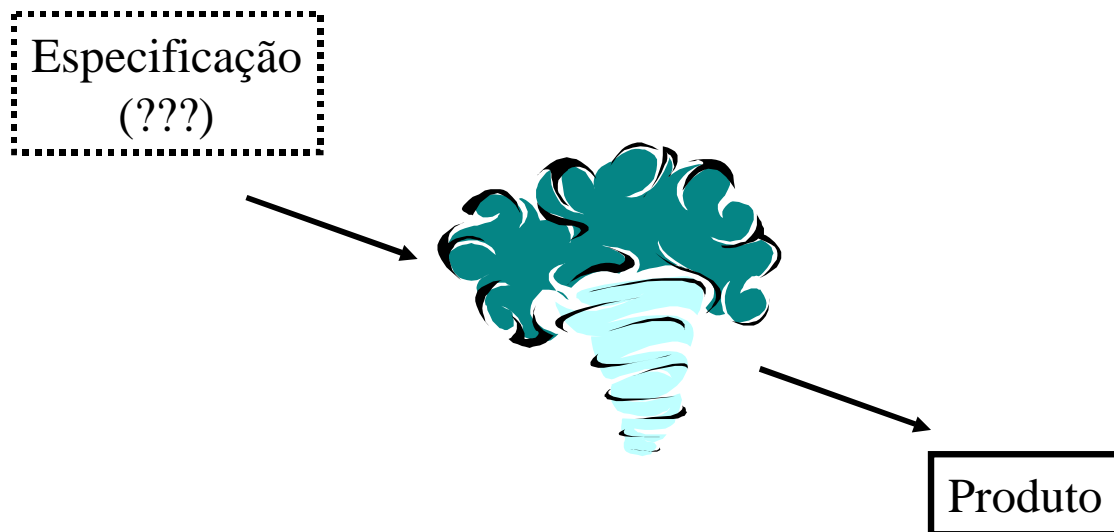


Figura 6 - O modelo de ciclo de vida Codifica e remenda

No modelo de ciclo de vida de **Cascata** (Figura 7), os principais subprocessos são executados em estrita sequência, o que permite demarcá-las com **pontos de controle** bem definidos. Estes pontos de controle facilitam muito a gestão dos projetos, o que faz com que esse processo seja, em princípio, confiável e utilizável em projetos de qualquer escala. Por outro lado, se interpretado literalmente, é um processo rígido e burocrático, em que as atividades de requisitos, análise e desenho têm de ser muito bem dominadas, pois não são permitidos erros. O modelo de cascata puro é de baixa visibilidade para o cliente, que só recebe o resultado final do projeto.

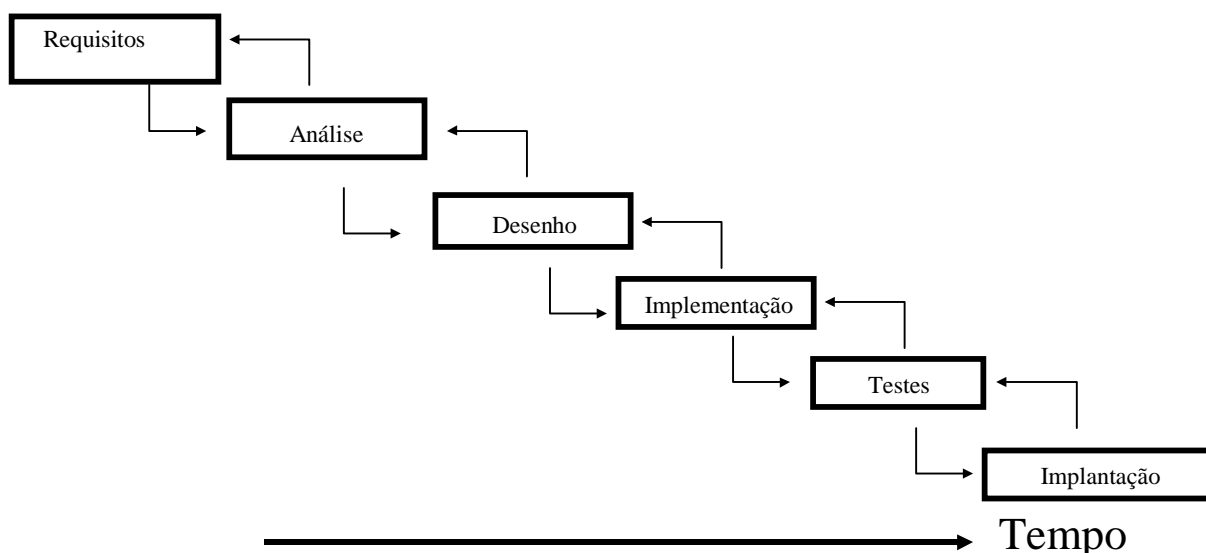


Figura 7 – O modelo de ciclo de vida em Cascata

Na prática, é sempre necessário permitir que, em fases posteriores, haja revisão e alteração de resultados das fases anteriores. Por exemplo, os modelos e documentos de especificação e desenho podem ser alterados durante a implementação, na medida em que problemas vão sendo descobertos. Uma variante que permite superposição entre fases e a realimentação de correções é o modelo

“**Sashimi**” (Figura 8). A superposição das fases torna difícil gerenciar projetos baseados nesse modelo de ciclo de vida.

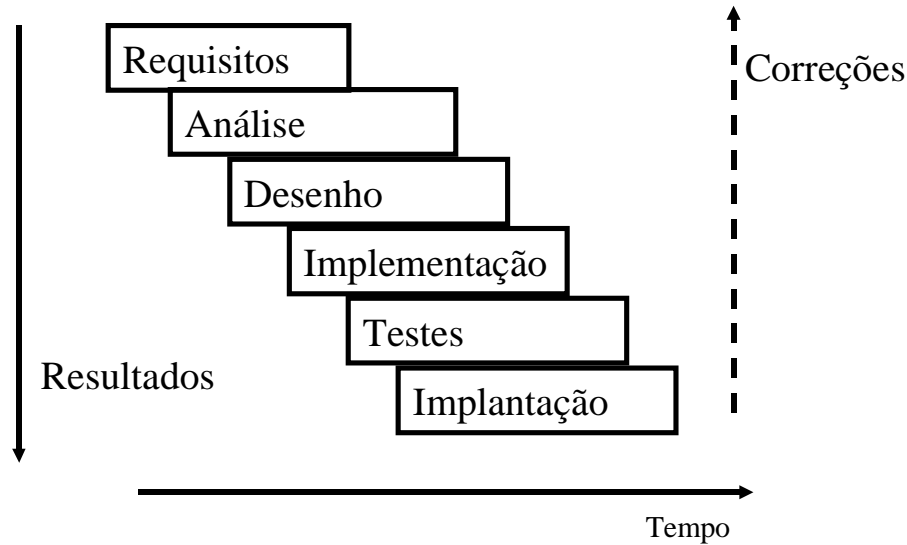


Figura 8 - O modelo de ciclo de vida em Sashimi

Um modelo de ciclo de vida radicalmente diferente é o modelo em **Espiral** (Figura 9). O produto é desenvolvido em uma série de iterações. Cada nova iteração corresponde a uma volta na espiral. Isso permite construir produtos em prazos curtos, com novas características e recursos que são agregados na medida em que a experiência descobre sua necessidade. As atividades de manutenção são usadas para identificar problemas; seus registros fornecem dados para definir os requisitos das próximas liberações. O principal problema do ciclo de vida em espiral é que ele requer gestão muito sofisticada para ser previsível e confiável.

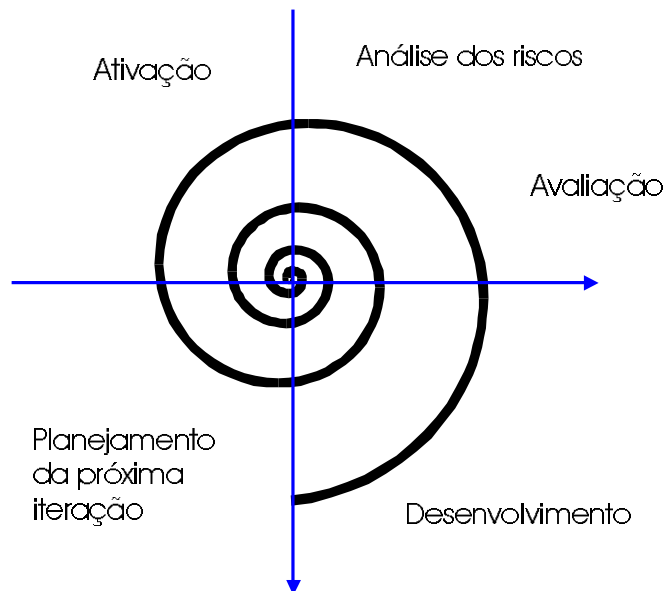


Figura 9 – O modelo de ciclo de vida em Espiral

Uma variante do modelo em espiral é o modelo de **Prototipagem evolutiva**. Neste modelo, a espiral é usada não para desenvolver o produto completo, mas para construir uma série de versões provisórias que são chamadas de protótipos. Os protótipos cobrem cada vez mais requisitos, até que se atinja o produto desejado. A prototipagem evolutiva permite que os requisitos sejam definidos progressivamente, e apresenta alta flexibilidade e visibilidade para os clientes. Entretanto, também

requer gestão sofisticada, e o desenho deve ser de excelente qualidade, para que a estrutura do produto não se degenere ao longo dos protótipos.

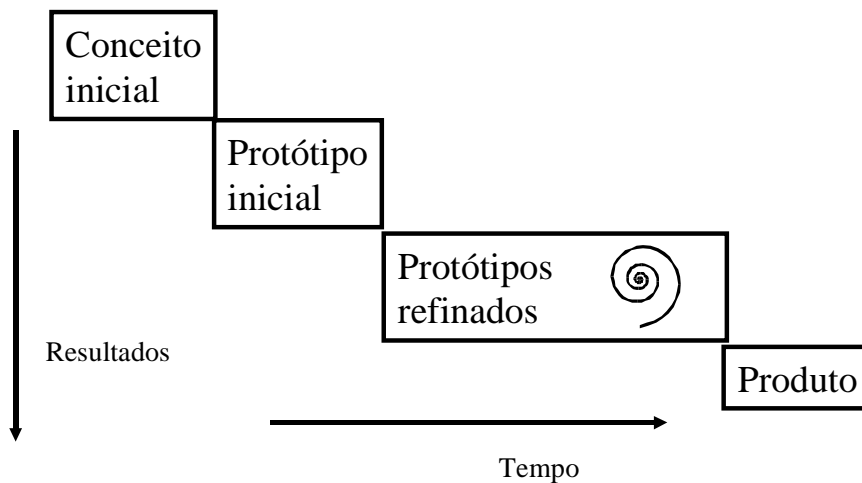


Figura 10 - O modelo de ciclo de vida de Prototipagem evolutiva

O modelo de **Entrega por estágios** (Figura 11) difere do modelo de cascata pela entrega ao cliente de liberações parciais do produto. Isso aumenta a visibilidade do projeto, o que geralmente é um fator muito importante no relacionamento com o cliente. Apresenta, entretanto, os demais defeitos do modelo em cascata.

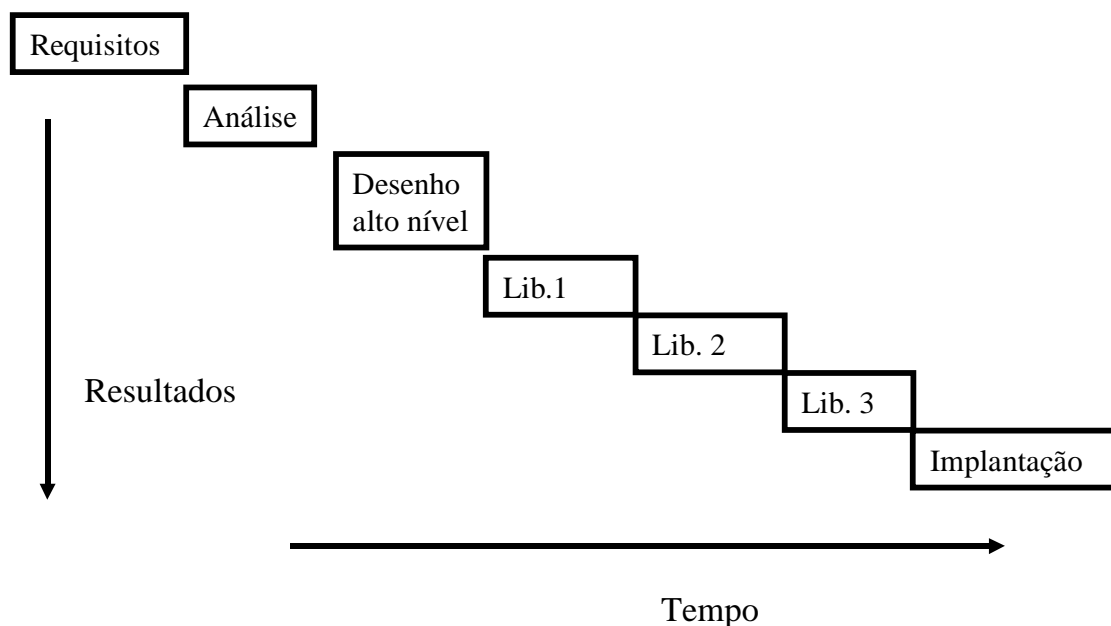


Figura 11 - O modelo de ciclo de vida de **Entrega por estágios**

Uma combinação dos modelos de Cascata e Prototipagem evolutiva forma o modelo de Entrega Evolutiva (Figura 12). Este modelo permite que, em pontos bem definidos, os usuários possam avaliar partes do produto e fornecer realimentação quanto às decisões tomadas. Facilita também o acompanhamento do progresso de cada projeto, tanto por parte de seus gerentes como dos clientes. A principal dificuldade continua ser a realização do Desenho Inicial: ele deve produzir uma arquitetura de produto robusta, que se mantenha íntegra ao longo dos ciclos de liberações parciais.

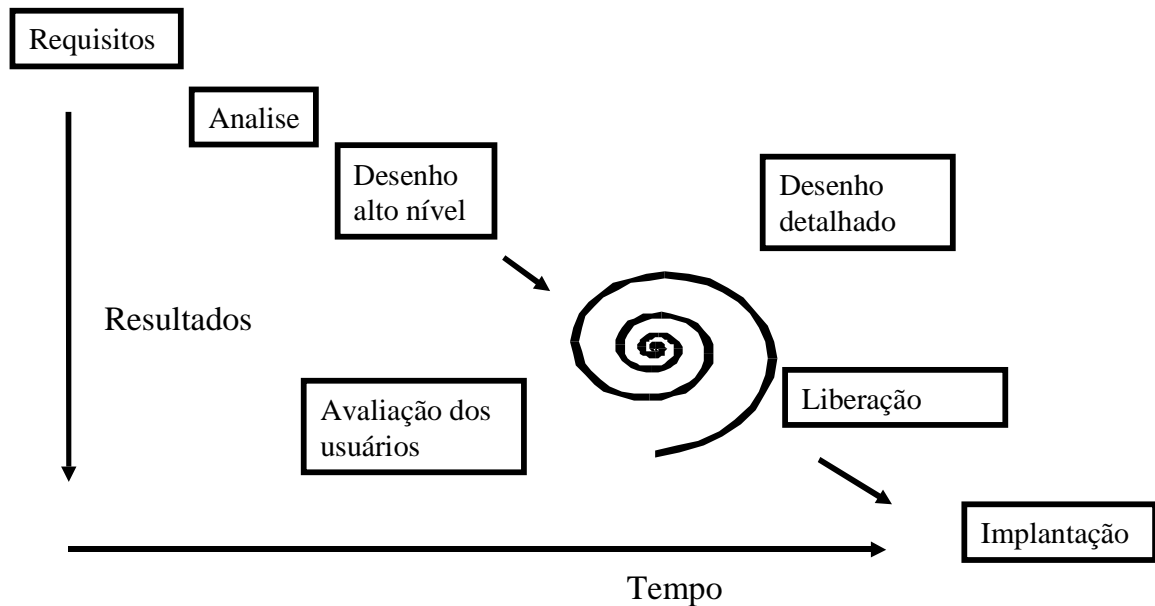


Figura 12 – O modelo de ciclo de vida de Entrega Evolutiva

Em modelos **dirigidos por prazo** (time-boxed), o produto é aquilo que se consegue fazer dentro de determinado prazo. Esses modelos podem ser razoáveis quando se consegue definir um conjunto de requisitos indispensáveis para os quais se sabe que os prazos estabelecidos são suficientes, e as folgas são usadas apenas para implementar requisitos opcionais. Na prática, os prazos costumam ser definidos de forma política, e o “produto” que se entrega no final do prazo é apenas um resultado parcial. Este será completado aos trancos e barrancos em desenvolvimento posterior, disfarçado de “manutenção”.

O ciclo de vida pode ser também **dirigido por ferramenta** (que pode ser chamada de ferramenta de CASE ou plataforma de desenvolvimento, pelos respectivos fabricantes). Algumas ferramentas impõem processos rígidos, que podem ser adequados para tipos bem específicos de produtos. A qualidade destes processos depende fundamentalmente da qualidade da ferramenta e de que o uso do processo seja restrito ao seu domínio de aplicação.

Uma solução tentadora pode ser comprar em vez de desenvolver. Quando se encontra um produto que atende aos requisitos desejados, é uma boa solução. Nesse caso, não há processo de desenvolvimento, mas existirão processos de aquisição, de implantação e, possivelmente, de adaptação e integração com outros sistemas. Dependendo dos casos, esse conjunto de processos pode ser mais sofisticado e caro do que um processo de desenvolvimento.

Muitos outros detalhes podem ser discutidos a respeito dos modelos de ciclo de vida de software. Um tratamento bastante aprofundado é encontrado em [McConnell96].

2 Exemplos de processos

2.1 O Processo Pessoal de Software

Como foi mencionado anteriormente, nos primeiros anos de existência do paradigma CMM, as organizações avaliadas nos níveis superiores de maturidade eram muito poucas. Uma característica destes níveis é o uso de processos definidos de forma precisa e quantitativa, que possam ser continuamente melhorados.

Partindo do princípio de que, para atingir os níveis superiores de maturidade, era necessário melhorar a prática dos processos em nível dos desenvolvedores individuais, Watts Humphrey propôs em [Humphrey95], uma série de processos pessoais que pudessem ser aprendidos em uma disciplina de

engenharia de software. Esse conjunto é chamado de Processo Pessoal de Software (“*Personal Software Process*”), ou **PSP**.

Esses processos são aprendidos através de uma seqüência de pequenos projetos, contida neste livro. Os projetos devem ser realizados seguindo rigorosamente os processos, que incluem um conjunto de formulários, scripts e relatórios predefinidos. Os projetos são individuais, com duração típica de cerca de 10 horas.

Classificação	Nome	Elementos novos de processo
Processos pessoais básicos	PSP0	Registro de tempos Registro de defeitos Padronização dos tipos de defeitos
	PSP0.1	Padronização da codificação Medição do tamanho Proposição de melhorias de processo
Processos pessoais com planejamento	PSP1	Estimativas de tamanho Relatórios de testes
	PSP1.1	Planejamento de tarefas Planejamento de cronogramas
Processos pessoais com gestão da qualidade	PSP2	Revisões de código Revisões de desenho
	PSP2.1	Modelos de desenho
Processos pessoais cíclicos	PSP3	Desenvolvimento cíclico

Tabela 4 – Os estágios do PSP

A Tabela 5 apresenta detalhes do PSP3, versão final do processo, que inclui os elementos introduzidos em todos os processos anteriores. O PSP3 tem um ciclo de vida de entrega em estágios. Não existe um tratamento separado dos requisitos; estes são muito simples em todos os projetos, e as respectivas atividades são consideradas parte do planejamento. O planejamento inclui a estimativa de tamanhos (medidos em linhas de código, com base em um modelo conceitual orientado a objetos), de esforços (medidos em tempo de desenvolvimento), de cronograma (tempo físico) e de defeitos.

O desenho é feito de acordo com padrões rigorosos, que usam conceitos de orientação a objetos, síntese lógica e máquinas seqüenciais, e submetido a uma fase rigorosa de verificação. Com base no desenho, a fase de desenvolvimento é dividida em ciclos; cada ciclo inclui desenho detalhado, codificação, revisão do código, compilação e testes de unidade dos respectivos módulos. Ao final de cada ciclo, o planejamento é reavaliado. O PSP sempre termina com uma fase de post-mortem, na qual é feito um balanço final do projeto. As lições aprendidas são documentadas e analisadas, para melhoria do processo no projeto seguinte.

Uma versão bastante simplificada do PSP foi apresentada em [Humphrey97]. Esta versão introdutória tem o objetivo de ensinar o uso de processos bem definidos no início de cursos de graduação em informática. O processo é introduzido através dos exercícios das disciplinas iniciais de algoritmos e programação.

Fase	Atividades	Resultados
Planejamento	Especificação dos requisitos. Estimativa de tamanho. Estratégia. Estimativa de recursos. Estimativa de prazos. Estimativa de defeitos.	Documentos dos requisitos. Modelo conceitual. Planos de recursos, prazos e qualidade. Registro de tempos.
Desenho de alto nível	Especificações externas. Desenho dos módulos. Prototipagem. Estratégia de desenvolvimento. Documentação da estratégia de desenvolvimento. Registro de acompanhamento de problema.	Especificações funcionais. Especificações de estados. Roteiros operacionais. Especificações de reutilização. Estratégia de desenvolvimento. Estratégia de testes. Registro de tempos.
Revisão do desenho de alto nível	Verificação da cobertura do desenho. Verificação da máquina de estados. Verificação lógica. Verificação da consistência do desenho. Verificação da reutilização. Verificação da estratégia de desenvolvimento. Conserto de defeitos.	Desenho de alto nível revisto. Estratégia de desenvolvimento revista. Estratégia de testes revista. Registro de defeitos de desenho de alto nível. Registro de problemas de desenho de alto nível. Registro de tempos.
Desenvolvimento	Desenho do módulo. Revisão do desenho. Codificação. Revisão do código. Compilação. Teste. Reavaliação e reciclagem.	Desenho detalhado dos módulos. Código dos módulos. Registro de defeitos dos módulos. Registro de problemas dos módulos. Relatórios dos testes. Registro de tempos.
Post-mortem	Contagem de defeitos injetados e removidos. Contagem de tamanhos e tempos.	Resumo do projeto.

Tabela 5 - Fases do PSP3

2.2 O Processo de Software para Times

Como sequência natural do PSP, Humphrey introduziu em [Humphrey99] o Processo de Software para Times (*Team Software Process*), ou **TSP**. A Tabela 6 e a Tabela 7 apresentam as partes principais da versão publicada deste processo (TSPe), que é orientada para utilização educacional. O TSP usa um modelo em espiral; os passos mostrados nessas tabelas correspondem a um dos ciclos. Ao longo de 15 semanas, são executados tipicamente três ciclos de desenvolvimento de um produto.

Os participantes do time de desenvolvedores são organizados de tal forma que cada desenvolvedor desempenhe um ou dois papéis gerenciais bem definidos, além de dividir a carga de desenvolvimento. Os papéis suportados pelo processo são os de gerente de desenvolvimento, de planejamento, de qualidade, de processo e de suporte, além do líder do time.

Fase	Atividades
Lançamento	<p>Descrição do curso: visão geral; informação para os alunos; objetivos do produto.</p> <p>Formação dos times: integrantes, metas e reuniões.</p> <p>Primeira reunião do time: requisitos de dados.</p> <p>Ativação dos projetos.</p>
Estratégia	<p>Visão geral da estratégia de desenvolvimento.</p> <p>Critérios da estratégia de desenvolvimento.</p> <p>Seleção da estratégia de desenvolvimento.</p> <p>Documentação da estratégia de desenvolvimento.</p> <p>Estimativas de tamanho.</p> <p>Definição do processo de controle de mudanças.</p>
Planejamento	<p>Visão geral do plano de desenvolvimento.</p> <p>Produção do planos de tarefas.</p> <p>Produção do cronograma.</p> <p>Produção dos planos pessoais dos engenheiros</p> <p>Balanceamento de carga dos engenheiros.</p> <p>Produção do plano da qualidade.</p>
Requisitos	<p>Revisão do processo de requisitos.</p> <p>Revisão das demandas dos usuários.</p> <p>Esclarecimento das demandas dos usuários.</p> <p>Distribuição das tarefas de requisitos.</p> <p>Documentação dos requisitos.</p> <p>Revisão dos requisitos.</p> <p>Colocação dos requisitos na linha de base.</p> <p>Revisão dos requisitos pelos usuários.</p>

Tabela 6 - Fases do TSPE – parte 1

O planejamento e o controle rigoroso de tamanhos, esforços, prazos e defeitos, característicos do PSP, continuam a ser feitos. O TSP enfatiza algumas áreas que correspondem às áreas do nível 2 do CMM: gestão dos requisitos, planejamento e controle de projetos, garantia da qualidade e gestão de configurações. Estas áreas de processo não são tratadas pelo PSP por serem consideradas muito simples no caso de projetos individuais (exceto alguns aspectos do planejamento de projetos).

Fase	Atividades
Desenho	Revisão do processo de desenho. Desenho de alto nível. Distribuição das tarefas de desenho. Documentação do desenho. Revisão do desenho. Atualização do desenho, com colocação na linha de base.
Implementação	Revisão do processo de implementação. Distribuição das tarefas de implementação. Desenho detalhado Inspeção do desenho detalhado. Código. Inspeção do código. Teste de unidades. Revisão da qualidade dos componentes. Liberação dos componentes.
Testes	Revisão do processo de testes. Planejamento e desenvolvimento dos testes. Construção. Integração. Testes de sistema. Documentação dos testes.
Post-mortem	Revisão do processo de post-mortem. Revisão dos dados de processo. Avaliação do desempenho dos papéis. Preparação do relatório do ciclo. Revisão dos pares.

Tabela 7 - Fases do TSPe -- parte 2

2.3 O Processo Orientado a Objetos para Software Extensível

O Processo Orientado a Objetos para Software Extensível (PROSE) foi desenvolvido dentro do Departamento de Ciência da Computação da Universidade Federal de Minas Gerais, sob coordenação do autor. Originalmente, este processo foi desenvolvido para uso em projetos de desenvolvimento de sistemas de apoio à Engenharia de Telecomunicações, contratados pela Telemig (hoje Telemar-MG).

O PROSE foi concebido com um processo padrão que visava a cobrir todo o ciclo de vida dos produtos de software, especialmente de aplicativos extensíveis, através de sucessivas versões produzidas durante um ciclo de vida de produto com duração de vários anos. Esses produtos normalmente seriam aplicativos gráficos interativos, baseados na tecnologia orientada a objetos.

O processo completo inclui um conjunto de recomendações, padrões, roteiros de revisão, políticas e modelos. A última versão publicada desse processo está contida em [Paula+98a], [Paula+98b], [Paula+98c] e [Paula+98d]. Uma característica central do PROSE é o uso da tecnologia orientada a objetos nas atividades de análise, desenho e implementação. Os modelos produzidos no processo usam a notação UML (“Unified Modeling Language”), definida por Booch, Jacobson e Rumbaugh em [Booch+99] e [Rumbaugh+99]. A Tabela 8 descreve a estrutura básica do PROSE. O modelo de ciclo de vida de cada projeto é o da entrega evolutiva. O ciclo de vida de um produto é constituído por versões sucessivas, que são produzidas em diferentes projetos; portanto, o modelo de ciclo de vida de

produto é em espiral. Muitos dos elementos do PROSE serviram de base ao PRAXIS, que será descrito detalhadamente adiante.

Macroatividade	Fase	Subfase
Ativação		
Especificação	Engenharia dos Requisitos	Levantamento dos requisitos
		Detalhamento dos requisitos
	Planejamento	Planejamento do desenvolvimento
		Planejamento da qualidade
Desenvolvimento	Desenho	Desenho dos testes de aceitação
		Desenho arquitetônico
		Desenho das interfaces de usuário
		Planejamento das liberações executáveis
	Implementação	Construção da liberação executável 1
		Construção da liberação executável 2
		Construção da liberação executável ...
		Construção da liberação executável final
		Preparação da implantação
		Testes alfa
Implantação	Testes beta	
	Operação piloto	

Tabela 8 - Atividades do PROSE

2.4 O Processo Unificado

Booch, Jacobson e Rumbaugh propuseram a UML como uma notação de modelagem orientada em objetos, independente de processos de desenvolvimento. Além disso, propuseram o Processo Unificado (“*Unified Process*”) [Jacobson+99], que utiliza a UML como notação de uma série de modelos que compõem os principais resultados das atividades do processo. O Processo Unificado descende de métodos anteriores propostos pelos autores em [Booch94], [Booch96], [Jacobson94], [Jacobson+94a], [Jacobson+97] e [Rumbaugh91]. Um produto comercial baseado no Processo Unificado é o Rational Unified Process; ele contém uma base de conhecimento que detalha e estende o material apresentado em [Jacobson+99].

Fase	Descrição
Concepção	Fase na qual se justifica a execução de um projeto de desenvolvimento de software, do ponto de vista do negócio do cliente.
Elaboração	Fase na qual o produto é detalhado o suficiente para permitir um planejamento acurado da fase de construção.
Construção	Fase na qual é produzida uma versão completamente operacional do produto.
Transição	Fase na qual o produto é colocado à disposição de uma comunidade de usuários.

Tabela 9 - Fases do Processo Unificado

Segundo seus autores, o Processo Unificado apresenta as seguintes características centrais:

- é dirigido por casos de uso;
- é centrado na arquitetura;

- é iterativo e incremental.

O ciclo de vida de um produto tem um modelo em espiral, em que cada projeto constitui um ciclo, que entrega uma liberação do produto. O Processo Unificado não trata do que acontece entre ciclos. Cada ciclo é dividido nas fases mostradas na Tabela 9. Uma característica importante do Processo Unificado é que as atividades técnicas são divididas em subprocessos chamados de **fluxos de trabalho** (“workflows”), mostrados na Tabela 10. Cada fluxo de trabalho (que chamaremos simplesmente de fluxo) tem um tema técnico específico, enquanto as fases constituem divisões gerenciais, caracterizadas por atingirem metas bem definidas.

Fluxo	Descrição
Requisitos	Fluxo que visa obter um conjunto de requisitos de um produto, acordado entre cliente e fornecedor.
Análise	Fluxo cujo objetivo é detalhar, estruturar e validar os requisitos, de forma que estes possam ser usados como base para o planejamento detalhado.
Desenho	Fluxo cujo objetivo é formular um modelo estrutural do produto, que sirva de base para a implementação
Implementação	Fluxo cujo objetivo é realizar o desenho em termos de componentes de código.
Testes	Fluxo cujo objetivo é verificar os resultados da implementação

Tabela 10 - Fluxos do Processo Unificado

3 Praxis

3.1 Visão geral

3.1.1 Introdução

Esta seção define o **Praxis**, um processo destinado a suportar projetos didáticos em disciplinas de Engenharia de Software de cursos de Informática. A sigla Praxis significa P**RO**cesso para A**P**licativos e**X**tensíveis I**NT**erativo**S**, refletindo uma ênfase no desenvolvimento de aplicativos gráficos interativos, baseados na tecnologia orientada a objetos.

O Praxis é desenhado para suportar projetos de seis meses a um ano de duração, realizados individualmente ou por pequenas equipes. Com isso, pretende-se que ele seja utilizável para projetos de fim de curso, ou projetos de aplicação de disciplinas de engenharia de software. A cobertura de todo o material do Praxis normalmente exigirá dois semestres letivos. O Praxis abrange material relativo tanto a métodos gerenciais como técnicos.

Procurou-se combinar no Praxis a experiência de desenvolvimento do PROSE, assim como elementos selecionados do PSP, do TSP e do Processo Unificado. O Praxis não substitui nem concorre com nenhum desses processos, pois tem objetivos diferentes, podendo ser usado como base de treinamento preparatório para cada um deles.

A UML é a notação de modelagem utilizada no Praxis, em todos os passos em que for aplicável. As práticas gerenciais são inspiradas nas práticas-chaves dos níveis 2 e 3 do SW-CMM. Os padrões incluídos procuram ser conformes com os padrões correspondentes do IEEE [IEEE94]. O material inclui modelos de documentos, que facilitam a preparação dos documentos requeridos, e roteiros de revisão, que facilitam a verificação destes.

3.1.2 Nomenclatura

Elemento	Descrição
Passo	Divisão formal de um processo, com pré-requisitos, entradas, critérios de aprovação e resultados definidos.
Fase	Divisão maior de um processo, para fins gerenciais, que corresponde aos pontos principais de aceitação por parte do cliente.
Iteração	Passo constituinte de uma fase, no qual se atinge um conjunto bem definido de metas parciais de um projeto.
Fluxo	Subprocesso caracterizado por um tema técnico.
Etapas	Passo constituinte de um fluxo.
Atividade	Termo genérico para unidades de trabalho executadas em um passo.

Tabela 11 – Elementos constituintes de um processo

A nomenclatura apresentada na Tabela 11 é utilizada para descrição das unidades de trabalho que compõem o Praxis. No estilo do Processo Unificado, o Praxis abrange tanto fases (subprocessos gerenciais) quanto fluxos (subprocessos técnicos). Uma fase é composta por uma ou mais iterações. Um fluxo é dividido em uma ou mais etapas. Iterações e etapas são exemplos de passos (Figura 13). Este diagrama deve ser lido: “Um processo possui uma ou mais fases e um ou mais fluxos. Uma fase possui uma ou mais iterações. Um fluxo possui uma ou mais etapas. Etapas e iterações são espécies de passos.”

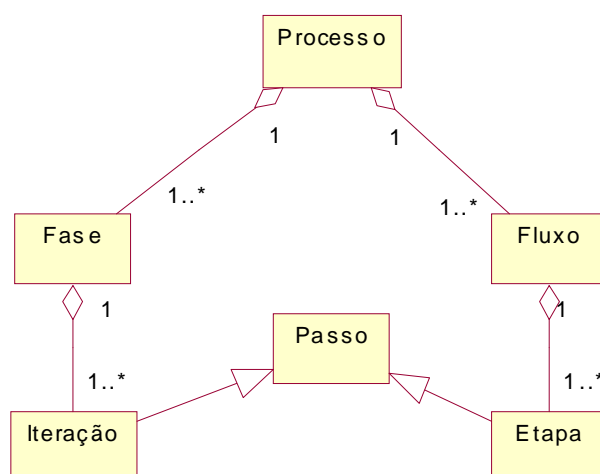


Figura 13 – Divisões de um processo

Elemento	Definição
Descrição	Frase que resume a natureza do passo.
Pré-requisitos	Condições que devam ser satisfeitas para o início de um passo, compreendendo tanto eventos externos quanto passos anteriores que devam ter sido completados e aprovados.
Insumos	Artefatos cuja disponibilidade prévia é necessária para a execução de uma atividade.
Atividades	Atividades executadas durante um passo.
Resultados	Artefatos produzidos durante a execução um passo.
Critérios de aprovação	Condições para que um passo seja considerado completo e aprovado.
Normas pertinentes	Normas aplicáveis a este passo

Tabela 12 – Elementos do script de um passo do Praxis

Um passo é definido através de um script, apresentado em forma de tabela. A Tabela 12 e a Figura 14 apresentam os elementos que serão utilizados na apresentação dos scripts. Cada passo está relacionado com artefatos que consome (insumos) e produz (resultados); está sujeito a condições de entrada (pré-requisitos) e de saída (critérios de aprovação); e executa uma série de atividades, sujeitas a normas pertinentes. Uma atividade pode ser executada por vários passos; pode ser divisível ou não em outras atividades; e está contida em um único fluxo.

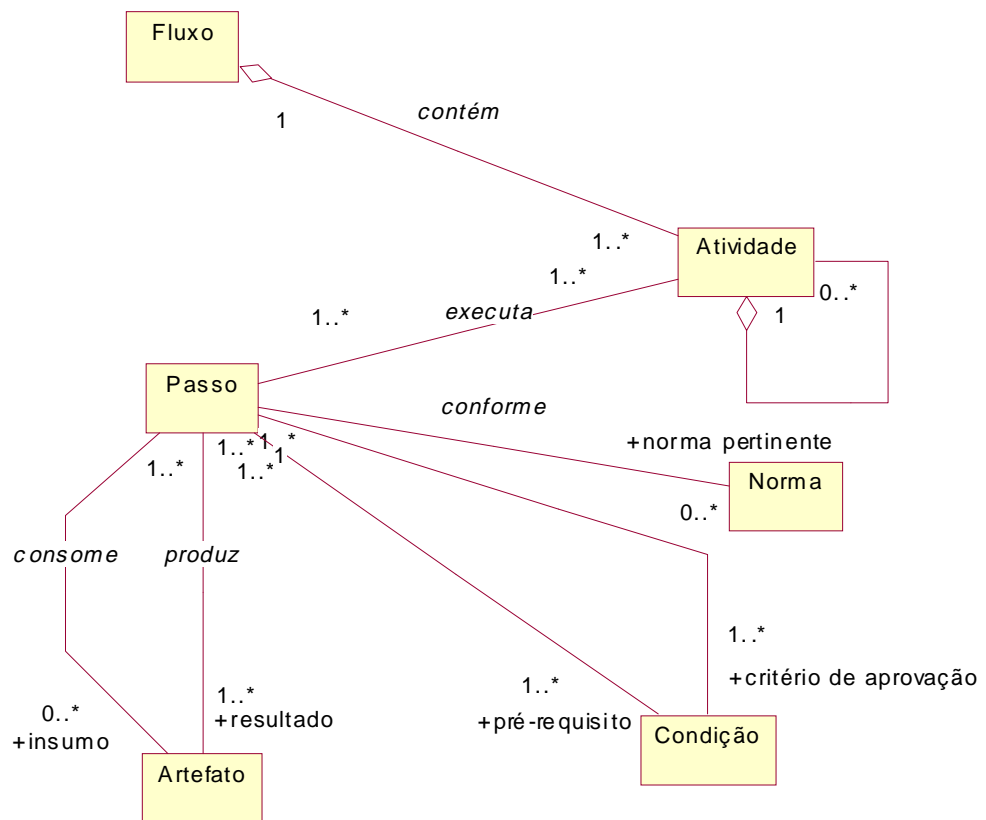


Figura 14 – Relacionamentos entre os elementos do script de um passo

3.1.3 Arquitetura

3.1.3.1 Fases

Os elementos maiores da arquitetura do Praxis são inspirados nos elementos correspondentes do Processo Unificado (Tabela 13 e Tabela 14), tendo-se em vista compatibilizar a nomenclatura com um processo que terá provavelmente grande aceitação na indústria de software. As definições das fases e fluxos são mais específicas em relação às opções de desenho de processo adotadas no Praxis.

Fase	Descrição
Concepção	Fase na qual necessidades dos usuários e conceitos da aplicação são analisados o suficiente para justificar a especificação de um produto de software, resultando em uma proposta de especificação.
Elaboração	Fase na qual a especificação do produto é detalhada o suficiente para modelar conceitualmente o domínio do problema, validar os requisitos em termos deste modelo conceitual e permitir um planejamento acurado da fase de construção.
Construção	Fase na qual é desenvolvida (desenhada, implementada e testada) uma liberação completamente operacional do produto, que atende aos requisitos especificados.
Transição	Fase na qual o produto é colocado à disposição de uma comunidade de usuários para testes finais, treinamento e uso inicial.

Tabela 13 - Fases do Praxis

Fluxo	Descrição
Requisitos	Fluxo que visa a obter um conjunto de requisitos de um produto, acordado entre cliente e fornecedor.
Análise	Fluxo que visa a detalhar, estruturar e validar os requisitos, em termos de um modelo conceitual do problema, de forma que estes possam ser usados como base para o planejamento e acompanhamento detalhados da construção do produto.
Desenho	Fluxo que visa a formular um modelo estrutural do produto que sirva de base para a implementação, definindo os componentes a desenvolver e a reutilizar, assim como as interfaces entre si e com o contexto do produto.
Implementação	Fluxo que visa a detalhar e implementar o desenho através de componentes de código e de documentação associada.
Testes	Fluxo que visa a verificar os resultados da implementação, através do planejamento, desenho e realização de baterias de testes.

Tabela 14 - Fluxos do Praxis

3.1.3.2 Iterações

A divisão das fases do Praxis obedece ao modelo de ciclo de vida de entrega evolutiva. Os nomes das iterações foram escolhidos de forma a ilustrar o tema principal de cada. Não devem ser confundidos com os nomes semelhantes de fluxos, embora as coincidências de nome ressaltem os fluxos mais importantes de cada iteração.

A concepção contém uma única iteração, chamada de "Ativação". A elaboração consta das iterações de "Levantamento dos Requisitos", em que o foco é a captura dos requisitos junto aos usuários do produto, e de "Análise dos Requisitos", que focaliza o detalhamento e a validação dos requisitos através de técnicas de análise.

A construção começa por uma iteração de "Desenho Inicial", em que é realizado o desenho do produto em nível mais alto de abstração, de forma a permitir a divisão das funções e componentes do produto, ao longo das iterações seguintes. Em seguida aparecem uma ou mais "Liberações", sendo a integração

final do produto testada na iteração dos "Testes Alfa". Note-se que todas as liberações, com exceção da última, são liberações parciais, por não contemplarem todos os requisitos especificados.

A transição começa pelos "Testes Beta", iteração na qual os testes de aceitação são repetidos no ambiente dos usuários. Na iteração de "Operação Piloto", o produto é usado de forma vigiada, de preferência em instalações que contemplem o caráter ainda experimental da operação. Em produtos comerciais de prateleira a transição geralmente termina nos testes beta, enquanto em sistemas de missão crítica a operação piloto pode levar um longo tempo, estendendo-se o uso do produto gradualmente através das instalações do cliente.

Terminado o ciclo de um projeto, começa a operação do produto, que durará enquanto o produto não for substituído por nova versão ou não for desativado. Durante a operação, problemas relativos ao produto são tratados através de um processo de manutenção. Este processo será considerado pertinente aos métodos gerenciais.

Fase	Iteração	Descrição
Concepção	Ativação	Levantamento e análise das necessidades dos usuários e conceitos da aplicação, em nível de detalhe suficiente para justificar a especificação de um produto de software.
Elaboração	Levantamento dos Requisitos	Levantamento detalhado das funções, interfaces e requisitos não funcionais desejados para o produto.
	Análise dos Requisitos	Modelagem conceitual dos elementos relevantes do domínio do problema e uso desse modelo para validação dos requisitos e planejamento detalhado da fase de Construção.
Construção	Desenho Inicial	Definição interna e externa dos componentes de um produto de software, em nível suficiente para decidir as principais questões de arquitetura e tecnologia, e para permitir o planejamento detalhado das atividades de implementação.
	Liberação 1	Implementação de um subconjunto de funções do produto que será avaliado pelos usuários.
	Liberação ...	Idem.
	Liberação Final	Idem.
	Testes Alfa	Realização dos testes de aceitação, no ambiente dos desenvolvedores, juntamente com elaboração da documentação de usuário e possíveis planos de Transição.
Transição	Testes Beta	Realização dos testes de aceitação, no ambiente dos usuários.
	Operação Piloto	Operação experimental do produto em instalação piloto do cliente, com a resolução de eventuais problemas através de processo de manutenção

Tabela 15 – Detalhamento das fases do Praxis

3.1.3.3 Elementos das iterações

Cada uma das iterações é detalhada na próxima subseção. Para cada iteração, é apresentado um script com os seguintes elementos:

- descrição sucinta da iteração;
- pré-requisitos internos e externos ao processo;
- insumos da iteração, inclusive antigos (consumidos também por iterações anteriores) e novos (consumidos pela primeira vez nesta iteração), identificados pelas respectivas siglas;
- atividades normalmente executadas na iteração, repartidas de acordo com os respectivos fluxos;

- resultados da iteração, indicando-se nome e sigla dos artefatos produzidos e, caso a iteração produza apenas partes de um artefato, que partes são estas;
- critérios de aprovação, que devem ser satisfeitos para que a iteração possa ser concluída;
- normas pertinentes às atividades da iteração.

Os artefatos, condições e normas citados são discutidos mais detalhadamente na seção seguinte. As descrições aqui apresentadas focalizam principalmente os elementos técnicos do processo. Entretanto, os elementos gerenciais mais importantes são indicados, agrupando-se as atividades gerenciais em um fluxo extra de Gestão.

3.1.3.4 Elementos dos fluxos

Para representação dos detalhes de cada fluxo serão utilizados os diagramas de atividades da UML. Estes diagramas são uma variante dos fluxogramas, sendo geralmente usados para descrever processos de negócio. Os fluxos podem ser encarados como processos de negócio dos desenvolvedores de software.

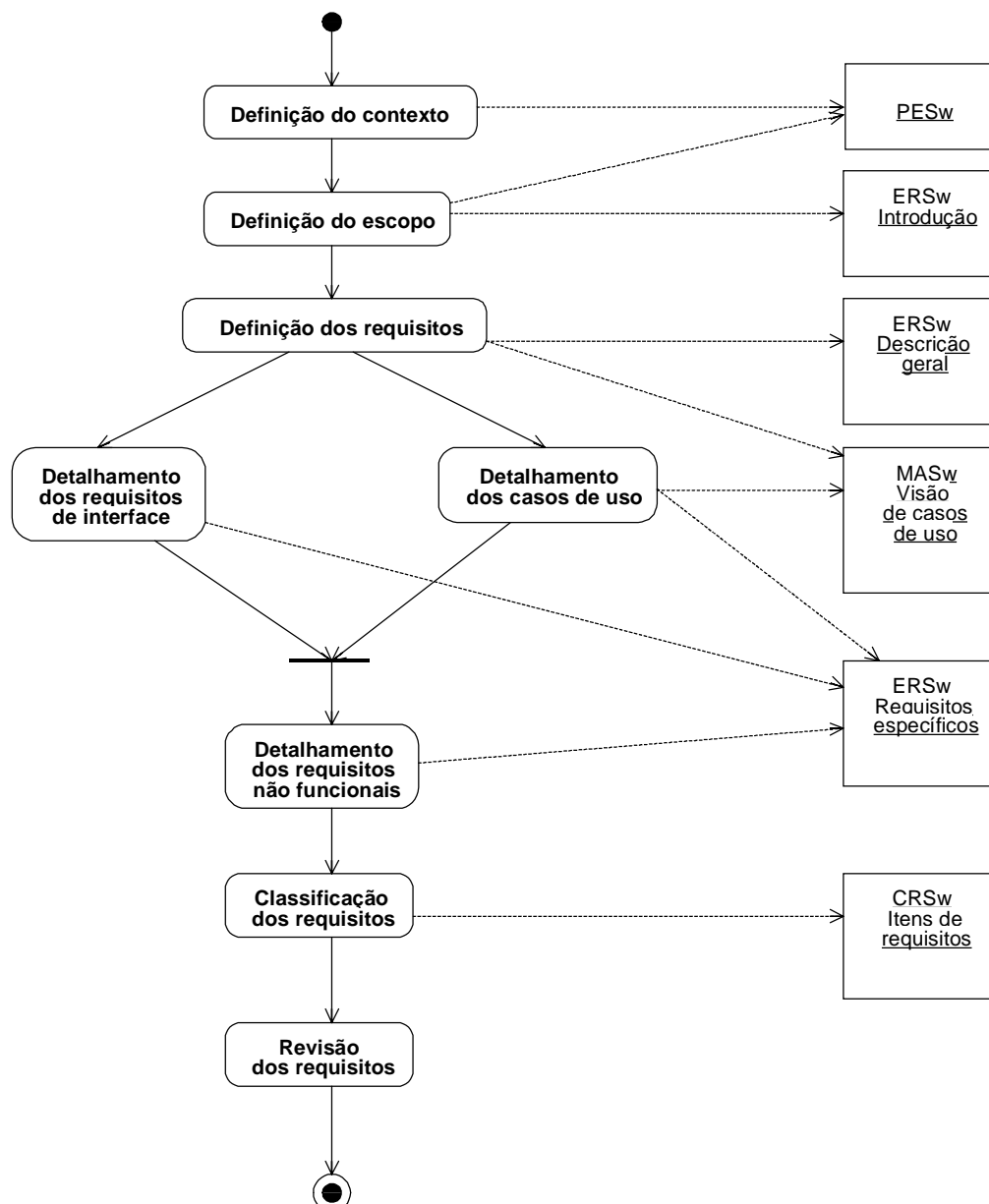


Figura 15 - Exemplo de detalhamento de fluxo

A Figura 15 apresenta como exemplo o diagrama de atividades que descreve o fluxo de Requisitos do Praxis. A Tabela 16 explica os símbolos usados.

Símbolo	Descrição
Retângulo ovalado	Atividade (passo do fluxo).
Retângulo	Objeto (artefato do fluxo).
Seta cheia	Relação de precedência entre atividades.
Seta pontilhada	Consumo ou produção de objeto por atividade.
Linha horizontal cheia	Ponto de sincronização (onde subfluxos paralelos se juntam).
Pequeno círculo cheio	Estado inicial.
Círculo cheio dentro de círculo vazio	Estado final.

Tabela 16 - Símbolos dos diagramas de atividades

Note-se que as atividades "Detalhamento dos requisitos de interface" e "Detalhamento dos casos de uso" são realizadas conceitualmente em paralelo. Um diagrama de fluxo é apenas um modelo simplificado do que realmente acontece. Para simplificar, não são mostradas setas de realimentação, que mostrariam o retorno a atividades anteriores; em qualquer atividade é possível voltar a qualquer das atividades anteriores para corrigir problemas.

Não são mostradas também setas que representam o consumo de artefatos, mas apenas a produção destes. Em princípio, cada atividade pode consumir artefatos gerados em todas as atividades anteriores. Geralmente é indicado o primeiro nível de divisão de cada artefato que é efetivamente produzido em uma atividade. Por exemplo, a atividade "Definição do escopo" produz a seção "Introdução" da Especificação dos Requisitos do Software, indicada pela sigla ERSw.

3.1.3.5 Distribuição dos esforços

O relacionamento entre fluxos e fases é matricial; a Tabela 17 mostra, para cada coluna, uma distribuição plausível do esforço da fase entre os fluxos. A Tabela 18 mostra, na última linha, uma distribuição plausível do esforço do projeto por cada fase. Estes números hipotéticos levariam ao gráfico da Figura 16, na qual se mostra a variação do esforço despendido em cada fluxo, ao longo de um projeto.

	Concepção	Elaboração	Construção	Transição
Requisitos	80,00%	35,00%	2,00%	1,00%
Análise	15,00%	55,00%	5,00%	2,00%
Desenho	3,00%	5,00%	30,00%	10,00%
Implementação	1,00%	3,00%	45,00%	20,00%
Testes	1,00%	2,00%	18,00%	67,00%
Total do fluxo	100,00%	100,00%	100,00%	100,00%

Tabela 17 – Uma possível distribuição do esforço de cada fase por fluxo

	Concepção	Elaboração	Construção	Transição	Total por fluxo
Requisitos	4,00%	7,00%	1,10%	0,20%	12,30%
Análise	0,75%	11,00%	2,75%	0,40%	14,90%
Desenho	0,15%	1,00%	16,50%	2,00%	19,65%
Implementação	0,05%	0,60%	24,75%	4,00%	29,40%
Testes	0,05%	0,40%	9,90%	13,40%	23,75%
Total por fase	5,00%	20,00%	55,00%	20,00%	100,00%

Tabela 18 - Distribuição do esforço total por fase e fluxo

Estes exemplos são inteiramente hipotéticos, mas representam uma distribuição razoável para um projeto bem conduzido. O fluxo de Requisitos domina a Concepção, quando são levantados os primeiros requisitos, e mantém-se durante a elaboração, quando eles são detalhados e validados. O esforço com os requisitos é pequeno nas fases seguintes, correspondendo apenas a eventuais correções e complementações. O fluxo de Análise é mais importante na Elaboração do que na Concepção, correspondendo nas fases seguintes apenas a eventuais correções. Os fluxos de Desenho e Implementação são pequenos nas fases iniciais, correspondendo apenas à confecção de protótipos; predominam durante a Construção, e na Transição correspondem apenas a correções de problemas. O fluxo de Testes pesa pouco nas duas primeiras fases, correspondendo apenas ao teste de protótipos; ele pesa mais na Construção, e domina a Transição.

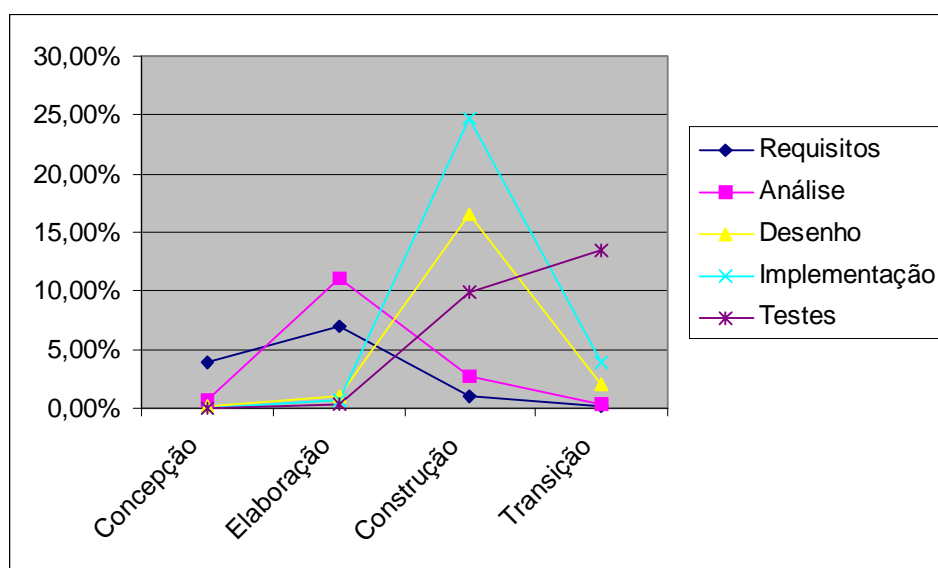


Figura 16 - Gráfico do esforço por fase e fluxo

As fases são divididas em unidades gerenciais menores, chamadas de iterações. Cada iteração é terminada pela produção de um conjunto de resultados. O Praxis pode ser moldado a diferentes ciclos de vida, variando-se o número de iterações por fase. Quando a distribuição dos esforços entre os fluxos e do número de iterações entre as fases é mais balanceada, aproxima-se do modelo em espiral. A distribuição de esforços exemplificada acima é característica de um modelo próximo da entrega evolutiva, que é coerente com as iterações que são detalhadas a seguir.

3.2 Detalhes das fases

3.2.1 *Concepção*

Descrição	Levantamento e análise das necessidades dos usuários e conceitos da aplicação, em nível de detalhe suficiente para justificar a especificação de um produto de software.		
Pré-requisitos	Solicitação de proposta.		
Insumos	(só documentos externos ao projeto).		
Atividades	Fluxo	Tarefas	
	Requisitos	Definição do escopo do produto. Definição dos requisitos (preliminar).	
	Análise	Estudos de viabilidade (opcional).	
	Desenho	Esboço da arquitetura do produto (opcional). Estudos de viabilidade (opcional).	
	Implementação	Prototipagem dos requisitos (opcional).	
	Testes	Testes dos protótipos dos requisitos (opcional).	
	Gestão	Levantamento das metas gerenciais. Estimativas da fase de Elaboração. Elaboração de proposta de especificação.	
Resultados	Artefato	Sigla	Partes
	Proposta de Especificação de Software	PESw	
Critérios de aprovação	Aprovação em revisão gerencial. Aprovação da Proposta de Especificação de Software pelo cliente.		
Normas pertinentes	Padrão de Proposta de Especificação de Software.		

Tabela 19 - Script da Ativação

A iteração de Ativação (Tabela 19) tem por objetivo verificar se o cliente tem necessidades de negócio suficientes para justificar estudos detalhados de especificação de um produto de software, que seriam então realizados na fase de Elaboração. A Ativação é, naturalmente, a menos formal das iterações do processo. Muitas vezes, ela não é cobrada do cliente, sendo assumida pelo fornecedor como investimento de risco. Por isso, deve durar o mínimo necessário para que se faça uma avaliação grosseira do escopo e da viabilidade de uma idéia de produto. Entretanto, deve-se produzir informação suficiente para dimensionar a Elaboração, já que esta geralmente envolve despesas não desprezíveis, quer sejam bancadas pelo cliente ou pelo fornecedor.

As principais atividades da Ativação fazem parte do fluxo de Requisitos: a definição do escopo do produto e o levantamento preliminar dos requisitos. É importante também o fluxo de Gestão, que inclui o levantamento das metas gerenciais (principalmente limites de prazo e custo aceitáveis para o cliente), e as estimativas de custo e prazo da fase de elaboração. Estas estimativas são usadas para produzir uma Proposta de Especificação de Software.

Geralmente esses levantamentos preliminares são conseguidos em poucos dias de negociação com o cliente, mas projetos mais complexos podem requerer estudos de viabilidade e confecção de protótipos até para uma verificação mínima da viabilidade dos conceitos. É geralmente interessante fazer um pequeno esboço da arquitetura do produto, definindo a estrutura deste em poucos blocos, e propondo as tecnologias que são mais fortes candidatas a serem usadas no projeto.

Note-se que a demanda por um projeto de software pode resultar de um outro projeto de maior porte, como um estudo de definição de novo produto, um desenho de um sistema informatizado complexo,

um projeto piloto de avaliação de tecnologia, ou mesmo de solicitações de melhorias e modificações de uma versão anterior do produto. Nesse caso, a iteração de Ativação faz a ponte entre a origem da demanda e o novo projeto.

A Ativação deve identificar todas as possíveis partes interessadas no produto a ser especificado. O cliente deve se comprometer a liberar representantes autorizados de todos os grupos de usuários para participar do trabalho de Elaboração.

3.2.2 Elaboração

3.2.2.1 Levantamento dos Requisitos

Descrição	Levantamento detalhado das funções, interfaces e requisitos não funcionais desejados para o produto.		
Pré-requisitos	Ativação terminada.		
Insumos	PESw.		
Atividades	Fluxo	Tarefas	
	Requisitos	Levantamento completo dos requisitos. Detalhamento das interfaces. Detalhamento dos casos de uso. Detalhamento dos requisitos não funcionais.	
	Análise	Estudos de viabilidade (opcional).	
	Desenho	Estudos de viabilidade (opcional).	
	Implementação	Prototipagem dos requisitos (opcional).	
	Testes	Testes dos protótipos dos requisitos (opcional).	
	Gestão	Cadastramento dos requisitos.	
Resultados	Artefato	Sigla	Partes adicionadas
	Modelo de Análise do Software	MASw	Visão de casos de uso.
	Especificação dos Requisitos do Software	ERSw	Corpo.
	Cadastro de Requisitos do Software	CRSw	Interfaces, casos de uso e requisitos não funcionais.
Crítérios de aprovação	Aprovação em revisão gerencial.		
Normas pertinentes	Padrão para Especificação de Requisitos de Software.		

Tabela 20 - Script do Levantamento dos Requisitos

A fase de Elaboração é iniciada depois que o cliente aprova a Proposta de Especificação. Ela contém duas iterações:

- o Levantamento dos Requisitos visa à captura das necessidades dos usuários em relação ao produto, expressas na linguagem desses usuários;
- a Análise dos Requisitos confecciona um modelo conceitual do produto, que é usado para validar os requisitos levantados e para planejar o desenvolvimento posterior.

Durante o Levantamento dos Requisitos (Tabela 20), os requisitos devem ser levantados em nível tão detalhado quanto necessário para que cliente, usuários e desenvolvedores se ponham de acordo quanto a eles. Os requisitos já mencionados na Proposta de Especificação são revisados, sendo geralmente ampliados devido à participação de um número maior de partes interessadas. Para maior eficácia na

captura de requisitos, o processo recomenda que esta seja feita através de oficinas (*workshops*) estruturadas, com participação ativa e intensiva de todas as partes interessadas. Protótipos e estudos de viabilidade são feitos quando necessário. Por exemplo, protótipos rápidos e rascunhos de papel podem ajudar a definir os requisitos das interfaces de usuário.

Ao final do Levantamento dos Requisitos, o corpo da Especificação de Requisitos deve estar pronto. Os requisitos funcionais são descritos através de casos de uso, que formam a primeira visão do Modelo de Análise. As interfaces de usuário do produto são esboçadas apenas o suficiente para definir os respectivos requisitos, evitando-se entrar em detalhes de desenho. Os casos de uso devem ser expressos em termos de ações pertinentes ao domínio do problema, e não de detalhes das interfaces.

Geralmente, uma revisão gerencial é suficiente para fechar essa iteração, pois é preferível realizar uma revisão técnica formal apenas quando de posse do Modelo de Análise. Os requisitos levantados são lançados em um Cadastro dos Requisitos, que posteriormente amarrará os requisitos com os respectivos elementos derivados nos demais fluxos. Os requisitos cadastrados devem ser de alta qualidade: corretos, precisos, completos, consistentes, verificáveis e modificáveis, com prioridades relativas bem definidas.

3.2.2.2 Análise dos Requisitos

Enquanto o Levantamento dos Requisitos focaliza a visão que cliente e usuários têm dos requisitos de um produto, a Análise dos Requisitos focaliza a visão dos desenvolvedores. Entretanto, o processo ainda está dentro do espaço de problema, e não dentro do espaço de soluções. O Modelo de Análise usa a notação orientada a objetos para descrever de forma mais precisa os conceitos do domínio da aplicação que sejam relevantes para o entendimento detalhado dos requisitos do produto. Essa notação é usada como notação de modelagem do problema, independentemente do uso posterior da tecnologia orientada a objetos para desenho e implementação.

As atividades iniciais de Análise dos Requisitos levam à identificação de classes que representem adequadamente os conceitos expressos nos requisitos, e à descoberta dos respectivos atributos e relacionamentos. Essa parte da análise fornece o modelo lógico de dados (equivalente a um modelo de entidades e relacionamentos), que pode corresponder ao modelo conceitual de um banco de dados usado pelo produto. Essas classes são incluídas no Cadastro dos Requisitos.

São então detalhadas as responsabilidades de cada classe, definindo-se as respectivas operações. Estas operações são usadas para produzir as realizações dos casos de uso, nas quais os fluxos dos casos de uso são descritos em termos de interações entre as classes identificadas. Geralmente o detalhamento das realizações dos casos de uso leva à descoberta de ambigüidades, omissões e inconsistências nos requisitos funcionais, contribuindo para o aperfeiçoamento destes. Protótipos e estudos de viabilidade podem complementar a análise; por exemplo, um protótipo implementado em um sistema de bancos de dados pode ajudar a decidir se é viável atender aos requisitos de desempenho especificados para transações com bancos de dados.

Provavelmente, as atividades gerenciais mais importantes são as que acontecem nessa iteração. A Especificação dos Requisitos agora se torna confiável o suficiente para servir de base ao planejamento detalhado do restante do projeto, permitindo confeccionar uma proposta da fase de Construção, com prazos e orçamentos firmes. Essa proposta é refletida em um Plano de Desenvolvimento. Além disso, existe informação suficiente para planejar as atividades de um grupo de garantia da qualidade, expressas dentro de um Plano da Qualidade.

Descrição	Modelagem conceitual dos elementos relevantes do domínio do problema e uso deste modelo para validação dos requisitos e planejamento detalhado da fase de Construção.		
Pré-requisitos	Levantamento dos requisitos terminado.		
Insumos	Antigos		Novos
	PESw.		ERSw - corpo; MASw - visão de casos de uso.
Atividades	Fluxo	Tarefas	
	Requisitos	Revisão e modificação dos requisitos (se necessário).	
	Análise	Identificação das classe. Identificação dos atributos e relacionamentos. Realização dos casos de uso. Revisão e iteração.	
	Desenho	Estudos de viabilidade (opcional). Desenho arquitetônico.	
	Implementação	Prototipagem dos requisitos (opcional).	
	Testes	Testes dos protótipos dos requisitos (opcional).	
	Gestão	Cadastramento dos itens de análise no cadastro de requisitos. Planejamento do desenvolvimento. Planejamento da qualidade.	
Resultados	Artefato	Sigla	Partes adicionadas
	Modelo de Análise do Software	MASw	Visão lógica.
	Especificação dos Requisitos do Software	ERSw	Anexo – listagem do modelo de análise.
	Cadastro de Requisitos do Software	CRSw	Classes.
	Memória de Cálculo do Projeto do Software	MCPSw	Total.
	Modelo de Planejamento do Projeto do Software	MPPSw	Total.
	Plano de Desenvolvimento do Software	PDSw	Total.
	Plano da Qualidade do Software	PQSw	Total (menos detalhes de implementação).
Crítérios de aprovação	Aprovação em revisão técnica. Aprovação em auditoria da qualidade. Aprovação em revisão gerencial. Aprovação da Especificação dos Requisitos do Software e do Plano de Desenvolvimento do Software pelo cliente.		
Normas pertinentes	Padrão para Especificação de Requisitos de Software. Padrão para Plano de Desenvolvimento de Software. Padrão para Plano da Qualidade de Software.		

Tabela 21 - Script da Análise dos Requisitos

O fechamento desta iteração é de enorme responsabilidade, pois a partir daqui o fornecedor estará possivelmente assumindo compromissos de grande monta e colocando a sua reputação em jogo. Por isto, o processo determina a realização dos seguintes procedimentos de controle:

- uma revisão técnica formal, na qual um grupo de revisores independentes (não pertencentes à equipe responsável pela Elaboração), composto de desenvolvedores e outros especialistas, verifica a qualidade técnica da Especificação;
- uma auditoria da qualidade, na qual um grupo de garantia da qualidade verifica se essa fase foi conduzida conforme determinado pelo processo;
- uma revisão gerencial da equipe do fornecedor responsável pela Elaboração, que verifica se os desenvolvedores concordam que os compromissos a serem assumidos com o cliente são factíveis, e em que se faz um balanço desta iteração.

Feitas essas revisões, o fornecedor emitirá uma proposta de desenvolvimento, indicando os compromissos de prazos e custos que serão assumidos durante a Construção e, possivelmente, durante a Transição. Essa proposta encaminhará a Especificação de Requisitos, e conterá a informação do Plano de Desenvolvimento, no todo ou em parte, conforme o relacionamento combinado entre cliente e fornecedor. A palavra final quanto ao prosseguimento ou não do projeto será dada pelo cliente, com base nesses documentos.

3.2.3 Construção

3.2.3.1 Desenho Inicial

O Desenho Inicial é o principal passo da Construção (Tabela 22). Um desenho bem feito resolve os seguintes problemas:

- produz uma arquitetura robusta, estável e flexível, que é indispensável para que o produto tenha alta qualidade e vida longa;
- produz interfaces de usuário que tornem o produto fácil de aprender e de usar de maneira produtiva;
- serve de base para o planejamento e desenho precisos dos testes de aceitação;
- escolhe as soluções tecnológicas mais adequadas para satisfazer os requisitos de forma rápida, barata e confiável;
- identifica componentes comerciais ou de projetos anteriores que possam ser reutilizados, reduzindo o esforço de desenvolvimento;
- decide as questões técnicas importantes para a implementação, deixando para a confecção das liberações apenas a realização de detalhes;
- divide adequadamente o produto em componentes que possam ser agrupados em um número satisfatório de liberações, permitindo a avaliação precoce por parte dos usuários;
- divide adequadamente o produto em componentes cuja implementação possa ser dividida eficazmente entre os membros de uma equipe de desenvolvedores, diminuindo os prazos de implementação.

A atividade inicial de desenho é o desenho da arquitetura, que é a estrutura arquitetônica principal do produto, dividindo-o em camadas, pacotes lógicos e subsistemas. Essa atividade envolve decisões técnicas estratégicas, identificando-se tecnologias adequadas para a implementação dos subsistemas e

localizando componentes externos reutilizáveis. O desenho dos subsistemas determina os aspectos principais dos componentes maiores do produto e as interfaces entre estes.

O desenho das interfaces de usuário ataca os métodos necessários para satisfazer os requisitos de usabilidade. Dependendo do peso que estes requisitos tenham na aceitação do produto, pode caber aqui a realização de toda uma bateria de estudos e experimentos de usabilidade. Desenhadas as interfaces, os casos de uso devem ser detalhados em termos dos elementos reais das interfaces. O tratamento de erros de usuário e outras exceções deve ser completamente definido. Geralmente o nível de detalhe das interfaces e dos casos de uso exigirá a modelagem delas através de diagramas de estado. Finalmente, devem ser refeitas as realizações dos casos de uso, em termos das classes de desenho encontradas. Os casos de uso detalhados fornecem os elementos necessários para planejar e desenhar os testes de aceitação.

O desenho dos dados persistentes identifica as soluções mais adequadas para o acoplamento entre o modelo interno de desenho, que é orientado a objetos, e a arquitetura de sistemas externos de armazenamento de dados persistentes, como os bancos de dados relacionais. Estas questões são fundamentais para se atingirem alguns dos mais importantes requisitos funcionais, como os requisitos de desempenho, confiabilidade, portabilidade e manutenibilidade.

O refinamento das classes de desenho vai ocorrendo durante essas atividades, aumentando bastante o tamanho do modelo de desenho, se comparado com o modelo de análise. Protótipos podem ser usados para resolver problemas específicos, em geral ligados aos requisitos não funcionais. Geralmente são descobertas algumas falhas na especificação de requisitos e no modelo de análise, que devem ser corrigidas.

O Desenho Inicial termina quando o produto foi repartido em um conjunto de subsistemas com interfaces bem definidas entre si, estando bem entendida a maneira como os casos de uso serão realizados. Pode-se então planejar as liberações, definindo-se os subsistemas e casos de uso que cada uma implementará. Os planos de desenvolvimento e da qualidade devem ser revistos para cobrir os detalhes de recursos, custos e prazos dessas liberações.

Os procedimentos de controle do Desenho Inicial compreendem:

- aprovação dos aspectos técnicos do desenho, registrados na Descrição do Desenho do Software, em revisão técnica;
- aprovação do desenho das interfaces de usuário (documentado na Descrição do Desenho do Software) pelos usuários chaves;
- aprovação da conformidade com o processo, em auditoria da qualidade;
- aprovação do gerente do projeto e da equipe em revisão gerencial, em que é feito o balanço da iteração.

Descrição	Definição interna e externa dos componentes de um produto de software, em nível suficiente para decidir as principais questões de arquitetura e tecnologia, e para permitir o planejamento detalhado das atividades de implementação.		
Pré-requisitos	Elaboração terminada.		
Insumos	Antigos		Novos
	MASw; ERSw; CRSw		MCPSw; MPPSw; PDSw; PQSw.
Atividades	Fluxo	Tarefas	
	Requisitos	Revisão e modificação dos requisitos (se necessário).	
	Análise	Revisão e modificação do modelo de análise (se necessário).	
	Desenho	Desenho dos subsistemas. Desenho do acesso a dados persistentes. Desenho das interfaces de usuário. Elaboração dos casos de uso de desenho. Desenho das liberações.	
	Implementação	Prototipagem de problemas de desenho (opcional).	
	Testes	Planejamento e desenho dos testes de aceitação.	
	Gestão	Cadastramento dos itens de teste no cadastro de requisitos. Cadastramento dos itens de desenho no cadastro de requisitos. Planejamento detalhado das liberações. Atualização dos planos de desenvolvimento e da qualidade.	
Resultados	Artefato	Sigla	Partes
	Insumos		Eventuais modificações e detalhes adicionais.
	Modelo de Desenho do Software	MDSw	Todas as visões, em descrição de alto nível.
	Descrição do Desenho do Software	DDSw	Partes 1 a 4, com colocação no anexo das partes já produzidas do MDSw
	Descrição dos Testes do Software	DTSw	Planos e especificações dos testes de aceitação.
	Bateria de Testes de Regressão do Software	BTRSw	Scripts para automação dos testes de aceitação (opcional).
Crítérios de aprovação	Aprovação em revisão técnica. Aprovação do desenho das interfaces de usuário pelos usuários chaves. Aprovação em auditoria da qualidade. Aprovação em revisão gerencial.		
Normas pertinentes	Padrão para Descrição de Desenho de Software Padrão para Documentação de Testes de Software		

Tabela 22 - Script do Desenho Inicial

3.2.3.2 Liberação

A Tabela 23 apresenta o script de uma Liberação típica. O desenho detalhado resolve os aspectos faltantes das unidades que serão implementadas nessa liberação, como nomes de todas as unidades, assinaturas e visibilidade das operações, implementação dos relacionamentos, questões de escopo,

tratamento de erros e exceções, algoritmos e estruturas de dados. A lógica de cada unidade é detalhada através de máquinas de estados, pseudolinguagem e outros meios.

Descrição	Implementação de um subconjunto de funções do produto que será avaliado pelos usuários.		
Pré-requisitos	Desenho Inicial terminado. Liberação anterior terminada e avaliada pelos usuários.		
Insumos	Antigos		Novos
	MASw; ERSw; CRSw; MCPSw; MPPSw; PDSw; PQSw; MDSw; DDSw; DTSw; BTRSw		CFSw (liberação anterior); CESw (liberação anterior); RTSw (liberação anterior).
Atividades	Fluxo	Tarefas	
	Requisitos	Revisão e modificação dos requisitos (se necessário).	
	Análise	Revisão e modificação do modelo de análise (se necessário).	
	Desenho	Revisão e modificação do modelo de desenho (se necessário).	
	Implementação	Desenho detalhado dos componentes desta liberação. Codificação dos componentes desta liberação. Compilação dos componentes desta liberação.	
	Testes	Planejamento e desenho dos testes de integração da liberação. Planejamento e desenho dos testes de unidade da liberação. Realização dos testes de unidade da liberação. Realização dos testes de integração da liberação.	
	Gestão	Revisão e modificação dos planos de desenvolvimento e da qualidade (se necessário).	
Resultados	Artefato	Sigla	Partes
	Insumos		Eventuais modificações e detalhes adicionais.
	Relatórios dos Testes do Software	RTSw	Relatórios dos testes de unidade e de integração da liberação (opcional).
	Códigos Fontes do Software	CFSw	Unidades da liberação. Estruturas provisórias de teste.
	Códigos Executáveis do Software	CESw	Unidades da liberação. Estruturas provisórias de teste.
CrITÉRIOS de aprovação	Aprovação em inspeção do desenho detalhado e código da liberação. Aprovação da liberação pelos usuários chaves. Aprovação em auditoria da qualidade. Aprovação em revisão gerencial.		
Normas pertinentes	Padrão para Descrição de Desenho de Software Padrão para Documentação de Testes de Software Padrão para Desenho Detalhado e Codificação de Software.		

Tabela 23 - Script da Liberação n

Os testes de integração são planejados e desenhados, geralmente aproveitando-se subconjuntos adequados dos testes de aceitação. A definição das interfaces e da lógica interna das unidades permite o desenho dos respectivos testes. O conjunto do desenho detalhado é submetido a uma inspeção.

Durante a codificação, o desenho detalhado é convertido em código executável das linguagens de implementação. O código é inspecionado, verificando-se sua conformidade tanto com o desenho

detalhado quanto com o padrão de codificação. Em seguida, o código novo é compilado e os testes de unidade são executados, possivelmente com o auxílio de estruturas provisórias de teste.

Na integração da liberação, o código novo é ligado com os componentes produzidos nas liberações anteriores e com os componentes externos necessários. O conjunto é submetido aos testes de integração. Uma vez executável, a liberação é submetida à avaliação dos usuários. Geralmente são descobertos alguns problemas, que levam a refazer alguns aspectos do desenho, e possivelmente da análise e dos requisitos.

Os procedimentos de controle cada liberação compreendem:

- inspeção do desenho detalhado e do código das novas unidades da liberação;
- aprovação da liberação pelos usuários chaves;
- aprovação da conformidade com o processo e dos resultados dos testes, em Auditoria da Qualidade;
- aprovação do gerente do projeto e da equipe em Revisão Gerencial, onde é feito o balanço da iteração.

3.2.3.3 Testes Alfa

A iteração dos Testes Alfa (Tabela 24) fecha a Construção. Os testes de aceitação são realizados no ambiente do fornecedor, embora seja recomendável que isso seja feito por uma equipe especializada em testes, independentemente dos desenvolvedores. Durante os testes, são identificados problemas remanescentes. Se os problemas identificados corresponderem a defeitos de desenho ou de requisitos, as correções necessárias devem ser feitas não só no código mas em toda a cadeia de artefatos afetada. O Cadastro de Requisitos, se adequadamente mantido, permitirá identificar quais os itens afetados dessa cadeia.

Nessa iteração é elaborado o material de suporte à Transição. O principal item deste é o Manual do Usuário, no qual os procedimentos dos usuários podem ser descritos a partir dos casos de uso detalhados e outros aspectos do desenho das interfaces de usuário. A documentação de usuário pode incluir outros itens de treinamento, promoção e suporte, como demonstrações e sítios de apoio. Além disso, se a Transição envolver um grande número de instalações ou procedimentos complexos, é recomendável acrescentar ao plano de desenvolvimento um plano de transição.

No encerramento dos testes alfa não são previstas revisões técnicas específicas, mas apenas uma auditoria da qualidade, que checará inclusive os resultados dos testes de aceitação, documentos nos respectivos relatórios. O produto é então entregue ao cliente para implantação nas instalações da fase de Transição.

Descrição	Realização dos testes de aceitação, no ambiente dos desenvolvedores, juntamente com elaboração da documentação de usuário e possíveis planos de Transição.		
Pré-requisitos	Última liberação terminada e avaliada pelos usuários.		
Insumos	Antigos		Novos
	MASw; ERSw; CRSw; MCPSw; MPPSw; PDSw; PQSw; MDSw; DDSw; DTSw; BTRSw.		CFSw última (liberação); CESw (última liberação); RTSw (última liberação).
Atividades	Fluxo	Tarefas	
	Requisitos	Revisão e modificação dos requisitos (se necessário).	
	Análise	Revisão e modificação do modelo de análise (se necessário).	
	Desenho	Revisão e modificação do desenho de alto nível (se necessário).	
	Implementação	Revisão e modificação do desenho detalhado e código (se necessário). Produção da documentação de usuário.	
	Testes	Realização dos testes alfa (aceitação no ambiente dos desenvolvedores).	
	Gestão	Planejamento detalhado da Transição. Atualização dos planos de desenvolvimento e da qualidade.	
Resultados	Artefato	Sigla	Partes
	Insumos		Eventuais modificações e detalhes adicionais.
	Relatórios dos Testes do Software	RTSw	Relatórios dos testes alfa.
	Manual do Usuário do Software	MUSw	
Crítérios de aprovação	Aprovação em auditoria da qualidade. Aprovação em revisão gerencial. Aprovação da entrega do produto pelo cliente.		
Normas pertinentes	Padrão para Documentação de Testes de Software Padrão para Documentação de Usuário de Software		

Tabela 24 - Script dos Testes Alfa

3.2.4 Transição

3.2.4.1 Testes Beta

Nos Testes Beta (Tabela 25), os testes de aceitação são repetidos nas instalações dos usuários. Problemas relacionados com o funcionamento em instalações reais são identificados e resolvidos. A documentação de usuário também é testada. A aprovação da iteração inclui o controle pelo fornecedor (através de Auditoria da Qualidade e Revisão Gerencial) e pelo cliente (que, ao aprovar os Testes Beta, autoriza o início da Operação Piloto).

Descrição	Realização dos testes de aceitação, no ambiente dos usuários.		
Pré-requisitos	Construção terminada. Aceitação da instalação do produto pelo cliente.		
Insumos	Antigos		Novos
	MASw; ERSw; CRSw; MCPSw; MPPSw; PDSw; PQSw; MDSw; DDSw; DTSw; BTRSw.		RTSw (testes alfa); MUSw.
Atividades	Fluxo	Tarefas	
	Requisitos	Revisão e modificação dos requisitos (se necessário).	
	Análise	Revisão e modificação do modelo de análise (se necessário).	
	Desenho	Revisão e modificação do desenho de alto nível (se necessário).	
	Implementação	Revisão e modificação do desenho detalhado e código (se necessário). Revisão e modificação da documentação de usuário (se necessário).	
	Testes	Realização dos testes beta (aceitação no ambiente dos usuários).	
	Gestão	Revisão e modificação dos planos de desenvolvimento e da qualidade (se necessário).	
Resultados	Artefato	Sigla	Partes
	Insumos		Eventuais modificações e detalhes adicionais.
	Relatórios dos Testes do Software	RTSw	Relatórios dos testes beta.
Crítérios de aprovação	Aprovação em auditoria da qualidade. Aprovação em revisão gerencial. Aprovação dos testes beta pelo cliente.		
Normas pertinentes	Padrão para Documentação de Testes de Software		

Tabela 25 - Script dos Testes Beta

3.2.4.2 Operação Piloto

A Operação Piloto (Tabela 26) representa um estado de "liberdade vigiada" do produto. Em sistemas de missão crítica, essa iteração será realizada inicialmente em instalações escolhidas como pilotos, tomando-se os devidos cuidados em relação a backups, segurança e treinamento dos operadores. Os problemas encontrados pelos usuários passam a ser tratados pelo processo de manutenção do produto, permitindo-se avaliar a qualidade dos serviços de manutenção e suporte ao usuário. Os registros de defeitos permitem uma primeira avaliação da qualidade final do produto.

Durante a Operação Piloto é feita uma análise post-mortem do projeto, focalizando os problemas de processo encontrados. Esse balanço é consolidado em um Relatório Final do Projeto, onde são resumidas as métricas importantes coletadas no projeto e as lições que possam levar à melhoria do processo em projetos futuros.

Descrição	Operação experimental do produto em instalação piloto do cliente, com a resolução de eventuais problemas através de processo de manutenção.		
Pré-requisitos	Testes beta terminados e aprovados pelo cliente.		
Insumos	Antigos		Novos
	MASw; ERSw; CRSw; MCPSw; MPPSw; PDSw; PQSw; MDSw; DDSw; DTSw; BTRSw; MUSw.		RTSw (testes beta).
Atividades	Fluxo	Tarefas	
	Requisitos	Revisão e modificação dos requisitos (se necessário).	
	Análise	Revisão e modificação do modelo de análise (se necessário).	
	Desenho	Revisão e modificação do desenho de alto nível (se necessário).	
	Implementação	Revisão e modificação do desenho detalhado e código (se necessário). Revisão e modificação da documentação de usuário (se necessário).	
	Testes	Revisão e modificação da documentação de testes (se necessário).	
	Gestão	Balanço final do projeto. Produção do Relatório Final do Projeto.	
Resultados	Artefato	Sigla	Partes
	Insumos		Eventuais modificações e detalhes adicionais.
	Relatório Final de Projeto de Software	RFPSw	
Crítérios de aprovação	Aprovação em Auditoria da Qualidade. Aprovação em Revisão Gerencial. Aceitação final do produto pelo cliente.		
Normas pertinentes	Padrão para Documentação de Testes de Software. Padrão para Relatório Final de Projeto de Software. Padrão para Processo de Manutenção de Software.		

Tabela 26 - Script da Operação Piloto

3.3 Artefatos

Tipo de artefato	Descrição
Modelo	Artefato de um ferramenta técnica específica, produzido e usado nas atividades de um dos fluxos do processo.
Documento	Artefato produzido por ferramenta de processamento de texto ou hipertexto, que pode ser consultado on-line ou em forma impressa, para fins de referência ou revisão.

Tabela 27 - Tipos de artefato do Praxis

Os resultados produzidos e insumos consumidos nos passos do Praxis são chamados de artefatos do processo. Como mostra a Tabela 27, os artefatos podem ser documentos e modelos, conforme seus consumidores primários sejam humanos ou ferramentas.

Os artefatos podem ser também permanentes ou transitórios em relação ao processo. Os artefatos permanentes são atualizados a cada iteração do processo, de acordo com procedimentos de gestão de configurações. Um conjunto de artefatos associados a um marco do projeto, consistentes entre si e conformes com os padrões do processo, constitui uma **linha de base** do projeto. As linhas de base são tipicamente montadas ao final de cada iteração, preservando-se assim um retrato completo do projeto em cada um desses instantes. Isso é muito importante para facilitar a localização posterior de defeitos.

Nome	Sigla	Descrição
Proposta de Especificação do Software	PESw	Documento que delimita preliminarmente o escopo de um projeto, contendo um plano da fase de Elaboração.
Especificação dos Requisitos do Software	ERSw	Documento que descreve, de forma detalhada, o conjunto de requisitos especificados para um produto de software.
Plano de Desenvolvimento do Software	PDSw	Documento que descreve, de forma detalhada, os compromissos que o fornecedor assume em relação ao projeto, quanto a recursos, custos, prazos, riscos e outros aspectos gerenciais.
Plano da Qualidade do Software	PQSw	Documento que descreve, de forma detalhada, os procedimentos de garantia da qualidade que serão adotados no projeto.
Descrição do Desenho do Software	DDSw	Documento que descreve, de forma detalhada, os aspectos mais importantes do desenho do software.
Descrição dos Testes do Software	DTSw	Documento que descreve, de forma detalhada, os planos e especificações dos testes que serão executados
Manual do Usuário do Software	MUSw	Documento que serve de referência para uso do produto.

Tabela 28 - Documentos permanentes do Praxis

A Tabela 28 apresenta os documentos permanentes oficiais do Praxis. Os dois planos são considerados documentos gerenciais, e os demais são documentos técnicos. Tipicamente, eles são produzidos através uma ferramenta de edição de textos. O formato desses documentos é conforme com os padrões do IEEE [IEEE94]. Esses padrões requerem documentos bastante detalhados, mas o processo inclui modelos que facilitam o seu preenchimento. Além disso, em alguns modelos são preenchidas previamente as partes que não variam muito entre projetos conformes com esse processo; assim, só as exceções precisam ser documentadas.

Nome	Sigla	Descrição	Ferramentas aplicáveis
Cadastro dos Requisitos do Software	CRSw	Modelo que contém os requisitos levantados, assim como referências aos itens correspondentes dos modelos seguintes.	Planilha, banco de dados
Modelo de Análise do Software	MASw	Modelo que detalha os conceitos do domínio do problema a resolver que sejam relevantes para a validação dos requisitos.	Ferramenta de modelagem orientada a objetos
Memória de Planejamento do Projeto do Software	MPPSw	Modelo que contém a informação necessária para o planejamento e acompanhamento de tamanhos, esforços, custos, prazos e riscos do projeto.	Planilha, ferramenta de gestão de projetos
Modelo de Desenho do Software	MDSw	Modelo que detalha a estrutura lógica e física do produto, em termos de seus componentes.	Ferramenta de modelagem orientada a objetos
Bateria de Testes de Regressão do Software	BTRSw	Conjunto dos scripts dos testes de regressão.	Ferramenta de desenvolvimento, ferramenta de testes
Códigos Fontes do Software	CFSw	Conjunto dos códigos fontes produzidos.	Ferramenta de desenvolvimento
Códigos Executáveis do Software	CESw	Conjunto dos códigos executáveis produzidos.	Ferramenta de desenvolvimento

Tabela 29 - Modelos permanentes do Praxis

A Tabela 29 apresenta os modelos permanentes do Praxis. A Memória de Planejamento do Projeto do Software é o único modelo gerencial. A última coluna indica o tipo de ferramenta necessário para processamento de cada modelo. Em alguns casos, são apresentadas alternativas de menor ou maior sofisticação tecnológica.

Nome	Sigla	Descrição
Relatórios dos Testes do Software	RTSw	Conjunto dos relatórios que descrevem os resultados dos testes realizados.
Relatórios de Revisão do Software	RRSw	Conjunto dos relatórios que descrevem as conclusões das revisões realizadas.
Relatórios das Auditorias da Qualidade do Software	RAQSw	Conjunto dos relatórios que descrevem as conclusões das auditorias da qualidade realizadas.
Relatórios de Acompanhamento do Projeto do Software	RAPSw	Conjunto dos relatórios de acompanhamento do projeto, que relatam esforços, custos, prazos e riscos do período relatado, comparados com o que foi planejado.
Relatório Final do Projeto do Software	RFPSw	Relatório de balanço final do projeto.

Tabela 30 - Relatórios do Praxis

A Tabela 30 apresenta os relatórios do Praxis. Os dois primeiros são de caráter técnico, e os demais de caráter gerencial. O plano da qualidade prevê as datas de emissão dos relatórios de testes, revisões e auditorias. Os relatórios de acompanhamento são produzidos com a periodicidade especificada no plano de desenvolvimento (geralmente mensal).

	PESw	ERSw	PDSw	PQSw	DDSw	DTSw	MUSw
Ativação	P						
Levantamento dos Requisitos		P					
Análise dos Requisitos		C	P	P			
Desenho Inicial		A	A	A	P	P	
Liberação ...		A	A	A	C	C	
Testes Alfa		A	A	A	A	A	P
Testes Beta		A	A	A	A	A	A
Operação Piloto		A	A	A	A	A	A

Tabela 31 - Relacionamento entre iterações e documentos

	CRSw	MASw	MPPSw	MDSw	BTRSw	CFSw	CESw
Ativação							
Levantamento dos Requisitos	P						
Análise dos Requisitos	C	P	P				
Desenho Inicial	C	A	A	P	P		
Liberação ...	C	A	A	C	C	P	P
Testes Alfa	A	A	A	A	A	A	A
Testes Beta	A	A	A	A	A	A	A
Operação Piloto	A	A	A	A	A	A	A

Tabela 32 - Relacionamento entre iterações e modelos

A Tabela 31 e a Tabela 32 mostram o relacionamento entre iterações e artefatos. Um P indica a iteração inicial em que um artefato é produzido; um C indica que o artefato é normalmente completado nessa iteração, e um A indica que o artefato pode ser alterado na iteração. A partir da Análise dos Requisitos, cada linha indica o conteúdo da linha de base de saída da iteração.

3.4 Procedimentos de controle

Os procedimentos de controle são executados de maneira uniforme, em diferentes iterações do ciclo de vida. A conclusão desses procedimentos é condição necessária para que uma iteração de um projeto seja considerada como aprovada, passando-se à iteração seguinte. Maiores detalhes sobre os diversos tipos de revisões são fornecidos no Padrão para Revisões.

As **revisões técnicas** são o principal meio de controle da qualidade quanto aos aspectos técnicos; no caso das liberações, usa-se para a revisão de desenho detalhado e código a Inspeção, que é um procedimento um pouco mais rigoroso. Os pontos para realização das Revisões Técnicas foram definidos procurando-se equilibrar os vários aspectos envolvidos, como o volume de material submetido à revisão, o risco a que estão expostas as atividades subsequentes e o custo das próprias revisões.

Na **revisão gerencial** o gerente do projeto determina se a atividade pode ser dada por concluída, ouvindo, obrigatoriamente, os membros da equipe do projeto envolvidos na atividade, ou que possam ser afetados por ela. Em caso negativo, o gerente do projeto solicita à equipe do projeto que refaça a atividade. Em caso positivo, o gerente do projeto conduz um balanço das atividades da iteração e toma as providências para a passagem à próxima iteração. O balanço tem por objetivo determinar que lições

importantes foram aprendidas; estas podem servir de base para a melhoria do processo em projetos futuros.

As auditorias da qualidade são tipicamente feitas por um grupo independente de Garantia da Qualidade. Este grupo checa principalmente a conformidade das atividades realizadas com os padrões determinados pelo processo. Ele não realiza diretamente as revisões técnicas, inspeções e testes de aceitação, mas verifica os relatórios desses procedimentos. Outras verificações típicas dessas auditorias são a conformidade com os procedimentos de gestão de configurações, a coerência entre os diversos documentos do processo e a rastreabilidade entre os requisitos e os demais artefatos.

Algumas iterações requerem aprovação por parte do cliente ou dos usuários chaves. As aprovações do cliente geralmente são necessárias em pontos que envolvem decisões de continuar ou não o projeto (fim da Concepção e Elaboração) ou aceitação formal do produto (fim da Construção e Transição). As avaliações pelos usuários chaves geralmente são feitas quando é necessário verificar se, em determinado estágio de construção, o produto atende às necessidades dos usuários. Nessas últimas avaliações, visa-se a principalmente determinar se os requisitos foram corretamente interpretados pelos desenvolvedores.

Métodos técnicos

Página em branco

Requisitos

1 Princípios

1.1 Visão geral

O fluxo de Requisitos reúne as atividades que visam a obter o enunciado completo, claro e preciso dos requisitos de um produto de software. Estes requisitos devem ser levantados pela equipe do projeto, em conjunto com representantes do cliente, usuários-chaves e outros especialistas da área de aplicação. O conjunto de técnicas empregadas para levantar, detalhar, documentar e validar os requisitos de um produto forma a Engenharia de Requisitos. O resultado principal do fluxo dos requisitos é um documento de Especificação de Requisitos de Software (que abreviaremos ERSw).

Projetos de produtos mais complexos geralmente precisam de maior investimento em Engenharia de Requisitos do que projetos de produtos mais simples. A Engenharia de Requisitos é também mais complexa no caso de produtos novos. Quando um projeto visa a desenvolver uma nova versão de um produto existente, a experiência dos usuários com as versões anteriores permite identificar de forma rápida e clara as necessidades prioritárias. No caso de um novo produto, é mais difícil para os usuários identificar quais as características de maior valor, e é mais difícil para os desenvolvedores entender claramente o que os usuários desejam.

Uma boa Engenharia de Requisitos é um passo essencial para o desenvolvimento de um bom produto, em qualquer caso. Este capítulo descreve de forma detalhada as atividades do fluxo de Requisitos do Praxis, assim como algumas das técnicas mais importantes para a obtenção de requisitos de alta qualidade. Para garantir ainda mais a qualidade, os requisitos devem ser submetidos aos procedimentos de controle previstos no processo, e devem ser verificados através das atividades de Análise.

Requisitos de alta qualidade são claros, completos, sem ambigüidade, implementáveis, consistentes e testáveis. Os requisitos que não apresentem estas qualidades são problemáticos: eles devem ser revistos e renegociados com os clientes e usuários.

Tipo	Nome	Sigla
Documentos	Proposta de Especificação do Software	PESw
	Especificação dos Requisitos do Software	ERSw
Modelos	Cadastro dos Requisitos do Software	CRSw
	Modelo de Análise do Software	MASw

Tabela 33 – Artefatos de Requisitos

No Praxis, os requisitos estão contidos nos artefatos enumerados na Tabela 33. A Proposta de Especificação do Software contém uma visão preliminar dos requisitos, que será usada apenas para iniciar o fluxo de Requisitos, e não será mantida dentro das linhas de base do projeto. Os demais artefatos fazem parte das linhas de base. O enunciado detalhado dos requisitos estará contido na Especificação dos Requisitos do Software. O modelo dos casos de uso, que é parte da descrição dos requisitos funcionais, estará contido no Modelo de Análise do Software. O Cadastro dos Requisitos do Software é a base de dados que contém uma lista resumida de todos os requisitos e dos relacionamentos destes com itens derivados, gerados pelos demais fluxos do processo.

As principais referências deste capítulo, quanto aos métodos e técnicas de Engenharia de Requisitos, são [Davis93], [Gause+89], [Jones94], [McConnell96] e [Robertson+99]. Quanto ao modelo de casos de uso, especificamente, as principais referências são: [Booch+99], [Jacobson94], [Jacobson+99], [Quatrani98], [Rumbaugh+99] e [Schneider98].

1.2 A Especificação dos Requisitos do Software

1.2.1 *Natureza*

A Especificação dos Requisitos do Software é o documento oficial de descrição dos requisitos de um projeto de software. Ela pode se referir a um produto indivisível de software, ou a um conjunto de componentes de software, que formam um produto quando usados em conjunto (por exemplo, um módulo cliente e um módulo servidor).

As características que devem estar contidas na Especificação dos Requisitos do Software incluem:

- Funcionalidade: O que o software deverá fazer?
- Interfaces externas: Como o software interage com as pessoas, com o hardware do sistema, com outros sistemas e com outros produtos?
- Desempenho: Quais a velocidade de processamento, o tempo de resposta e outros parâmetros de desempenho requeridos pela natureza da aplicação?
- Outros atributos: Quais as considerações sobre portabilidade, manutenibilidade e confiabilidade que devem ser observadas?
- Restrições impostas pela aplicação: Existem padrões e outros limites a serem obedecidos, como linguagem de implementação, ambientes de operação, limites de recursos etc.?

1.2.2 *Elaboração*

A Especificação dos Requisitos do Software deve ser escrita por membros da equipe de desenvolvimento de um projeto, com a participação obrigatória de um ou mais **usuários chaves** do produto em pauta. O usuário chave é aquele que é indicado pelo cliente como pessoa capacitada a definir requisitos do produto; normalmente, os usuários chaves são escolhidos entre profissionais experientes das diversas áreas que usarão o produto. Estes usuários chaves devem ser devidamente informados e treinados sobre as técnicas e notações que serão utilizadas no fluxo de Requisitos.

Geralmente, nem desenvolvedores nem clientes ou usuários são qualificados para escrever por si sós a Especificação dos Requisitos do Software, porque:

- os clientes nem sempre entendem os processos de desenvolvimento de software em grau suficiente para produzir uma especificação de requisitos de implementação viável;
- os desenvolvedores nem sempre entendem a área de aplicação de forma suficiente para produzir uma especificação de requisitos satisfatória.

Os usuários chaves devem ser conscientizados do papel essencial que desempenham na Especificação dos Requisitos do Software. Deve-se, também, comunicar-lhes o papel que terão no restante do projeto, tal como no desenho das interfaces de usuário (inclusive estudos de usabilidade), revisões técnicas e de apresentação, avaliação das liberações, testes de aceitação e todos os procedimentos de implantação.

1.2.3 *Ambiente*

Um software pode conter toda a funcionalidade necessária ao cliente, ou ser parte de um sistema maior. No caso de uma Especificação dos Requisitos do Software relativa a um software que é parte de um sistema maior, os requisitos de nível de sistema podem ser contidos em um dos seguintes documentos:

- um documento de Especificação de Requisitos de Sistema;

- um documento de definição de produto;
- uma proposta de projeto de sistema.

O mais completo desses documentos é a Especificação dos Requisitos do Sistema. Esta especificação definirá os requisitos aplicáveis ao sistema como um todo. Estes requisitos podem ser repassados aos componentes de software, ou realizados por outros componentes. Além disto, o Desenho do Sistema definirá as interfaces entre os componentes de software e os demais componentes. Estas interfaces podem resultar em requisitos adicionais de software. Os requisitos dos componentes do software não poderão entrar em conflito com os requisitos do sistema total.

Quando o software fizer parte de um sistema maior que está sendo especificado de forma concorrente, os requisitos de todo o sistema e de seus componentes separados passam a ser definidos em conjunto pelas diversas equipes do sistema e negociados entre elas. Exemplos de equipes com as quais a equipe de especificação de software pode ter de interagir incluem os desenvolvedores de hardware, redes e bancos de dados e os especialistas da área de aplicação, além de pessoal de marketing e de áreas administrativas e financeiras.

A equipe do projeto de software deve atuar juntamente com esses demais grupos, e com clientes e usuários-chaves, na definição dos requisitos de nível de sistema. Ela deve sempre indicar para os demais participantes do levantamento de requisitos de sistema se os requisitos que se pretende implementar por meio de software são viáveis. Todo requisito de sistema que tenha impacto no desenvolvimento de software deve ser aprovado pelo gerente do projeto de software.

Durante o desenvolvimento dos requisitos de sistema, os grupos participantes devem definir quais características dos requisitos são críticas, do ponto de vista dos clientes e usuários. Devem também estabelecer critérios de aprovação para cada componente do sistema que um grupo deva fornecer a outros grupos.

1.2.4 *Evolução*

Os requisitos de um produto podem alterar-se ao longo de seu desenvolvimento, por diversos motivos:

- descoberta de defeitos e inadequações nos requisitos originais;
- falta de detalhes suficientes nos requisitos originais;
- alterações incontornáveis no contexto do projeto (por exemplo, mudanças de legislação).

Mesmo reconhecendo esse fato, todo esforço deve ser feito para que a Especificação dos Requisitos do Software seja tão completa quanto possível. No caso de alterações serem indispensáveis, elas devem obedecer aos procedimentos de Gestão de Requisitos de Software (neste livro, fazem parte dos métodos de Gestão de Projetos).

Segundo o paradigma SW-CMM, uma organização considerada madura na gestão de requisitos de software deve atingir as seguintes metas.

- Os requisitos de software são controlados para estabelecer uma base para as atividades gerenciais e de engenharia de software, dentro de um projeto.
- Os planos, resultados, produtos e atividades de software são mantidos consistentes com os requisitos de software.

1.2.5 *Limites*

Normalmente, a Especificação dos Requisitos do Software não deve incluir decisões de desenho e implementação, nem aspectos gerenciais de projeto. Uma exceção é o caso em que estes aspectos são

restrições definidas pelo cliente. Por exemplo, este pode definir que serão usadas determinadas linguagens de programação, determinados componentes ou determinadas plataformas de bancos de dados. Por isso, a Especificação dos Requisitos do Software deverá satisfazer os seguintes critérios.

- Definir completa e corretamente todos os requisitos do produto do software. Requisitos podem existir em virtude da natureza do problema a ser resolvido, ou em virtude de outras características específicas do projeto.
- Não descrever qualquer detalhe de desenho ou de implementação. Estes devem ser descritos nos modelos e documentos produzidos pelos respectivos fluxos.
- Não descrever aspectos gerenciais do projeto, como custos e prazos. Estes devem ser especificadas em outros documentos, tais como o Plano de Desenvolvimento do Software ou o Plano da Qualidade do Software.

Normalmente, os seguintes itens são considerados parte do desenho, e não devem figurar na Especificação dos Requisitos do Software:

- partição do produto em módulos;
- alocação de funções aos módulos;
- fluxo de informação entre módulos;
- estruturas internas de dados.

Os requisitos a seguir são considerados requisitos gerenciais do projeto, e não devem ser incluídos na Especificação dos Requisitos do Software:

- custo;
- cronograma de entregas;
- relatórios requeridos;
- métodos requeridos de desenvolvimento;
- procedimentos de controle da qualidade;
- critérios de verificação e validação.

1.3 Qualidade dos requisitos

1.3.1 *Características de qualidade*

Para servir de base a um produto de boa qualidade, a própria Especificação de Requisitos deve satisfazer uma série de características de qualidade. Uma Especificação de Requisitos deve ser:

- **Correta** - Todo requisito presente realmente é um requisito do produto a ser construído.
- **Precisa** - Todo requisito presente possui apenas uma única interpretação, aceita tanto pelos desenvolvedores quanto pelos usuários chaves.
- **Completa** - Reflete todas as decisões de especificação que foram tomadas.
- **Consistente** - Não há conflitos entre nenhum dos subconjuntos de requisitos presentes.

- **Priorizada** - Cada requisito é classificado de acordo com a sua importância, estabilidade e complexidade.
- **Verificável** - Todos os seus requisitos são verificáveis.
- **Modificável** - Sua estrutura e estilo permitem a mudança de qualquer requisito, de forma fácil, completa e consistente.
- **Rastreável** - Permite a fácil determinação dos antecedentes e conseqüências de todos os requisitos.

1.3.2 Correção

Uma Especificação dos Requisitos é correta se todo requisito presente nela realmente é um requisito do produto a ser construído. Não existe ferramenta que garanta a correção de uma Especificação dos Requisitos. Para verificá-la, deve-se checar a coerência da Especificação dos Requisitos do Software com outros documentos da aplicação, tais como a Proposta de Especificação do Software, a Especificação dos Requisitos do Sistema e outros padrões referentes à área de aplicação. Deve-se ainda solicitar a aprovação formal da Especificação dos Requisitos do Software por parte do cliente, sem a qual o projeto não poderá prosseguir.

1.3.3 Precisão

Uma Especificação dos Requisitos é precisa se todo requisito presente possuir apenas uma única interpretação, aceita tanto pelos desenvolvedores quanto pelos usuários-chaves. Em particular, uma Especificação dos Requisitos deve ser compreensível para todo o seu público-alvo, e deve ser suficiente para o desenho dos testes de aceitação. Recomenda-se a inclusão no glossário da Especificação dos Requisitos de todos os termos contidos no documento que possam causar ambigüidades de interpretação.

Os seguintes meios devem ser usados para garantir maior precisão da Especificação dos Requisitos:

- revisões técnicas;
- uso de notações e ferramentas de análise, orientadas a objetos no caso do Praxis.

1.3.4 Completeza

Uma Especificação dos Requisitos é completa se reflete todas as decisões de especificação que foram tomadas, não contendo cláusulas de pendências. Uma Especificação dos Requisitos completa deve:

- conter todos os requisitos significativos relativos a funcionalidade, desempenho, restrições de desenho, atributos e interfaces externas;
- definir as respostas do software para todas as entradas possíveis, válidas e inválidas, em todas as situações possíveis;
- conter um glossário de todos os termos técnicos e unidades de medida, assim como referências completas a todos os diagramas, figuras e tabelas.

1.3.5 Consistência

Uma Especificação dos Requisitos é consistente se não há conflitos entre nenhum dos subconjuntos de requisitos presentes. Existem três tipos comuns de conflitos entre requisitos:

- conflito entre características de objetos do mundo real - por exemplo, formatos de relatórios ou cores de sinalização;

- conflito lógico ou temporal entre ações - por exemplo, um requisito diz que a ação A deve ser realizada antes da ação B, e outro diz o contrário;
- uso de termos diferentes para designar o mesmo objeto do mundo real - por exemplo, “lembrete” versus “mensagem”.

1.3.6 Priorização

Uma Especificação dos Requisitos é priorizada se cada requisito é classificado de acordo com a respectiva importância e estabilidade. A estabilidade estima a probabilidade de que o requisito venha a ser alterado no decorrer do projeto, com base na experiência de projetos correlatos. A priorização classifica o requisito de acordo com um dos seguintes graus:

- **requisito essencial** – requisito sem cujo atendimento o produto é inaceitável;
- **requisito desejável** – requisito cujo atendimento aumenta o valor do produto, mas cuja ausência pode ser relevada em caso de necessidade (por exemplo, de prazo);
- **requisito opcional** – requisito a ser cumprido se houver disponibilidade de prazo e orçamento, depois de atendidos os demais requisitos.

1.3.7 Verificabilidade

Uma Especificação dos Requisitos é verificável se todos os seus requisitos são verificáveis. Um requisito é verificável se existir um processo finito, com custo compensador, que possa ser executado por uma pessoa ou máquina, e que mostre a conformidade do produto final com o requisito. Em geral requisitos ambíguos não são verificáveis, assim como requisitos definidos em termos qualitativos, ou contrários a fatos técnicos e científicos.

1.3.8 Modificabilidade

Uma Especificação dos Requisitos é modificável se sua estrutura e estilo permitirem a mudança de qualquer requisito, de forma fácil, completa e consistente. A modificabilidade geralmente requer:

- organização coerente, com índices e referências cruzadas;
- ausência de redundância entre requisitos;
- definição separada de cada requisito.

1.3.9 Rastreabilidade

Uma Especificação dos Requisitos é rastreável se permite a fácil determinação dos antecedentes e conseqüências de todos os Requisitos. Dois tipos de rastreabilidade devem ser observados.

- **Rastreabilidade para trás** - deve ser possível localizar a origem de cada requisito. Deve-se sempre saber por que existe cada requisito, e quem ou o que o originou. Isso é importante para que se possa avaliar o impacto da mudança daquele requisito, e dirimir dúvidas de interpretação.
- **Rastreabilidade para a frente** - deve ser possível localizar quais os resultados do desenvolvimento que serão afetados por cada requisito. Isso é importante para garantir que os itens de análise, desenho, código e testes abranjam todos os requisitos, e para localizar os itens que serão afetados por uma mudança nos requisitos.

2 Atividades

2.1 Visão geral

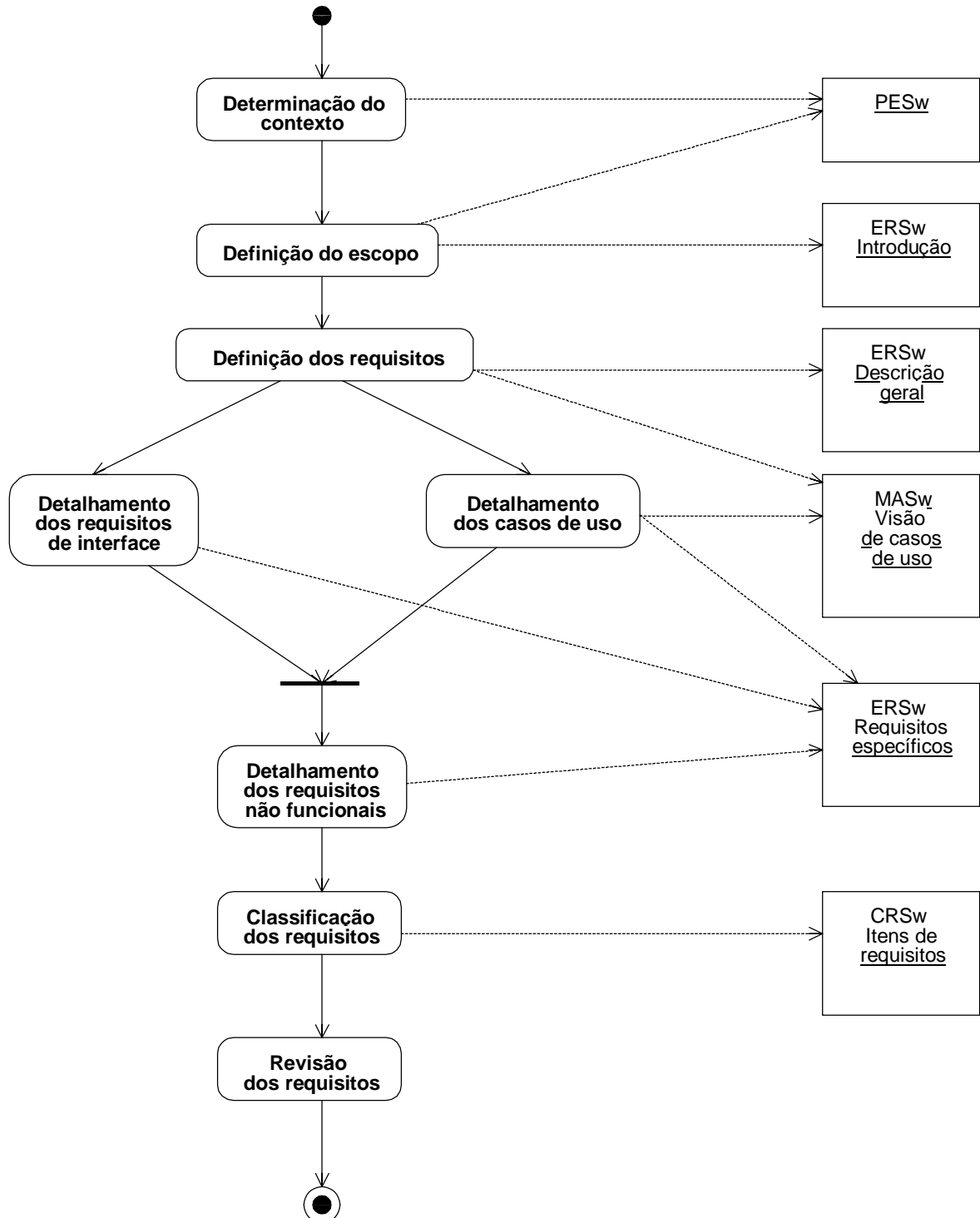


Figura 17 – Atividades e artefatos do fluxo de Requisitos

A Figura 17 apresenta as atividades do fluxo de Requisitos do Praxis. O fluxo é iniciado através da "**Determinação do contexto**", que levanta os aspectos dos processos de negócio ou de um sistema maior que sejam relevantes para a determinação dos requisitos do produto. A "**Definição do escopo**" delimita os problemas que o produto se propõe a resolver. Estas duas atividades são realizadas

principalmente na fase de Concepção, embora seus resultados possam ser revistos posteriormente, quando necessário. Os dados levantados servem de base para a Proposta de Especificação de Software³, e são posteriormente resumidos na seção "Introdução" da Especificação dos Requisitos do Software.

A "**Definição dos requisitos**" produz uma lista de todos os requisitos funcionais e não funcionais. Estes requisitos são descritos de forma sucinta, ainda sem entrar-se em detalhes. São também identificados os grupos de usuários do produto, e as demais restrições aplicáveis. Essas características compõem a seção "Descrição geral do produto" da Especificação dos Requisitos do Software.

É recomendável, até este ponto, que os tomadores de decisão do cliente participem do levantamento dos requisitos. As atividades seguintes cobrem aspectos mais detalhados, sendo provavelmente mais adequado que participem os usuários-chaves e não necessariamente pessoal gerencial do cliente. Esta atividade é visitada uma primeira vez durante a Concepção, para determinar-se pelo menos uma lista preliminar dos requisitos que permita dimensionar a fase de Elaboração. Ela será revista nesta fase, normalmente com a participação de um número maior de partes interessadas.

As três atividades seguintes correspondem ao detalhamento dos requisitos de interface, funcionais e não funcionais. Os dois primeiros são mostrados como atividades paralelas, pois existem interações fortes entre os requisitos de interface e os requisitos funcionais. Cada uma dessas atividades corresponde a uma das subseções da seção "Requisitos específicos" da Especificação dos Requisitos do Software.

O "**Detalhamento dos requisitos de interface**" levanta os aspectos das interfaces do produto que os usuários consideram requisitos. Normalmente, é feito um esboço das interfaces de usuário, levantado através de um protótipo executável ou de estudos em papel. Esses esboços, entretanto, não devem descer a detalhes de desenho, mas apenas facilitar a visualização dos verdadeiros requisitos (por exemplo, que informação a interface deve captar e exibir). São também detalhadas as interfaces com outros sistemas e componentes de sistema.

No Praxis, o "**Detalhamento dos requisitos funcionais**" utiliza a notação de casos de uso. Cada caso de uso representa uma fatia de funcionalidade do produto. Os relacionamentos dos casos de uso com os grupos de usuários e entre si são descritos dentro da visão de casos de uso, que é parte do Modelo de Análise do Software. O fluxo de execução das funções é descrito de forma padronizada, dentro da Especificação dos Requisitos do Software.

O "**Detalhamento dos requisitos não funcionais**" completa os requisitos, descrevendo os requisitos de desempenho e outros aspectos considerados necessários para que o produto atinja a qualidade desejada. Inclui-se aqui também o detalhamento de requisitos derivados de outros tipos de restrições (por exemplo, restrições de desenho).

A "**Classificação dos requisitos**" determina as prioridades relativas dos requisitos e avalia a estabilidade e a complexidade de realização. Os requisitos aprovados são lançados no Cadastro dos Requisitos do Software, para que sejam posteriormente registrados e rastreados seus relacionamentos com itens derivados, em outros artefatos do projeto.

Finalmente, a "**Revisão dos requisitos**" determina se todos eles satisfazem os critérios de qualidade de requisitos e se a Especificação dos Requisitos do Software está clara e bem entendida por todas as partes interessadas.

³ O padrão para Proposta de Especificação de Software apresentado neste livro é muito resumido e não inclui detalhes do contexto e do escopo. Caso necessário, esses detalhes podem ser apresentados em um anexo dessa Proposta, ou em um documento de Especificação de Requisitos de Sistema (que está fora do escopo deste livro).

2.2 Detalhes das atividades

2.2.1 *Determinação do contexto*

Usamos o termo "Determinação do contexto" para indicar todo o conjunto de tarefas que determina os aspectos relevantes do contexto em que um produto de software operará. Essa atividade é a mais variada de todas. Dependendo da complexidade e da responsabilidade do produto, pode ser resolvida em uma conversa informal com o cliente ou requerer a execução de um processo complexo de:

- definição de produto (por exemplo, vide [Floyd+93]);
- engenharia de requisitos de sistema (por exemplo, vide [Laplante93]);
- modelagem de processos de negócio (por exemplo, vide [Jacobson+94a]).

Todos os modelos da UML podem ser usados para capturar os aspectos relevantes de um sistema maior ou dos processos de negócio. Neste último caso, é particularmente útil o diagrama de atividades. Este diagrama foi usado na Figura 17, para descrever o próprio fluxo de Requisitos. A Figura 18 apresenta um exemplo de modelo de processo de negócio. Este modelo fornece um contexto para um sistema de informatização de mercearia, que será usado nos exemplos posteriores deste livro.

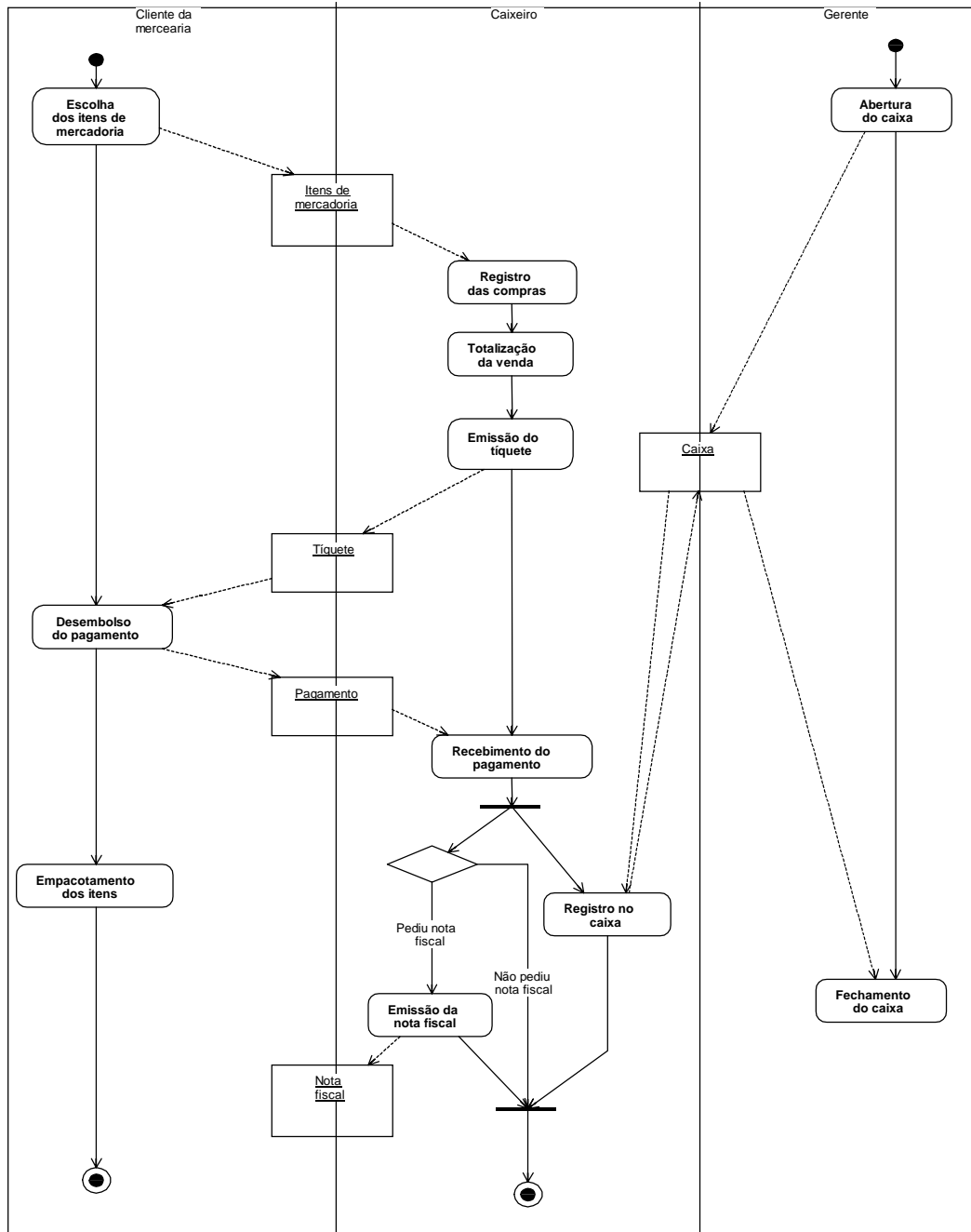


Figura 18 – Exemplo de modelo de processo de negócio

Em relação à Figura 17, a Figura 18 introduz alguns elementos adicionais. O diagrama é particionado em **raias** (*swimlanes*), que representam os diversos papéis de pessoas ou grupos envolvidos no processo (no exemplo, cliente da mercearia, caixeiro e gerente). Os objetos situados sobre as fronteiras das raias representam objetos partilhados entre os papéis. Em um processo de negócio, eles podem representar tanto objetos físicos quanto informacionais. Um losango de decisão tem como saída subfluxos alternativos (no exemplo, a emissão opcional de nota fiscal).

A determinação do contexto, a rigor, está fora do escopo da Engenharia de Software. Disciplinas correlatas incluem a Engenharia de Sistemas, as Engenharias das áreas de aplicação, e as técnicas de Organização e Métodos. Fica lembrado aqui, entretanto, que em muitas situações o engenheiro de software é chamado a participar desta atividade, pelo menos parcialmente. Da realização adequada dessa atividade básica pode depender o sucesso de todo o resto de um projeto.

2.2.2 Definição do escopo

2.2.2.1 Missão

O ponto focal do escopo de um produto é a missão dele (Tabela 34). A missão sintetiza que valor o produto acrescenta para o cliente e os usuários. Conseguir descrever o objetivo de um produto em um parágrafo curto, sem abuso de conjunções, é um indicador de que se conseguiu uma visão coerente do papel deste nos processos do cliente. A declaração da missão delimita as responsabilidades do produto e sintetiza o comprometimento entre cliente e fornecedor.

O produto Merci 1.0 visa oferecer apoio informatizado ao controle de vendas, de estoque, de compra e de fornecedores da mercearia Pereira & Pereira.

Tabela 34 - Exemplo de Missão do produto

2.2.2.2 Limites

Deve-se determinar os limites do produto (Tabela 35), ou seja, o que o produto não fará. Isso evita falsas expectativas por parte do cliente e usuários, e pode ressaltar funções e atributos que serão implementadas por outros componentes de um sistema maior, ou em versões futuras desse produto.

O Merci não fará vendas parceladas e só receberá dinheiro ou cheque.
 O Merci só fará a Emissão de Nota Fiscal durante a Operação de Venda.
 O Merci não fará um cadastro de clientes da mercearia Pereira & Pereira Comercial Ltda.
 O preço de venda deverá ser calculado pela mercearia Pereira & Pereira Comercial Ltda. e informado ao Merci.
 Atividades como backup e recuperação das bases de dados do sistema ficam a cargo da administração de dados e não serão providas no Merci.
 Não haverá tolerância a falhas no Merci.

Tabela 35 - Exemplo de Limites do produto

2.2.2.3 Benefícios

Deve-se identificar os benefícios que se espera obter com o produto e o valor destes para o cliente. O valor pode ser descrito simplesmente pela importância atribuída pelo cliente, ou pode ser expresso de outras formas, inclusive quantitativas. O levantamento dos benefícios (Tabela 36) é necessário para determinar se o valor dele compensará o investimento no projeto. Associando-se posteriormente funções e benefícios, será possível fazer a priorização dos requisitos funcionais com base concreta. Para isso, é necessário desde já distinguir quais benefícios são essenciais para justificar o produto, e quais podem ser considerados desejáveis ou opcionais.

Número de ordem	Benefício	Valor para o Cliente
1	Agilidade na compra e venda de mercadorias.	Essencial
2	Conhecimento do mercado de fornecedores visando uma melhor conjugação de qualidade, preço e prazo.	Essencial
3	Diminuição de erros na compra e venda de mercadorias.	Essencial
4	Economia de mão de obra.	Essencial
5	Eliminação da duplicidade de pedidos de compra.	Essencial
6	Qualidade na emissão da Nota Fiscal e Ticket de Venda, em relação à emissão manual.	Essencial
7	Diminuição do custo de estocagem.	Desejável
8	Identificação de distorções entre o quantitativo vendido e o ainda existente no estoque.	Desejável
9	Maior agilidade nas decisões de compra.	Desejável

Tabela 36 - Exemplo de Benefícios do produto

2.2.2.4 Referências

A partir desta atividade, é preciso identificar e catalogar todos os materiais cuja consulta possa ser necessária para melhor entendimento dos requisitos. As referências devem ser completas, para que todas as fontes de dados citadas na Especificação dos Requisitos do Software possam ser recuperadas, caso necessário.

Outra lista de referência que deve ser levantada é o glossário do projeto. Este glossário permitirá que as definições e siglas sejam coerentes com aquelas usadas em todos os documentos do projeto. Ele incluirá siglas, abreviações e termos relevantes para todas as partes interessadas. Por isso, deve conter as definições tanto de termos relevantes da área de aplicação, quanto de termos relevantes de informática que não sejam do conhecimento do público em geral.

2.2.2.5 Aspectos gerenciais

É preciso determinar quais as faixas de custo e prazo que o cliente espera desse projeto. Naturalmente, é muito cedo para fazer-se qualquer estimativa decentes desses parâmetros (embora muitos clientes esperem isso). Durante a fase de Concepção, entretanto, é importante determinar pelo menos o prazo e o custo da fase de Elaboração. Para isso, é geralmente necessário dispor de uma lista preliminar dos requisitos, levantada em uma passada pela atividade seguinte.

2.2.3 *Definição dos requisitos*

2.2.3.1 Introdução

Os pontos mais importantes dessa atividade são a identificação dos **casos de uso** (representações de funções do produto) e dos **atores** (representações dos usuários e outros sistemas que interagem com o produto). Os relacionamentos entre casos de uso e atores são representados através de **diagramas de casos de uso**, dos quais o principal é o **diagrama de contexto** do produto.

2.2.3.2 Casos de uso

Os casos de uso representam funções completas do produto. Um caso de uso realiza um aspecto maior da funcionalidade do produto: deve gerar um ou mais benefícios para o cliente ou os usuários. O modelo de casos de uso serve de base para determinar:

- classes e operações, durante a Análise;
- casos de testes de aceitação, durante os Testes;
- roteiros de manual de usuário, durante a Implementação.

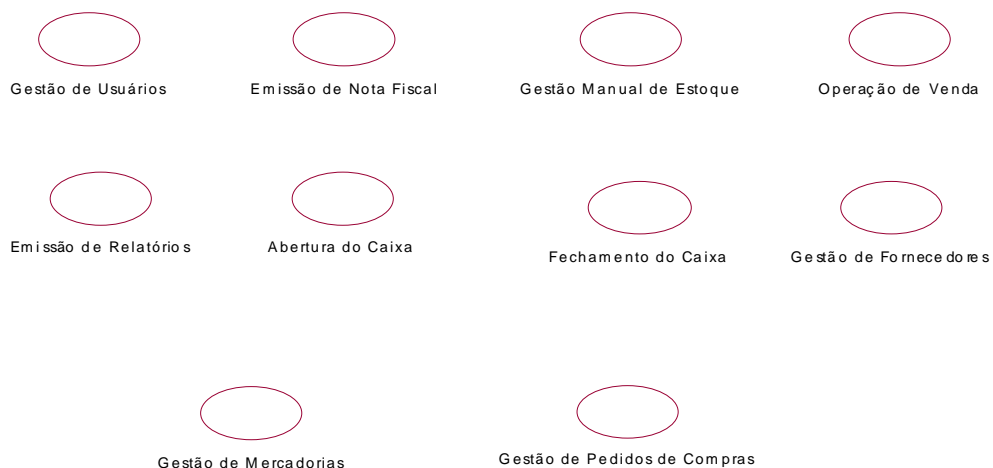


Figura 19 - Exemplos de casos de uso

A Figura 19 mostra os casos de uso de um sistema de informatização de mercearia. Durante a “Definição dos requisitos”, basta resumir cada caso de uso em uma descrição sucinta (Tabela 37). O fluxo do caso de uso, que detalhará os passos correspondentes, será definido no “Detalhamento dos requisitos funcionais”.

Número de ordem	Caso de uso	Descrição
1	Abertura do Caixa	Passagem para o Modo de Venda, liberando assim o caixa da mercearia para a Operação de Venda. O Gerente da mercearia deve informar o valor inicial deste caixa.
2	Emissão de Nota Fiscal	Emissão de Nota Fiscal para o cliente da mercearia (extensão da Operação de Venda).
3	Emissão de Relatórios	Emissão de relatórios com as informações das bases de dados do Merc.
4	Fechamento do Caixa	Totalização das vendas do dia e mudança para o Modo de Gestão.
5	Gestão de Fornecedores	Processamento de inclusão, exclusão e alteração de fornecedores.
6	Gestão de Mercadorias	Processamento de inclusão, exclusão e alteração de mercadorias.
7	Gestão de Pedidos de Compra	Processamento de inclusão, exclusão e alteração de pedidos de compra de mercadorias.
8	Gestão de Usuários	Controle de usuários que terão acesso ao Merc.
9	Gestão Manual de Estoque	Controle manual de entrada e saída de mercadorias.
10	Operação de Venda	Operação de venda ao cliente da mercearia.

Tabela 37 - Exemplo de lista de casos de uso

2.2.3.3 Atores

Os papéis dos usuários do produto são modelados através dos atores (Figura 20). Cada ator representa uma classe de usuários definida na Especificação dos Requisitos do Software. Os atores modelam os papéis e não as pessoas dos usuários; por exemplo, o mesmo usuário físico pode agir como gerente, gestor de estoques ou gestor de compras. Pode ser útil também definir atores não humanos, para modelar outros sistemas que devam interagir com o produto em questão.



Figura 20 - Exemplos de atores

Atores podem ser identificados através dos seguintes critérios:

- quem está interessado em certo requisito;
- onde o produto será usado;
- quem se beneficiará do produto;
- quem fornecerá informação ao produto;
- quem usará informação do produto;
- quem removerá informação do produto;
- quem dará suporte e manutenção ao produto;
- quais os recursos externos usados pelo produto;
- quais os papéis desempenhados por cada usuário;
- quais os grupos de usuários que desempenham o mesmo papel;
- quais os sistemas legados com os quais o produto deve interagir.

Para cada ator, deve-se incluir uma descrição sucinta das responsabilidades do respectivo papel (Tabela 38). Deve-se também identificar as características mais importantes do respectivo grupo de usuários. Exemplos de características importantes são cargo ou função, permissão de acesso, frequência de uso, nível educacional, proficiência no processo de negócio e proficiência em informática (Tabela 39).

Número de ordem	Ator	Definição
1	Caixeiro	Funcionário operador comercial de caixa.
2	Gerente	Funcionário responsável pela abertura e fechamento do caixa, além do cadastro de usuários.
3	Gestor de Compras	Funcionário responsável por: <ul style="list-style-type: none"> • cadastramento das mercadorias pertencentes ao estoque; • manter os níveis do estoque em valores acima do mínimo permitido para cada mercadoria; • emissão dos pedidos de compra da mercearia.
4	Gestor de Estoque	Funcionário responsável pela elaboração do inventário do estoque da mercearia e por manter estes níveis coerentes com as bases de dados do Mercê.

Tabela 38 - Exemplo de descrição de atores

Número de ordem	Atores	Permissão de acesso	Frequência de uso	Nível educacional	Proficiência na aplicação	Proficiência em Informática
1	Caixeiro	Operação de Venda, Emissão de Nota Fiscal.	Diário em horário comercial	1º Grau	Operacional	Aplicação
2	Gerente	Abertura do Caixa, Fechamento do Caixa, Gestão de Usuários.	Diário	2º Grau	Completa	Aplicação Windows 95
3	Gestor de Compras	Gestão de Mercadorias, Gestão de Fornecedores	Diária	3º grau	Completa	Aplicação Windows 95
4	Gestor de Estoque	Gestão Manual de Estoque.	Diário	1º Grau	Operacional	Aplicação

Tabela 39 - Exemplo de descrição de características dos usuários

Atores são usados para representar também sistemas externos. Estes podem incluir sistemas legados, produtos comerciais de software e outros componentes de um sistema maior. Podem incluir recursos de hardware e comunicação que devam receber um tratamento específico por parte do produto (por exemplo, dispositivos de hardware que normalmente não fazem parte do ambiente operacional).

2.2.3.4 Relacionamentos entre casos de uso e atores

Cada diagrama de casos de uso especifica os relacionamentos entre casos de uso e atores (Figura 21). Os relacionamentos indicam a existência de comunicação entre atores e casos de uso. Um caso de uso pode estar associado a mais de um ator, quando a sua execução requer a participação de diferentes atores.

Normalmente, a comunicação será representada como ligação sem direção; convencionou-se, nesse caso, que a iniciativa de comunicação parte do ator. Quando a iniciativa parte do caso de uso (por exemplo, alarmes, mensagens, dados enviados para outros sistemas etc.), a comunicação deve ser direcionada para o ator (Figura 22).

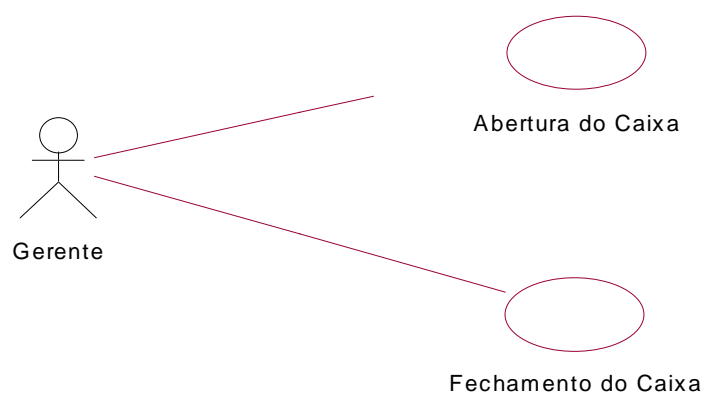


Figura 21 - Exemplo de casos de uso de um ator

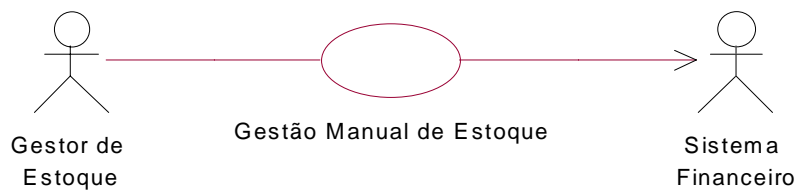


Figura 22 - Relacionamento entre caso de uso com mais de um ator

Caso exista grande número de atores, deve-se procurar agrupá-los em atores genéricos, que representem características comuns a vários grupos de usuários de comportamento semelhante em relação ao produto. Atores genéricos e específicos são ligados por relacionamentos de herança. Na Figura 23, indica-se que “Gerente de Vendas” e “Gerente de Compras” têm alguns aspectos em comum, que são abstraídos através do ator “Gerente”. Os diagramas de casos de uso podem ser simplificados, mostrando-se um caso de uso comum (aos atores específicos) comunicando-se apenas com o ator genérico (Figura 24).

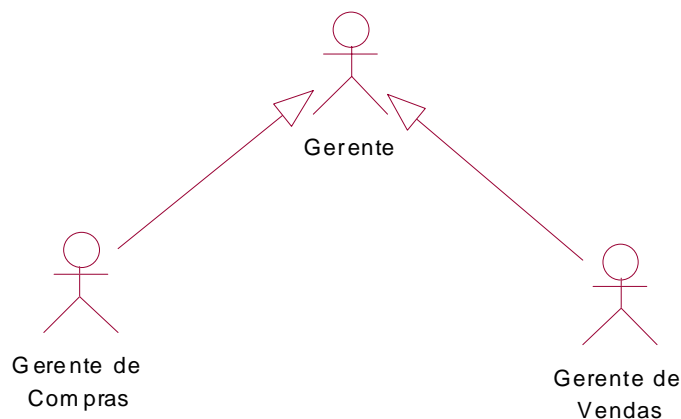


Figura 23 – Herança entre atores

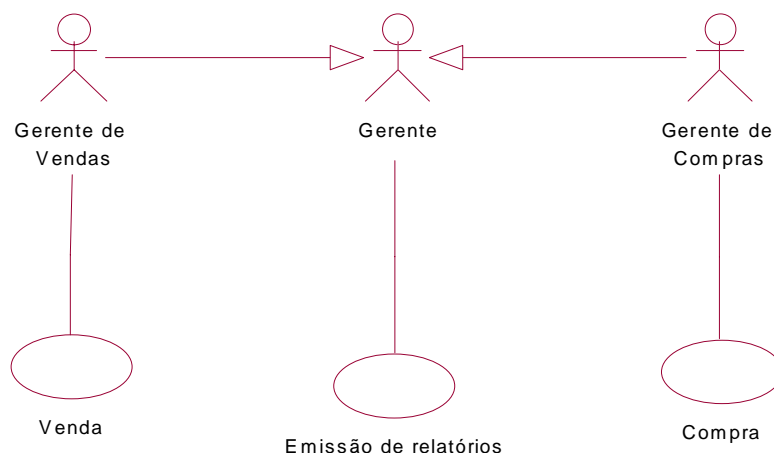


Figura 24 – Uso de atores genéricos

A identificação dos atores, além ser o ponto de partida para a especificação das interfaces de usuário, ajuda bastante a identificar os casos de uso relevantes. Os casos de uso normalmente expressam:

- quais as tarefas de cada ator;
- que informação cada ator cria, armazena, consulta, altera ou remove;
- que informação cada caso de uso cria, armazena, consulta, altera ou remove;
- que mudanças externas súbitas devem ser informadas ao produto pelos atores;
- que ocorrências no produto devem ser informadas a algum ator;
- que casos de uso darão suporte e manutenção ao sistema;
- quais os casos de uso necessários para cobrir todos os requisitos funcionais.

2.2.3.5 Diagrama de contexto

O diagrama de casos de uso mais importante é o **diagrama de contexto**. Este é um diagrama de blocos que mostra as interfaces do produto com seu ambiente de aplicação, inclusive os diversos tipos de usuários e outros sistemas com os quais o produto deva interagir. O diagrama de contexto deve indicar fontes e sorvedouros de dados. Se o produto fizer parte de um sistema maior, deve também identificar as interfaces entre o produto e o restante do sistema.

Nesse diagrama, os usuários, sistemas externos e outros componentes de um sistema maior são representados por atores, enquanto os casos de uso representam as possíveis formas de interação do produto com os atores. Devem ser mostrados apenas os casos de uso que se comunicam diretamente com os atores, através de interfaces (Figura 25).

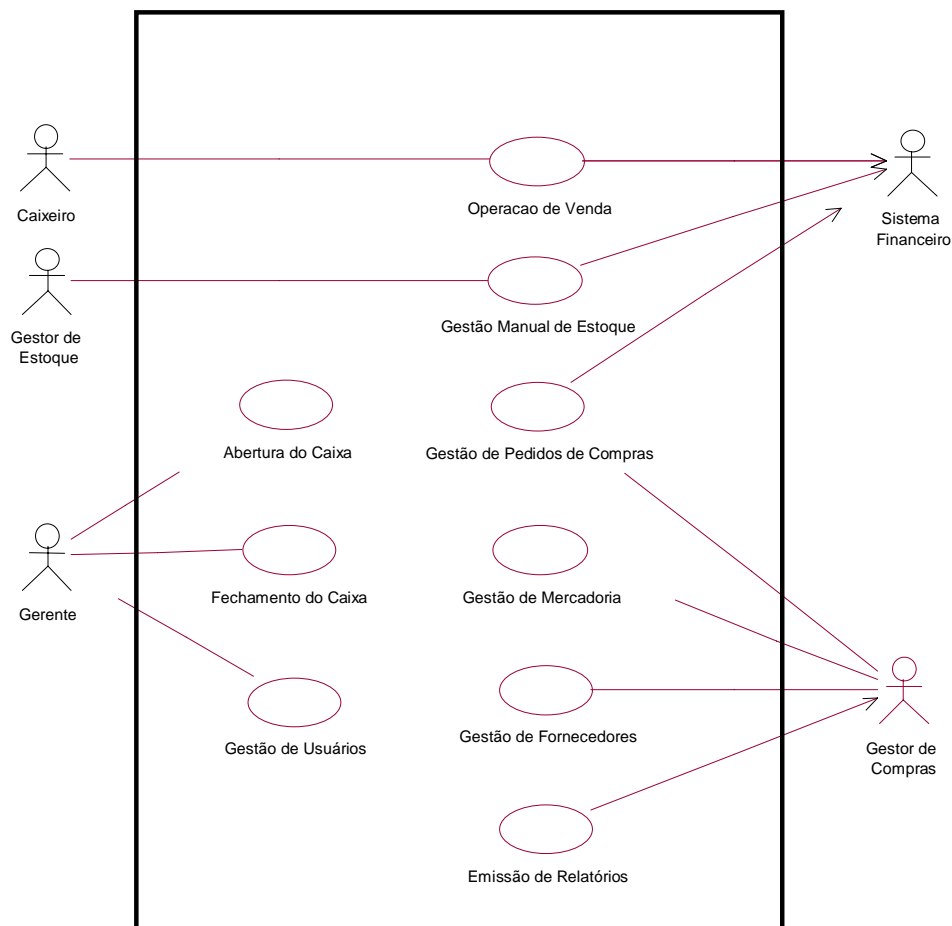


Figura 25 - Diagrama de contexto

Uma definição completa do contexto de um produto pode incluir outros aspectos, como:

- restrições de memória principal ou secundária;
- modos de operação (por exemplo, processamento em lote ou manutenção);
- requisitos de adaptação a ambientes específicos (por exemplo, que aspectos devem ser configuráveis durante a instalação em determinado local).

2.2.3.6 Outros requisitos

Completa-se a definição dos requisitos através da descrição das restrições aplicáveis. Estas provavelmente serão detalhadas em diversos tipos de requisitos não funcionais. Deve-se também definir as hipóteses de trabalho. O objetivo aqui é deixar claro, para todas as partes interessadas, quais suposições em que os demais requisitos se baseiam. Casos essas hipóteses sejam posteriormente invalidadas, fica claro que alguns requisitos possivelmente terão que ser renegociados.

Número de ordem	Restrição	Descrição
1	Ambiente	O ambiente operacional a ser utilizado é o Windows 95 (ou compatível).
2	Ambiente	O sistema deverá executar em um Pentium 133 MHz, com impressora de tecnologia laser ou de jato de tinta, a ser usada para impressão de todos os relatórios, exceto os tickets de venda.
3	Ambiente	Será utilizada uma impressora específica para a emissão dos tickets de venda, configurável como impressora suportada pelo ambiente operacional.
4	Expansibilidade	O produto deve ser desenvolvido levando-se em consideração que poderá ser expandido para mais de um caixa.
5	Legal	O produto deverá estar em conformidade com as leis e regulamentos vigentes na época do início do Desenvolvimento.
6	Segurança	O produto deverá restringir o acesso através de senhas individuais para cada usuário.

Tabela 40 – Exemplo de restrições

Número de ordem	Hipótese	De quem depende
1	Deve ser utilizado o sistema de gestão de bancos de dados Microsoft Access.	Pereira & Pereira Comercial Ltda deve adquirir, instalar e povoar.

Tabela 41 – Exemplo de hipóteses de trabalho

2.2.4 Detalhamento dos requisitos de interface

2.2.4.1 Interfaces genéricas

Esta atividade levanta, de forma detalhada, todos os requisitos referentes a entradas e saídas do produto. As interfaces externas não incluem arquivos de trabalho usados apenas pelo produto, mas incluem qualquer tipo de dados partilhados com outros produtos e componentes de sistema. Se uma interface é externa, ela faz parte do problema, e portanto deve fazer parte dos requisitos.

Detalhes importantes dos requisitos de interface incluem as fontes e destinos dos dados, assim como os requisitos de formatos destes. Estes detalhes podem ter resultado do desenho de um sistema maior, ou ser derivados dos requisitos destes.

3.1.3.1 Interface de software Sistema Financeiro

3.1.3.1.1 Fonte da entrada

Não aplicável.

3.1.3.1.2 Destino da saída

Arquivo texto para o Finance 98.

3.1.3.1.3 Relacionamentos com outras interfaces

As interfaces de Estoque e de Venda geram lançamentos para a interface com o Sistema Financeiro.

Tabela 42 - Exemplo de requisitos para interface de software

2.2.4.2 Interfaces gráficas de usuário

Nas interfaces gráficas de usuário, existem questões que claramente representam **requisitos** dos produto, tais como formatos de dados e comandos. Outros detalhes, como formatos de telas e janelas, são aspectos de desenho da interface de usuário.

Entretanto, pode ser recomendável que a Especificação dos Requisitos contenha um esboço gráfico da interface, já que estes esboços ajudam a identificar mais claramente os requisitos, e muitas vezes resultam naturalmente de atividades de prototipagem usadas para realizar a Engenharia de Requisitos. Caso esses esboços sejam incluídos, fica entendido que eles representam sugestões; o detalhamento definitivo será feito dentro do fluxo de Desenho.

Os leiautes de telas e janelas devem ser descritos em nível de detalhe suficiente para que o usuário possa aprová-los ou não, mas sem entrar em considerações de desenho para a usabilidade. Por exemplo, o destaque que se dá a um item de uma janela pode ser um requisito, como no caso de alarmes e advertências, ou pode ser um aspecto de desenho, quando simplesmente contribui para melhorar a legibilidade.

Os campos e comandos incluídos em cada interface de usuário devem representar requisitos de captura e exibição de informação. Durante o fluxo de Desenho, eles poderão ser substituídos por soluções funcionalmente equivalentes. Por exemplo, um botão constante do esboço da interface pode ser substituído por uma hiperligação, caso a interface seja implementada através de um navegador. Um campo de texto livre pode ser substituído por uma lista de seleção, caso o desenho conclua pela conveniência dessa no sentido de reduzir erros do usuário.

Figura 26 – Exemplo de leiaute de interface

2.2.5 Detalhamento dos requisitos funcionais

2.2.5.1 Introdução

Os requisitos funcionais descrevem as funções que o produto deverá realizar em benefício dos usuários. Existem muitas maneiras de descrição dessas funções. No Praxis, cada função será descrita por um caso de uso. A descrição dos fluxos dos casos de uso define os detalhes dos requisitos funcionais.

2.2.5.2 Organização dos casos de uso

O diagrama de contexto fornece a principal referência para visualização dos casos de uso. Em sistemas com número maior de casos de uso, ou casos de uso mais complexos, diversos tipos de diagramas adicionais podem ser usados para facilitar o seu entendimento. Diagramas específicos podem ser usados para fornecer visões mais localizadas, como os casos de uso acessíveis a determinados atores, ou que se pretende implementar em determinada liberação. Grupos correlatos de casos de uso podem ser agrupados em pacotes lógicos (pastas do Modelo de Análise).

Os casos de uso geralmente possuem um fluxo principal e vários subfluxos alternativos, que são detalhados conforme a próxima subseção. Entretanto, notações especiais são utilizadas para facilitar a descrição de funcionalidade mais complexa. Entre estas notações, destacam-se os casos de usos secundários, que simplificam o comportamento dos casos de uso primários através dos mecanismos de **inclusão** e **extensão**.

O caso de uso B estende o caso de uso A quando B representa uma situação opcional ou de exceção, que normalmente não ocorre durante a execução de A. Essa notação pode ser usada para representar fluxos alternativos ou anormais complexos. O caso de uso “estendido” é referenciado nas precondições do caso de uso “extensor”. As precondições são a primeira parte do fluxos dos casos de uso

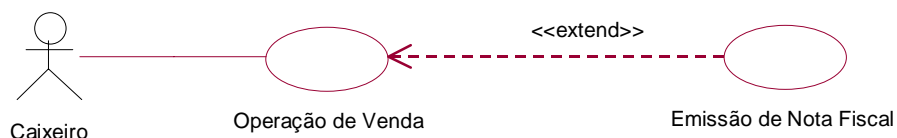


Figura 27 - Exemplo de caso de uso de extensão

O relacionamento “estende” é usado para representar:

- comportamento opcional;
- comportamento que só ocorre sob certas condições (por exemplo, alarmes);
- fluxos alternativos cuja realização depende de escolha de um ator.

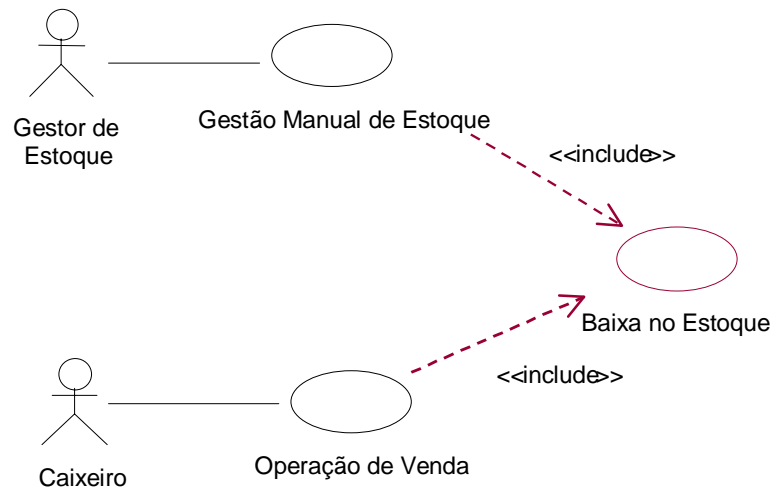


Figura 28 - Exemplo de caso de uso de inclusão

O caso de uso A inclui o caso de uso B quando B representa uma atividade complexa, comum a vários casos de uso. Essa notação pode ser usada para representar subfluxos complexos e comuns a vários casos de uso. O caso de uso “incluído” é referenciado no fluxo do caso de uso “includor”.

Note-se que um caso de uso incluído não representa uma subrotina comum. Interações entre casos de uso que têm acesso aos mesmos dados não são representadas nos diagramas de casos de uso, e sim nos diagramas de interação que resultam do fluxo de Análise. Um caso de uso incluído deve ser considerado uma espécie de macro, cuja única finalidade é abreviar as descrições dos fluxos dos casos de uso includores.

2.2.5.3 Fluxos dos casos de uso

O detalhamento dos fluxos dos casos de uso constitui geralmente uma das tarefas mais demoradas, difíceis e importantes do fluxo de Requisitos. Para cada caso de uso, é preciso levantar as precondições, ou seja, as condições que supõem estejam satisfeitas, ao iniciar a execução de um caso de uso (Tabela 43). Em seguida, levanta-se o fluxo principal, que representa a execução mais normal da função, e os subfluxos e fluxos alternativos, que representam variantes que são executadas sob certas condições.

Os fluxos são comumente descritos em linguagem natural, na forma de uma sequência de passos (Tabela 44). Cada passo corresponde a uma ação de um ator ou do produto; estes devem aparecer explicitamente como sujeitos da frase. Outros atores podem aparecer como objetos verbais de uma ação. Condições e iterações podem aparecer, mas os detalhes destas devem ser descritos em subfluxos, de preferência. Isso ajuda a manter a legibilidade do fluxo, que é essencial para garantir o bom entendimento de todas as partes.

Os subfluxos descrevem geralmente detalhes de iterações, ou de condições executadas com frequência (Tabela 45). Os fluxos alternativos descrevem condições pouco habituais ou exceções (Tabela 46). Os subfluxos devem ser detalhados dentro do fluxo de Requisitos; por outro lado, apenas os fluxos

alternativos mais importantes são detalhados como parte desse fluxo, deixando-se o tratamento detalhado de exceções para o desenho das interfaces de usuário.

Toda mercadoria a ser vendida (item de venda) deve estar previamente cadastrada.

Merci deve estar no Modo de Vendas.

Tabela 43 - Exemplo de precondições do caso de uso Operação de Venda

O Caixeiro faz a abertura da venda.

O Merci gera o código da operação de venda.

Para cada item de venda aciona o subfluxo Registro.

O Caixeiro registra a forma de pagamento.

O Caixeiro encerra a venda.

Para cada item aciona o subfluxo Impressão de Linha do Ticket.

O Merci notifica o Sistema Financeiro informando: Data, Número da Operação de Venda, “Receita”, Valor Total”, Nome do Cliente (caso tenha sido emitida a nota fiscal).

Tabela 44 - Exemplo de fluxo principal do caso de uso Operação de Venda

O Caixeiro registra o item de venda, informando a identificação e a quantidade.

O Merci totaliza a venda para o cliente da mercearia.

Tabela 45 - Exemplo de subfluxo: registro de item em Operação de Venda

Se o Gestor de Compras solicitar:

o Merci imprime o pedido de compra.

Tabela 46 - Exemplo de fluxo alternativo: Impressão de Pedido de Compras

A determinação dos fluxos de um caso de uso pode ser feita a partir das seguintes observações:

- quando e como o caso de uso principia;
- como o caso de uso interage com os atores;
- de que dados o caso de uso necessita;
- seqüência normal dos passos do caso de uso;
- possíveis seqüências anormais e alternativas dos passos do caso de uso;
- quando e como o caso de uso termina.

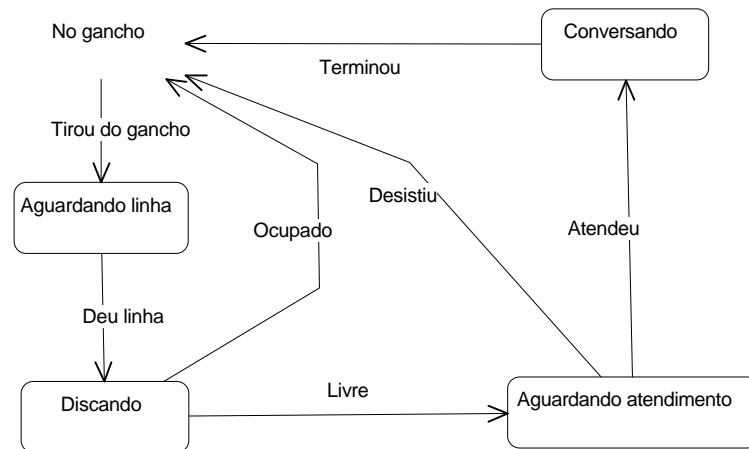


Figura 29 - Exemplo de diagrama de estado: fluxo de uma ligação telefônica

Para casos de uso complexos, pode-se descrever seu comportamento de maneira mais formal, através de um diagrama de estado ou diagrama de atividade da UML. Os detalhes dos estados e transições destes diagramas devem limitar-se ao que for requerido para fechar o detalhamento dos requisitos, evitando-se entrar em detalhes de desenho. Fluxos que precisam de diagramas de estado raramente são usados na Especificação de Requisitos. Durante o Desenho, entretanto, estes diagramas geralmente são necessários para gerir-se a complexidade introduzida pelo detalhamento adicional dos casos de uso.

2.2.6 Detalhamento dos requisitos não funcionais

2.2.6.1 Visão geral

Os requisitos não funcionais incluem os requisitos de desempenho e outros atributos de qualidade do produto. Incluem-se aqui também os requisitos lógicos de dados e as restrições ao desenho. Os requisitos não funcionais devem ser enunciados de forma precisa e quantitativa, mesmo que seja difícil formular valores razoáveis no levantamento dos requisitos de uma primeira versão de um produto.

Muitos requisitos não funcionais são globais, aplicando-se ao produto como um todo. Um requisito não funcional pode ser específico de um caso de uso; por exemplo, a duração máxima de uma transação descrita por um caso de uso. Nessa situação, o caso de uso deve ser indicado na descrição do requisito não funcional.

2.2.6.2 Requisitos de desempenho

Os requisitos de desempenho são requisitos numéricos, estáticos e dinâmicos a que o produto ou o sistema maior devem obedecer. Requisitos estáticos podem incluir, por exemplo:

- números de terminais suportados;
- números de usuários simultâneos;
- volume de informação que deve ser tratado.

Os requisitos dinâmicos podem incluir, por exemplo, o número esperado de transações por unidade de tempo, indicando-se condições de normalidade e de pico.

Todos os requisitos de desempenho devem ser especificados de forma quantitativa e mensurável. Por exemplo: “O produto deverá ter resposta rápida” não é um requisito aceitável; “90% das vezes o tempo de resposta do produto deverá ser inferior a 2 segundos” é um requisito aceitável.

A totalização da Operação de Venda não pode gastar mais do que 5 segundos, devendo ser realizada em 2 segundos, 80% das vezes.

Tabela 47 - Exemplo de requisito de desempenho

Um requisito quantitativo de desempenho é parte do problema e não da solução, e não adianta ignorá-lo se ele realmente existir. Portanto, se um dado requisito de desempenho é necessário para que o produto seja útil, ele deve ser enunciado sempre. Nesse caso, fica entendido que sua medição será obrigatória nos testes de aceitação.

Deve ficar claro para os clientes que, toda vez que questões de desempenho forem deixadas em aberto, os desenvolvedores têm liberdade para interpretá-las da forma mais conveniente para o desenho do produto. Todos devem estar conscientes de que as restrições de desempenho geralmente têm impacto profundo e irreversível nas decisões sobre a arquitetura do produto.

2.2.6.3 Requisitos de dados persistentes

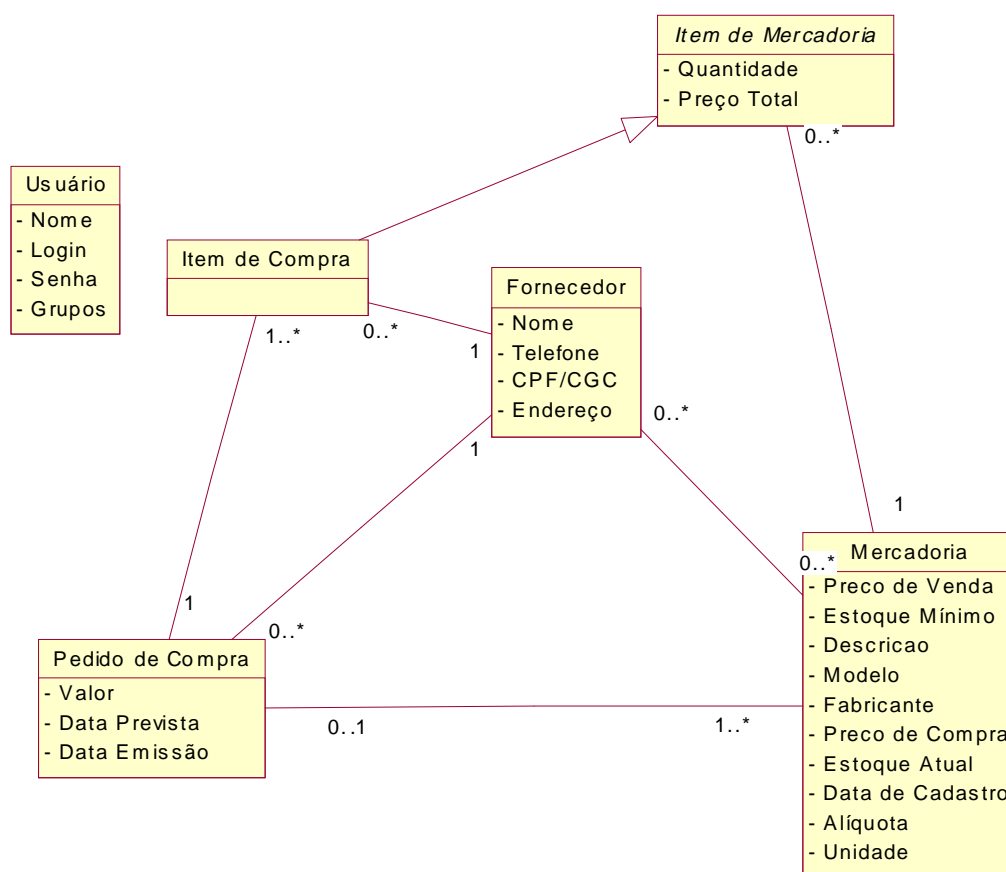


Figura 30 - Exemplo de diagrama de dados persistentes

Deve-se levantar as estruturas lógicas dos dados persistentes que sejam usadas pelo produto (Figura 30). Os dados persistentes são aqueles cujo valor subsiste após cada execução do produto, estando geralmente armazenados em memória secundária. Cada estrutura lógica de dados pode ser, por exemplo, um arquivo convencional, uma tabela em um banco de dados relacional ou uma classe persistente.

Um modelo completo de dados persistentes será obtido no fluxo de Análise. Entretanto, pode-se levantar provisoriamente um modelo de entidades e relacionamentos do problema, tendo em vista que esse tipo de modelo geralmente é entendido pela maioria dos usuários. O modelo de entidades e relacionamentos representa um primeiro esboço do Modelo de Análise.

2.2.6.4 Restrições ao desenho

As restrições ao desenho representam condições que limitam o espaço de soluções, e não o espaço de problemas, como ocorre com os requisitos normais. Elas devem ser incluídas entre os requisitos quando decorrem de imposições do cliente ou de terceiros, como padrões e restrições legais.

O Merci deverá ser desenhado de forma que possa ser expandido para mais de um terminal de caixa.

Tabela 48 - Exemplos de restrição ao desenho (expansibilidade)

O leiaute da nota fiscal utilizada pela mercearia deve ter sido previamente aprovado pela Secretaria de Receita.

Tabela 49 - Exemplos de restrição ao desenho (legal)

2.2.6.5 Atributos de qualidade

Os atributos de qualidade são características não funcionais requeridas pelo cliente. Uma classificação recomendável usa as características e subcaracterísticas definidas pela norma ABNT ISO-9126. Essa norma inclui atributos de funcionalidade (Tabela 52), confiabilidade (Tabela 53), usabilidade (Tabela 54), manutenibilidade (Tabela 55) e portabilidade (Tabela 56), além dos requisitos de desempenho. Os atributos devem sempre ser quantificados através de métricas adequadas, para que possam servir de base aos testes de aceitação.

Um operador de caixa proficiente em máquina registradora deverá ser capaz de aprender a operar o Merci com um dia de treinamento.

Tabela 50 - Exemplos de atributos de qualidade (apreensibilidade)

O Merci deverá restringir o acesso através de senhas para os usuários, conforme especificado na Seção 2.3 Características dos Usuários.

Tabela 51 - Exemplos de atributos de qualidade (segurança do acesso)

Subcaracterística	Definição	Possíveis métricas
Acurácia	Correção dos resultados	Correlação entre os resultados de casos de teste padronizados e os resultados obtidos por um método de referência (por exemplo, cálculo manual ou medidas em campo).
Interoperabilidade	Capacidade de interação com outros produtos especificados.	Índice de compatibilidade com um produto de referência.
Segurança do acesso	Resistência ao acesso não autorizado.	Índice de sucesso em testes de resistência à penetração.

Tabela 52 - Propostas da ISO-9126 para subcaracterísticas de funcionalidade

Subcaracterística	Definição	Possíveis métricas
Maturidade	Frequência de falhas	Tempo médio entre falhas (MTBF).
Tolerância a falhas	Capacidade de operação parcial em presença de falhas.	Índice de degradação por falha.
Recuperabilidade	Tempo e esforço necessários para restabelecer níveis de desempenho especificados.	Tempo médio para recuperação (MTTR).

Tabela 53 - Propostas da ISO-9126 para subcaracterísticas de confiabilidade

Subcaracterística	Definição	Possíveis métricas
Inteligibilidade	Esforço para reconhecimento do conceito lógico.	Tempo necessário para transmissão do conceito lógico, durante o treinamento dos usuários.
Apreensibilidade	Esforço para aprendizado da operação do produto.	Tempo necessário para que determinado tipo de usuário seja treinado no produto.
Operacionalidade	Esforço para operação do produto.	Produtividade do usuário em operações realizadas por unidade de tempo, taxas de erros de utilização.

Tabela 54 - Propostas da ISO-9126 para subcaracterísticas de usabilidade

Subcaracterística	Definição	Possíveis métricas
Analisabilidade	Esforço necessário para diagnosticar problemas.	Tempo de diagnóstico de problemas, tal como registrado pelo sistema de manutenção.
Modificabilidade	Esforço necessário para modificar, adaptar ou corrigir problemas.	Tempo de resolução de problemas, tal como registrado durante a manutenção.
Estabilidade	Risco de efeitos inesperados por ocasião das modificações.	Porcentagem de defeitos detectados que foram introduzidos não durante o desenvolvimento, mas em operações de manutenção.
Testabilidade	Esforço para validação do produto modificado.	Esforço associado aos testes de regressão (vide capítulo sobre Testes).

Tabela 55 - Propostas da ISO-9126 para subcaracterísticas de manutenibilidade

Subcaracterística	Definição	Possíveis métricas
Adaptabilidade	Capacidade de adaptação a ambientes específicos.	Alguma medida da fração do sistema (em linhas de código, número de classes, número de métodos etc.) que é independente de ambiente.
Capacidade para ser instalado	Esforço para instalação em um ambiente especificado.	Alguma medida do custo e duração da fase de implantação.
Conformidade	Obediência a padrões de portabilidade.	Fração das recomendações do padrão desejado que foram implementadas.
Capacidade para substituir	Esforço necessário para substituir outro produto especificado	Fração das funções que operam de forma idêntica às funções correspondentes do produto de referência.

Tabela 56 - Propostas da ISO-9126 para subcaracterísticas de portabilidade

2.2.7 *Classificação dos requisitos*

Nesta atividade, são marcados os requisitos que formarão o Cadastro dos Requisitos do Software (CRSw). No Praxis, serão considerados como requisitos diferentes e individuais:

- cada interface;
- cada caso de uso;
- cada requisito não funcional.

Todo requisito cadastrado deverá receber um identificador único dentro do projeto, que permitirá sua referência nos documentos de descrição do desenho do software e descrição dos testes do software. No caso de projetos de sistemas, cada requisito de software deverá ser associado com o respectivo item da especificação de requisitos de sistema, da definição de produto ou da proposta de projeto, por meio dos mecanismos apropriados do Cadastro dos Requisitos.

A Figura 31 mostra um exemplo de organização do cadastro inicial. Os campos dessa tabela devem ser assim preenchidos:

- **Número** - número de ordem do requisito no cadastro.
- **Nome do requisito** - nome dado à interface, ao caso de uso ou ao requisito não funcional, na Especificação de Requisitos do Software.
- **Tipo** - interface, caso de uso ou requisito não funcional.
- **Importância** - essencial, desejável ou opcional, conforme definido na priorização dos requisitos.
- **Complexidade** - estimativa do esforço e riscos de implementação, em comparação com outros requisitos do projeto; pode ser alta, média ou baixa.
- **Estabilidade** - estimativa da probabilidade de que o requisito venha a ser alterado no decorrer do projeto, com base na experiência de projetos correlatos; pode ser alta, média ou baixa.

Número	Nome do requisito	Tipo	Importância	Complexidade	Estabilidade
1	Interface de usuário Tela de Abertura do Caixa	Interface	Essencial	Baixa	Média
2	Interface de usuário Tela de Compras	Interface	Essencial	Média	Média
3	Interface de usuário Tela de Estoque	Interface	Desejável	Média	Baixa
4	Interface de usuário Tela de Fechamento do Caixa	Interface	Essencial	Baixa	Média
5	Interface de usuário Tela de Fornecedores	Interface	Essencial	Média	Baixa
6	Interface de usuário Tela de Mercadorias	Interface	Essencial	Média	Alta
7	Interface de usuário Tela de Nota Fiscal	Interface	Desejável	Baixa	Alta
8	Interface de usuário Tela de Pedidos de Compras	Interface	Opcional	Baixa	Baixa
9	Interface de usuário Tela de Relatórios	Interface	Essencial	Baixa	Baixa
10	Interface de usuário Tela de Usuários	Interface	Essencial	Baixa	Média
11	Interface de usuário Tela de Vendas	Interface	Essencial	Alta	Média
12	Interface de usuário Relatório de Estoque Baixo	Interface	Desejável	Média	Baixa
13	Interface de usuário Relatório de Fornecedores	Interface	Desejável	Média	Baixa
14	Interface de usuário Relatório de Mercadorias	Interface	Desejável	Média	Alta
15	Interface de usuário Nota Fiscal	Interface	Essencial	Média	Baixa
16	Interface de usuário Pedido de Compra	Interface	Opcional	Baixa	Baixa
17	Interface de usuário Relação de Pedidos de Compra	Interface	Opcional	Média	Média
18	Interface de usuário Ticket de Venda	Interface	Essencial	Média	Média
19	Interface de software Sistema Financeiro	Interface	Desejável	Média	Média
20	Caso de uso Abertura do Caixa	Caso de uso	Essencial	Baixa	Média
21	Caso de uso Emissão de Nota Fiscal	Caso de uso	Desejável	Média	Média
22	Caso de uso Emissão de Relatórios	Caso de uso	Essencial	Baixa	Baixa
23	Caso de uso Fechamento do Caixa	Caso de uso	Essencial	Baixa	Média
24	Caso de uso Gestão de Fornecedores	Caso de uso	Essencial	Média	Baixa
25	Caso de uso Gestão de Mercadorias	Caso de uso	Essencial	Média	Média
26	Caso de uso Gestão de Pedidos de Compra	Caso de uso	Opcional	Baixa	Baixa
27	Caso de uso Gestão de Usuários	Caso de uso	Essencial	Baixa	Média
28	Caso de uso Gestão Manual de Estoque	Caso de uso	Desejável	Média	Baixa
29	Caso de uso Operação de Venda	Caso de uso	Essencial	Alta	Média
30	Requisito de desempenho Tempo de resposta	Não funcional	Desejável	Baixa	Alta
31	Restrição ao desenho Padrão de Nota Fiscal	Não funcional	Essencial	Média	Alta
32	Restrição ao desenho Expansibilidade	Não funcional	Opcional	Alta	Média
33	Atributo da qualidade Segurança do Acesso	Não funcional	Desejável	Média	Média
34	Atributo da qualidade Apreensibilidade	Não funcional	Desejável	Média	Alta

Figura 31 – Cadastro inicial dos requisitos

2.2.8 Revisão dos requisitos

A revisão dos requisitos tem por objetivo assegurar que a Especificação dos Requisitos do Software:

- esteja conforme com o respectivo padrão, e com outros padrões aplicáveis ao projeto em questão;
- atenda aos critérios de qualidade dos requisitos;
- forneça informação suficiente para o desenho do produto, de seus testes de aceitação e do seu manual de usuário.

Inicialmente, os requisitos são revistos informalmente pelos participantes do levantamento. O Praxis prevê, ao final da iteração de Análise dos Requisitos, uma revisão técnica formal. Nessa altura, o fluxo de análise terá completado o Modelo de Análise do Software, permitindo uma verificação mais rigorosa dos requisitos. Um roteiro de revisão é fornecido neste livro, dentro do padrão para Especificação de Requisitos de Software.

Em princípio, não cabe à revisão dos requisitos entrar no mérito do atendimento das necessidades dos usuários por parte dos requisitos especificados. Estes aspectos devem ser estabelecidos através das técnicas de Engenharia de Requisitos (discutidas na próxima seção), através de revisões de apresentação para o cliente e usuários e através da aceitação formal pelo cliente. Caso este tipo de questionamento surja no decorrer de uma revisão formal, o líder da reunião deve anotá-lo na lista de tópicos da revisão, não permitindo sua discussão.

3 Técnicas

3.1 Introdução

Esta subseção trata de técnicas que são aplicáveis a várias atividades do fluxo de requisitos. Os protótipos e as oficinas de requisitos aumentam a eficácia das atividades de levantamento de requisitos, e a qualidade dos requisitos resultantes. A aplicação dessas técnicas contribui para que os requisitos sejam levantados em tempo curto, com participação ativa das partes interessadas.

As técnicas de relacionamento com os clientes tratam dos problemas de relacionamento humano que surgem entre desenvolvedores e usuários, envolvendo os requisitos. Muitas vezes, esses problemas se transformam em pendências políticas que põem a perder os resultados do esforço técnico.

3.2 Protótipos

3.2.1 *Tipos de protótipos*

O uso de protótipos tem sido uma técnica muito popular na literatura, principalmente quando associado ao uso de métodos orientados a objetos. Para colocar essa técnica em uma perspectiva correta, dentro dos processos de desenvolvimento do software, é importante distinguir entre protótipo descartável e protótipo evolucionário [Davis93].

O **protótipo descartável** é tipicamente construído durante a Engenharia de Requisitos, com a única finalidade de demonstrar aos usuários chaves o que o analista captou quanto aos requisitos do produto, ou parte deles. No contexto do Praxis, “prototipagem” sem adjetivos sempre se refere ao protótipo descartável. Na construção de um protótipo descartável, o fundamental é a rapidez de construção; um protótipo descartável deve ser produzido em poucas horas, ou no máximo em poucos dias.

O **protótipo evolucionário** é chamado de liberação no contexto do Praxis. Ele conterá um subconjunto dos requisitos do produto final, mas nenhum dos padrões de engenharia de software é afrouxado em sua construção. O objetivo da partição da construção de um produto em liberações é permitir a implementação incremental⁴ e iterativa⁵ de um aplicativo complexo. Ele não faz parte da Engenharia de Requisitos, embora seja um prosseguimento natural do uso de protótipos descartáveis e da análise orientada a objetos.

3.2.2 *Objetivos*

O protótipo descartável tem por objetivo explorar aspectos críticos dos requisitos de um produto, implementando de forma bastante rápida um pequeno subconjunto da funcionalidade deste. Ele ajuda a decidir sobre questões que sejam vitais para o sucesso de um produto, e que sejam tratáveis em um experimento de curta duração. Por exemplo, ele é indicado para se estudar:

- alternativas de interface de usuário;
- problemas de comunicação com outros produtos;
- viabilidade de atendimento dos requisitos de desempenho.

Em projetos maiores, o protótipo descartável pode também ser empregado em problemas de desenho e implementação, sempre que seja possível decidir uma questão importante através de um pequeno experimento.

⁴ Por partes.

⁵ Que pode ser revista várias vezes, em função da avaliação dos Usuários.

3.2.3 *Técnicas*

O protótipo descartável pode ser construído no mesmo ambiente que o produto final, em ambiente de desenvolvimento mais rápido, ou mesmo em um ambiente considerado ferramenta de documentação. A escolha é mera questão de conveniência. Como exemplos de áreas tratáveis por protótipos e os respectivos ambientes de prototipagem citam-se os seguintes::

- **Interface de usuário** – protótipo visual (fachada de Hollywood), escrito seja no ambiente definitivo, seja em um ambiente de programação rápida.
- **Relatórios textuais** – processador de textos ou ferramenta de geração de relatórios.
- **Relatórios gráficos** – ferramenta de desenho ou ambiente de programação rápida com biblioteca gráfica.
- **Organização e desempenho de bancos de dados** ferramenta de desenvolvimento rápido integrada ao sistema de gerência de bancos de dados que se pretenda usar.
- **Cálculos complexos** – planilha ou ferramenta matemática.
- **Partes de tempo de resposta crítico**, em sistemas de tempo real – pequeno programa de teste no ambiente alvo.
- **Tecnologia no limite do estado da arte** – pequeno programa de teste, no ambiente que for mais adequado.

3.2.4 *Riscos e benefícios*

O material de código dos protótipos descartáveis não deve ser reaproveitado. Antes que se decidam usar protótipos descartáveis, o cliente e os usuários devem ser plenamente esclarecidos quanto aos objetivos, e comprometer-se com seu descarte. Argumentos de custo dos protótipos nunca são válidos: quando se decide usá-los, é porque se concluiu que o custo de construí-los e depois descartá-los é compensado pela redução dos riscos. Quem quiser usar protótipos evolutivos deve planejá-los como tais desde o começo; pode ser o caso, por exemplo, em relação às interfaces de usuário.

O prazo de construção e uso dos protótipos descartáveis é obrigatoriamente curto. Eles devem sempre ter um objetivo simples e muito bem definido, que geralmente consiste em confirmar ou refutar uma hipótese. Devem também ter um prazo máximo, curto em relação à duração total esperada para o projeto; ao final desse prazo, devem ser cancelados se não estiverem prontos.

Os relatos de experiência de uso de protótipos descartáveis indicam os seguintes benefícios:

- redução dos riscos na Construção;
- aumento da manutenibilidade;
- aumento da estabilidade dos requisitos;
- oportunidade para treinamento dos programadores menos experientes, trabalhando como codificadores⁶.

Segundo [Jones94], o uso de protótipos descartáveis oferece um retorno de investimento de um fator de 200%, em um ano, e de 1.000%, em quatro anos.

⁶ Porque o código de um protótipo descartável não precisa ter o mesmo padrão de qualidade que o código definitivo, já que não será mantido.

Os seguintes pontos são particularmente importantes quanto ao sucesso dos protótipos descartáveis:

- escolha do ambiente de prototipagem;
- entendimento dos objetivos do protótipo por parte de todos os interessados no projeto;
- focalização em áreas menos compreendidas;
- tratamento da prototipagem como experimento científico, sujeito a monitoração e controle.

3.3 Oficinas de requisitos

3.3.1 *Visão geral*

As oficinas de requisitos são reuniões estruturadas para definição conjunta dos requisitos, envolvendo desenvolvedores, usuários e outros especialistas. O tipo de oficina que será aqui discutido é baseado nas técnicas de JAD, tais como descritas em [McConnell96]. Existem variantes na literatura.

JAD é um acrônimo para “Joint Application Development”. Trata-se de uma técnica estruturada de condução de reuniões de desenvolvimento de material, aplicável a diversas atividades do ciclo de vida do software, como engenharia de requisitos, desenho dos testes e desenho do produto. É particularmente aplicável ao levantamento e negociação de requisitos, onde estes são realizados em um conjunto de reuniões em que participam desenvolvedores e usuários chaves, assim como gerentes de ambos os lados.

A técnica de JAD apresenta as seguintes vantagens:

- comprometer com os requisitos os usuários com poder de decisão sobre estes;
- reduzir o prazo de levantamento da especificação dos requisitos;
- eliminar requisitos de valor questionável;
- reduzir diferenças de interpretação dos requisitos entre usuários e desenvolvedores;
- produzir um primeiro esboço das interfaces de usuário;
- trazer à baila, o mais cedo possível, problemas políticos que possam interferir no projeto.

A técnica de JAD compreende as seguintes tarefas:

- **Personalização** – adaptação do método à tarefa específica, geralmente feita pelo líder do JAD.
- **Sessões** – parte principal do método, na qual participam todos os interessados na tarefa.
- **Fechamento** – produção dos documentos resultantes.

A seguir, é apresentado um esquema genérico de realização de JAD, seguido de um esquema específico para o fluxo de Requisitos. O JAD de requisitos pode, caso conveniente, ser repartido em JADs separados para cada fase ou cada iteração do processo.

3.3.2 *Tarefas do JAD*

3.3.2.1 **Personalização**

A subdivisão de personalização dura tipicamente de 1 a 10 dias, compreendendo:

- organização da equipe de JAD;
- orientação dos participantes quanto à técnica de JAD (muitas vezes chamada de “pre-JAD”);
- determinação de aspectos particulares em relação ao projeto;
- preparação das instalações e material das sessões.

3.3.2.2 Sessões

As sessões duram de um a vários dias. Todos os integrantes da equipe de JAD devem participar das sessões em tempo integral, para que não se perca tempo em recapitulações para os ausentes em sessões anteriores. Idealmente, as sessões devem ser conduzidas em local afastado das organizações dos participantes. Interrupções são altamente prejudiciais, e todos os participantes devem ser previamente avisados disso. Telefonemas não devem ser atendidos, os telefones celulares devem ser desligados, e a agenda de todos os participantes deve estar livre de outros compromissos durante as sessões.

Deve-se planejar a disponibilidade dos equipamentos e suprimentos que sejam necessários, tais como:

- computadores;
- projetores;
- copiadoras;
- blocos de escrever;
- flip-charts;
- quadros brancos;
- câmeras instantâneas (para fotografar quadros brancos);
- lápis e canetas;
- cartões de visita;
- lanches.

As sessões devem contar com os seguintes tipos de participantes:

- o **líder da sessão**, responsável por manter o bom clima e o foco das discussões, que deve dominar técnicas de condução de reunião, e ser treinado em JAD, em particular;
- os **patrocinadores**, representantes do cliente, responsáveis pela decisão de continuar ou não o projeto;
- os **representantes dos usuários**, com autoridade para tomar decisões em nome da respectiva comunidade;
- os **desenvolvedores**, cuja função básica durante o JAD deve ser fornecer informação sobre a viabilidade das idéias levantadas;
- o **relator**, participante da equipe do projeto, encarregado de redigir as atas de reunião.

Durante a discussão as idéias vão sendo registradas e simultaneamente exibidas a todos os participantes, seja através de quadros brancos, seja através de computador com projetor. O líder deve estimular a participação de todos, manter o foco, encaminhar a resolução de conflitos e identificar questões que não possam ser resolvidas durante a sessão. Deve ficar claro para todos o comprometimento que deverá existir em relação às conclusões do JAD.

3.3.2.3 Fechamento

Na subdivisão de fechamento, devem-se produzir os seguintes documentos:

- uma coletânea do material produzido durante as sessões;
- os resultados para a etapa do processo tratada no JAD.

É feita então uma apresentação da parte já produzida desses resultados, para os responsáveis pelo projeto, juntamente com um balanço dessa etapa. Isso é feito em uma reunião também chamada de revisão do JAD (“*JAD review*”).

3.3.3 JAD para Requisitos

3.3.3.1 Personalização

A subdivisão de personalização compreende, além dos aspectos genéricos citados anteriormente, as seguintes tarefas:

- preparação das instalações e material para as sessões de levantamento de requisitos;
- preparação de formulários e material audiovisual;
- preparação de software para documentação dos requisitos;
- preparação de software para modelagem e documentação de casos de uso;
- preparação de software para desenho de interfaces e prototipagem.

3.3.3.2 Sessões

Uma sessão típica da JAD para levantamento dos requisitos tem as seguintes etapas:

1. abertura – apresentação das finalidades da sessão, agenda e tempos previstos;
2. definição de alto nível dos requisitos:
 - 2.1. identificação das necessidades do cliente às quais o produto deve atender;
 - 2.2. objetivos do produto;
 - 2.3. benefícios esperados;
 - 2.4. possíveis funções (com as prioridades aproximadas);
 - 2.5. considerações estratégicas.
3. delimitação do escopo do produto – identificação de funções que o produto poderia conter, mas decidiu-se não incluir nesta versão;
4. levantamento dos casos de uso do produto;

5. detalhamento dos casos de uso do produto:

5.1. identificação dos fluxos principais dos casos de uso;

5.2. identificação dos subfluxos e fluxos alternativos dos casos de uso;

6. definição dos requisitos e esboço do layout das interfaces de usuário do produto;

7. definição dos requisitos para interfaces com outros produtos;

8. planejamento do JAD de análise:

8.1. estimativas do JAD de análise;

8.2. identificação dos participantes do JAD de análise;

8.3. elaboração do cronograma do JAD de análise.

9. documentação dos problemas e considerações – documentação das opções consideradas e o porquê das decisões tomadas;

10. fechamento das sessões.

3.3.3.3 Fechamento

Na subdivisão de fechamento, deve-se produzir os seguintes documentos:

- uma coletânea do material produzido durante as sessões;
- o corpo da Especificação de Requisitos do Software;
- os diagramas e fluxos dos casos de uso, no Modelo de Análise;
- uma apresentação da parte já produzida da Especificação de Requisitos do Software para os responsáveis pela decisão de continuar o projeto.

3.3.4 Riscos e benefícios

O JAD é um processo sofisticado, que exige intenso comprometimento dos desenvolvedores e usuários, embora dure normalmente um período bastante curto, se comparado com a duração do projeto. Deve-se enfatizar junto ao cliente os benefícios que o uso do JAD pode trazer, de forma que este libere o respectivo pessoal com a necessária disponibilidade.

Alguns problemas podem comprometer seriamente a eficácia do JAD:

- não participação das pessoas que desempenham papéis-chaves nos processos de uso do produto;
- participação de pessoas não comprometidas com o produto (observadores não devem ser permitidos, ou devem transformar-se em participantes integrais);
- número excessivo de participantes (o número máximo recomendado varia de 8, para grupos iniciantes, a 15, para grupos experientes).

Caso o cliente não libere seu pessoal para participar com a dedicação necessária, o JAD não é viável. Muitas das técnicas mencionadas anteriormente ainda podem ser usadas, mas deve-se deixar claros para o cliente os riscos de que seja produzida uma especificação insatisfatória e de que o levantamento

dos requisitos tome mais tempo do que o devido. Deve-se deixar especialmente claro para o cliente e os usuários que o tempo adicional gasto com a Engenharia de Requisitos não poderá ser recuperado durante o desenvolvimento, e que cada dia de atraso da especificação representa no mínimo um dia de atraso do projeto.

A literatura relata muitos benefícios dos JADs bem realizados. [Jones94] relata, para a técnica de JAD, um retorno típico de investimento da ordem de 200% para o primeiro ano de uso, e 1.000% ao final de quatro anos. Relata também que JAD e prototipagem são técnicas sinérgicas, que podem reduzir as mudanças de requisitos a um nível abaixo de 5%, na maioria dos projetos. [McConnell96] relata reduções de 20% a 60% no tempo e esforço gastos no levantamento de requisitos. Isso se traduz em uma redução de 10% a 30% no tempo total de projetos típicos.

Os seguintes pontos são particularmente importantes quanto ao sucesso das técnicas de JAD:

- o líder das sessões deve ser treinado e experiente;
- os gerentes com poder de decisão devem estar comprometidos com o JAD;
- os participantes devem assistir a todas as sessões em tempo integral;
- as sessões devem ser conduzidas fora das instalações do cliente, em horários reservados na agenda de todos os participantes, sem direito a telefonemas ou interrupções;
- os participantes devem estar preparados, entendendo perfeitamente os procedimentos e os objetivos do JAD;
- ao final do JAD, o cliente deve receber informação realista quanto ao prazo e esforço restantes do projeto.

O JAD de levantamento de requisitos dá melhores resultados em projetos de aplicações em áreas bem entendidas. Caso contrário, corre-se o risco de que as discussões sobre os processos e modelos de negócio dominem a reunião, perdendo-se de vista a questão dos requisitos do software. Esse problema é particularmente sério quando os usuários provêm de grupos diferentes dentro da organização cliente, que não tenham normalmente oportunidade de discutir entre si a modelagem de negócios. Se os processos de negócio forem realmente complexos ou pouco conhecidos, é preferível organizar JADs específicos para resolvê-los primeiro.

3.4 Relacionamento com os clientes

3.4.1 *Importância*

Segundo pesquisa citada em [McConnell96], o envolvimento dos usuários é um dos fatores mais importantes para o sucesso de um projeto. As três principais causas de atraso, estouro de custos ou redução de funções dos projetos foram identificadas como sendo falta de informação de/sobre os usuários, requisitos incompletos e mudanças de requisitos. Essas causas podem ser tratadas através de técnicas de relacionamento com os clientes.

Um bom relacionamento com os clientes contribui para acelerar os projetos, já que o atrito com os clientes é fonte de ineficiência e erros. Segundo [Jones94], ocorrem atritos com os clientes em pelo menos metade dos projetos pagos por administração e mais de dois terços dos projetos de preço fixo. O atrito tende a crescer muito com o tamanho do projeto, tornando-se particularmente grave em contratos acima de US\$100.000,00.

Além disso, o bom relacionamento melhora a velocidade **percebida** pelos clientes. Muitas vezes a visibilidade do progresso de um projeto é mais importante para inspirar confiança nos clientes do que a velocidade propriamente dita.

Muitos dos problemas dos projetos de software são gerados por expectativas irreais, principalmente quanto aos prazos. Um levantamento mostrou que cerca de 10% de todos os projetos são cancelados por causa de expectativas irreais quanto ao prazo ou a produtos [McConnell96]. A aceitação de expectativas irreais, principalmente quanto a prazos, é um dos erros mais clássicos que é possível cometer.

3.4.2 *Fontes de problemas*

Os seguintes fatores contribuem para perdas de eficiência dos projetos, decorrentes de problemas de relacionamento com os clientes:

- falta de entendimento, por parte do cliente, da necessidade de boas práticas de engenharia de software, como planos e revisões;
- atrasos em avaliações e decisões críticas, por parte dos clientes;
- existência de múltiplos contatos da parte do cliente, sem que se saiba exatamente quem está autorizado a decidir.

Segundo [Jones94], as principais causas de atritos com os clientes são:

- Por culpa do fornecedor:
 - promessas de prazo inexecutáveis;
 - custos artificialmente baixos;
 - falta de competências requeridas pelo tipo de projeto;
 - baixa qualidade dos produtos;
 - compromissos não cumpridos;
 - relatórios inadequados ou inexatos sobre o progresso do projeto.
- Por culpa do cliente:
 - exigência de prazos impossíveis;
 - exigência de novos requisitos, sem alterações de prazo e custo;
 - não formulação de requisitos de qualidade e critérios de aceitação;
 - falta de acompanhamento do progresso do projeto.

Algumas das situações de maior risco são aquelas em que o cliente ou usuário:

- não sabe exatamente o que quer;
- não quer se comprometer com requisitos escritos;
- insiste em alterações de requisitos, sem mudança de prazos e custos;
- não se comunica bem com os desenvolvedores (e vice-versa);
- não participa de revisões;

- não entende o processo de desenvolvimento do software;
- tem receios em relação ao produto.

O risco geralmente aumenta no caso de usuários sem sofisticação técnica. Novos usuários podem representar novos riscos. Faz parte do relacionamento com os clientes identificar esses riscos, e monitorá-los ao longo do projeto.

3.4.3 *Práticas orientadas para o cliente*

No levantamento dos requisitos, o maior problema é identificar todos os requisitos reais, e somente eles. Frequentemente os requisitos levantados:

- não contêm todos os requisitos reais;
- entram em conflito com os requisitos reais;
- não são suficientemente profundos;
- são vagos, permitindo-se interpretações diferentes por usuários e desenvolvedores.

A Figura 32, inspirada em [McConnell96], ilustra a diferença entre requisitos reais e levantados.

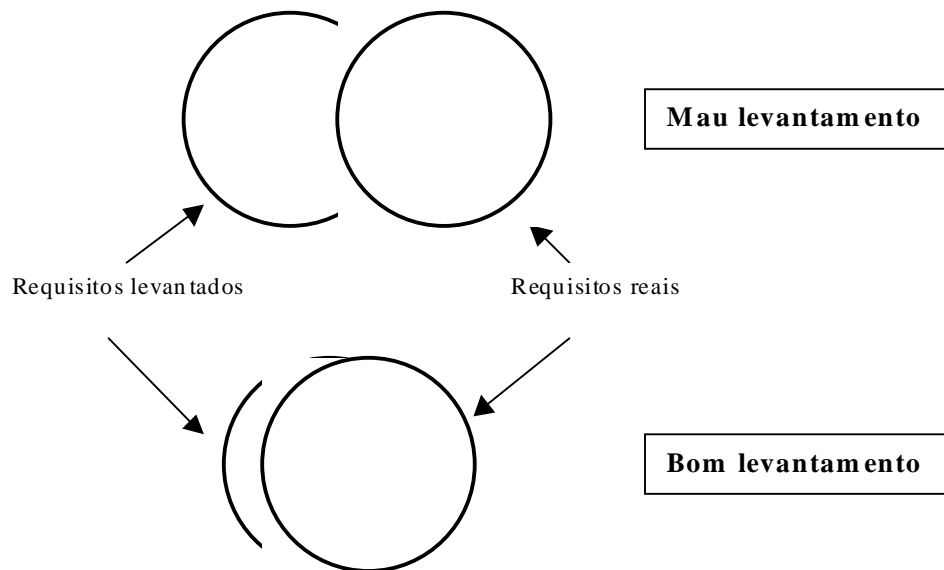


Figura 32 – Requisitos reais versus levantados

O JAD é uma técnica que contribui, de forma comprovada, para melhorar a qualidade dos requisitos levantados. São também úteis as técnicas da área de avaliação de produtos, como levantamentos de satisfação, registros de uso dos produtos (por exemplo, gravados em vídeo) e reuniões de grupos de foco.

Algumas técnicas de planejamento facilitam o relacionamento com os clientes:

- seleção de modelos incrementais de ciclo de vida;
- identificação de quem é o cliente real, com poder de decisão (nem sempre é um usuário);
- uso de métodos eficientes de comunicação com os clientes;
- criação de situações em que todos os lados ganham;

- gestão dos riscos, inclusive análise e monitoração.

Quanto às fases de desenho e implementação, as seguintes práticas contribuem para o bom relacionamento com os usuários:

- técnicas de desenho e implementação que reduzam os custos de modificações ocasionais (desenho robusto e extensível, código bem documentado etc.);
- pontos de controle frequentes e visíveis para o cliente.

O desenvolvimento incremental, baseado em um desenho arquitetônico robusto, seguido de liberações bem planejadas, atende a esses requisitos. As próprias liberações se constituem em resultados intermediários concretos, muito mais eficazes do que relatórios de progresso no que diz respeito ao convencimento dos clientes.

Análise

1 *Princípios*

1.1 **Objetivos**

O fluxo da Análise visa aos seguintes objetivos:

- modelar de forma precisa os conceitos relevantes do domínio do problema;
- verificar a qualidade dos requisitos obtidos através do fluxo de Requisitos;
- detalhar esses requisitos o suficiente para que atinjam o nível de detalhe adequado aos desenvolvedores.

No Praxis, os métodos de Análise são baseados na Tecnologia Orientada a Objetos, resultando em um Modelo de Análise do Software, expresso na notação UML. As principais referências sobre Análise Orientada a Objetos são [Booch94], [Booch96], [Booch+97], [Booch+99], [Jacobson94], [Jacobson+94a], [Jacobson+99], [Love93], [Quatrani98], [Rumbaugh+99], [Schneider98], [Taylor90] e [White94].

O Modelo de Análise deve conter os detalhes necessários para servir de base para o desenho do produto, mas deve-se evitar a inclusão de detalhes que pertençam ao domínio da implementação e não do problema. Quando se usa um Modelo de Análise orientado a objetos, os requisitos funcionais são tipicamente descritos e verificados através dos seguintes recursos de notação:

- Os casos de uso descrevem o comportamento esperado do produto como um todo. Os diagramas de casos de uso descrevem os relacionamentos dos casos de uso entre si e com os atores, enquanto os fluxos descrevem os detalhes de cada caso de uso.
- As classes representam os conceitos do mundo da aplicação que sejam relevantes para a descrição mais precisa dos requisitos. Os diagramas de classes mostram os relacionamentos entre estas, e as especificações das classes descrevem os respectivos detalhes.
- As realizações dos casos de uso mostram como objetos das classes descritas colaboram entre si para realizar os principais roteiros que podem ser percorridos dentro de cada caso de uso.

1.2 **Objetos e classes**

O foco da Análise é a modelagem dos conceitos presentes no domínio do problema. Nas metodologias de modelagem orientadas a objetos, as entidades do domínio do problema são representadas por **objetos**. Objetos podem ser vistos como estruturas de dados encapsuladas por procedimentos. Os campos das estruturas de dados são os **atributos** do objeto, e os procedimentos são as respectivas **operações**. Os objetos interagem entre si trocando **mensagens**, que são invocações das operações.

Objetos similares são agrupados em **classes**. Na maioria dos casos, a Análise dos Requisitos está mais interessada nas classes do que em objetos específicos de determinadas classes. Os diagramas de interação, entretanto, representam exemplos de **interações entre objetos de certas classes**.

Na UML, um objeto é representado por um retângulo, onde o nome do objeto é sublinhado. Quando um objeto aparece em um diagrama UML, podem acontecer as seguintes situações:

- Um objeto pode ter classe indeterminada. Esta é uma situação transitória que pode acontecer durante a análise, mas deve ser resolvida.
- Um objeto pode ter denominação própria e pertencer a determinada classe. Isso pode acontecer com objetos que têm um significado especial dentro do modelo. O nome da classe é separado do nome do objeto por um sinal de dois pontos.
- Um objeto pode ser anônimo, representando uma instância genérica de determinada classe. É o caso mais comum. O campo à esquerda dos dois pontos fica vazio.

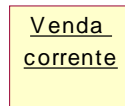


Figura 33 - Objeto sem classe determinada

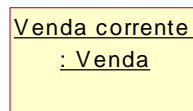


Figura 34 - Objeto com indicação da respectiva classe

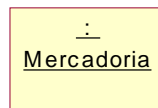


Figura 35 - Objeto anônimo

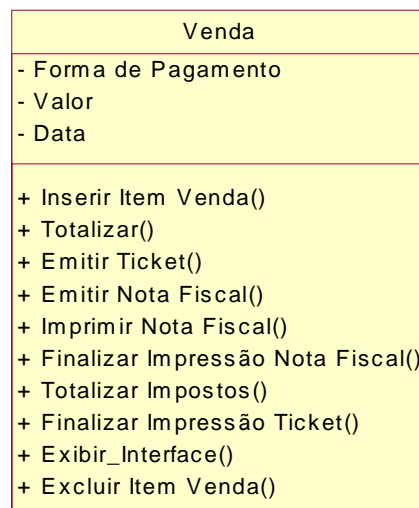


Figura 36 - Representação de classe

Na UML, a classe é representada por um retângulo dividido em três compartimentos, que contêm respectivamente o nome da classe, os atributos (chamados em algumas linguagens de variáveis, propriedades ou membros de dados) e as operações (chamados em algumas linguagens de métodos ou membros de função).

Para maior clareza nos diagramas, pode-se suprimir cada um dos compartimentos de atributos e operações, ou deixar de mostrar determinados atributos ou operações. São opcionais também a indicação da visibilidade por caracteres ou ícones, a assinatura (lista de argumentos e tipo de retorno)

das operações, e o tipo e valor padrão dos atributos. Normalmente, não é necessário chegar a esse nível de detalhe no Modelo de Análise.

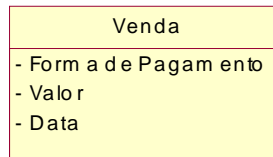


Figura 37 - Representação de classe com supressão das operações

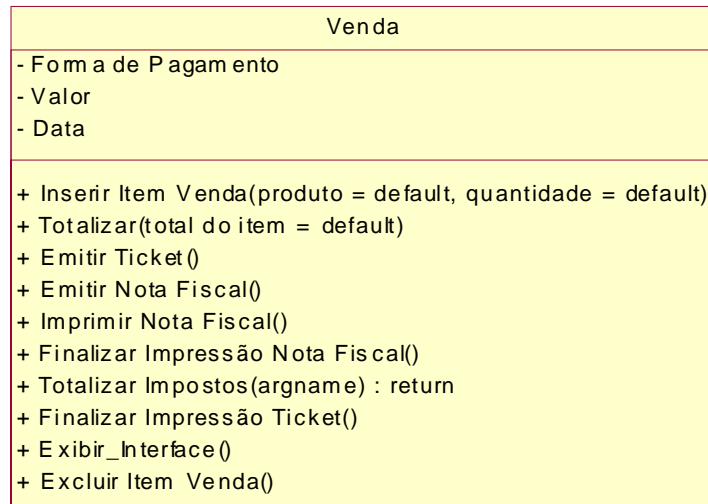


Figura 38 - Representação de classe com detalhamento das assinaturas das operações

1.3 Uso de notações alternativas

O Praxis é essencialmente baseado em notações orientadas a objetos para análise, desenho e implementação, e, em particular, na notação UML. O uso de outras notações (por exemplo, por imposição de um cliente) requereria considerável adaptação de grande parte do processo. A tabela a seguir mostra a correspondência entre os artefatos da notação UML e algumas outras notações de grande difusão. Para maiores informações sobre notações estruturadas de análise, vide [Davis93] e [Pressman95], entre outros.

Notação orientada a objetos (UML)	Notações estruturadas
Casos de uso	Português estruturado
Diagramas de atividade	Fluxogramas, DFD
Diagramas de estado	Fluxogramas, variantes de diagramas de estado
Diagramas de classes	Modelos E-R
Diagramas de interação	DFD, DSSD, JSD, SADT
Especificações de classes e relacionamentos	Dicionários de dados

Tabela 57 – Notações estruturadas correspondentes à UML

2 Atividades

2.1 Visão geral

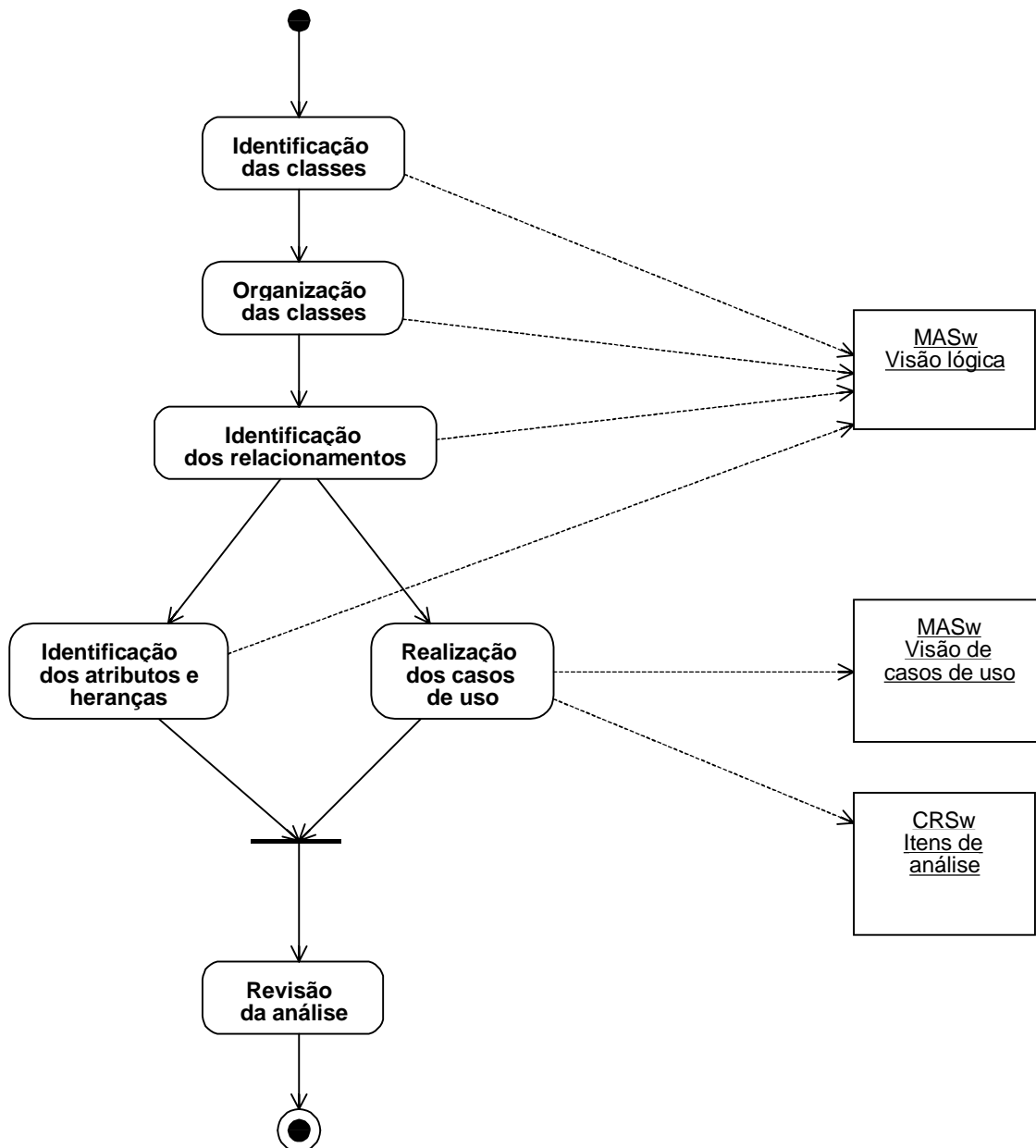


Figura 39 – Atividades e artefatos do fluxo de Análise

A Análise principia pela "**Identificação das classes**", na qual são analisados os fluxos dos casos de uso e outros documentos relevantes em relação ao produto desejado. Os conceitos candidatos a classes são localizados, e filtrados de acordo com vários critérios. Prossegue com a "**Organização das classes**", que organiza as classes em pacotes lógicos (agrupamentos de classes correlatas), e lhes atribui os estereótipos de entidade, fronteira e controle, dependendo do papel que desempenham no modelo. A "**Identificação dos relacionamentos**" determina os relacionamentos de vários tipos que podem existir entre os objetos das classes identificadas. Todas essas atividades alimentam a visão lógica do Modelo de Análise, que contém as especificações e diagramas de classes.

Em seguida, a "**Identificação dos atributos e heranças**" levanta os atributos de análise, isto é, propriedades que fazem parte do conceito expresso pela classe. Essa informação também vai para a

visão lógica. Mais ou menos em paralelo, a "**Realização dos casos de uso**" verifica os fluxos dos casos de uso, representando-os através de diagramas de interação. Estes mostram como os fluxos são realizados através de trocas entre mensagens dos objetos das classes encontradas. Isso ajuda a localizar imprecisões e outros tipos de problemas nos fluxos, e permite definir as operações das classes de análise. Essa informação vai para a visão de casos de uso do Modelo de Análise; serve também para determinar a amarração entre as classes de análise e os requisitos, registrando-se essa informação no Cadastro dos Requisitos.

Finalmente, a "**Revisão da análise**" valida o esforço de Análise e o correspondente esforço de Requisitos. Podem acontecer várias revisões informais, individuais ou em grupo (por exemplo, dentro de um JAD). No final da iteração de Análise dos Requisitos, o Praxis prevê uma revisão técnica formal.

2.2 Detalhes das atividades

2.2.1 *Identificação das classes*

2.2.1.1 Identificação das classes chaves

Nessa tarefa, deve ser feita a identificação das classes chaves, que são as classes iniciais do modelo. Uma técnica básica consiste em procurar os substantivos existentes nos fluxos dos casos de uso e outros requisitos do produto. Na pesquisa dos substantivos, devem ser observados os seguintes detalhes,:

- eliminar aspectos de implementação e informações irrelevantes quanto à missão do produto;
- resolver ambigüidades da linguagem;
- considerar também locuções verbais, desde que equivalentes a substantivos;
- considerar que substantivos podem não resultar em classes, mas em objetos, relacionamentos ou atributos de classes;
- permanecer no nível lógico, não incluindo detalhes de interfaces, arquivos, estruturas de dados etc.;
- permanecer dentro do escopo do produto, evitando classes não conexas com a missão deste.

Outros critérios podem ser usados para filtrar as classes:

- identificar coisas tangíveis e papéis que estas desempenham;
- identificar objetos que são necessários para completar os casos de uso;
- identificar as responsabilidades, o conhecimento e as ações providas por cada classe;
- listar as classes que colaboram para o cumprimento das responsabilidades.

As **responsabilidades** representam o conhecimento e ações que possibilitam às classes cumprir seu papel nos casos de uso. As **colaborações** representam outras classes que colaboram para o cumprimento das responsabilidades das classes já descobertas. Uma técnica de levantamento que é fácil e de larga utilização é baseada nos **cartões CRC**⁷. Estes cartões são usados pelos participantes da sessão de modelagem (por exemplo, em um JAD) para lançar propostas de classes à medida que os

⁷ CRC vem de Classes-Responsabilidades-Colaborações.

casos de uso vão sendo discutidos. O pequeno tamanho do cartão obriga a ser o mais sintético possível na descrição das classes candidatas.

Nome da classe	
Responsabilidades	Colaborações

Figura 40 - Exemplo de cartão CRC

As abstrações candidatas que não forem consideradas como classes devem ser registradas à parte. Posteriormente, elas poderão se transformar em relacionamentos ou atributos.

Por exemplo, a Figura 41 mostra a versão inicial do fluxo da Operação de Venda de um sistema de informatização de mercearia. O nome do produto aparece em negrito, e os nomes dos atores em itálico. Os substantivos simples ou compostos aparecem sublinhados.

<p>O <i>caixeiro</i> faz a <u>abertura</u> da <u>venda</u>.</p> <p>O <i>caixeiro</i> registra os <u>itens vendidos</u>, informando a <u>identificação</u> e a <u>quantidade</u> do item.</p> <p>O Merci totaliza a venda para o <u>cliente da mercearia</u>.</p> <p>O <i>caixeiro</i> encerra a venda.</p> <p>O Merci emite o <u>ticket de caixa</u> para o cliente da mercearia.</p> <p>O <i>caixeiro</i> registra a <u>forma de pagamento</u>.</p> <p>O Merci faz a <u>baixa</u> no <u>estoque</u> das <u>mercadorias</u> vendidas.</p>
--

Figura 41 - Exemplo de fluxo de caso de uso

Os substantivos descobertos são os seguintes: abertura, venda, item vendido, identificação, quantidade do item, cliente da mercearia, ticket de caixa, forma de pagamento, baixa, estoque, mercadoria. Faz-se então a análise mostrada na Tabela 58. As classes candidatas são representadas na Figura 42.

Classe candidata	Análise
abertura	operação
venda	provável classe
item vendido	provável classe, melhor descrita como Item de Venda
identificação	atributo de Item de Venda
quantidade	atributo de Item de Venda
cliente da mercearia	entidade fora do escopo do produto
ticket de caixa	relatório (entidade de implementação)
forma de pagamento	atributo de Venda
baixa	operação
estoque	conjunto das mercadorias cadastradas, sendo uma possível classe
mercadoria	provável classe

Tabela 58 - Exemplo de análise de classes candidatas

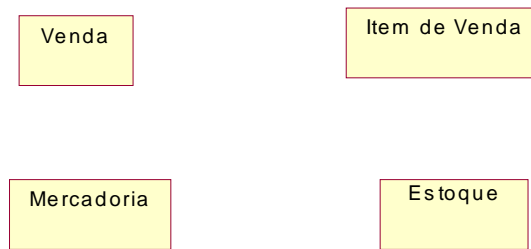


Figura 42 - Classes candidatas encontradas

Quando se tem um modelo de Entidades - Relacionamentos disponível, as entidades desse modelo correspondem naturalmente às classes. Geralmente serão **classes persistentes**, isto é, classes que sobrevivem a cada execução do programa, por serem guardadas em bancos de dados ou arquivos.

2.2.1.2 Especificações das classes

A denominação das classes identificadas é um passo importante da análise. Deve-se escolher para as classes nomes significativos, isto é, substantivos singulares, com ou sem adjetivo, que melhor caracterizem as abstrações. Devem-se evitar nomes vagos, assim como nomes reservados da própria metodologia (classe, tipo etc.).

A documentação de cada classe, incluída em sua especificação, deve conter:

- uma definição clara e concisa da classe;
- uma lista de responsabilidades e colaborações da classe;
- uma lista de regras e restrições aplicáveis;
- possíveis exemplos.

<p>Descrição:</p> <p>armazena a informação relativa a um item de uma venda.</p> <p>Responsabilidades:</p> <p>comandar baixa no estoque; calcular impostos; imprimir linha de ticket e da nota fiscal.</p> <p>Colaborações:</p> <p>Venda, Mercadoria</p> <p>Regras e restrições:</p> <p>Cada Item de Venda corresponde a uma linha do ticket de caixa e da nota fiscal. Todo Item de Venda deve corresponder a uma mercadoria no estoque.</p> <p>Exemplos:</p> <p>seis cervejas Rottenbeer em lata; duas caixas de pregos tamanho 2.</p>

Figura 43 - Exemplo de documentação de classe

Ao longo da análise a especificação das classes será completada com outros aspectos relevantes:

- operações necessárias para cumprir as responsabilidades;
- atributos necessários para cumprir as responsabilidades;

- relacionamentos com as classes colaboradoras;
- eventualmente, arquivos ou páginas da Web com informação adicional.

A Tabela 59 apresenta alguns sintomas de problemas com denominação e documentação de classes, juntamente com a respectiva solução.

Sintoma de problema	Solução
Classes com diferentes nomes e documentação parecida.	Combinar as classes.
Classes com documentação muito longa.	Dividir a classe.
Classe difícil de denominar ou documentar.	Aprofundar a análise..

Tabela 59 - Sintomas de problemas na documentação das classes

2.2.1.3 Término da atividade

Ao fim dessa atividade, devem estar:

- identificadas as classes iniciais;
- batizadas e definidas as classes, indicando-se suas responsabilidades e colaborações;
- anotadas as regras ou restrições sobre as classes;
- registradas as abstrações que não foram aprovadas como classes, mas continuam como candidatas a outros elementos de modelagem.

São os seguintes os resultados dessa atividade:

- um ou mais diagramas de classes, contendo as classes descobertas;
- uma especificação de classe para cada classe descoberta;
- um registro de outras abstrações já descobertas.

Segue-se, na Figura 44 um exemplo das classes iniciais que poderiam ser encontradas na análise do sistema de informatização de mercearia.

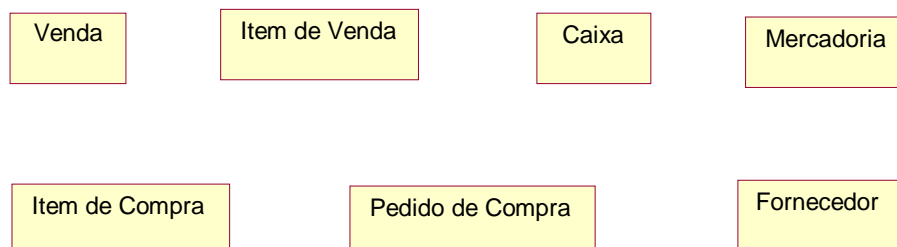


Figura 44 – Classes chaves de um sistema de informatização de mercearia

2.2.2 Organização das classes

2.2.2.1 Modos de organização

Na modelagem de problemas reais, atinge-se facilmente a marca de dezenas ou até centenas de classes. A UML contém alguns recursos de notação para facilitar a organização dos modelos. Esses recursos permitem dividir as classes em grupos significativos, sem alterar a semântica do modelo.

Os **pacotes lógicos** são agrupamentos de elementos de um modelo. No modelo de análise, eles podem ser utilizados para formar grupos de classes com um tema comum. Na Figura 45, os pacotes lógicos Compras, Vendas e Administração são usados para agrupar classes especializadas em relação a esses aspectos do funcionamento da mercearia.

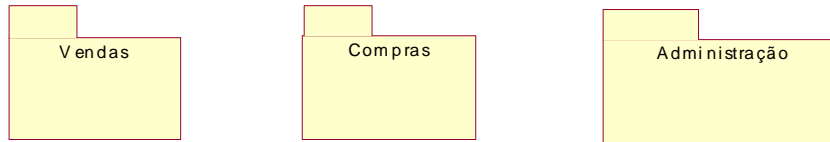


Figura 45 - Exemplos de pacotes lógicos

Os **estereótipos** são extensões de elementos do modelo. Podem ser usados para denotar especializações significativas de classes. Os atores, por exemplo, podem ser tratados pelas ferramentas de modelagem como classes estereotipadas. Os estereótipos podem ser indicados através de ícones próprios, ou incluindo-se o nome do estereótipo em aspas francesas (aqui representadas por << >>).

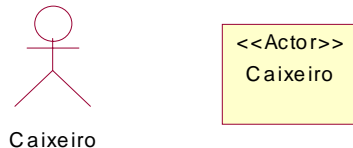


Figura 46 - Representações alternativas do estereótipo "Ator"

Jacobson ([Jacobson94], [Jacobson+99]) propõe a divisão das classes do Modelo de Análise de acordo com os seguintes estereótipos.

- **Entidades** ("entity") – modelam informação persistente, sendo tipicamente independentes da aplicação. Geralmente são necessárias para cumprir alguma responsabilidade do produto, e freqüentemente correspondem a entidades de bancos de dados.
- **Fronteiras** ("boundary")– tratam da comunicação com o ambiente do produto. Modelam as interfaces do produto com usuários e outros sistemas, e surgem tipicamente de cada par ator – caso de uso.
- **Controles** ("control") – coordenam o fluxo de um caso de uso complexo, encapsulando lógica que não se enquadra naturalmente nas responsabilidades das entidades. São tipicamente dependentes de aplicação.

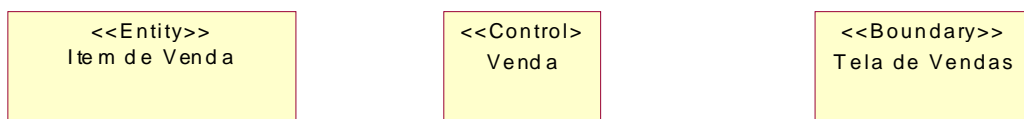


Figura 47 - Exemplo de classes de análise com estereótipos textuais

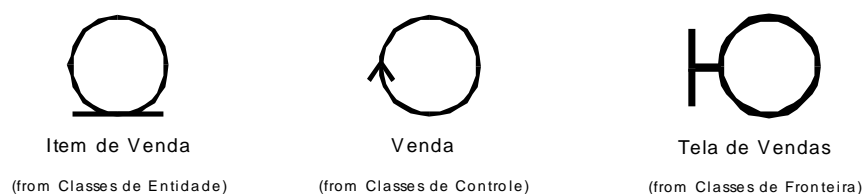


Figura 48 - Exemplo de classes de análise com estereótipos icônicos

Para simplificar, usaremos geralmente a notação estereótipos textuais para as classes de análise⁸. Geralmente, a maioria das classes localizadas através dos métodos descritos para a atividade "Identificação das classes" corresponde a classes <<Entidade>>. Por isso, as classes de <<Fronteira>> e <<Controle>> serão aqui tratadas com maior detalhe.

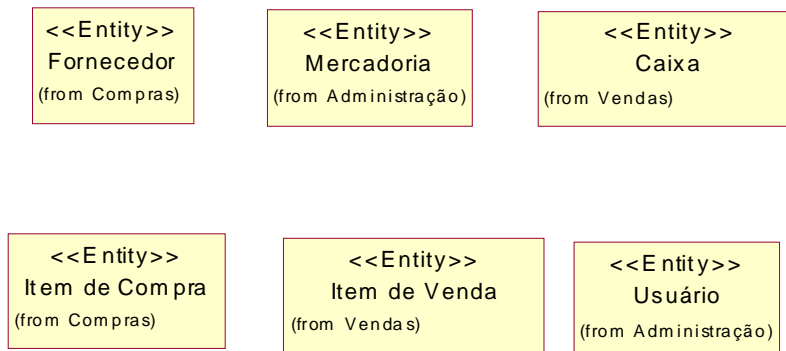


Figura 49 - Exemplos de classes de entidade

2.2.2.2 Classes de fronteira

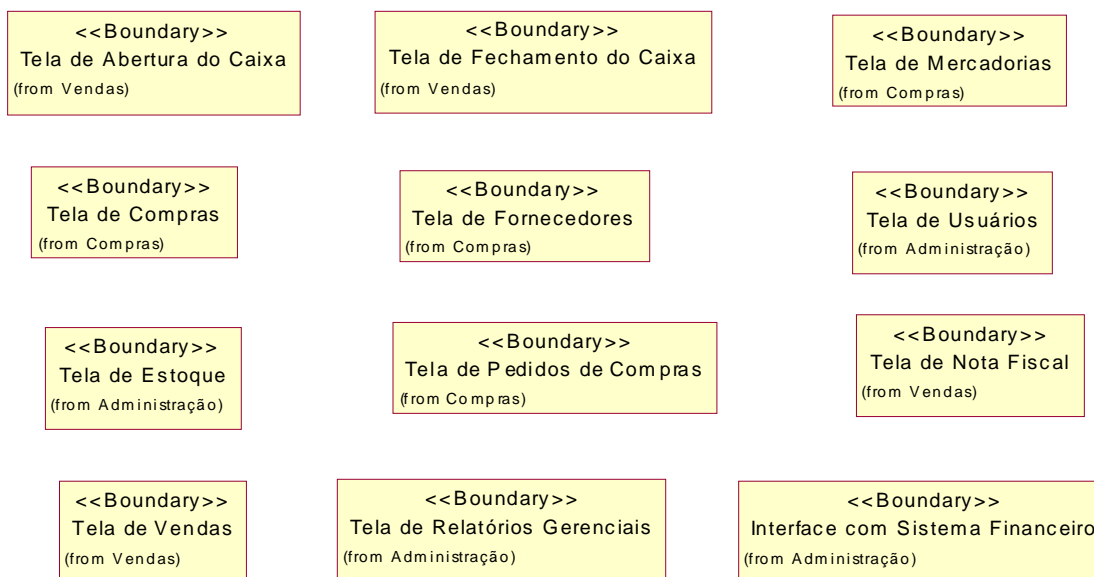


Figura 50 – Exemplo de classes de fronteira

As **classes de fronteira** representam as interfaces de usuário que serão desenvolvidas dentro do produto. Normalmente estão associadas a relacionamentos entre atores e casos de uso. As interfaces gráficas de usuário e as interfaces com outros sistemas geralmente são fontes de classes de fronteira. Nem sempre os analistas mostram as classes de fronteira, pois elas podem ser consideradas implícitas no relacionamento entre atores e as demais classes (principalmente de controle). Entretanto, o uso das classes de fronteira no modelo de análise facilita posteriormente o desenho das interfaces de usuário.

⁸ E também porque, na época de redação deste texto, não se dispunha de uma ferramenta capaz de desenhar os estereótipos icônicos com boa qualidade...

2.2.2.3 Classes de controle

As **classes de controle** realizam funções que não pertencem naturalmente a nenhuma das classes de entidade. Um caso típico é a geração de relatórios. Espalhar esse tipo de funções entre classes de entidade não é uma solução robusta; é preferível concentrá-las em uma classe específica, que unirá os objetos que colaboram para produzir o resultado desejado.

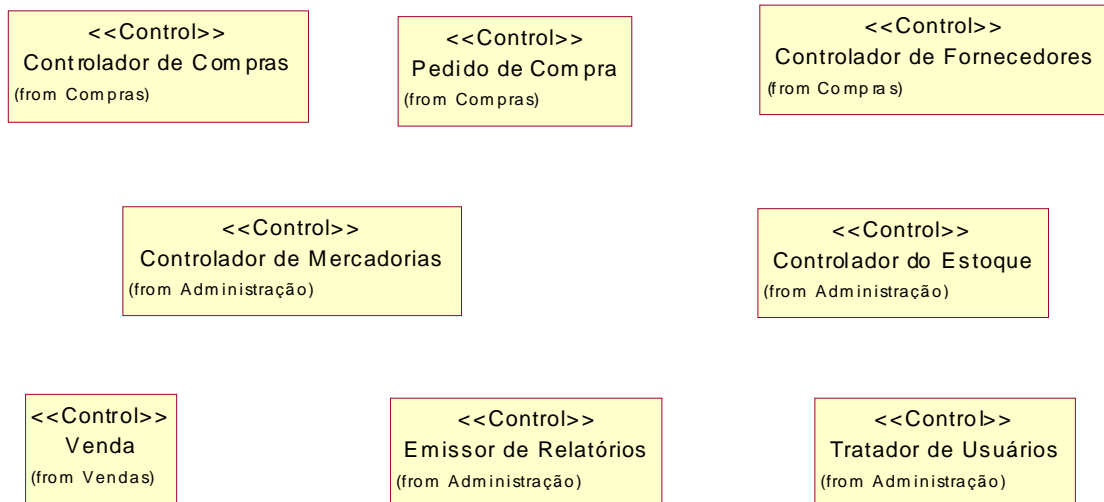


Figura 51 – Exemplo de classes de controle

Um caso particularmente importante de classes de controle é representado pelas classes que coordenam a realização de casos de uso. Utilizando-se classes de controle do caso de uso, elas podem encapsular aspectos de lógica que são específicos da aplicação em questão, permitindo deixar nas classes de entidade apenas aspectos que são inerentes a cada conceito modelado. Com isso, as classes de entidade se tornam mais reaproveitáveis em outras aplicações. Por outro lado, concentrar a realização do caso de uso apenas nas classes de controle e tratar as classes de entidade como meras estruturas de dados é regredir à modelagem funcional, desprezando a filosofia de orientação a objetos.

Na maioria dos casos, uma classe de controle corresponde a um caso de uso. Dependendo da complexidade do fluxo, esse número pode aumentar, definindo-se por exemplo classes de controle para coordenar um subfluxo mais complexo, um algoritmo ou uma regra de negócio importante. Por outro lado, um caso de uso muito simples pode ser coordenado por uma classe de fronteira.

2.2.2.4 Término da atividade

Ao fim dessa atividade, devem estar:

- identificadas as classes de entidade, fronteira e controle;
- organizadas as classes em pacotes lógicos, de acordo com os respectivos temas.

São os seguintes os resultados dessa atividade:

- um ou mais diagramas de classes, contendo as classes já estereotipadas;
- um conjunto de pacotes lógicos que particiona as classes de forma adequada;
- pelo menos um diagrama de pacotes lógicos.

2.2.3 Identificação dos relacionamentos

2.2.3.1 Introdução

Nessa atividade devem ser definidos os relacionamentos entre as classes descobertas. Os relacionamentos ajudam a filtrar e refinar as classes. Os principais relacionamentos são os **relacionamentos de associação**, que denotam as dependências semânticas entre classes que sejam relevantes para o modelo. Eles correspondem, normalmente, aos relacionamentos encontrados nos diagramas de Entidade - Relacionamento.

São também definidos os relacionamentos de agregação, que são associações que expressam conceitos de “todo-parte”, lógicos ou físicos.

2.2.3.2 Definição dos relacionamentos de associação

As associações expressam relações bidirecionais de dependência semântica entre duas classes. Associações entre classes indicam que os objetos de uma das classes têm conhecimento dos objetos da outra. Por exemplo, um pedido é emitido por um cliente e um cliente tem diversos pedidos pendentes.

Associações entre classes indicam que existe a possibilidade de comunicação direta entre os respectivos objetos. Isso significa que faz parte das responsabilidades de um objeto de uma das classes determinar os objetos correspondentes da outra classe. Normalmente, existirão em cada classe operações para cumprir essa responsabilidade.



Figura 52 - Relacionamento de associação

2.2.3.3 Especificação das associações

A especificação das associações deve incluir o seu nome, descrição e possíveis restrições. Normalmente, devem ser definidas as **multiplicidades** dos **papéis** dos participantes do relacionamentos. Em certos casos, é útil batizar também os papéis, assim como especificar vários detalhes desses.

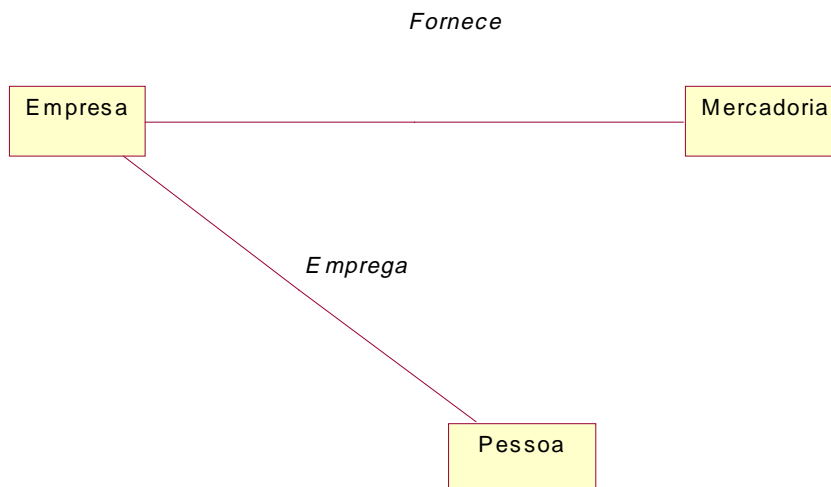


Figura 53 - Relacionamento com nome

Os nomes das associações devem ser simples e significativos. A dificuldade de encontrar nomes simples para os relacionamentos geralmente indica falha de modelagem. Recomenda-se usar um substantivo que descreva bem a semântica do relacionamento. Pode-se também usar um verbo, desde que esteja claro qual classe é sujeito e qual classe é objeto desse verbo. Uma convenção habitual é batizar o relacionamento de modo que ele seja lido corretamente de cima para baixo ou da esquerda para a direita. Na última versão da UML, um pequeno triângulo pode ser usado para indicar a direção de leitura, caso necessário. Os relacionamentos só devem ser batizados quando o nome contribuir significativamente para o entendimento do modelo.

Normalmente, os relacionamentos são binários. Um relacionamento binário tem dois participantes. Os papéis são denominações que exprimem em que qualidade um objeto de uma das classes do relacionamento se relaciona com um objeto da outra classe.

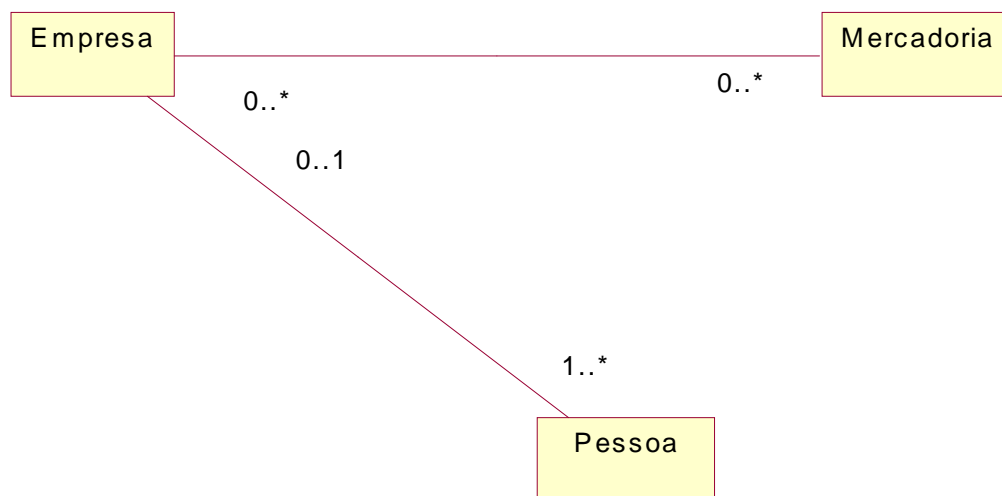


Figura 54 - Relacionamentos com multiplicidades

A multiplicidade de um participante indica quantos objetos de uma classe se relacionam com cada objeto da outra classe. Relacionamentos obrigatórios têm multiplicidade mínima 1. A multiplicidade máxima indica o número máximo de instâncias da classe alvo que podem existir simultaneamente.

Os papéis dos participantes devem ser batizados explicitamente quando participarem de um relacionamento em uma qualidade que não é implícita no respectivo nome da classe. Não é conveniente denominar relacionamentos e papéis cujo significado seja óbvio, dados os nomes das classes.

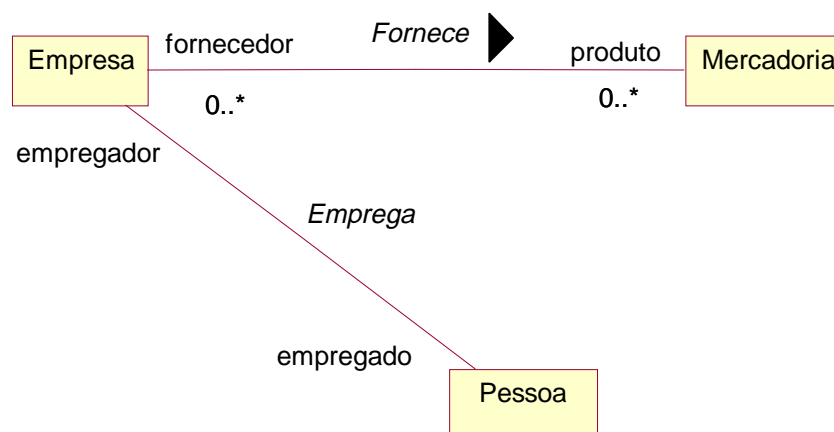


Figura 55 - Relacionamentos com denominação dos participantes

Na Figura 55 indica-se que um objeto da classe "Empresa" participa no papel de "fornecedor" do relacionamento "Fornece", com zero ou mais objetos da classe "Mercadoria", e um objeto dessa classe se relaciona com zero ou mais objetos da classe "Empresa" na qualidade de "produto". Uma "Empresa" pode participar de outros relacionamentos em outras qualidades; por exemplo, no papel de "empregador", em um relacionamento "Emprega" com um objeto da classe "Pessoa".

Os relacionamentos podem ter direção de navegação. Um relacionamento é navegável da classe A para a classe B se, dado um objeto da classe A, consegue-se obter de forma direta (por exemplo, através de uma operação da classe A) os objetos relacionados da classe B. Normalmente, é preferível tratar todos os relacionamentos do Modelo de Análise como bidirecionais, sem restrições de navegação. Estas devem ser introduzidas na fase de desenho.



Figura 56 - Relacionamento com restrição de navegabilidade

Por exemplo, a Figura 56 indica que, dada uma "Mercadoria", é possível localizar diretamente o respectivo "Fornecedor", mas a recíproca não é verdadeira.

2.2.3.4 Relacionamentos avançados

Um **relacionamento de agregação** é uma associação que reflete a construção física ou a posse lógica. Relacionamentos de agregação são casos particulares dos relacionamentos de associação, e só é necessário distingui-los quando for conveniente enfatizar o caráter "todo-parte" do relacionamento. Geralmente um relacionamento de agregação é caracterizado pela presença da expressão "parte de" na descrição do relacionamento, e pela assimetria da navegação.

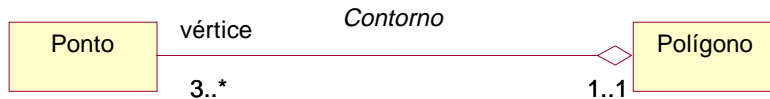


Figura 57 - Relacionamento de agregação

Um tipo mais forte de relacionamento todo-parte é o **relacionamento de composição**. Nesse caso, os objetos da classe parte não têm existência independente da classe todo. A Figura 58 indica que o "centro" de um "Círculo" não tem existência independente deste, enquanto que a Figura 57 indica que cada "ponto" existe independentemente do "Polígono" ao qual serve de "vértice".



Figura 58 - Relacionamento de composição

Uma associação pode ser reflexiva. Uma auto-associação indica um relacionamento entre objetos de mesma classe que desempenham diferentes participações.

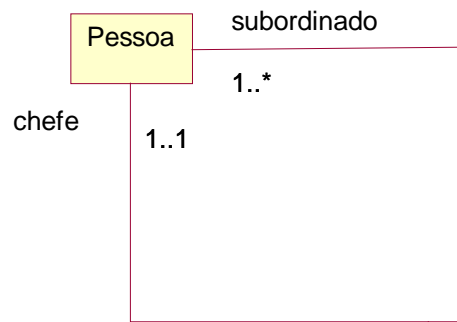


Figura 59 - Associação reflexiva

Os relacionamentos podem ser de natureza complexa. Em certos casos, um relacionamento é mais bem explicado através de uma classe de associação, que exprime atributos e até operações que são propriedades do relacionamento como um todo e não de cada participante isoladamente.

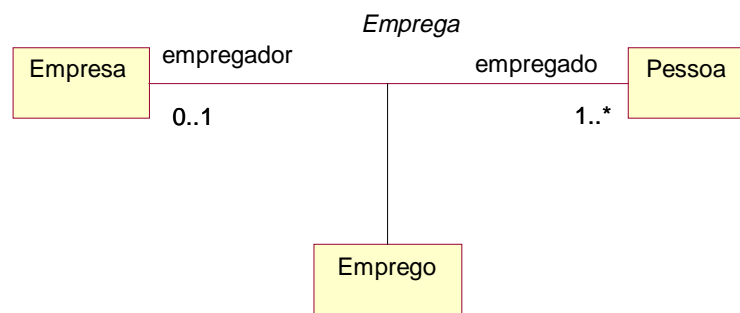


Figura 60 - Classe de associação

Um qualificador é um atributo que restringe um relacionamento através de uma seleção. Na maioria dos casos, cada valor do qualificador está associado a uma única instância da classe alvo. Na Figura 61, o qualificador "número de conta" restringe a participação de objetos da classe "Pessoa" no relacionamento. Um "número de conta" pode estar associado a zero ou um cliente (contas conjuntas não são modeladas).

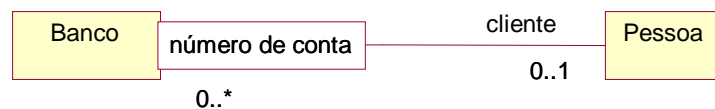


Figura 61 - Relacionamento com qualificador

2.2.3.5 Término da atividade

Ao fim dessa tarefa, devem estar:

- identificados os principais relacionamentos entre classes chaves;
- descobertas novas classes, tais como as classes de associação, pela análise dos relacionamentos;
- especificados todos os relacionamentos, inclusive com multiplicidades.

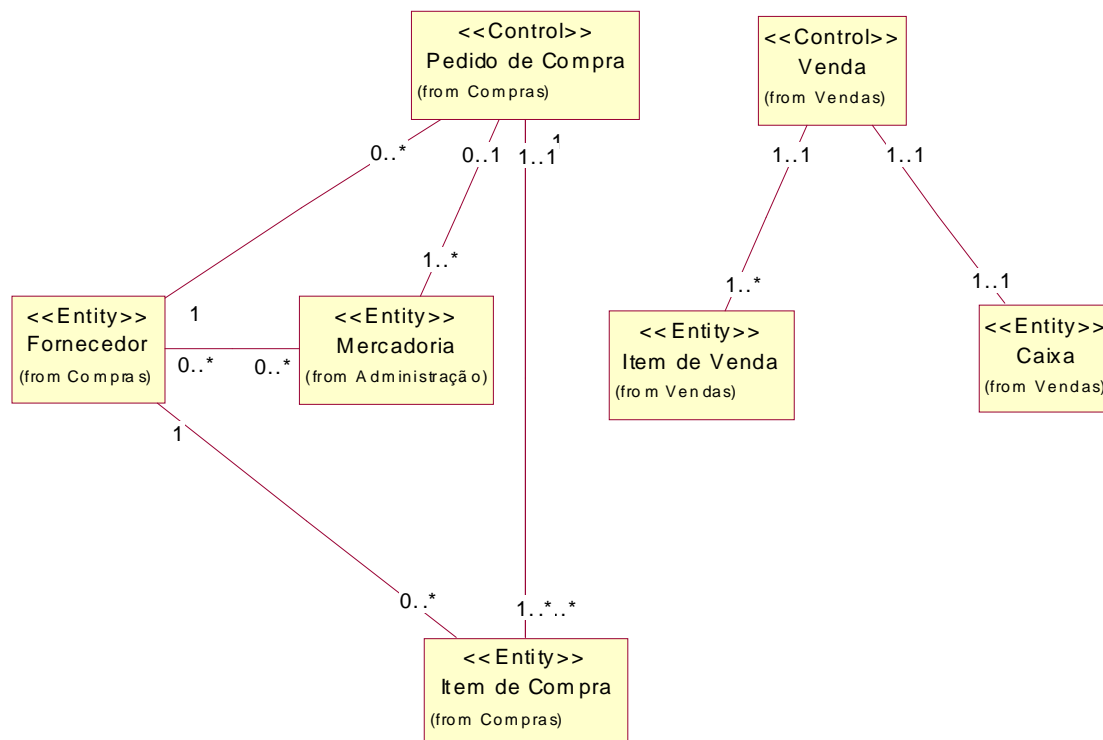


Figura 62 - Exemplo de diagrama de classes com relacionamentos

São os seguintes os resultados dessa atividade:

- diagramas de classes com relacionamentos anotados, inclusive com as multiplicidades que se julgar necessário explicitar;
- especificações de classe para todas as classes, inclusive as novas;
- especificações de relacionamento para todos os relacionamentos.

2.2.4 Realização dos casos de uso

2.2.4.1 Introdução

Nessa atividade devem ser definidas as operações de cada classe. As operações devem ser suficientes para se fazer a **realização** dos casos de uso, isto é, a tradução dos fluxos destes em termos de interações entre objetos das classes achadas. Essas interações são expressas através de **diagramas de interação**.

2.2.4.2 Mensagens e operações

Em modelos orientados a objetos, as mensagens representam os mecanismos de interação entre estes. Um objeto só pode receber mensagens que correspondam à invocação de uma operação da respectiva classe. Durante a execução da modelagem, os diagramas podem conter, em caráter provisório, mensagens não associadas a operações de alguma classe. Entretanto, ao término da análise todas as mensagens têm de ser resolvidas em termos de operações de alguma classe.

Uma mensagem tem as seguintes partes:

- receptor - o objeto que recebe a mensagem;

- operação - a função requisitada do receptor;
- parâmetros - os dados para a operação.

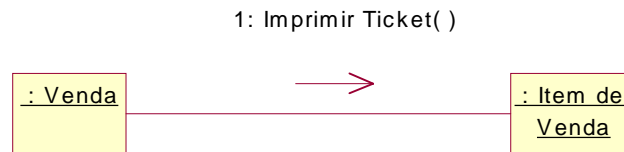


Figura 63 - Exemplo de mensagem

2.2.4.3 Diagramas de interação

2.2.4.3.1 Introdução

O método principal de determinação das operações consiste em construir diagramas de interação para realizar cada caso de uso, e determinar as operações necessárias para executar esses diagramas. Cada caso de uso pode ter várias realizações, correspondendo a diferentes **roteiros**, ou seja, seqüências de ações que exprimem o comportamento.

Por exemplo, um roteiro primário pode corresponder ao fluxo principal do caso de uso, e podem ser detalhados roteiros secundários para combinações importantes de subfluxos e fluxos alternativos. Durante a Análise, tipicamente, são elaborados os roteiros para 80% dos fluxos primários e para os principais fluxos secundários.

Para serem úteis, os diagramas de interação devem ser simples. Normalmente, em cada diagrama as ações são realizadas em ordem seqüencial. Lógica simples de seleção e repetição pode ser expressa através de anotação nos diagramas. Lógica mais complexa é mais bem representada por vários diagramas, correspondentes a diferentes roteiros.

A UML prevê dois tipos de diagramas de interação:

- diagramas de seqüência;
- diagramas de colaboração.

2.2.4.3.2 Diagramas de seqüência

Os diagramas de seqüência enfatizam o ordenamento temporal das ações. Eles são construídos de acordo com as seguintes convenções:

- linhas verticais representam os objetos;
- setas horizontais representam as mensagens passadas entre os objetos;
- rótulos das setas são os nomes das operações;
- a posição na vertical mostra o ordenamento relativo das mensagens;
- o diagrama pode ser complementado e esclarecido por anotações.

<p>O <i>Gestor de Compras</i> seleciona a mercadoria.</p> <p>O Merci verifica se existe algum pedido pendente que contenha esta mercadoria.</p> <p>Se não houver pedido pendente contendo a mercadoria a ser excluída:</p> <p> O Merci desvincula a mercadoria dos fornecedores (os fornecedores não mais fornecerão a mercadoria que esta sendo excluída).</p> <p> O Merci faz a remoção da mercadoria.</p> <p>Se houver pedido pendente contendo a mercadoria a ser excluída</p> <p> O Merci emite uma mensagem de erro.</p>

Tabela 60 - Exemplo de fluxo de caso de uso

Os diagramas de seqüência são orientados para exprimir, de preferência, o desenrolar temporal de seqüências de ações. É mais difícil representar lógicas de seleção e repetição sem prejudicar a inteligibilidade do diagrama. Os roteiros representam desdobramentos da lógica do caso de uso. É preferível usar diagramas separados para representar roteiros resultantes de diferentes caminhos lógicos. Por exemplo, a lógica contida no fluxo da Figura 64 pode ser descrita através dos diferentes roteiros (Figura 65 e Figura 66). Para simplificar o exemplo, os diagramas não usam classes de fronteira.

<p>O <i>Caixeiro</i> registra itens de mercadoria.</p> <p>O Merci totaliza venda.</p> <p>O <i>Caixeiro</i> registra modo de venda.</p> <p>Se venda a prazo</p> <p> O <i>Caixeiro</i> insere venda em contas a receber.</p> <p>Senão</p> <p> O <i>Caixeiro</i> registra pagamento.</p>
--

Figura 64 – Exemplo de fluxo com lógica de seleção: Operação de Venda

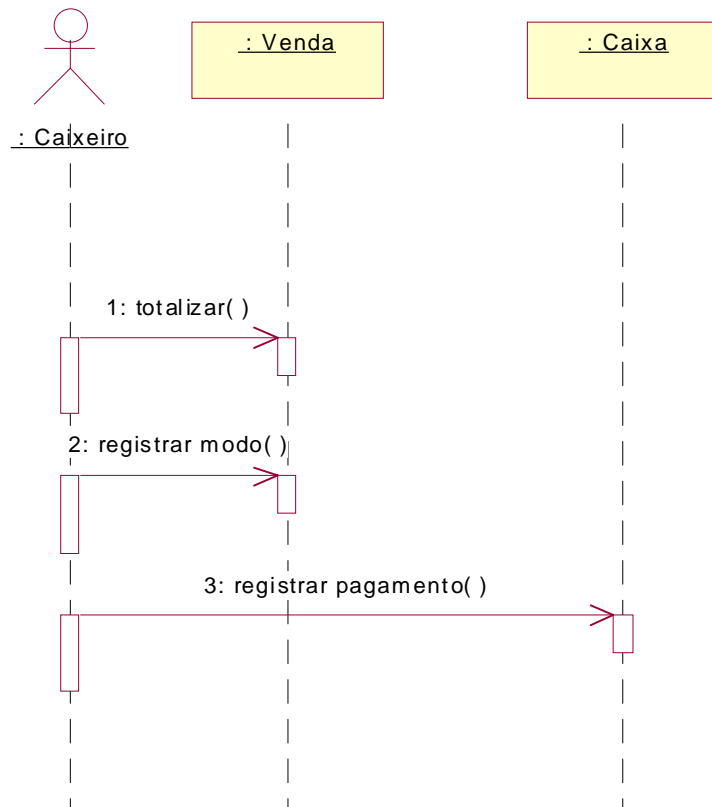


Figura 65 – Exemplo de roteiro alternativo: Operação de Venda à Vista

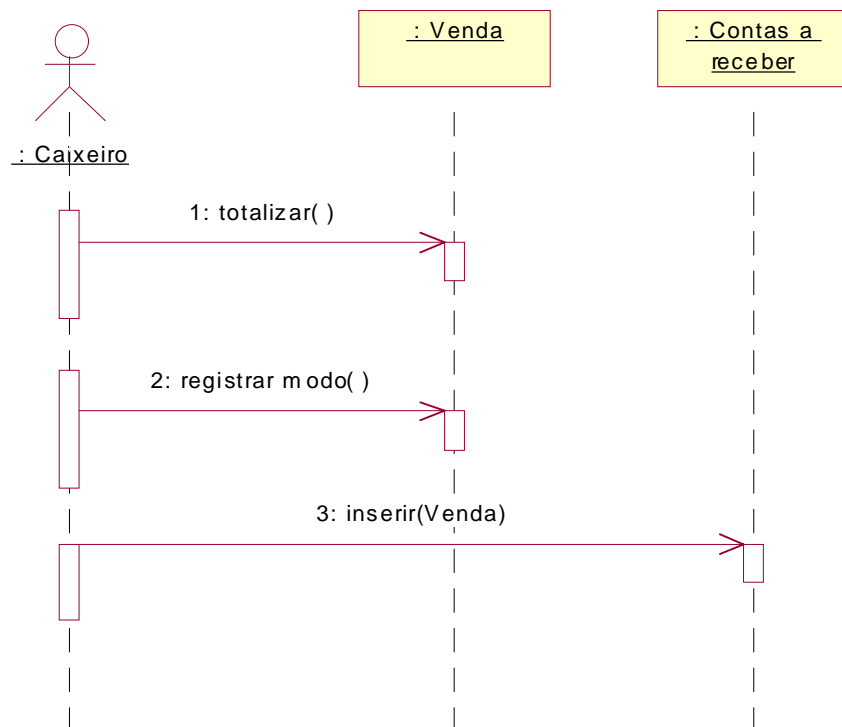


Figura 66 – Exemplo de roteiro alternativo: Operação de Venda a Prazo

Por outro lado, só há necessidade de desenhar diagramas de sequência para os roteiros mais importantes ou mais difíceis. Subfluxos curtos podem ser inseridos em um roteiro, indicando-se a lógica de ativação deles por meio de **expressões de recorrência**:

- [condição] - lógica de seleção;
- *[condição] - lógica de iteração.

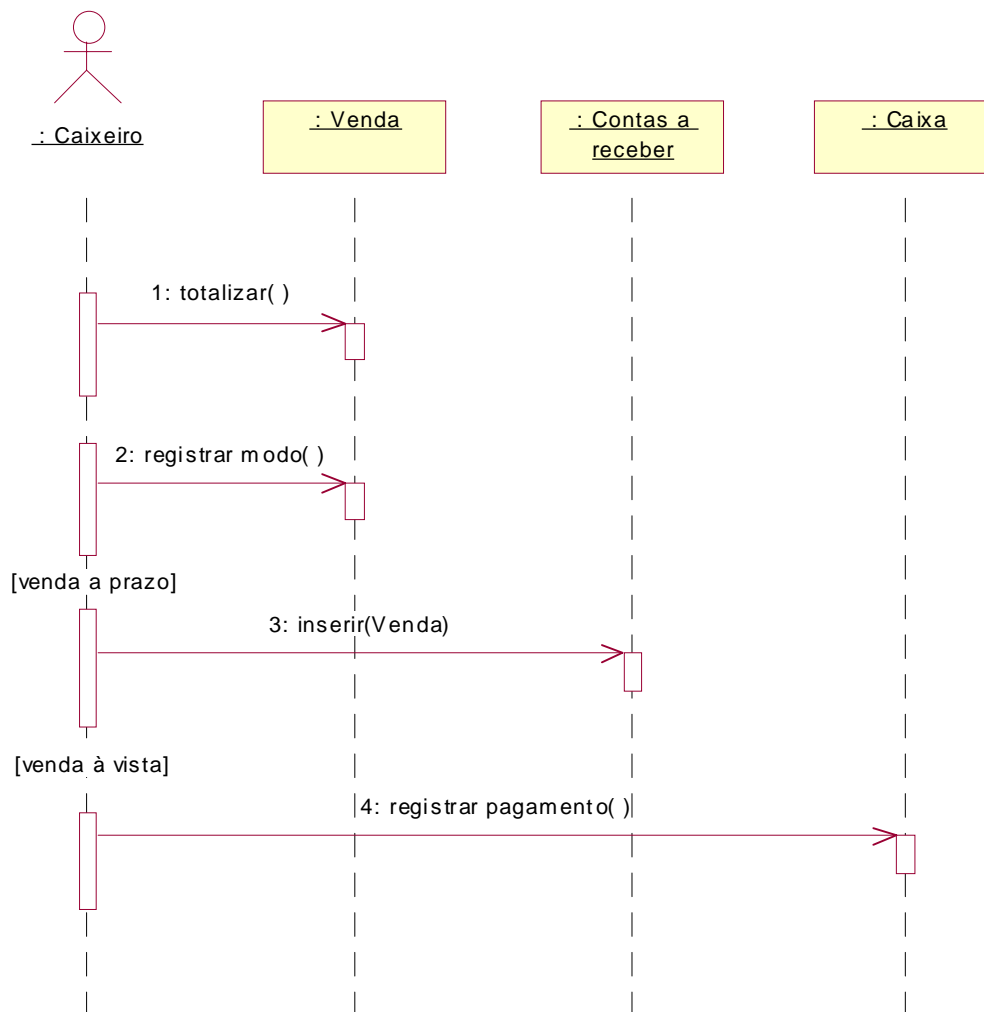


Figura 67 – Representação de lógica por meio de restrições

2.2.4.3.3 Diagramas de colaboração

Os diagramas de colaboração enfatizam os relacionamentos entre os objetos participantes. Eles são construídos de acordo com as seguintes convenções:

- nodos representam os objetos;
- arcos representam as mensagens passadas entre os objetos;
- rótulos dos arcos são os nomes das operações;
- os números de sequência mostram o ordenamento relativo das mensagens;

- anotações podem complementar o diagrama.

Nos diagramas de colaboração a ordenação das mensagens é mostrada apenas pela numeração delas. Alguns preferem utilizar para a Análise os diagramas de colaboração, por facilitarem a descoberta de relacionamentos entre as classes.

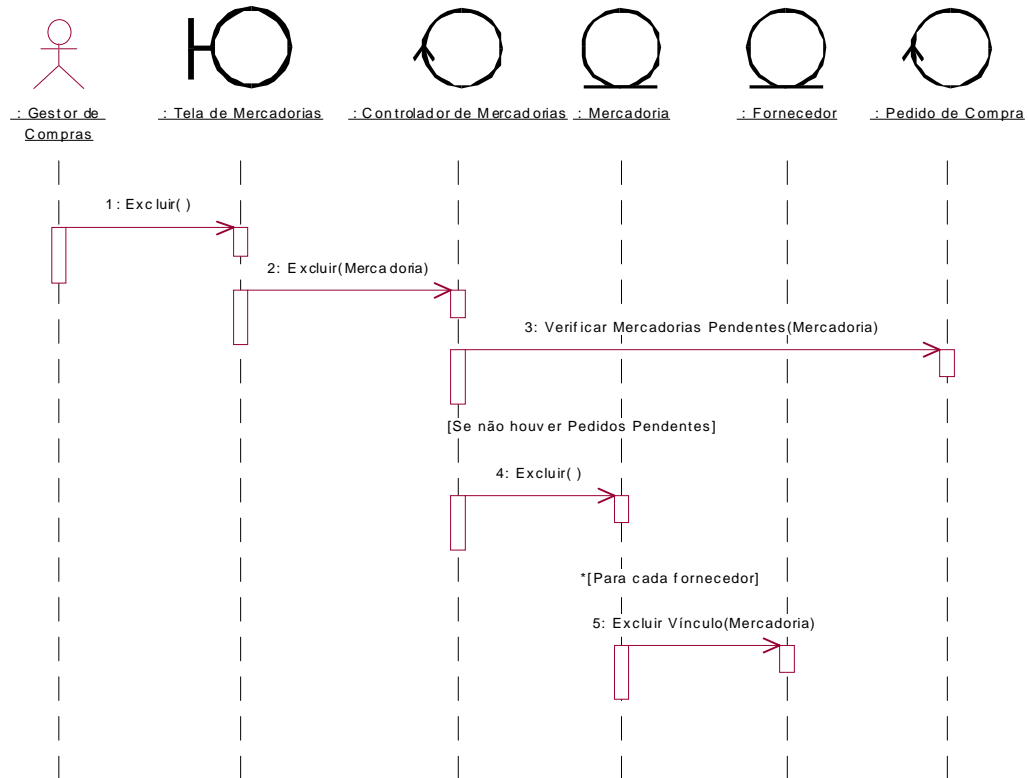


Figura 68 - Realização de fluxo através de diagrama de seqüência

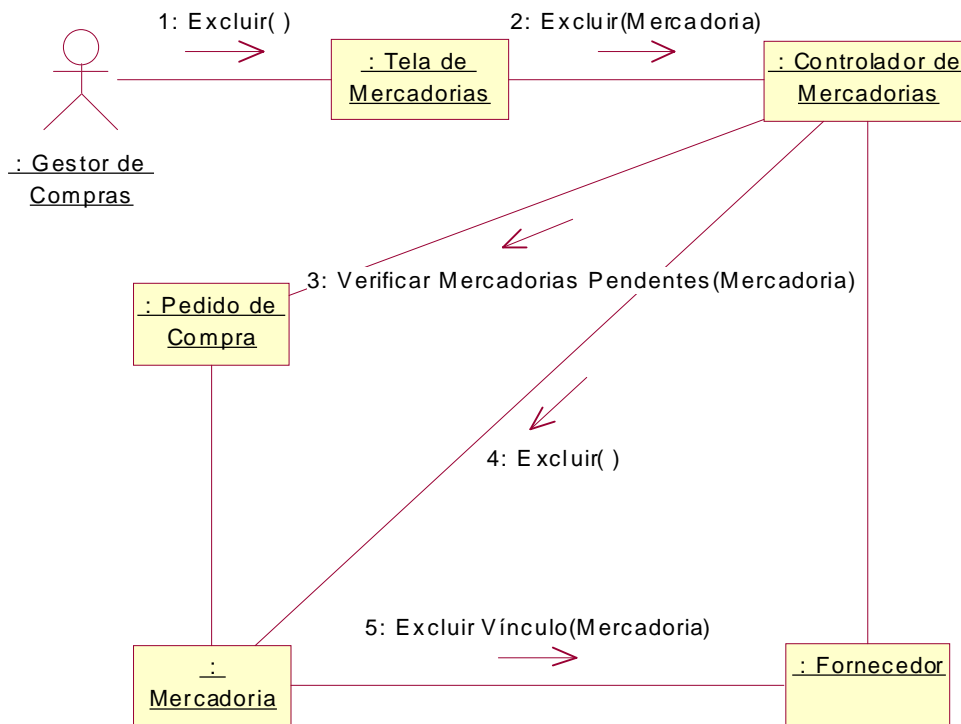


Figura 69 - Realização de fluxo através de diagrama de colaboração

2.2.4.4 Identificação das operações

Todas as mensagens dos diagramas de interação devem ser mapeadas em operações da classe receptora. Quando os diagramas de interação são construídos, deve-se verificar, para cada nova mensagem, se esta correspondente a uma operação já existente da classe receptora. Quando não, nova operação deve ser criada. Se for conveniente, a criação das operações pode ser adiada enquanto se discutem alternativas de realização do caso de uso.

A identificação das operações deve ser completada através da análise das responsabilidades das classes. Por exemplo, se foi identificada uma operação de “Incluir”, é provável que a classe precise de uma operação de “Excluir”, e talvez de uma operação de “Alterar”, com o mesmo argumento. Dadas as responsabilidades de cada classe, deve ser definido um conjunto de operações suficiente para satisfazer essas responsabilidades.

Novos relacionamentos entre classes podem ser descobertos através das operações. A existência de mensagens entre objetos nos diagramas de interação sugere a necessidade de relacionamentos entre as respectivas classes (Figura 70). Outro sintoma de relacionamento é a presença de objetos nas assinaturas das operações. A Figura 71 indica um relacionamento entre “Venda” e “Contas a receber”.

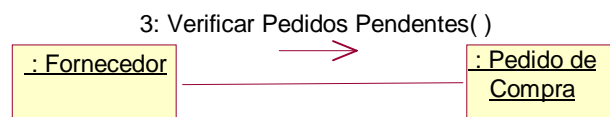


Figura 70 – Existência de mensagens entre objetos

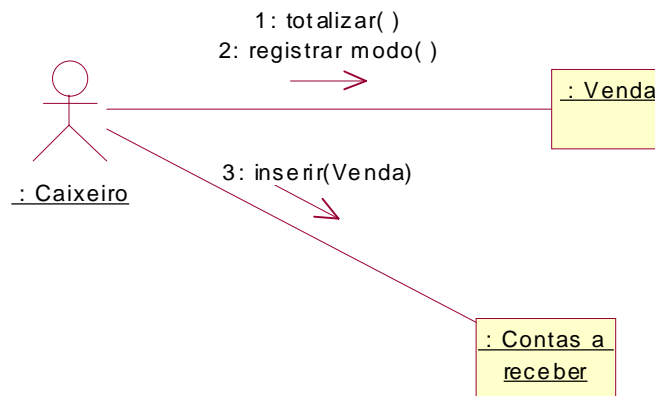


Figura 71 – Presença de objeto em assinatura de operação

2.2.4.5 Especificação das operações

O batismo das operações deve obedecer a regras que facilitem a leitura dos diagramas. Os nomes das operações devem ser verbos cujo sujeito é um objeto da classe receptora. Preferimos usar o imperativo, na forma infinitiva. A descrição de cada operação deve descrever sua funcionalidade, entradas e saídas.

Cada operação deve realizar uma função bem definida, que possa ser sintetizada em uma frase curta. O nome deve refletir o resultado da operação, e não os detalhes de processamento desta.

No modelo de análise, geralmente não é necessário detalhar os argumentos das operações. Caso argumentos sejam necessários para explicar a operação, deve-se evitar excesso de argumentos. Isso geralmente indica a necessidade de partir as operações em outras mais simples. Devem-se evitar argumentos que funcionem apenas como chaves de entrada, isto é, de condição para uma seleção interna à operação. Isso geralmente indica uma falha de modelagem.

2.2.4.6 Cadastramento dos itens de análise

			Item de requisitos																																								
Num	Nome do item	Tipo	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34							
1	Tela de Abertura do Caixa	Cl. fronteira	x																																								
2	Tela de Compras	Cl. fronteira		x																																							
3	Tela de Estoque	Cl. fronteira			x																																						
4	Tela de Fechamento do Caixa	Cl. fronteira				x																																					
5	Tela de Fornecedores	Cl. fronteira					x																																				
6	Tela de Mercadorias	Cl. fronteira						x																																			
7	Tela de Nota Fiscal	Cl. fronteira							x								x																										
8	Tela de Pedidos de Compras	Cl. fronteira								x																																	
9	Tela de Relatórios	Cl. fronteira									x																																
10	Tela de Usuários	Cl. fronteira										x																															
11	Tela de Vendas	Cl. fronteira											x																														
12	Sistema Financeiro	Cl. fronteira																																									
13	Caixa	Cl. entidade																																									
14	Fornecedor	Cl. entidade																																									
15	Item de Compra	Cl. entidade																																									
16	Item de Mercadoria	Cl. entidade																																									
17	Item de Venda	Cl. entidade																																									
18	Mercadoria	Cl. entidade																																									
19	Usuário	Cl. entidade																																									
20	Nota Fiscal	Cl. entidade																																									
21	Pedido de Compra	Cl. controle																																									
22	Venda	Cl. controle																																									
23	Controlador de Mercadorias	Cl. controle																																									
24	Controlador de Estoque	Cl. controle																																									
25	Controlador de Compras	Cl. controle																																									
26	Controlador de Fornecedores	Cl. controle																																									
27	Emissor de Relatórios	Cl. controle																																									
28	Tratador de Usuários	Cl. controle																																									

Figura 72 – Exemplo de rastreamento entre itens de análise e de requisitos

Número	Nome do requisito	Tipo	Importância	Complexidade	Estabilidade
1	Interface de usuário Tela de Abertura do Caixa	Interface	Essencial	Baixa	Média
2	Interface de usuário Tela de Compras	Interface	Essencial	Média	Média
3	Interface de usuário Tela de Estoque	Interface	Desejável	Média	Baixa
4	Interface de usuário Tela de Fechamento do Caixa	Interface	Essencial	Baixa	Média
5	Interface de usuário Tela de Fornecedores	Interface	Essencial	Média	Baixa
6	Interface de usuário Tela de Mercadorias	Interface	Essencial	Média	Alta
7	Interface de usuário Tela de Nota Fiscal	Interface	Desejável	Baixa	Alta
8	Interface de usuário Tela de Pedidos de Compras	Interface	Opcional	Baixa	Baixa
9	Interface de usuário Tela de Relatórios	Interface	Essencial	Baixa	Baixa
10	Interface de usuário Tela de Usuários	Interface	Essencial	Baixa	Média
11	Interface de usuário Tela de Vendas	Interface	Essencial	Alta	Média
12	Interface de usuário Relatório de Estoque Baixo	Interface	Desejável	Média	Baixa
13	Interface de usuário Relatório de Fornecedores	Interface	Desejável	Média	Baixa
14	Interface de usuário Relatório de Mercadorias	Interface	Desejável	Média	Alta
15	Interface de usuário Nota Fiscal	Interface	Essencial	Média	Baixa
16	Interface de usuário Pedido de Compra	Interface	Opcional	Baixa	Baixa
17	Interface de usuário Relação de Pedidos de Compra	Interface	Opcional	Média	Média
18	Interface de usuário Ticket de Venda	Interface	Essencial	Média	Média
19	Interface de software Sistema Financeiro	Interface	Desejável	Média	Média
20	Caso de uso Abertura do Caixa	Caso de uso	Essencial	Baixa	Média
21	Caso de uso Emissão de Nota Fiscal	Caso de uso	Desejável	Média	Média
22	Caso de uso Emissão de Relatórios	Caso de uso	Essencial	Baixa	Baixa
23	Caso de uso Fechamento do Caixa	Caso de uso	Essencial	Baixa	Média
24	Caso de uso Gestão de Fornecedores	Caso de uso	Essencial	Média	Baixa
25	Caso de uso Gestão de Mercadorias	Caso de uso	Essencial	Média	Média
26	Caso de uso Gestão de Pedidos de Compra	Caso de uso	Opcional	Baixa	Baixa
27	Caso de uso Gestão de Usuários	Caso de uso	Essencial	Baixa	Média
28	Caso de uso Gestão Manual de Estoque	Caso de uso	Desejável	Média	Baixa
29	Caso de uso Operação de Venda	Caso de uso	Essencial	Alta	Média
30	Requisito de desempenho Tempo de resposta	Não funcional	Desejável	Baixa	Alta
31	Restrição ao desenho Padrão de Nota Fiscal	Não funcional	Essencial	Média	Alta
32	Restrição ao desenho Expansibilidade	Não funcional	Opcional	Alta	Média
33	Atributo da qualidade Segurança do Acesso	Não funcional	Desejável	Média	Média
34	Atributo da qualidade Apreensibilidade	Não funcional	Desejável	Média	Alta

Figura 73 – Cadastro inicial dos requisitos

A realização dos casos de uso permite determinar quais classes são dependentes de quais requisitos. Essa informação é lançada na Cadastro dos Requisitos do Software, como mostra a Figura 72. Os números dos itens de requisitos correspondem à Figura 73. Comparando-se as duas tabelas, verifica-se que, em geral:

- as classes de fronteira derivam dos requisitos de interfaces externas e dos casos de uso em que participam;
- as classes de controle e de entidade derivam dos casos de uso em que participam;
- os requisitos de interfaces correspondentes a relatórios em geral não geram classes de fronteira, mas são relacionados com as classes participantes dos casos de uso em que são emitidos;
- os requisitos não funcionais específicos de caso de uso são relacionados com as respectivas classes.

2.2.4.7 Término da atividade

Ao fim dessa atividade, devem estar:

- determinadas as operações requeridas para realizar os casos de uso através de diagramas de interação;
- determinados os argumentos de algumas dessas operações, quando necessário para o entendimento do modelo;

- atribuídas as operações às respectivas classes.

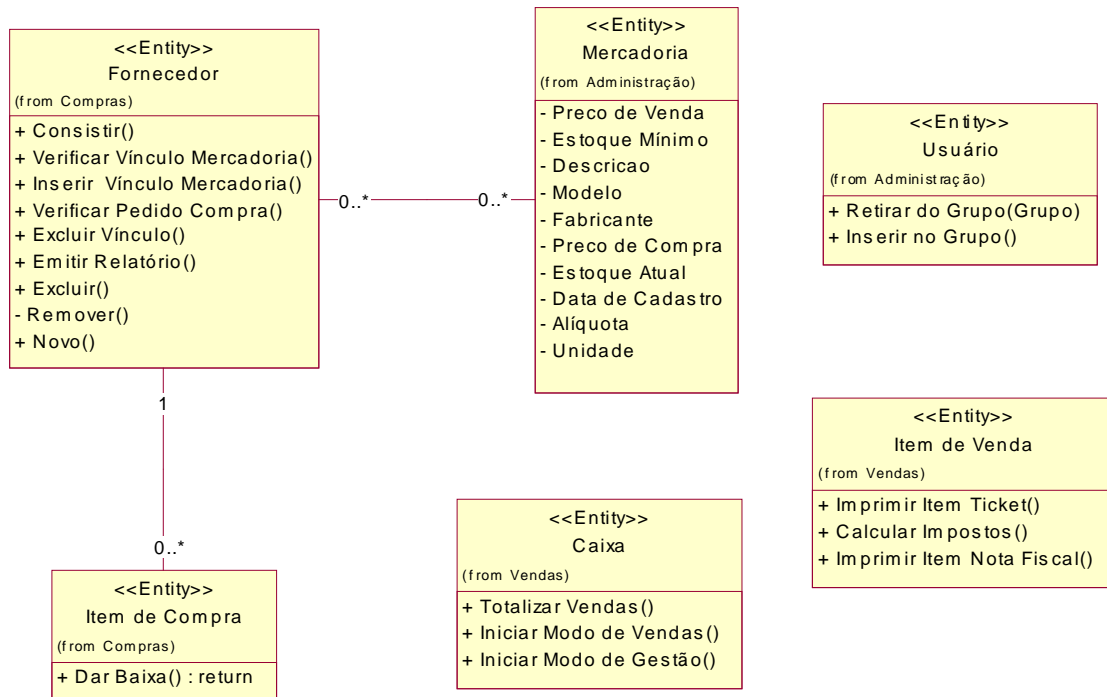


Figura 74 - Diagrama de classes com operações

São os seguintes os resultados dessa atividade:

- diagramas de classes mostrando as operações e os relacionamentos de dependência que se considerar necessário explicitar;
- especificações de classes atualizadas para mostrar as operações;
- diagramas de colaboração e seqüência mostrando as interações entre objetos que realizam os casos de uso;
- especificações dos objetos, quando se julgar necessário incluir no modelo instâncias específicas, ou seja, objetos não anônimos.

2.2.5 Identificação dos atributos e heranças

2.2.5.1 Definição de atributos

Atributos são propriedades que descrevem as classes. Atributos equivalem a relacionamentos de composição em que:

- a classe parte é o tipo do atributo;
- o papel é o nome do atributo.

Atributos e relacionamentos podem ser alternativas para se expressar os mesmos conceitos (Figura 75). A escolha entre atributo e relacionamento deve visar à clareza do modelo. Geralmente atributos são de tipos equivalentes a classes pequenas e reutilizáveis, que representam abstrações de nível superior ao do domínio do problema.

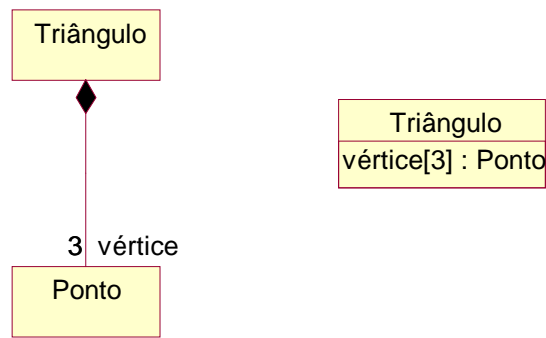


Figura 75 – Equivalência entre atributos e composições

2.2.5.2 Identificação de atributos

Os seguintes métodos devem ser utilizados para a identificação dos atributos:

- Listar as propriedades de uma classe que sejam relevantes para o domínio em questão. Deve-se procurar um compromisso entre objetividade (procurar atender a determinado projeto, com o mínimo custo) e generalidade (permitir a reutilização da classe em outros projetos).
- Localizar, nos documentos dos requisitos, atributos que ainda não tenham sido incluídos nas classes. Os atributos frequentemente são adjetivos ou possessivos que descrevem um nome de classe.
- Evitar, nessa etapa, a inclusão de atributos que só são necessários para a implementação.

Só há necessidade de explicitar os atributos das classes do Modelo de Análise nos casos em que esses atributos:

- representam propriedades relevantes dos objetos do domínio da aplicação;
- representam campos de interfaces externas, no caso de classes de fronteira;
- justificam hierarquias de herança.

2.2.5.3 Definição das heranças

O relacionamento de herança existe entre classes de natureza mais geral (superclasses, classes bases) e suas especializações (subclasses, classes derivadas). A identificação de superclasses é feita quando são localizados atributos ou operações comuns a um grupo de classes. Nas subclasses devem ser localizadas as operações ou atributos que só se aplicam a um subconjunto das instâncias de uma classe.

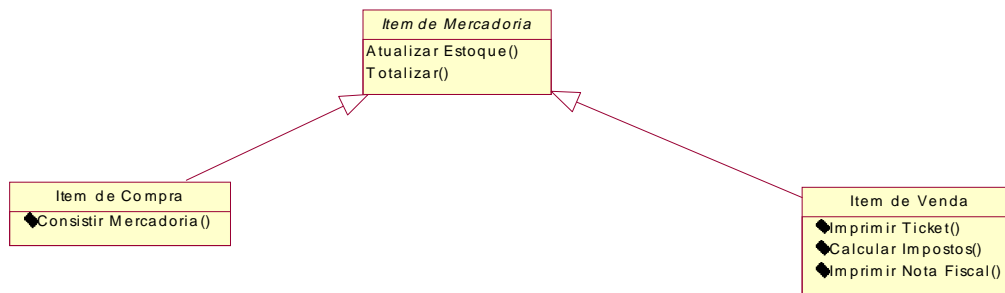


Figura 76 - Exemplo de relacionamentos de herança

2.2.5.4 Término da atividade

Ao fim dessa atividade, devem estar:

- localizados, em sua maioria, os atributos de dados das classes principais;
- descobertos e atribuídos os tipos desses atributos;
- descobertos alguns relacionamentos que se transformam em classes porque têm atributos de dados;
- determinadas as subclasses e superclasses.

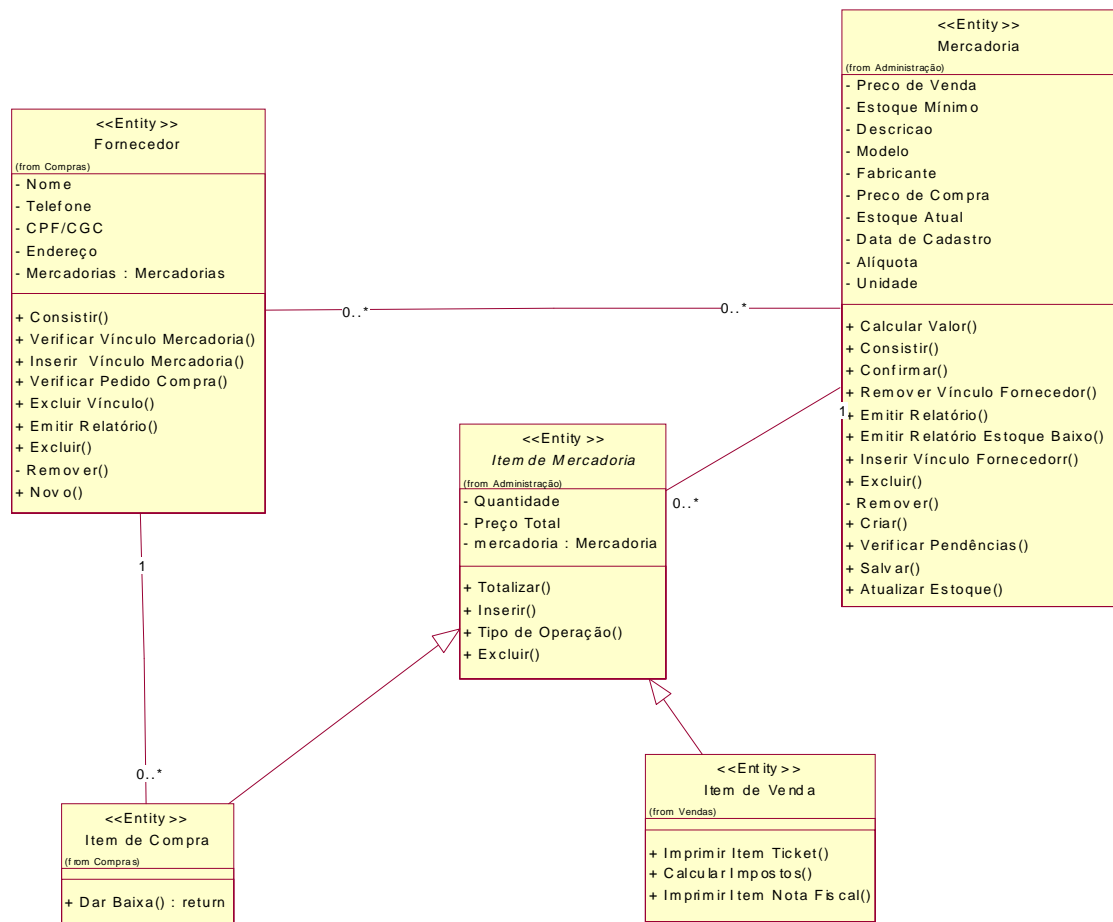


Figura 77 – Diagrama de classes com atributos e relacionamentos de herança

São os seguintes os resultados dessa atividade:

- diagrama aumentado de classes, mostrando:
 - a estrutura de herança;
 - novas classes e relacionamentos.
- especificações para as novas classes e relacionamentos.

2.2.6 *Revisão da análise*

2.2.6.1 Forma da revisão

O Praxis prevê, ao final da última iteração da fase de Elaboração, uma revisão técnica formal, em que são analisados em conjunto o Modelo de Análise e a Especificação dos Requisitos do Software. Para facilitar essa revisão, o padrão para Especificação dos Requisitos do Software prevê que a informação de suporte conterá listagens dos diagramas e especificações do Modelo de Análise. De preferência, a ferramenta de modelagem deve permitir a extração dessa informação em formato com qualidade para publicação, evitando a extração manual tediosa e propensa a erros.

Um roteiro de revisão deve ser utilizado para que a revisão técnica tenha uma cobertura completa. Esse roteiro deve pedir a checagem de todos os itens previstos no padrão para Especificação dos Requisitos do Software e conter uma lista de conferência do Modelo de Análise. Um exemplo de roteiro de revisão do Modelo de Análise está contido no padrão para Especificação de Requisitos de Software.

2.2.6.2 Validação do Modelo de Análise

A validação do modelo deve incluir os seguintes passos:

- percorrer os casos de uso, verificando se existem caminhos para realizar todas as operações necessárias, dentro dos diagramas de colaboração;
- verificar, para cada campo de saída requerido nas interfaces, se existe uma maneira de obter esse campo, através de alguma colaboração entre classes.

A confecção do Modelo de Análise prossegue até que:

- tenham sido associadas a cada classe todas as operações realizadas por elas ou sobre elas;
- seja entendido o que cada operação deve fazer e que outras classes estão envolvidas.

Normalmente, a fase de análise envolverá várias iterações das etapas descritas anteriormente. Alterações na Especificação dos Requisitos do Software podem também revelar-se necessárias. Pode-se considerar a análise encerrada, ao menos provisoriamente, quando:

- foram identificadas todas as classes que representam conceitos do domínio relevantes para o produto em questão;
- foram identificados os relacionamentos relevantes entre cada uma dessas classes;
- foram identificadas, para cada classe, todas as operações realizadas por ela ou sobre ela;
- foram descritas colaborações que realizam todos os casos de uso de fluxo não trivial;
- foram analisadas todas as operações até entender-se o que cada uma deve fazer, e que outras classes são invocadas;
- foram identificados todos os atributos necessários para descrever os conceitos do domínio representados pelas classes.

2.2.6.3 Término da Análise

O fluxo de Análise, como todos os outros, pode ser revisitado em todas as fases. Um ponto importante, entretanto, é o fim da Elaboração, quando devem estar:

- identificadas todas as classes de análise, assim como os relacionamentos entre elas;

- identificada a grande maioria das operações e dos atributos necessários para realizar os casos de uso especificados.

São os seguintes os resultados da Análise:

- um diagrama de classes completo;
- especificações de classes para todas as classes, com a definição de cada classe, juntamente com:
 - seus relacionamentos;
 - seus atributos;
 - suas operações;
 - suas superclasses;
- diagramas de colaboração e seqüência, mostrando a realização dos principais roteiros dos casos de uso do produto.

3 Técnicas

3.1 Introdução

A maioria das técnicas usadas na análise foi apresentada dentro da descrição das atividades desse fluxo. Esta subseção trata de técnicas adicionais que são aplicáveis a várias atividades do fluxo de Análise. Oficinas de análise são utilizadas para realização da análise em reuniões estruturadas, se possível com a participação dos usuários. A documentação do Modelo de Análise envolve aspectos que são tratados aqui com maior detalhe.

3.2 Oficinas de análise

3.2.1 Introdução

A Análise pode ser efetuada usando a técnica de JAD, embora a participação dos usuários não seja tão indispensável quanto no fluxo de Requisitos. Em muitos casos, entretanto, a contribuição dos usuários é essencial para a modelagem dos conceitos do domínio do problema.

A maioria das pessoas não tem maiores dificuldades em entender a parte estática dos modelos de análise (classes, atributos e relacionamentos). Os desenvolvedores deverão avaliar se os usuários do projeto em questão podem ser treinados de forma rápida para ler as realizações dos casos de uso. Note-se que, para participar dos JADs de análise, basta que os usuários tenham razoável capacidade de leitura dos diagramas UML que forem empregados. Essa capacidade pode ser adquirida à medida que cada notação é introduzida.

3.2.2 JADs de Análise

3.2.2.1 Personalização

A subdivisão de personalização é semelhante à subdivisão correspondente do JAD para Requisitos, compreendendo:

- preparação das instalações e material para as sessões de análise;

- treinamento, quanto à técnica de JAD, dos participantes não familiarizados;
- treinamento, quanto à notação UML, dos participantes não familiarizados;
- preparação de formulários e material audiovisual;
- preparação de documentos que deverão ser consultados;
- preparação de software para apresentação e edição dos requisitos;
- preparação de software para modelagem orientada a objetos;
- preparação de software para desenho de interfaces e prototipagem.

3.2.2.2 Sessões

A subdivisão de sessões pode durar de um a muitos dias. Os recursos necessários e a forma de condução das reuniões são semelhantes aos dos JADs de requisitos.

Essa subdivisão abrange tipicamente as seguintes tarefas:

1. abertura – apresentação das finalidades da sessão, agenda e tempos previstos;
2. revisão e refinamento dos resultados da fase de levantamento do JAD;
3. identificação dos principais grupos de dados do produto (classes que fazem parte do modelo de análise do produto);
4. identificação dos relacionamentos entre grupos de dados (relacionamentos entre classes do modelo de análise);
5. identificação das estruturas principais dos grupos de dados do produto (atributos e relacionamentos de herança das classes do modelo conceitual);
6. realização dos casos de uso do produto (diagramas de interação para os principais roteiros dos casos de uso, levado à identificação das operações das classes do modelo de análise);
7. documentação dos problemas e considerações – documentar as opções consideradas e o porquê das decisões tomadas;
8. fechamento das sessões.

3.2.2.3 Fechamento

Na subdivisão de fechamento, devme-se produzir os seguintes resultados:

- uma coletânea do material produzido durante as sessões;
- o Modelo de Análise do Software, completo;
- uma Especificação de Requisitos do Software revisada e completada com a documentação do Modelo de Análise, que deve ser submetida a Revisão Técnica;
- possivelmente, um ou mais protótipos descartáveis do produto;
- uma apresentação da Especificação de Requisitos do Software completa, para os responsáveis pela decisão de dar continuidade ao projeto.

O Plano de Desenvolvimento do Software deve ser elaborado em paralelo com esta subdivisão de fechamento, e apresentado ao cliente juntamente com a Especificação de Requisitos do Software. Isso permite que o cliente avalie os requisitos propostos em conjunto com as estimativas de prazos e custos. Assim, ele terá fundamentos mais concretos para uma decisão de prosseguir com o projeto, ou pedir a reformulação dos requisitos.

3.3 Documentação do Modelo de Análise

3.3.1 Introdução

No Praxis, a documentação do Modelo de Análise deve ser incluída como parte da Informação de Suporte da Especificação dos Requisitos do Software. Isso permite que o grupo de revisão técnica dessa especificação tenha à mão os elementos do Modelo de Análise. Estes elementos são essenciais para validar os fluxos dos casos de uso e outros requisitos.

3.3.2 Diagramas de classes

Recomenda-se incluir na documentação os seguintes diagramas:

- diagramas que mostrem grandes grupos de classes e seus relacionamentos, sem exibir seus atributos e operações;
- diagramas que mostrem, com maiores detalhes, grupos mais coesos de classes (por exemplo, que colaboram para realizar um caso de uso importante, ou um grupo distinto de casos de uso correlatos);
- diagramas que mostrem hierarquias de agregação e de herança.

Os diagramas do Modelo de Análise devem conter o mínimo de informação sobre os relacionamentos que seja necessário para completar a realização dos casos de uso. Para cada relacionamento, devem-se indicar as multiplicidades das classes participantes. A denominação dos relacionamentos e dos papéis das classes participantes só deve ser feita quanto isso contribuir para melhor entendimento do modelo. Restrições que não possam ser expressas através da UML devem ser indicadas por meio de anotações nos diagramas.

3.3.3 Especificações das classes

Deve-se descrever no Modelo de Análise o mínimo de informação necessário para completar o detalhamento dos requisitos, sem antecipar detalhes de desenho, como visibilidade, continência por valor ou referência e aspectos específicos do ambiente de implementação.

Devem-se, por outro lado, incluir no Modelo de Análise todas as operações que correspondem a mensagens dos diagramas de interação. Devem-se incluir aqui apenas operações que contribuem para a realização dos casos de uso. Normalmente, não são incluídas no Modelo de Análise operações de significado óbvio (tais como operadores de igualdade e cópia), ou operações dependentes do ambiente de implementação (tais como construtoras e destruidoras).

Detalhes como tipos e valores iniciais dos atributos e assinaturas das operações só devem ser incluídos quando forem relevantes do ponto de vista de especificação de requisitos. Restrições e requisitos específicos de determinados atributos ou operações devem ser descritos junto a estes. Restrições e requisitos aplicáveis à classe como um todo devem ser descritos no respectivo campo de documentação.

3.3.4 *Realizações dos casos de uso*

As realizações dos casos de uso devem mostrar como as instâncias das classes do Modelo de Análise interagem para realizar os casos de uso, servindo para validar tanto casos de uso quanto classes. Um caso de uso pode ter:

- nenhuma realização (tolerável apenas para casos de uso extremamente simples);
- uma única realização, correspondente a um roteiro que percorre o fluxo principal do caso de uso;
- várias realizações, correspondentes a roteiros importantes que percorrem também subfluxos e fluxos alternativos do caso de uso.

Desenho

1 *Princípios*

1.1 **Objetivos**

O fluxo de Desenho (design ou projeto⁹) tem por objetivo definir uma estrutura implementável para um produto de software que atenda aos requisitos especificados para este. O desenho de um produto de software deve considerar os seguintes aspectos:

- o atendimento dos requisitos não funcionais, como os requisitos de desempenho;
- a definição de classes e outros elementos de modelo em nível de detalhe suficiente para a respectiva implementação;
- a decomposição do produto em componentes cuja construção seja relativamente independente, de forma que eventualmente possa ser realizada por pessoas diferentes, possivelmente trabalhando em paralelo;
- a definição adequada e rigorosa das interfaces entre os componentes do produto, minimizando os efeitos que problemas em cada um dos componentes possam trazer aos demais elementos;
- a documentação das decisões de desenho, de forma que estas possam ser comunicadas e entendidas por quem vier a implementar e manter o produto;
- a reutilização de componentes, mecanismos e outros artefatos, para aumentar a produtividade e a confiabilidade;
- o suporte a métodos e ferramentas de geração semi-automática de código.

1.2 **O modelo de desenho**

O modelo de desenho representa a continuação do modelo de análise, mas geralmente apresenta muito mais detalhes; tipicamente, o modelo de desenho apresenta até cinco vezes mais detalhes que o modelo de análise. A Tabela 61 mostra uma comparação entre os dois modelos; vários aspectos serão detalhados nas subseções seguintes. Uma comparação bem mais detalhada é encontrada em [Jacobson+99].

As estruturas de implementação são introduzidas no modelo de desenho. Mesmo as estruturas do domínio passam a ser descritas de forma mais detalhada quando é necessário resolver questões de implementação. Os casos de uso passam a se referir aos detalhes das interfaces de usuário, e são considerados aspectos como implementação dos relacionamentos, navegabilidade, controle de acessos, assinaturas das operações, criação e destruição de objetos.

⁹ O termo projeto, como tradução de *design*, só deve ser usado quando não houver confusão possível com o sentido de *project*.

Modelo de análise	Modelo de desenho
Descreve o problema	Descreve uma solução
Conceitual (não trata de implementação)	Físico (base para implementação)
Suporta vários possíveis desenhos	Específico em relação a uma implementação
Classes estereotipadas conceituais (fronteiras, entidades e controle)	Classes estereotipadas de acordo com o ambiente de implementação (formulários, módulos etc.)
Pouco formal e detalhado	Muito formal e detalhado
Poucos pacotes lógicos	Mais pacotes lógicos, organizados em camadas
Criação manual (por exemplo, em oficinas de análise)	Criação parcialmente automatizada (usando engenharia reversa)
Mantido opcionalmente	Mantido obrigatoriamente

Tabela 61 – Comparação dos modelos de análise e desenho

1.3 Desenho para a testabilidade

A testabilidade é um objetivo chave no desenho de software. Os engenheiros de software devem desenhar os componentes de um produto pensando em como estes serão testados, como fazem habitualmente seus colegas de hardware. As liberações devem ser desenhadas de tal modo que cada módulo só seja integrado uma vez. Quando possível, o desenho deve ser feito de modo a reduzir a necessidade de estruturas provisórias de teste. Este tipo de código chega a representar metade do volume de código útil [Humphrey99], e é mais uma fonte de defeitos.

Algumas vezes, é necessário alterar o desenho de módulos de um produto para facilitar os testes deste. Pode ser necessário expor operações ou parâmetros adicionais que permitam melhor exercitar as funções dos módulos. Como isso pode aumentar o acoplamento entre módulos, essas escolhas devem ser feitas com cuidado. É normal a interação entre várias das atividades do fluxo de Desenho e a atividade de desenho dos testes, do fluxo de Testes, para resolver questões de testabilidade.

2 Atividades

2.1 Visão geral

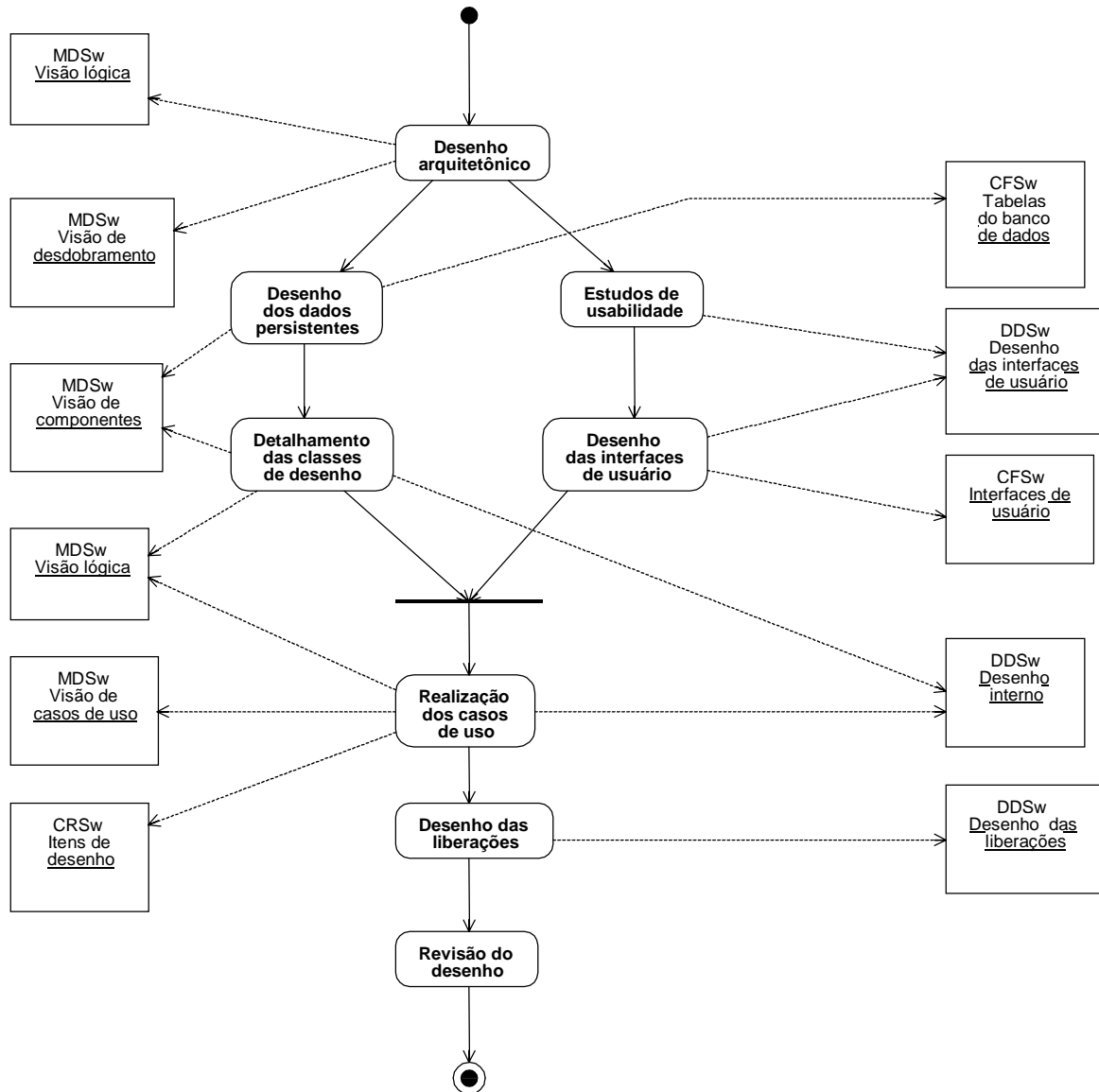


Figura 78 – Atividades e artefatos do fluxo de Desenho

A atividade de "**Desenho arquitetônico**" trata de aspectos estratégicos de desenho interno. Ela define a divisão do produto em subsistemas, escolhendo as tecnologias mais adequadas. As decisões sobre tecnologia devem viabilizar o atendimento dos requisitos não funcionais e a execução do projeto dentro do prazo e custos estimados. Uma decisão fundamental é a escolha do ambiente definitivo de implementação, se isso já não fizer parte das restrições de desenho constantes da Especificação dos Requisitos do Software.

A atividade de "**Estudos de usabilidade**" é também estratégica, focalizando a visão externa do produto. Ela levanta os dados sobre os usuários do produto e a forma destes de trabalhar. Estes dados serão necessários para conseguir-se um bom desenho das interfaces de usuário, e mesmo para melhor detalhamento da execução dos casos de uso.

Pelo caráter estratégico, é recomendável que essas duas primeiras atividades sejam realizadas principalmente na fase de Elaboração, sobretudo tratando-se de produtos mais complexos. As

atividades seguintes são mais típicas da fase de Construção. No Praxis, elas são realizadas principalmente na iteração de Desenho inicial. Tipicamente, entretanto, muitos detalhes ficam para ser resolvidos durante as Liberações, requerendo novas visitas a essas atividades do fluxo de Desenho.

O "**Desenho das interfaces de usuário**" trata do desenho das interfaces reais do produto, em seu ambiente definitivo de implementação. Essa atividade é baseada nos requisitos de interfaces constantes da Especificação dos Requisitos do Software. Estes requisitos e o leiaute ali sugerido são detalhados e realizados levando-se em conta as características do ambiente de implementação e os resultados dos estudos de usabilidade.

A atividade de "**Desenho dos dados persistentes**" trata da definição de estruturas externas de armazenamento persistente, como arquivos e bancos de dados. O principal problema aqui é a realização de uma ponte entre o modelo de desenho orientado a objetos e os paradigmas das estruturas de armazenamento, que geralmente são de natureza bastante diferente.

O "**Detalhamento das classes de desenho**" é uma atividade que agrupa muitas tarefas distintas, destinadas a fazer a transformação do Modelo de Análise no Modelo de Desenho. Muitos detalhes irrelevantes para a Análise devem ser então decididos, como visibilidade das operações e a navegabilidade dos relacionamentos. Inclui-se aqui também a tomada de decisões sobre como serão representadas as coleções de objetos.

A "**Realização dos casos de uso**" determina como os objetos das classes de desenho colaborarão para realizar os casos de uso. Os próprios fluxos devem ser reescritos com muito mais detalhe, para levar em conta os detalhes do desenho das interfaces de usuário. Essa realização dos casos de uso serve para validar muitas das decisões tomadas nas atividades anteriores.

O "**Desenho das liberações**" determina como a construção do produto será dividida em Liberações. Casos de uso e classes de desenho são repartidos entre as liberações, procurando-se mitigar primeiro os maiores riscos, obter realimentação crítica dos usuários a intervalos razoáveis, e possivelmente dividir as unidades de implementação entre a força de trabalho.

Finalmente, a "**Revisão do desenho**" valida o esforço de Desenho, confrontando-o com os resultados dos Requisitos e da Análise. Podem acontecer várias revisões informais, individuais ou em grupo. No final da iteração de Desenho Inicial, o Praxis prevê uma revisão técnica formal.

2.2 Detalhes das atividades

2.2.1 *Desenho arquitetônico*

2.2.1.1 **Arquitetura**

A **arquitetura** de um produto expressa uma divisão desse produto em subsistemas e outros componentes de nível mais baixo, as interfaces entre os componentes e as interações através das quais eles realizam as funções do produto, atendendo aos requisitos não funcionais. Na definição da arquitetura, deve-se buscar um equilíbrio entre o atendimento de requisitos atuais e futuros. A definição de estruturas e mecanismos genéricos demais produz sistemas ineficientes e fora do prazo, enquanto a definição de estruturas e mecanismos específicos demais produz sistemas de manutenção e extensão difíceis.

2.2.1.2 **Pacotes lógicos de desenho**

Na UML, os subsistemas e outros componentes são representados por **pacotes lógicos de desenho**. Estes pacotes lógicos são grupos de classes e outros elementos de modelagem, que apresentam fortes relacionamentos entre si (**alta coesão interna**) e poucos relacionamentos com elementos de outros pacotes lógicos (**baixo acoplamento externo**).

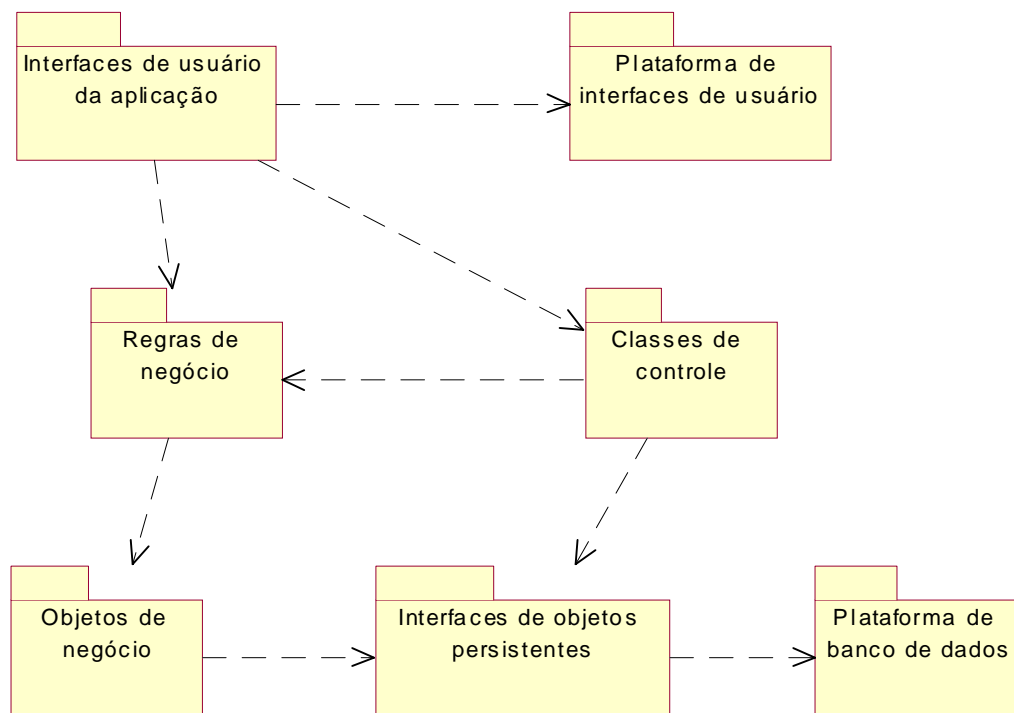


Figura 79 – Exemplo de pacotes lógicos

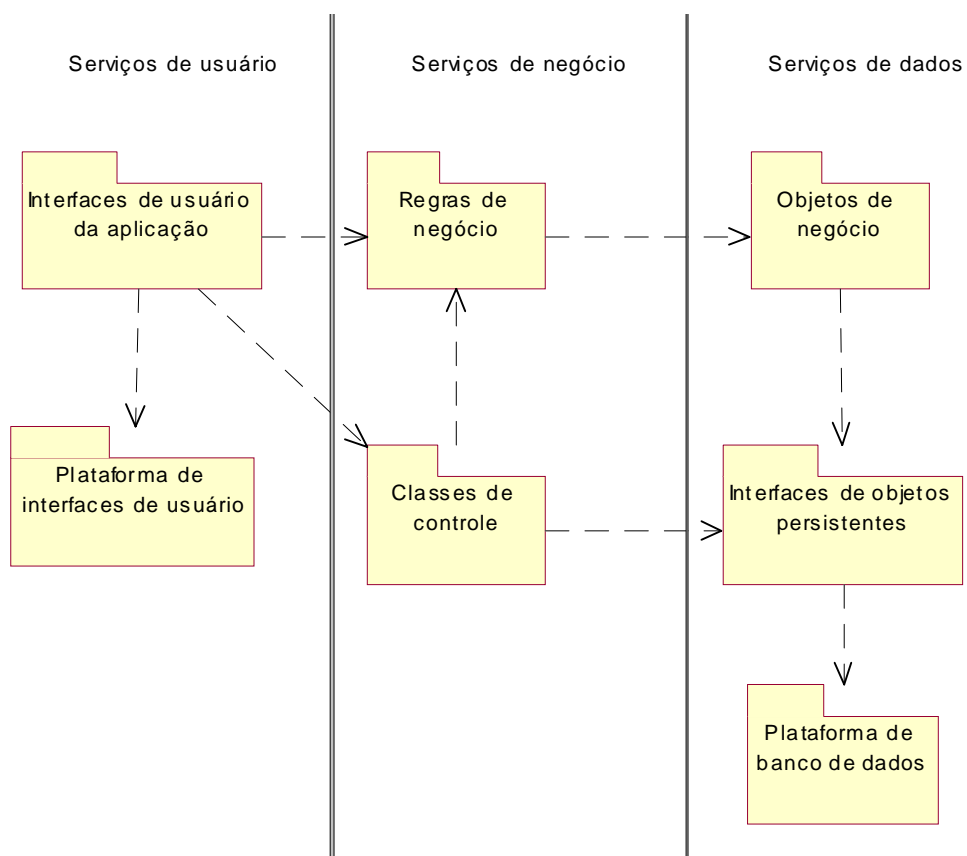


Figura 80 – Exemplo de arquitetura em camadas

2.2.1.3 Organização da arquitetura

Um tipo comum de arquitetura organiza os pacotes lógicos em **camadas**. Cada camada pode usar os serviços das camadas inferiores, mas desconhece as camadas que estão acima dela. Uma arquitetura usual é a arquitetura do modelo de três camadas, que divide os produtos em camadas correspondentes aos serviços de usuário (contendo as interfaces de usuário), serviços de negócio (contendo os elementos que realizam os casos de uso, implementando as regras de negócio) e serviços de dados (contendo os objetos persistentes e os mecanismos para armazenamento destes). Na UML, as camadas podem ser representadas por meio de **partições** ou **raias** (*swimlanes*).

Outras organizações arquitetônicas podem ser mais adequadas. Em ambientes Unix, por exemplo, é comum a organização em filtros, onde cada componente processa um fluxo recebido do componente anterior.

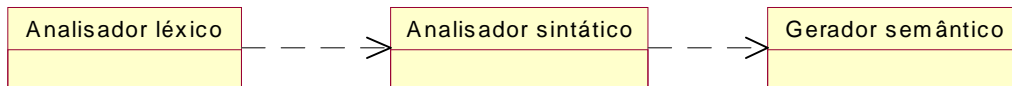


Figura 81 – Exemplo de arquitetura em filtros

Pacotes do domínio	Interfaces de usuário da aplicação Regras de negócio Objetos de negócio
Pacotes da implementação	Plataforma de interfaces de usuário Classes de controle Interfaces de objetos persistentes Plataforma de banco de dados

Tabela 62 – Exemplo de separação do domínio e da implementação

O desenho deve promover a separação entre o domínio e a implementação. As únicas alterações que podem ocorrer na descrição do domínio são aquelas devidas à descoberta de novos aspectos e detalhes do próprio domínio. As estruturas de implementação, por outro lado, devem ser mantidas livres de detalhes do domínio. Com isso, as estruturas de implementação podem ser reaproveitadas para resolver problemas de outros domínios, e as estruturas do domínio podem ser transportadas para outras plataformas de implementação. Domínio e implementação devem ser organizados em pacotes lógicos separados. Na Figura 79, por exemplo, os pacotes obedecem à divisão mostrada na Tabela 62.

2.2.1.4 Definição dos pacotes lógicos

Na definição da arquitetura devem ser tomadas decisões estratégicas quanto à separação de componentes importantes em pacotes lógicos. Os pacotes lógicos devem ser definidos de acordo com os seguintes critérios:

- encapsular classes correlatas;
- facilitar o arquivamento e a recuperação das classes;
- agrupar-se em torno das principais funções arquitetônicas.

Possíveis candidatos a pacotes lógicos incluem:

- interfaces de usuário da aplicação;
- componentes da biblioteca de interfaces de usuário do ambiente de desenvolvimento;
- componentes que realizam regras de negócio e outras características da aplicação do produto;

- componentes que representam entidades de dados do domínio da aplicação;
- interfaces com bancos de dados e outras estruturas de dados persistentes;
- interfaces de comunicação de dados, em sistemas distribuídos, quando não transparentes para o desenvolvedor;
- interfaces com dispositivos especiais de hardware, não transparentes para o desenvolvedor;
- interfaces de uso dos serviços embutidos nos ambientes de desenvolvimento e operação, não transparentes para o desenvolvedor.

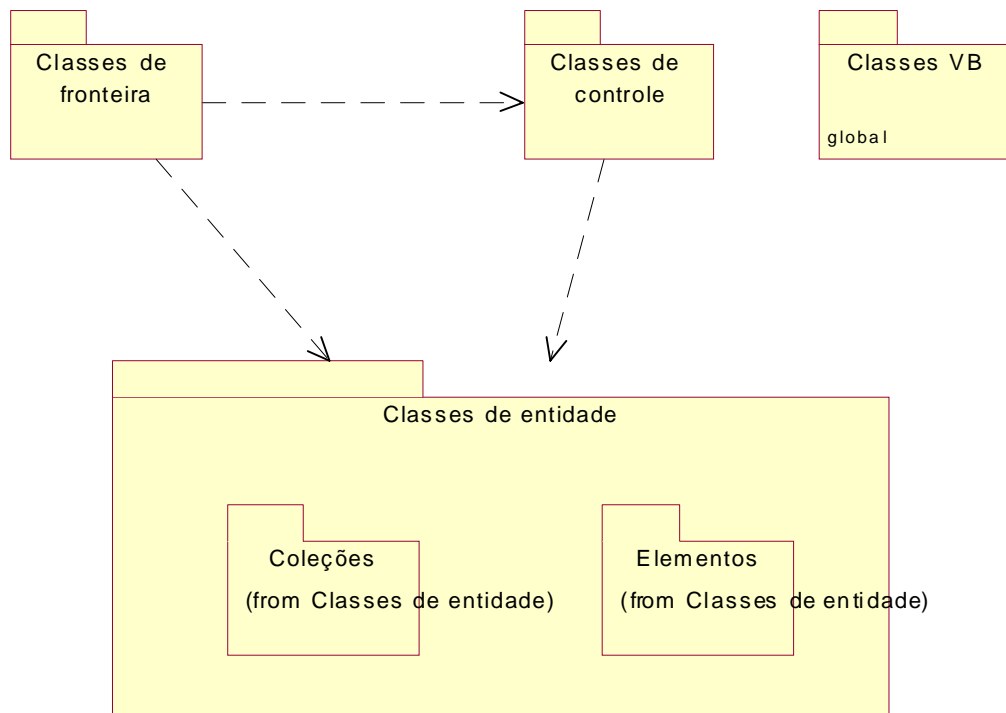


Figura 82 – Exemplo de organização de pacotes lógicos

A Figura 82 mostra parte dos pacotes lógicos desenhados para um sistema de informatização de Mercearia. Aparecem os pacotes descritos na Tabela 63. Essa descrição corresponde a um momento do desenho no qual os pacotes relativos ao armazenamento dos dados persistentes ainda não foram definidos.

Pacote lógico	Descrição	Exemplo de componente
Classes de fronteira	Realizam as interfaces de usuário da aplicação.	frmFornecedores
Classes de controle	Coordenam as colaborações entre as demais classes, que realizam os casos de uso.	Controle_de_Gestao_de_Fornecedores
Classes VB	Classes integrantes do ambiente de desenvolvimento Visual Basic.	modErrorHandler
Classes de entidade	Classes que representam os elementos do modelo do domínio.	Incluem as Coleções e os Elementos.
Coleções	Representam os conjuntos de dados do domínio.	Fornecedores
Elementos	Representam as entidades de dados do domínio, que são os elementos básicos das coleções.	Fornecedor

Tabela 63 – Descrição de pacotes lógicos

2.2.2 Estudos de usabilidade

Para assegurar a qualidade das interfaces de usuário de um produto de software, é preciso ser capaz de medir a **usabilidade** deste, ou seja, sua facilidade de uso. [Shneiderman92] propõe as métricas seguintes para a avaliação de sistemas interativos.

- **Facilidade de aprendizado:** quanto tempo é necessário para que os usuários aprendam a utilizar cada função?
- **Produtividade dos usuários:** quanto tempo é necessário para que os usuários executem cada tarefa, em uso de rotina?
- **Taxa de erros:** qual a quantidade e tipos de erros ocorridos na execução das tarefas?
- **Retenção ao longo do tempo:** por quanto tempo os usuários conseguem manter o conhecimento necessário sobre o produto, considerando-se a frequência de uso?
- **Satisfação do usuário:** quão satisfeitos os usuários se sentem com o produto, de maneira geral?

Esses fatores podem ser balanceados. Se for viável um longo tempo de aprendizado, a produtividade dos usuários pode ser melhorada por meio da utilização de recursos de aceleração, como atalhos e macros. Se a taxa de erros tiver que ser muito baixa, a produtividade precisa ser sacrificada. Essas decisões devem ser baseadas nos atributos de usabilidade constantes da especificação de requisitos do produto.

A engenharia de usabilidade é uma disciplina bastante ampla, cujo estudo detalhado está fora do escopo deste texto. O custo de hardware e software de um sistema é pago apenas uma vez, mas o custo de uso é pago todos os dias. Este custo decorre da queda de produtividade causada por dificuldades de utilização e pela perda de tempo com recuperação de erros. Por isso, o desenho das interfaces de usuário é um problema fundamental no desenvolvimento de um produto de software. Para o tratamento mais detalhado de questões de engenharia da usabilidade, recomenda-se consultar a bibliografia especializada, como [Constantine+99], [Hix+93] e [Shneiderman92].

2.2.3 Desenho das interfaces de usuário

2.2.3.1 Visão geral

O desenho das interfaces de usuário abrange tanto o desenho externo (gráfico e funcional) das interfaces quanto o desenho das classes de fronteira correspondentes. Diretrizes mais específicas para o desenho externo são apresentadas na subseção e no padrão para Desenho de Interfaces de Usuário.

Em alguns ambientes, como o Visual Basic, as interfaces de usuário são realizadas através de módulos do tipo "formulário". A construção dos formulários abrange simultaneamente o respectivo desenho externo e interno. Nesse ambiente, existem ferramentas que permitem extrair as classes de fronteira correspondentes a partir do código dos formulários (técnica de engenharia reversa). Esse assunto é discutido na subseção seguinte.

2.2.3.2 Classes de fronteira

As **classes de fronteira** representam as interfaces de usuário que serão desenvolvidas dentro do produto. Normalmente estão associadas a relacionamentos entre atores e casos de uso. No modelo de desenho, elas devem ser aderentes aos artefatos que o ambiente de desenvolvimento utiliza para a implementação das interfaces de usuário. Por exemplo, em ambientes Visual Basic correspondem geralmente a classes do tipo formulário ("form"); no ambiente Visual C++ correspondem normalmente a classes herdeiras de classes componentes das Microsoft Foundation Classes.

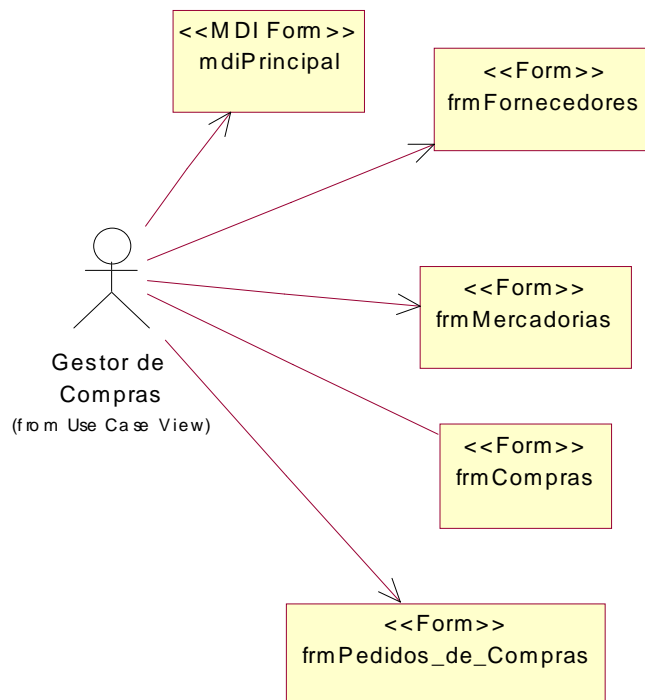


Figura 83 – Exemplo de classes de interfaces de usuário

Quando é necessário mostrar como as classes de fronteira do produto se relacionam com as classes constituintes da plataforma de desenvolvimento, as classes necessárias dessa plataforma devem ser importadas para o modelo. Na Figura 83, que representa um modelo a ser implementado em Visual Basic, os estereótipos são suficientes para indicar os elementos utilizados do ambiente de desenvolvimento. Vê-se que `mdiPrincipal` é um quadro MDI, enquanto as demais interfaces são formulários. Em um ambiente mais complexo, como o Visual C++, pode ser conveniente mostrar explicitamente os relacionamentos de herança. Por exemplo, uma caixa de diálogo seria construída como herdeira de uma classe de diálogo da biblioteca MFC (Figura 84).

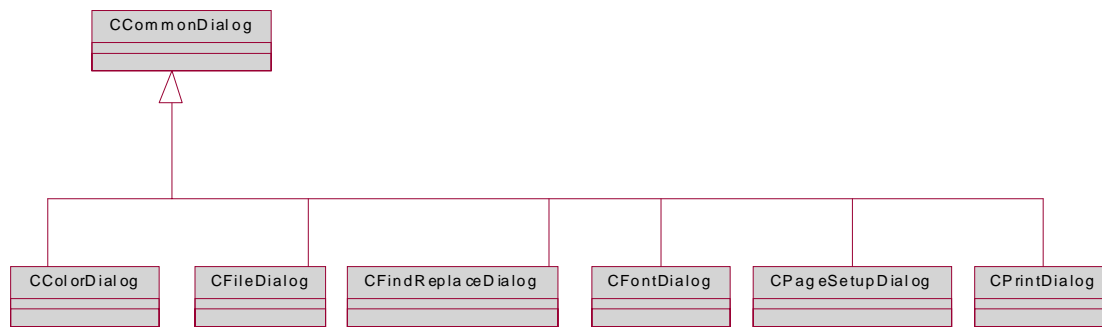


Figura 84 – Classes de diálogo MFC

2.2.4 Desenho dos dados persistentes

Os objetos persistentes são aqueles que continuam a existir após a execução dos programas que os criaram ou atualizaram. Objetos persistentes podem ser guardados em arquivos ou em bancos de dados de diversas tecnologias. Sistemas de gerência de bancos de dados orientados a objetos são a solução mais direta para a implementação de objetos persistentes, principalmente aqueles que representam dados mais complexos. Entretanto, a tecnologia de bancos de dados orientados a objetos é ainda pouco difundida. A tecnologia dominante na área de bancos de dados é a tecnologia relacional.

Versões mais recentes de alguns sistemas de gestão de bancos de dados relacionais têm oferecido extensões de orientação a objetos, criando-se uma tecnologia híbrida objeto-relacional. Essas extensões são geralmente dependentes de fabricantes. De uma maneira geral, a ponte entre o paradigma relacional dominante nos bancos de dados e o paradigma orientado a objetos dominante no software é baseada em soluções específicas de cada fabricante.

Por isso, discutiremos aqui apenas as linhas gerais dos problemas envolvidos quando se quer armazenar objetos persistentes em bancos de dados relacionais puros. O tratamento detalhado deste assunto é feito na subseção 3.4 Interfaces com bancos de dados relacionais.

2.2.5 Detalhamento das classes de desenho

2.2.5.1 Visão geral

A obtenção das classes de desenho a partir das classes de análise requer atenção a muitos detalhes. Classes de análise podem ser partidas ou agrupadas, já que razões tecnológicas ou requisitos não funcionais podem prevalecer sobre a visão basicamente funcional do modelo de análise. Novas classes podem ser criadas, para realizar atributos que não são tipos simples da linguagem de implementação, para realizar mecanismos de desenho, ou para encapsular componentes externos ou sistemas legados.

São tratados a seguir em maior detalhe alguns problemas mais complexos em relação às classes de desenho. As classes de coleção ajudam a implementar relacionamentos de multiplicidade maior que um. Outros aspectos dos relacionamentos precisam ser detalhados para decidir-se quanto à sua implementação. Os aspectos de visibilidade dos elementos das classes devem ser resolvidos.

2.2.5.2 Adição de classes de coleção

Relacionamentos de 1..n ou 0..n podem ser implementados através de classes de coleção, que contêm uma estrutura de dados cujos itens são objetos de mesma classe. A representação de coleções é dependente do ambiente de desenvolvimento.

Uma representação conveniente para coleções em C++ usa as **classes parametrizadas** (“templates”). Estas classes têm uso genérico, independentemente da natureza dos objetos. As classes parametrizadas são **instanciadas** para formar as classes do domínio. Na Figura 85, o contorno de um Polígono é uma objeto da classe Pontos, que é uma “list” cujos elementos são da classe Ponto. “list” é

importada da biblioteca STL; seu parâmetro formal é “_Ty”, que indica a classe do elemento da lista e é substituído por Ponto para formar a classe instanciada Pontos.

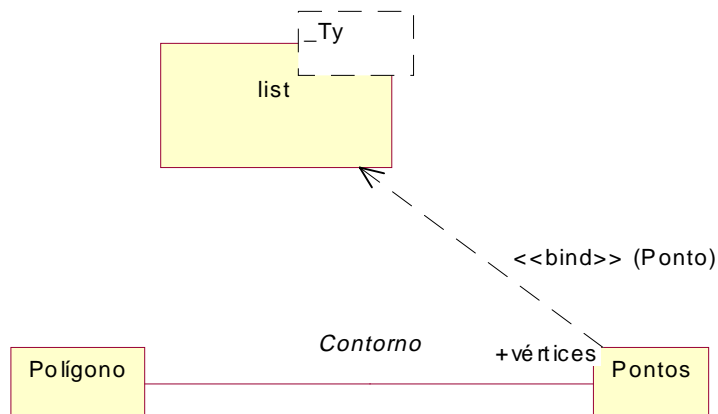


Figura 85 - Exemplo de uso de classe parametrizada

A classe parametrizada corresponde ao código

```
template <class _Ty> class list;
```

enquanto a declaração da classe instanciada é

```
typedef list < Ponto > Pontos;
```

No ambiente Visual Basic, por outro lado, o relacionamento entre uma classe coleção e a respectiva classe elemento é determinado por meio de um tipo de dados específico da linguagem (Figura 86).

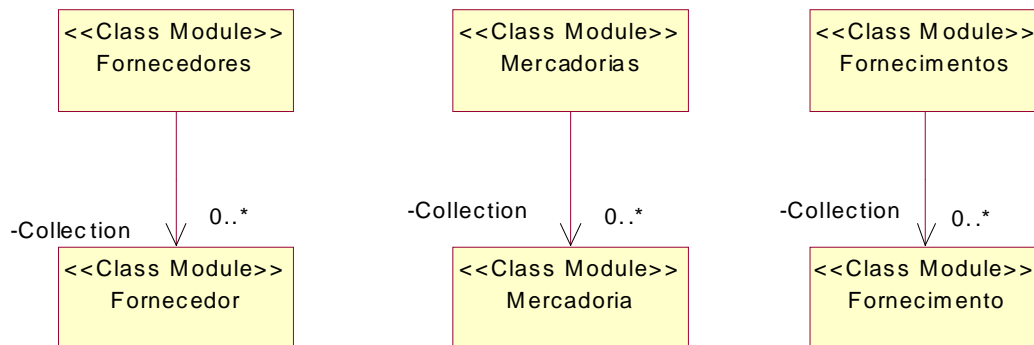


Figura 86 – Coleções em Visual Basic

Para maiores detalhes, o leitor deve consultar as referências dos respectivos ambientes.

2.2.5.3 Detalhamento dos relacionamentos

As associações são bidirecionais, mas algumas só são atravessadas em uma direção. A **direção de navegação** determina como as associações podem ser implementadas. Por exemplo, uma associação que só é atravessada em uma direção pode ser implementada por um apontador. A existência de navegabilidade de uma classe A para uma classe B indica que, dado um objeto de A, é possível localizar de forma direta e eficiente os objetos correspondentes de B; a localização de objetos de A correspondentes a um objeto de B requer meios indiretos, como a execução de um algoritmo de pesquisa.

A Figura 87 indica que é possível localizar eficientemente a mercadoria e o fornecedor correspondentes a um determinado fornecimento, mas a localização dos fornecimentos que correspondem a um fornecedor ou mercadoria exigem uma pesquisa. O estereótipo << Class

Module >> indica que essas classes serão implementadas como módulos de classe do Visual Basic. Os papéis indicam os nomes das variáveis que serão usadas para implementar os relacionamentos.¹⁰

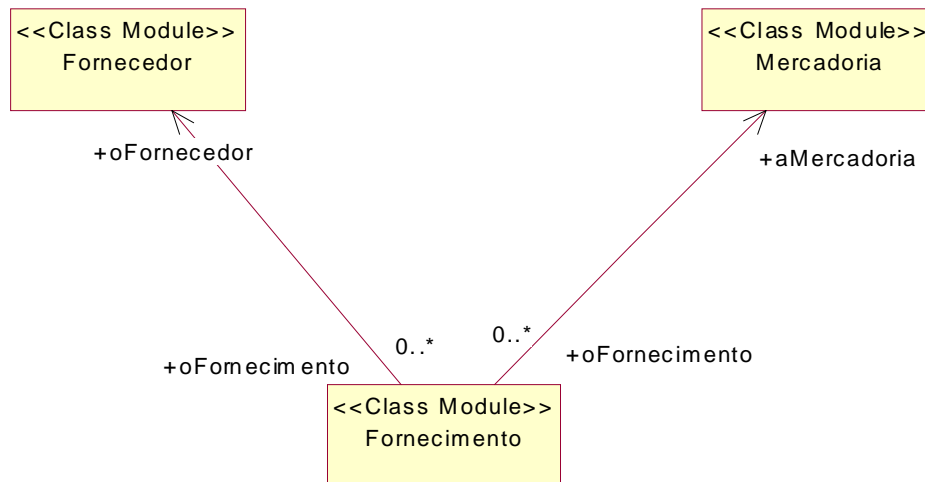


Figura 87 – Relacionamentos com navegação restrita

Durante a análise, os relacionamentos de agregação são usados para indicar que objetos da classe parte são contidos lógica ou fisicamente em objetos da classe todo. No desenho, deve-se decidir como esses relacionamentos serão implementados, observando-se os tempos de vida dos objetos.

- **Agregação por valor:** o objeto todo contém o objeto parte em seu espaço¹¹. O objeto parte tem o mesmo tempo de vida que o objeto todo. É também chamada de composição.
- **Agregação por referência:** o objeto todo contém uma referência ao objeto parte, ou um apontador para o objeto parte. O objeto parte tem vida independente do objeto todo, e deve ser construído e destruído separadamente.

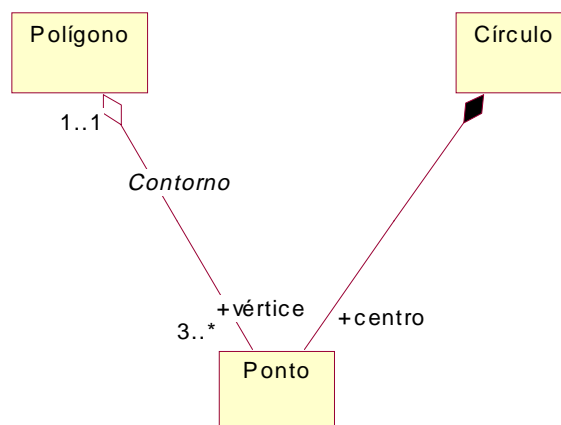


Figura 88 – Agregação versus composição

Na Figura 88, por exemplo, indica-se que um Ponto, no papel de centro, está contido em um Círculo por valor, não tendo existência independente deste. Por outro lado, um conjunto de três ou mais pontos, no papel de vértices, têm existência independente do Polígono do qual formam o

¹⁰ Os sinais + indicam a visibilidade dos papéis, semelhante à visibilidade dos atributos, discutida adiante.

¹¹ A declaração da classe parte pode estar aninhada ou não na declaração da classe todo.

Contorno. Isso faz sentido, por exemplo, se os pontos representarem entidades físicas, como pontos de um relevo, enquanto o polígono representa uma curva de nível que passa por esses pontos.

2.2.5.4 Definição do controle de acesso

Na análise do domínio, não há limitações de acesso ou visibilidade, pois o objetivo é ter uma visão geral e verificar a completeza das abstrações. Já no desenho as classes devem idealmente ser tratadas como caixas pretas, limitando-se os efeitos de mudanças e aspectos de implementação. Deve-se nesta atividade decidir os tipos de acesso a operações ou atributos de cada classe:

- **público**: qualquer outra classe pode usar (símbolo na UML +);
- **protegido**: só subclasses dessa classe podem usar (símbolo na UML #);
- **privado**: nenhuma outra classe pode usar diretamente (símbolo na UML -);
- **implementação**: declarado no corpo de uma operação.

Para determinar o tipo de acesso, devem ser usados os seguintes critérios:.

- Os dados devem ser normalmente privados. O acesso a eles deve ser feito através de operações específicas de consulta e atualização.
- Dados de curtíssima duração (por exemplo, iteradores de estruturas de dados) podem ser de implementação.
- As operações que fornecem resultados de interesse interno da classe devem ser privadas.
- As operações que só são invocadas por subclasses devem ser protegidas.
- As operações que fornecem resultados de interesse de outras classes devem ser públicas.
- Exceções ao controle normal de acesso, para determinadas classes clientes, podem ser feitas declarando-se essas classes como **amigas** da classe fornecedora.

Na Figura 89 são mostrados os controles de acesso dos atributos e operações de uma classe implementada em Visual Basic. Note-se que todos os atributos são privados. São públicas as operações provenientes do modelo de análise e as operações de acesso aos atributos; são privadas as operações de início e término da vida dos objetos.

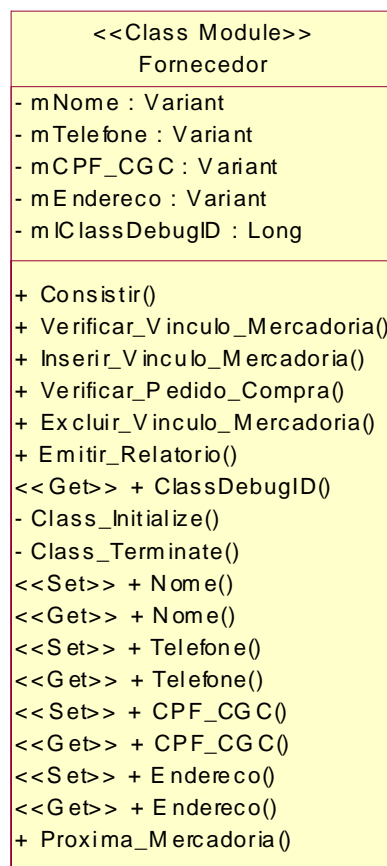


Figura 89 - Exemplo de controles de acesso

Note-se também, na Figura 89, o uso de estereótipos para indicar que construções da linguagem alvo serão utilizadas para implementar a classe (<< Class Module >>) e as operações de acesso aos atributos (<< Get >>, << Set >>).

Caso se use geração automática de código, os atributos serão normalmente gerados como privados, e a geração de operações de acesso obedecerá às seguintes regras:

- para os atributos declarados como privados no modelo de desenho não serão geradas operações de acesso;
- para os atributos declarados como públicos ou protegidos no modelo de desenho serão geradas operações de consulta e alteração, com o grau de visibilidade que o respectivo atributo tiver no modelo de análise.

2.2.6 Realização dos casos de uso

2.2.6.1 Detalhamento dos fluxos dos casos de uso

O detalhamento dos fluxos dos casos de uso (Figura 90) mostra como cada roteiro importante será realizado em termos das interfaces de usuário desenhadas. A lógica mostrada é derivada do modelo de análise, mas a execução do roteiro mostra os elementos reais das interfaces de usuário, podendo servir de especificação para os procedimentos de teste e os roteiros de uso incluídos na documentação de usuário.

Se a interface estiver no estado "Alterada", o Merci emite a mensagem MGF-08, solicitando confirmação.
 Se o Gestor de Compras negar a confirmação, o Merci abandona este subfluxo.
 O Merci pesquisa se existe fornecedor com este Código na tabela Fornecedores do banco de dados Merci.
 Se não existir um fornecedor com este Código, o Merci executa a exceção EGF-01.
 Se existir um fornecedor com este código:
 O Merci exibe o Nome, Endereço, Telefone e CGC do fornecedor.
 O Merci exibe a lista de mercadorias fornecidas pelo fornecedor (campos Código e Descrição), como grade no quadro Mercadorias Fornecidas.
 O Merci coloca a interface no estado "Atualizada"

Figura 90 – Exemplo de caso de uso detalhado no desenho

Fluxos complexos podem ser mais bem descritos por meio de diagramas de estado (Figura 91). O diagrama de estado permite visualizar a execução do caso de uso de forma mais abrangente que os roteiros textuais, embora menos detalhada. O diagrama de estado do caso de uso pode servir de base para derivação dos diagramas de estado das classes de interface de usuário (Figura 92) e de controle.

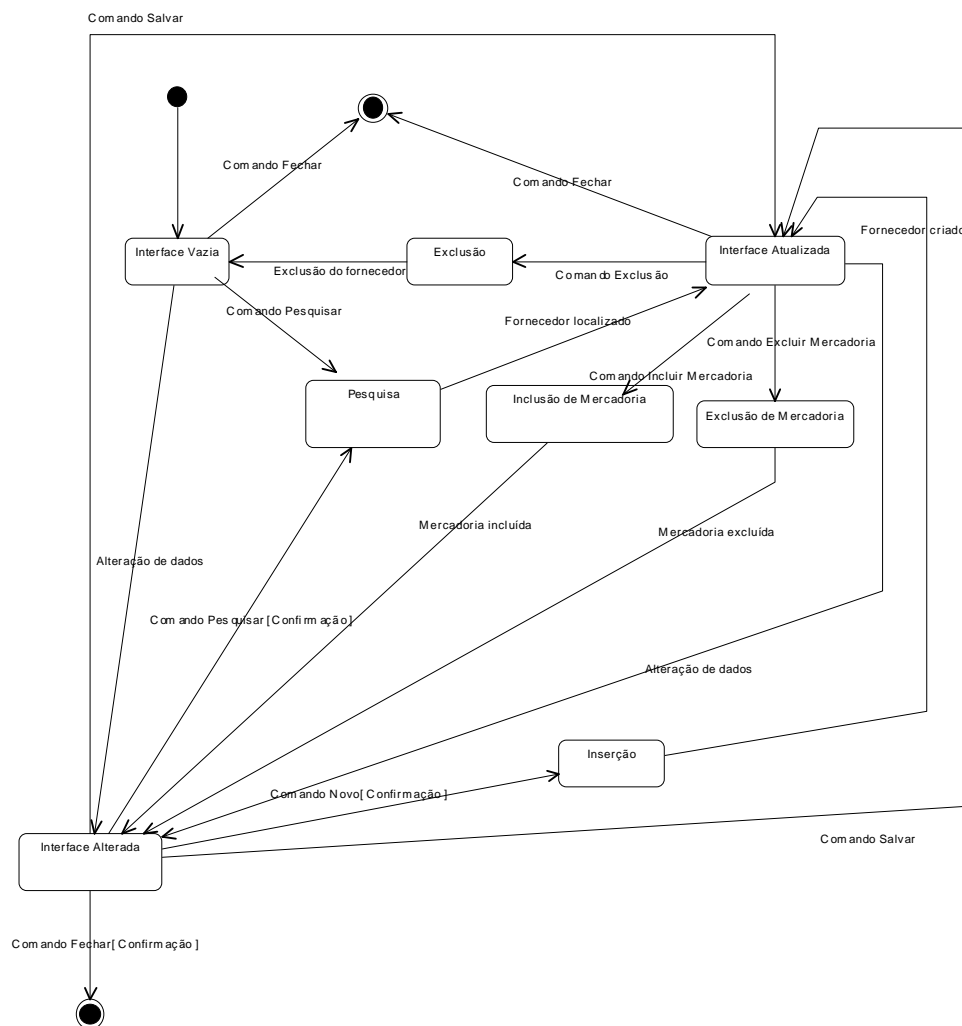


Figura 91 – Exemplo de diagrama de estado de caso de uso

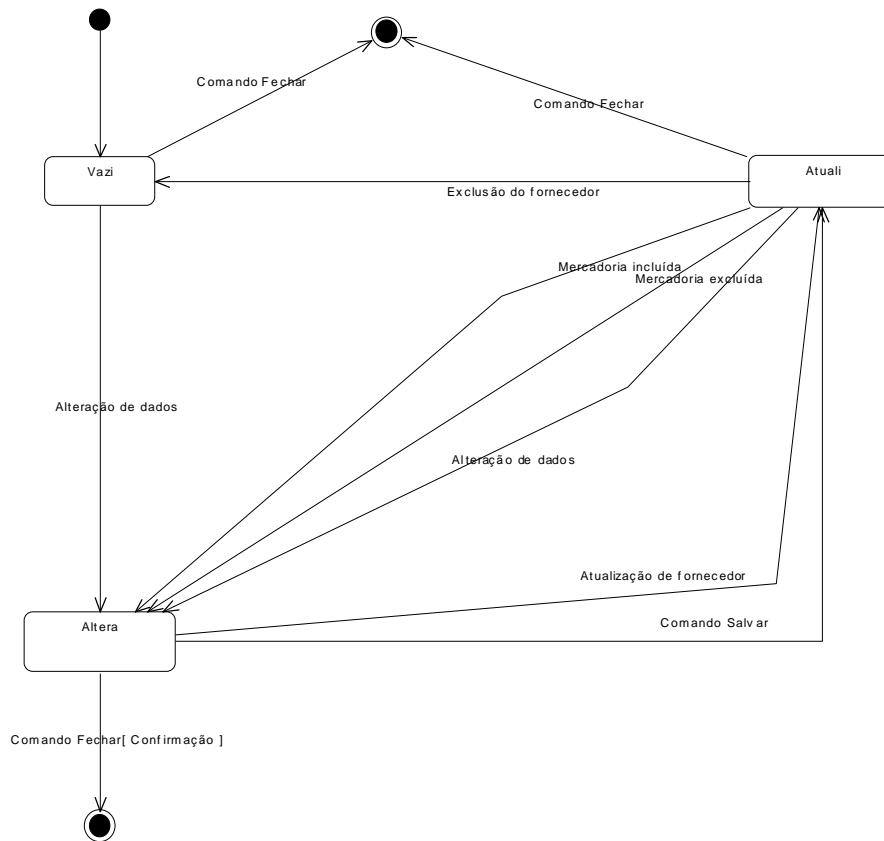


Figura 92 – Exemplo de diagrama de estado de classe de interface de usuário

O modelo de desenho deve mostrar a realização dos casos de uso através de diagramas de seqüência e colaboração. Os diagramas de seqüência (Figura 93) devem ser preferidos quando se quer enfatizar o ordenamento temporal das mensagens, enquanto os diagramas de colaboração (Figura 94) devem ser usados para se enfatizar os mecanismos através dos quais as classes envolvidas colaboram para a execução do caso de uso.

Os detalhes de processamento que não correspondem a interações entre classes foram lançados como anotações no diagrama da Figura 93. Estas anotações podem servir como especificações das operações das classes envolvidas. Operações ainda mais complexas podem precisar de notações mais formais para descrição, tais como algoritmos codificados em linguagem semelhante à linguagem de implementação.

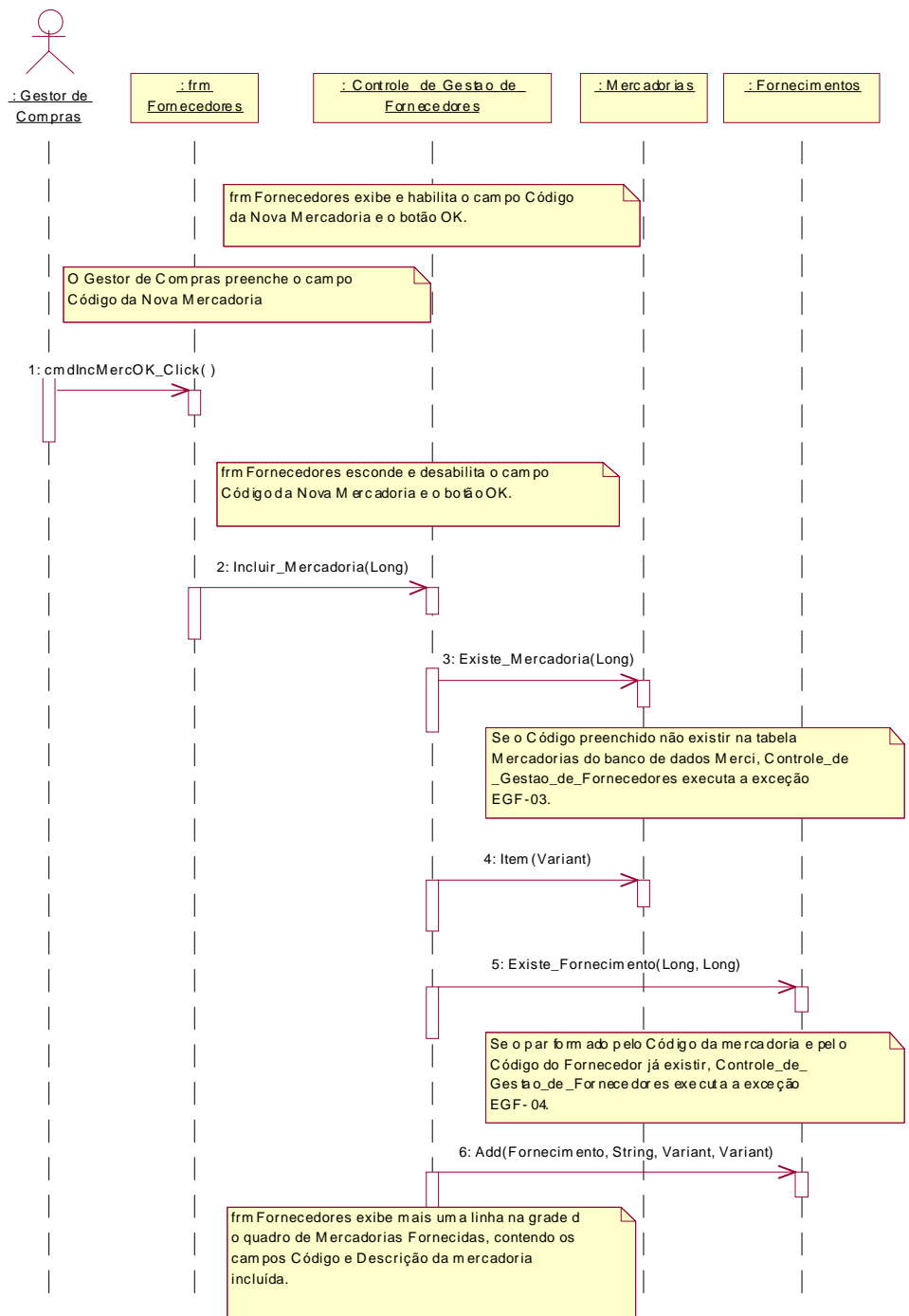


Figura 93 – Exemplo de diagrama de seqüência para realização de um caso de uso

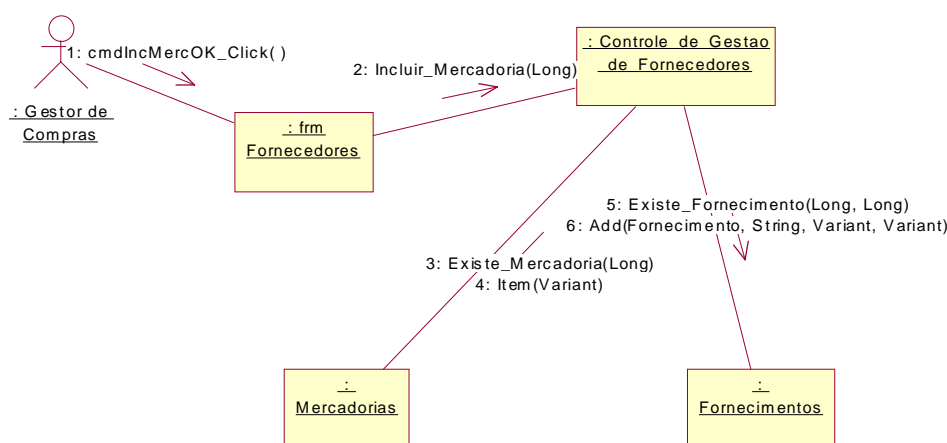


Figura 94 – Exemplo de diagrama de colaboração para realização de um caso de uso

2.2.7 Desenho das liberações

2.2.7.1 Planejamento das liberações

O desenho deve usar uma abordagem progressiva e iterativa, através da construção de versões operacionais parciais, que sirvam como pontos de controle e permitam a avaliação por parte dos usuários. Essas versões parciais são as **liberações**.

A ordem das liberações deve procurar reduzir o risco do desenvolvimento. São exemplos usuais de áreas de risco, que devem ser postas à prova nas primeiras liberações:

- os requisitos funcionais mais importantes do produto;
- as interfaces gráficas de usuário;
- novos equipamentos, sistemas e serviços, tais como sistemas de gerência de bancos de dados, com os quais a equipe de desenvolvimento não está familiarizada.

Outros objetivos das liberações são a verificação da eficácia de mecanismos chaves, ou seja, de colaborações necessárias para implementar os casos de uso, e a avaliação e realimentação precoces, por parte do usuário. Em projetos maiores, é possível planejar liberações paralelas de subsistemas independentes, para dividi-los entre a força de trabalho.

Número de ordem	Nome da liberação executável	Objetivos
1	Compras	Implementar os casos de uso relacionados com a Gestão de Compras, permitindo o povoamento do banco de dados do Mercú.
2	Vendas	Implementar os casos de uso relacionados com a Gestão de Vendas, completando as funções do Mercú.

Tabela 64 - Exemplo de Lista das liberações

O planejamento do restante da fase de Construção deve ser baseado nas liberações. O ordem de implementação das liberações define a ordem de integração do produto. O plano de liberações deve ser consistente com os planos de testes do software e com o plano de desenvolvimento do software. Na Descrição do Desenho do Software, o plano é refletido através da Lista das liberações (Tabela 64)

2.2.7.2 Especificação das liberações

As liberações são versões parciais do produto em desenvolvimento, formadas por uma combinação de:

- **código novo**, que deverá ser parte do produto definitivo;
- **código já escrito**, revisado e testado em liberações anteriores;
- **componentes de teste**, tais como cotos (*stubs*), controladores (*drivers*) e bases de dados de teste;
- **implementações simplificadas** de algumas das classes mais complexas, que serão detalhadas em liberações posteriores.

Classes a serem implementadas	Controle_Gestao_de_Mercadorias, Controle_de_Gestao_de_Pedidos_de_Compras, Controle_de_Gestao_Manual_de_Estoque, Mercadorias, Mercadoria, Fornecimentos, Fornecimento, Pedidos_de_Compras, Pedido_de_Compras, Compras, Compra, frmMercadorias, frmPedidos_de_Compras, frmCompras
Classes a serem alteradas	mdiPrincipal, frmFornecedores, Controle_Gestao_de_Fornecedores.
Casos de uso a serem implementados	Gestão de Mercadorias Gestão de Pedidos de Compra Gestão Manual de Estoque
Casos de uso a serem complementados	Gestão de Fornecedores - exclusão
Componentes de teste	Cotos para os casos de uso não implementados.
Componentes reutilizados	MS Flex Grid

Tabela 65 – Exemplo de especificação de liberação

De preferência, as liberações são divididas de forma que cada liberação implemente classes inteiras e casos de uso inteiros. Em certos casos, é admissível que uma liberação implemente apenas algumas das operações de uma classe ou algum dos roteiros de um caso de uso, quando isso for indispensável para tratar de riscos maiores. Por exemplo, pode ser conveniente submeter à avaliação precoce dos usuários os roteiros mais comuns de um caso de uso importante, deixando-se para depois os roteiros de execução menos freqüente.

A especificação de cada liberação (Tabela 65) deve conter pelo menos os seguintes elementos:

- classes novas a serem implementadas;
- classes a serem alteradas;
- casos de uso a serem implementados;
- casos de uso a serem complementados;
- componentes de testes;
- componentes reutilizados.

2.2.8 *Revisão do desenho*

Diversos tipos de revisão informal do desenho, realizadas de forma individual e em grupo, podem ser empregados ao longo das iterações do processo. Entretanto, pelo menos uma revisão técnica formal deve ser realizada, tendo em vista o peso que defeitos de desenho têm em relação aos prazos e custos dos projetos.

O Praxis prevê, ao final da primeira iteração da fase de Construção, uma revisão técnica formal, em que são analisados em conjunto o Modelo de Desenho, a Descrição do Desenho do Software e a Descrição dos Testes do Software, na parte relativa aos testes de aceitação. Para facilitar essa revisão, o padrão para Descrição do Desenho do Software prevê que a informação de suporte conterá listagens dos diagramas e especificações do Modelo de Desenho. De preferência, a ferramenta de modelagem deve permitir a extração dessa informação em formato com qualidade para publicação, evitando a extração manual tediosa e propensa a erros.

Um roteiro de revisão deve ser utilizado para que a revisão técnica tenha uma cobertura completa. Esse roteiro deve pedir a checagem de todos os itens previstos no padrão para Descrição do Desenho do Software, e conter uma lista de conferência do Modelo de Desenho. Além disso, deve-se usar um roteiro para checar se foram seguidas as diretrizes contidas no padrão para Desenho das Interfaces de Usuário. Caso seja adotado um manual de estilo de interfaces de usuário, específico do ambiente de implementação, a aderência a este deve ser checada. Exemplos de roteiros de revisão do Modelo de Desenho e das interfaces de usuário estão contidos no padrão para Descrição de Desenho de Software.

3 *Técnicas*

3.1 *Visão geral*

As técnicas tratadas aqui em detalhe dizem respeito ao desenho de interfaces de usuário, ao desenho para a reutilização e às interfaces com bancos de dados relacionais. A reutilização é um fator muito significativo para a redução dos prazos e custos dos projetos, e geralmente o grau de reutilização conseguido é muito influenciado pelo desenho. As interfaces com bancos de dados relacionais representam um problema difícil e freqüente, dados que esses tipos de bancos de dados são atualmente muito comuns.

Não será tratada especificamente a realização de oficinas de Desenho. Muitas atividades de desenho não se prestam adequadamente à realização em grupo. Eventualmente, as atividades de "Desenho arquitetônico" e "Estudos de usabilidade" podem ser discutidas por grupos de desenvolvedores, na forma de oficinas. Os resultados demais atividades podem ser revisados informalmente em grupos, antes de se fazer uma revisão formal.

Além disso, a participação dos usuários geralmente é muito menor do que nos fluxos de Requisitos e mesmo de Análise. Uma exceção, naturalmente, pode ser representada pela atividade de "Desenho das interfaces de usuário". Quando for conveniente realizar oficinas de Desenho no estilo de JAD, estas podem ser adaptadas a partir das diretrizes para oficinas de Análise.

3.2 *Desenho de interfaces de usuário*

3.2.1 *Modelos de conhecimento*

3.2.1.1 *Visão geral*

O conhecimento das interfaces por parte dos usuários tem os seguintes aspectos:

- conhecimento sintático;
- conhecimento semântico de computação;

- conhecimento semântico das tarefas.

O desenho de interfaces capazes de permitir uma interação eficiente depende do entendimento das necessidades dos usuários em relação a esses três tipos de conhecimento. A mesma pessoa pode ter conhecimento avançado das tarefas e ser um principiante em computação, ou vice-versa.

3.2.1.2 Conhecimento sintático

O conhecimento sintático abrange os detalhes de formato das interações, que geralmente são dependentes de dispositivos. Esses detalhes incluem, por exemplo:

- como excluir um objeto;
- como abrir ou fechar um arquivo;
- como movimentar objetos;
- quais os mecanismos aceleradores existentes.

O domínio desse conhecimento é dificultado por vários fatores:

- depende da plataforma, por causa dos detalhes dependentes de dispositivos, como resolução da tela ou tipo de teclado;
- possui aspectos arbitrários, que reduzem a possibilidade de criar associações que ajudem a memorizar;
- requer prática para retenção, sendo esquecido quando o uso não é frequente;
- tarefas semelhantes em diferentes níveis de hierarquia de um ambiente podem requerer comandos de aspecto muito diferente.

Um exemplo do último problema é a diversidade de comandos existente no ambiente Unix tradicional. O editor de texto "vi", por exemplo, oferece comandos diferentes para excluir caracteres, palavras e linhas. Em ambientes mais modernos, os fabricantes recomendam aos desenvolvedores guias de estilo que ajudam a obter maior uniformidade entre os comandos dos diversos aplicativos. É fortemente recomendada a aderência ao guia de estilo oficial do ambiente alvo de um projeto.

3.2.1.3 Conhecimento semântico

O conhecimento semântico de computação é geralmente transmitido por meio de estruturas conceituais, exemplos de uso e analogias com outros conhecimentos. Ele é independente de sintaxe, e de retenção mais fácil que o conhecimento sintático.

O conhecimento semântico de computação é organizado hierarquicamente. Por exemplo, a informação armazenada em um sistema de computação geralmente é vista pelos usuários como um conjunto de pastas que contêm arquivos. Os arquivos, por sua vez, são divididos em objetos menores dependentes do tipo. Arquivos de texto, por exemplo, se decompõem em seções, parágrafos, palavras e caracteres.

O conhecimento semântico sobre tarefas também pode ser organizado de forma hierárquica. As ações para realizar uma tarefa complexa são decompostas em ações mais simples. A Tabela 66 mostra como os níveis de conhecimento são empregados ao se utilizar um computador para escrever textos.

Tipo de conhecimento		Alto nível	Baixo nível
Semântico	Das tarefas	Redação	Ortografia
	De computação	Arquivos	Objetos do editor de texto
Sintático		Como salvar e abrir arquivos	Comandos de edição de texto

Tabela 66 - Exemplo de níveis de conhecimento

Tipicamente, a semântica das tarefas deve ser definida durante o levantamento dos requisitos. A semântica computacional deve começar a ser definida na especificação das interfaces de usuário, e pode ser completada na fase de desenho. A sintaxe pode ser definida parcialmente durante o desenho, sendo completada durante a implementação.

3.2.2 *Desenho centrado no usuário*

3.2.2.1 Conhecimento dos usuários

O desenho de interfaces de uso fácil é difícil. Via de regra, quanto mais fácil de usar é um produto, maiores os esforços de desenho e implementação de suas interfaces de usuário. Cabe aos gerentes de projeto decidir quanto é justificável gastar, em tempo e em recursos, a troco do aumento da usabilidade. Geralmente é necessário desenhar as interfaces de forma interativa, submetendo as soluções encontradas a vários ciclos de avaliação por parte dos usuários.

O desenhista de interfaces deve conhecer muito bem os usuários destas e os processos que o produto irá automatizar. Métodos de marketing, como realização de entrevistas, aplicação de questionários e sessões de grupos de foco são mecanismos úteis também para o desenho das interfaces. Devem ser analisadas as tarefas que se pretende automatizar, incluindo os processos que serão executados manualmente ou por outros sistemas de informática. Em certos casos, o desenhista deve fazer um trabalho de campo, convivendo com os usuários na rotina de trabalho destes últimos.

3.2.2.2 Participação dos usuários no desenho

Os usuários devem participar do desenho das interfaces, através de técnicas de desenho participativo, como o JAD. É necessário envolver os usuários que irão efetivamente utilizar o produto, ou seja, os usuários chaves. O desenho participativo ajuda a compreender:

- quais são as tarefas que os usuários precisam executar;
- com que frequência eles executam essas tarefas;
- em que condições eles executam essas tarefas;

No caso de produtos em evolução, as versões correntes devem ser avaliadas. Nessa avaliação, é preciso identificar o que a versão atual faz, e como o faz. Pontos fracos e fortes devem ser identificados.

3.2.2.3 Prevenção de erros dos usuários

As ações que podem provocar erros dos usuários devem ser previstas e inibidas. Isso é feito, por exemplo, desabilitando-se as opções inválidas em cada estado das interfaces. Com isso, apenas um subconjunto das opções ficará habilitado e disponível, em um dado momento, o que facilita o aprendizado e diminui os erros, principalmente os mais graves. A documentação de usuário deve explicar claramente os estados das interfaces, de maneira que os usuários entendam o porquê da desativação de certos comandos.

Do lado dos desenhistas, requer-se um desenho cuidadoso dos estados de cada interface. Por outro lado, a limitação das opções reduz também a carga de trabalho dos desenvolvedores, diminuindo a

quantidade de procedimentos de exceção e mensagens de erro que deverão ser desenhadas e implementadas.

Deve-se solicitar confirmação dos usuários ao executar comandos que possam resultar em perda de informação ou outros resultados indesejáveis. Essa confirmação é ainda mais necessária quando o produto não oferecer meios para que o usuário desfça ações das quais se arrependeu.

3.2.2.4 Adequação aos usuários

Os usuários experientes devem ter acesso a mecanismos que tornem a interação mais rápida, como aceleradores e atalhos. Para esses usuários, a necessidade de apontamento para utilizar cardápios e outros itens reduz a produtividade.

O controle da interação deve estar sempre nas mãos do usuário. Por exemplo, ao preencher formulários interativos, os usuários devem poder editar os campos na ordem em que desejarem, embora uma ordem padrão de deslocamento entre campos deva ser oferecida. Devem-se evitar ações disparadas de forma implícita, como o salvamento automático de um registro após ser preenchido o último campo de um formulário.

Os usuários iniciantes devem ter à disposição um conjunto básico de comandos que permita realizar um mínimo de trabalho útil. Tipicamente, uma página de manual deve ser suficiente para descrever o uso desse conjunto. Essa informação deve incluir os mecanismos para ativar as funções básicas, e a descrição das principais áreas da interface.

3.3 Desenho para a reutilização

3.3.1 Visão geral

Quando uma organização começa a usar processos definidos de desenvolvimento de software, os maiores ganhos iniciais resultam da redução dos defeitos introduzidos em cada iteração. Isso ocorre por causa da redução do desperdício de tempo e dinheiro, principalmente aquele que é causado por defeitos de requisitos, análise e desenho. A partir daí, ganhos significativos de produtividade só são conseguidos por meio da reutilização. A automação de algumas tarefas de desenvolvimento, através de ferramentas bem escolhidas, contribui para a redução de defeitos, mas não resulta em um ganho de produtividade comparável com a reutilização. Embora a reutilização possa e deva ser feita em relação aos artefatos de todo o ciclo de vida, ela é um aspecto dominante em um bom desenho.

Durante as atividades de desenho, é preciso considerar tanto a reutilização de material do passado quanto o desenho de material reutilizável no futuro. O desenho deve levar em conta que material será aproveitado do ambiente de desenvolvimento, de bibliotecas comerciais de componentes e de componentes produzidos em projetos anteriores. É preciso considerar aspectos de reutilização de código e do próprio desenho.

Quanto à produção de material reutilizável, deve haver uma solução de compromisso entre o desenho para a reutilização e o cumprimento de metas específicas de cada projeto. O material reutilizável deve apresentar níveis de qualidade ainda maiores que o material normal dos projetos. Isso exige um desenho especialmente cuidadoso.

3.3.2 Reutilização de código

3.3.2.1 Formas de reutilização de código

Exemplos de reutilização de código incluem:

- reutilização de **classes**, como as incluídas em classes fundamentais e bibliotecas padrão;
- reutilização de **objetos**, isto é, módulos de código binário de interface padronizada;

- reutilização de **plataformas** (*frameworks*), isto é, de camadas inteiras da arquitetura.

3.3.2.2 Reutilização de classes

A reutilização de classes aproveita o código de classes e sub-rotinas preexistentes, por meio de chamada de operações, agregação (incorporação de objetos, por valor ou por referência) ou herança. Para a reutilização de classes, é preciso ter acesso ao código fonte pelo menos da definição das classes, que contém sua interface, com a declaração das operações. A definição das operações, naturalmente, pode estar em arquivos binários ligáveis.

Os mais modernos ambientes de desenvolvimento são baseados em famílias de classes fundamentais, com cujo desenho o engenheiro de software deve procurar estar bastante familiarizado. Por exemplo, no ambiente Microsoft Visual C++ existem as seguintes famílias:

- a biblioteca de classes fundamentais MFC, para o desenvolvimento de aplicativos com interfaces gráficas de usuário, em ambientes Windows;
- a biblioteca de classes fundamentais ATL, para o desenvolvimento de componentes dentro do padrão ActiveX;
- a biblioteca de classes padronizadas STL, para a manipulação de estruturas de dados, fluxos de entrada e saída e operações básicas da linguagem C++.

O engenheiro de software que trabalhe com esses ambientes deve ter à mão os modelos UML dessas bibliotecas. Estes modelos costumam ser fornecidos pelos fabricantes de ferramentas de análise e desenho orientados a objetos. Por outro lado, não é necessário importar para o modelo de desenho todos os elementos dos modelos das bibliotecas, mas apenas aqueles que são necessários para descrever os mecanismos importantes do desenho.

3.3.2.3 Reutilização de objetos

A reutilização de objetos utiliza componentes de código binário, com formatos de arquivo e interfaces de uso padronizadas. São exemplos os componentes baseados nas arquiteturas:

- COM – arquitetura padronizada pela Microsoft, que inclui as variantes DCOM e ActiveX;
- CORBA – arquitetura genérica e portátil para objetos distribuídos, padronizada pelo consórcio OMG ("Object Management Group");
- JavaBeans – arquitetura específica para Java, padronizada pela Sun.

3.3.2.4 Reutilização de plataformas

A reutilização de plataformas utiliza ambientes completos para desenvolvimento de aplicativos em áreas específicas. Uma plataforma é constituída de famílias de classes especializadas.

São exemplos:

- a plataforma ARX para o ambiente AutoCAD (plataforma de desenho técnico e CAD);
- as classes predefinidas do Lotus Notes (plataforma de trabalho em grupos e suporte a fluxos de trabalho);
- as classes predefinidas dos membros da família Microsoft Office (plataforma de automação de escritórios).

3.3.3 *Reutilização de desenho*

A reutilização do desenho reaproveita idéias de colaborações entre classes para resoluções de determinados problemas de desenho comumente encontrados. Ela é baseada em **padrões**¹² (“*patterns*”) de desenho, geralmente apresentados na forma de modelos de classes, juntamente com instruções de utilização e sugestões de implementação. Na maioria das vezes esses modelos são baseados em classes abstratas, das quais são derivadas por herança as classes concretas necessárias.

O engenheiro de software deve consultar catálogos de padrões, procurando soluções mais próximas dos problemas encontrados em cada projeto. O catálogo mais difundido de padrões de desenho está em [Gamma+94].

3.4 Interfaces com bancos de dados relacionais

3.4.1 *Visão geral*

Os bancos de dados relacionais podem ser usados para a implementação de objetos persistentes, por meio de mecanismos de montagem e desmontagem desses objetos. A desmontagem dos objetos traduz os dados contidos nestes objetos em tabelas do modelo relacional, e a montagem realiza a operação inversa.

As diferenças entre os paradigmas relacional e orientado a objeto acarretam várias dificuldades. É necessário muito código para tradução entre paradigmas; a montagem de objetos persistentes pode requerer acesso a muitas tabelas; e vários problemas de desempenho e integridade têm de ser resolvidos. Para o leitor interessado em aprofundar o assunto, recomendam-se [Jacobson94, cap. 9], [Booch94, cap. 10], [Blaha+98], [Blaha+99] e [Loomis94].

O aspecto estático do armazenamento dos objetos persistentes em bancos de dados relacionais envolve o desenho das tabelas que os representarão. Os seguintes problemas devem ser resolvidos:

- mapeamento entre objetos e tabelas;
- mapeamento entre tipos complexos e tipos simples;
- representação da herança;
- integridade referencial;
- índices;
- normalização.

O aspecto dinâmico do armazenamento dos objetos persistentes em bancos de dados relacionais envolve o desenho de classes e mecanismos que farão, em tempo de execução, a tradução entre os paradigmas. Os seguintes problemas devem ser resolvidos:

- isolamento das dependências de tecnologia;
- consistência entre memória principal e banco de dados;
- representação de transações.
- armazenamento de operações.

¹² Não confundir com padrões no sentido de *standards*.

3.4.2 Representação de objetos por tabelas

3.4.2.1 Mapeamento entre objetos e tabelas

As seguintes regras são básicas para o mapeamento entre objetos e tabelas:

- cada classe é normalmente representada por uma tabela;
- atributos primitivos são representados por colunas;
- atributos complexos são representados por múltiplas colunas ou tabelas adicionais;
- a coluna da chave primária será um indicador da instância, que pode ser:
 - um atributo designado como chave;
 - um identificador invisível gerado pelo sistema.
- cada instância da classe será representada por uma linha da tabela.

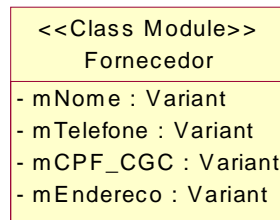


Figura 95 - Exemplo de classe persistente

```

CREATE TABLE T_Fornecedor(
  mNome VARCHAR( ),
  mTelefone VARCHAR( ),
  mCPF_CGC VARCHAR( ),
  mEndereco VARCHAR( ),
  FornecedorId NUMBER(5),
  PRIMARY KEY(FornecedorId))
  
```

Figura 96 - Definição de tabela para classe persistente

mNome	mTelefone	mCPF_CGC	mEndereco	FornecedorId

Figura 97 - Tabela correspondente a classe persistente

A Figura 95 mostra um exemplo de classe persistente, que é traduzida na Figura 97, através do comando SQL mostrado na Figura 96. Este comando DDL¹³, expresso na linguagem ANSI SQL, foi gerado automaticamente por uma ferramenta de modelagem.

¹³ Data Description Language.

3.4.2.2 Representação dos relacionamentos

A representação dos relacionamentos obedecerá às seguintes regras:

- cada relacionamento **1:1** será representado por uma coluna em uma das classes;
- cada relacionamento **1:n** ou **0:n** será representado por uma coluna de chave estrangeira na classe de maior multiplicidade;
- cada relacionamento **m:n** será representado, de preferência, por uma tabela separada, que corresponderá a uma classe de associação.



Figura 98 - Exemplo de classes de análise persistentes com relacionamento

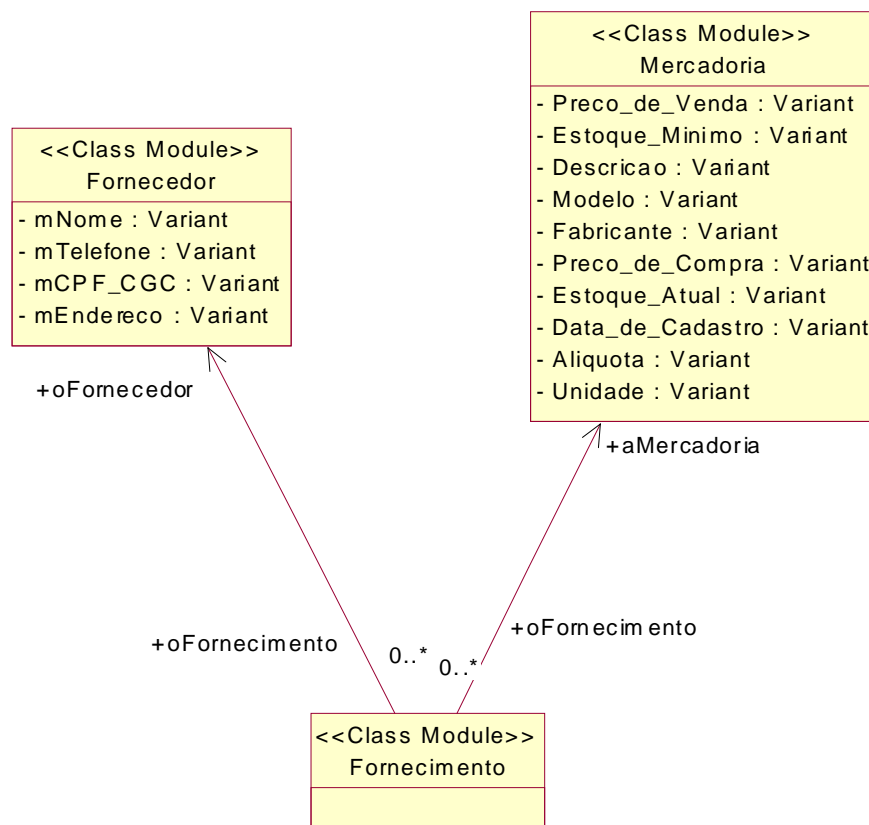


Figura 99 - Exemplo de classes de desenho persistentes com relacionamentos

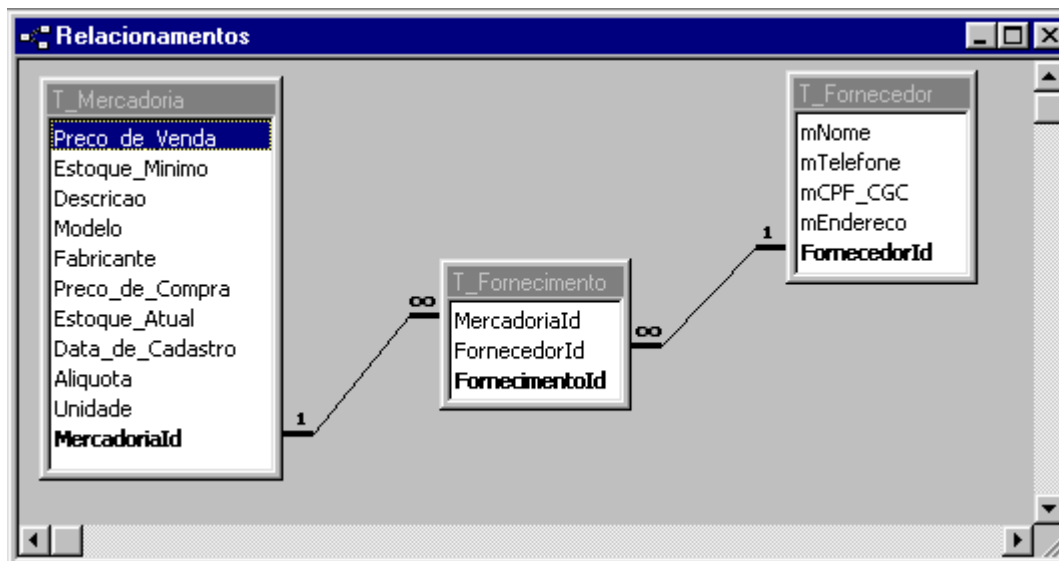


Figura 100 - Tabelas construídas no Microsoft Access

As classes do modelo de análise da Figura 98 foram convertidas, no modelo de desenho, para a Figura 99. Um relacionamento **m:n** foi substituído por uma classe intermediária. A Figura 100 mostra as tabelas correspondentes em um sistema de gestão¹⁴ de bancos de dados.

3.4.2.3 Representação da herança

Para a representação da herança, existem as seguintes possibilidades.

- Os atributos herdados são copiados em todas as tabelas das classes herdeiras. Não há tabela para representar a classe base.
- A classe base tem sua tabela, que é referida pelas tabelas das classes herdeiras. As tabelas das classes herdeiras só representam os atributos próprios. Uma visão pode fornecer os atributos herdados.



Figura 101 - Exemplo de classes persistentes com relacionamento de herança

¹⁴ Normalmente, traduzimos *management* por gestão. Neste caso, deve-se notar que o termo "sistemas de **gerência** de bancos de dados" é usual na respectiva comunidade.

quantidade	preco_total	Item_de_mercadoria_Id

numero_da_venda	Item_de_venda_Id

Figura 102 - Tabelas correspondentes a classes com herança

numero_da_venda	quantidade	preco_total

Figura 103 - Visão correspondente a classe herdeira

3.4.2.4 Outras questões

A parte estática da modelagem orientada a objetos é semelhante à modelagem de Entidades - Relacionamentos e, por isso, as tabelas geradas a partir de modelos orientados a objetos geralmente estarão na terceira forma normal, automaticamente. Entretanto, tabelas podem ser partidas por questões de desempenho. Quando há necessidade de desnormalizar, os mecanismos de tradução entre paradigmas devem ocultar esses aspectos de implementação.

Outros aspectos mais avançados incluem a definição de restrições de integridade referencial e índices. As restrições de integridade protegem os bancos de dados contra possíveis estados inconsistentes. Os índices aumentam a eficiência de navegação do banco de dados. Para maiores detalhes de como derivá-los do modelo orientado a objetos, vide [Blaha+98] e [Blaha+99].

3.4.3 Tradução entre paradigmas

3.4.3.1 Mecanismos de tradução

A tradução entre os paradigmas deve ser feita por meio de um mecanismo contido em um pacote lógico específico. Quando objetos devam ser armazenados no banco de dados, as classes desse mecanismo são responsáveis por retirar dos objetos do domínio os atributos que serão guardados no banco de dados, convertê-los para os tipos correspondentes suportados pelo sistema de gestão do banco de dados e gerar os comandos correspondentes da linguagem desse sistema. Essa linguagem é um dialeto da linguagem SQL, cujos detalhes variam conforme o fabricante.

Além disso, os fabricantes oferecem componentes de acesso a bancos de dados, dos quais essas classes devem tirar partido, para tornar o código mais simples e mais confiável. O pacote lógico que contém os mecanismos de tradução isola as porções de código que são dependentes de tecnologia específica.

3.4.3.2 Consistência de objetos

Deve-se cuidar de manter a consistência entre os objetos (existentes na memória principal) e suas imagens, existentes como linhas das tabelas do banco de dados. Em particular, não se deve criar objetos diferentes para representar a mesma linha de tabela.

Podem-se colocar nos mecanismos de tradução lógica adequada para manter a consistência. Essa lógica deve verificar se um objeto cuja construção é solicitada corresponde a uma linha de tabela já representada na memória principal.

3.4.3.3 Transações

A representação de transações deve ser feita por meio de um mecanismo que garanta a sua atomicidade, isto é, nenhuma transação pode ser executada parcialmente, deixando o banco de dados em estado inconsistente. Uma classe Transação pode ser usada para ocultar um protocolo de *commit/rollback*.

Por esse protocolo, as partes de uma transação são efetuadas de forma provisória. Só ao final da transação as modificações se tornam definitivas (*committed*), se toda a transação for bem sucedida, ou todas são anuladas (*rolled-back*) se o sucesso da transação não for completo. Tipos específicos de transação (consulta, atualização etc.) podem ser realizados por classes derivadas.

3.4.3.4 Representação das operações

A representação de operações é um problema inerente ao modelo relacional, que, na sua forma pura, só representa os atributos. As operações normalmente existirão apenas no código das classes correspondentes às tabelas.

Para guardá-las no próprio banco de dados, é preciso recorrer a extensões do modelo relacional, como gatilhos (*triggers*), procedimentos armazenados (*stored procedures*) e extensões de orientação a objetos. Estas extensões são dependentes do fabricante, embora exista um esforço de padronização delas na linguagem SQL.

Testes

1 *Princípios*

1.1 Visão geral

Apresenta-se neste capítulo um conjunto de procedimentos e técnicas para a realização do fluxo de Testes de software. Embora menos eficazes que as revisões para a remoção de defeitos, os testes são indispensáveis para remover os defeitos que ainda escapam das revisões e para avaliar-se o grau de qualidade de um produto e de seus componentes.

O teste exaustivo é geralmente impossível, mesmo para produtos relativamente simples. Para testar um programa cuja entrada seja um texto de dez caracteres, por exemplo, é necessário analisar 26^{10} combinações. Enquanto muitos desses testes são redundantes, a utilização de força bruta pode deixar a descoberto muitos casos com alta probabilidade de defeitos, como entradas com mais de 10 caracteres. Focaliza-se aqui o desenho de testes que tenham a mais alta probabilidade de descobrir defeitos com o mínimo de esforço.

As principais referências deste capítulo são os padrões de testes contidos em [IEEE94]. Outras referências importantes são [Humphrey90], [Perry95] e [Pressman95].

1.2 Objetivos

A atividade de testes é uma etapa crítica para o desenvolvimento de software. Frequentemente, a atividade de testes insere tantos erros em um produto quanto a própria implementação. Por outro lado, o custo para correção de um erro na fase de manutenção é de sessenta a cem vezes maior que o custo para corrigi-lo durante o desenvolvimento [Pressman95].

Embora as revisões técnicas sejam mais eficazes na detecção de defeitos, os testes são importantes para complementar as revisões e aferir o nível de qualidade conseguido. A realização de testes é, quase sempre, limitada por restrições de cronograma e orçamento. É importante que os testes sejam bem planejados e desenhados, para conseguir-se o melhor proveito possível dos recursos alocados para eles.

Um objetivo central de toda a metodologia dos testes é maximizar a **cobertura** destes. Deseja-se conseguir detectar a maior quantidade possível de defeitos que não foram apanhados pelas revisões, dentro de dados limites de custo e prazos.

1.3 Métodos

Para serem eficazes, os testes devem ser cuidadosamente desenhados e planejados. Testes irreproduzíveis e improvisados devem ser evitados. Durante e após a realização dos testes, os resultados de cada teste devem ser minuciosamente inspecionados, comparando-se resultados previstos e obtidos.

Os desenvolvedores não são as pessoas mais adequadas para testar seus próprios produtos. Assim como nas revisões, os autores têm maior dificuldade em enxergar problemas, comparados com pessoas que não participaram do desenho e da implementação. O trabalho de testadores independentes é particularmente importante no caso dos testes de aceitação. Possivelmente, os mesmos testadores independentes se encarregarão também dos testes de integração.

O planejamento e o desenho dos testes devem ser feitos por pessoas experientes, que conheçam adequadamente a metodologia de testes usada. Por outro lado, a realização dos testes baseados em planos e especificações de testes bem definidos pode ser feita por pessoas menos experientes, ou até parcialmente automatizada.

Os testes são indicadores da qualidade do produto, mais do que meios de detecção e correção de erros. Quanto maior o número de defeitos detectados em um software, provavelmente maior também o número de defeitos não detectados. A ocorrência de um número anormal de defeitos em uma bateria de testes indica uma provável necessidade de redesenho dos itens testados.

Existem basicamente duas maneiras de se construírem testes:

- **Método da caixa branca:** tem por objetivo determinar defeitos na estrutura interna do produto, através do desenho de testes que exercitem suficientemente os possíveis caminhos de execução. Os testes de unidade são geralmente de caixa branca.
- **Método da caixa preta:** tem por objetivo determinar se os requisitos foram total ou parcialmente satisfeitos pelo produto. Os testes de caixa preta não verificam como ocorre o processamento, mas apenas os resultados produzidos. Os testes de aceitação e de regressão normalmente são de caixa preta. Os testes de integração geralmente misturam testes de caixa preta e de caixa branca.

1.4 Baterias de testes

A lista a seguir enumera e descreve brevemente os tipos de **baterias de testes**, ou conjuntos de testes de software de determinada categoria. Estes conjuntos estão relacionados com as iterações do Praxis.

4. **Testes de Unidade** têm por objetivo **verificar** um elemento que possa ser logicamente tratado como uma unidade de implementação; por exemplo, uma subrotina ou um módulo. Em produtos implementados com tecnologia orientada a objetos, uma unidade é tipicamente uma classe. Em alguns casos, pode ser conveniente tratar como unidade um grupo de classes correlatas. No Praxis, cada Liberação pode ter várias baterias de testes de unidade, correspondentes às unidades que as constituem.
5. **Testes de Integração** têm por objetivo **verificar** as interfaces entre as partes de uma arquitetura de produto. Dentro do Praxis, esses testes têm por objetivo verificar se as unidades que compõem cada Liberação funcionam corretamente, em conjunto com as unidades já integradas, implementando corretamente os casos de uso cobertos por essa Liberação Executável. No Praxis, cada Liberação tipicamente tem uma bateria de testes de integração.
6. **Testes de Aceitação** têm por objetivo **validar** o produto, ou seja, verificar se este atende aos requisitos especificados. Eles são executados em ambiente o mais semelhante possível ao ambiente real de execução. No Praxis, existem duas baterias de testes de aceitação: ao final da Construção (Testes Alfa) e no início da Transição (Testes Beta). Os testes de aceitação podem ser divididos em testes funcionais e não funcionais.

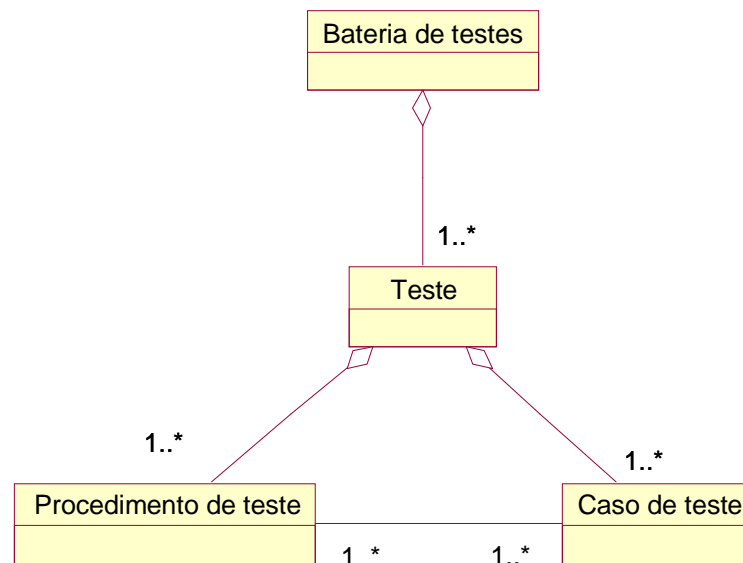


Figura 104 – Estrutura de testes

No padrão de nomenclatura de software do IEEE (IEEE Standard Glossary, in [IEEE94]), no Processo Unificado [Jacobson+99] e no TSP [Humphrey+99], a bateria de testes para validação do produto como um todo é chamada de Testes de Sistema, reservando-se o termo Testes de Aceitação para os testes feitos pelo cliente como parte de um procedimento de aceitação do produto. Como no Praxis os conteúdo desses dois grupos de testes é idêntico, preferiu-se chamá-los de Testes de Aceitação, sejam eles realizados no ambiente do fornecedor (Testes Alfa), ou do cliente (Testes Beta).

Uma categoria especial é formada pelos **Testes de Regressão**, que executam novamente um subconjunto de testes previamente executados. Seu objetivo é assegurar que alterações em partes do produto não afetem as partes já testadas. As alterações realizadas, especialmente durante a manutenção, podem introduzir erros nas partes previamente testadas.

A maior utilidade dos testes de regressão aparece durante o processamento de solicitações de manutenção. Entretanto, testes de regressão podem ser executados em qualquer passo do desenvolvimento. Por exemplo, para cada Liberação, deve ser feita uma regressão com os testes de integração das Liberações anteriores.

2 Atividades

2.1 Visão geral

O processo de testes tem dois grandes grupos de atividades para cada bateria de testes: **preparação e realização** dos testes. Durante a preparação, é elaborado o plano da bateria e são desenhadas as especificações de cada teste. Durante a realização, os testes são executados, os defeitos encontrados são, se possível, corrigidos, e os relatórios de teste são redigidos.

Normalmente, a preparação se inicia pelos testes de mais alto nível, ou seja, os testes de aceitação, baseados principalmente na Especificação de Requisitos do Software. Desses testes e do Desenho das Liberações, elaborado no fluxo de Desenho e contido na Descrição do Desenho do Software, são derivados os testes de integração. O desenho detalhado das unidades, realizado durante cada Liberação, serve de base para o desenho dos testes de unidade.

	Entradas
Testes de aceitação	Especificação dos Requisitos Plano de Desenvolvimento Plano da Qualidade
Testes de integração	Descrição do Desenho (desenho de alto nível) Descrição dos Testes (testes de aceitação)
Testes de unidade	Descrição do Desenho (desenho detalhado da unidade) Descrição dos Testes (testes de integração) Código Fonte da unidade

Tabela 67 – Entradas para o desenho dos testes

A realização dos testes ocorre em ordem inversa, dando origem a uma estrutura em V (Figura 105). As atividades de preparação produzem os documentos de Descrição dos Testes, que servem de insumo para as atividades de realização. Estas produzem os Relatórios dos Testes.

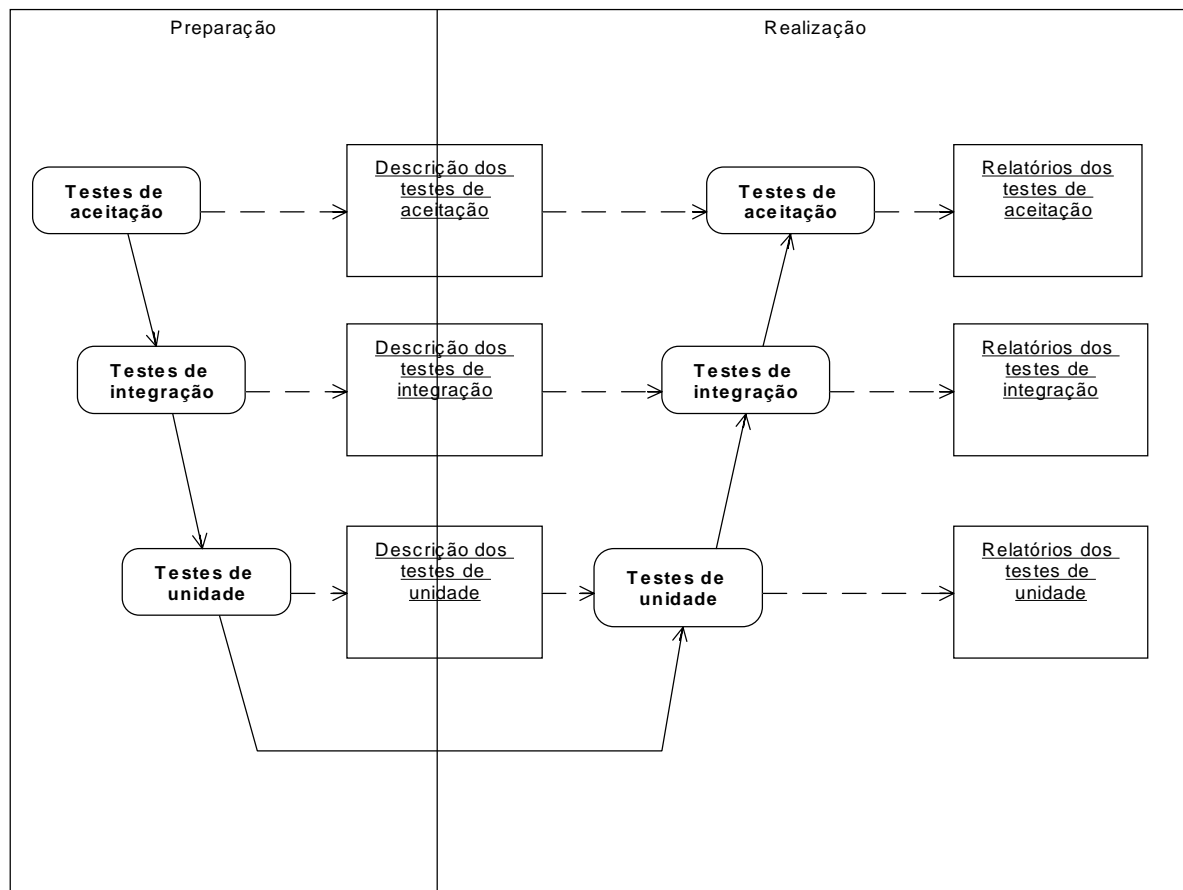


Figura 105 – Estrutura em V do fluxo de testes

A preparação e realização de cada bateria correspondem a uma passada completa pelo fluxo de Testes. Este fluxo é composto das atividades mostradas na Figura 106. Essas atividades são adaptadas a partir da norma para testes de unidade do IEEE:

IEEE. *IEEE Std. 1008 – 1987. IEEE Standard for Software Unit Testing*, in [IEEE94].

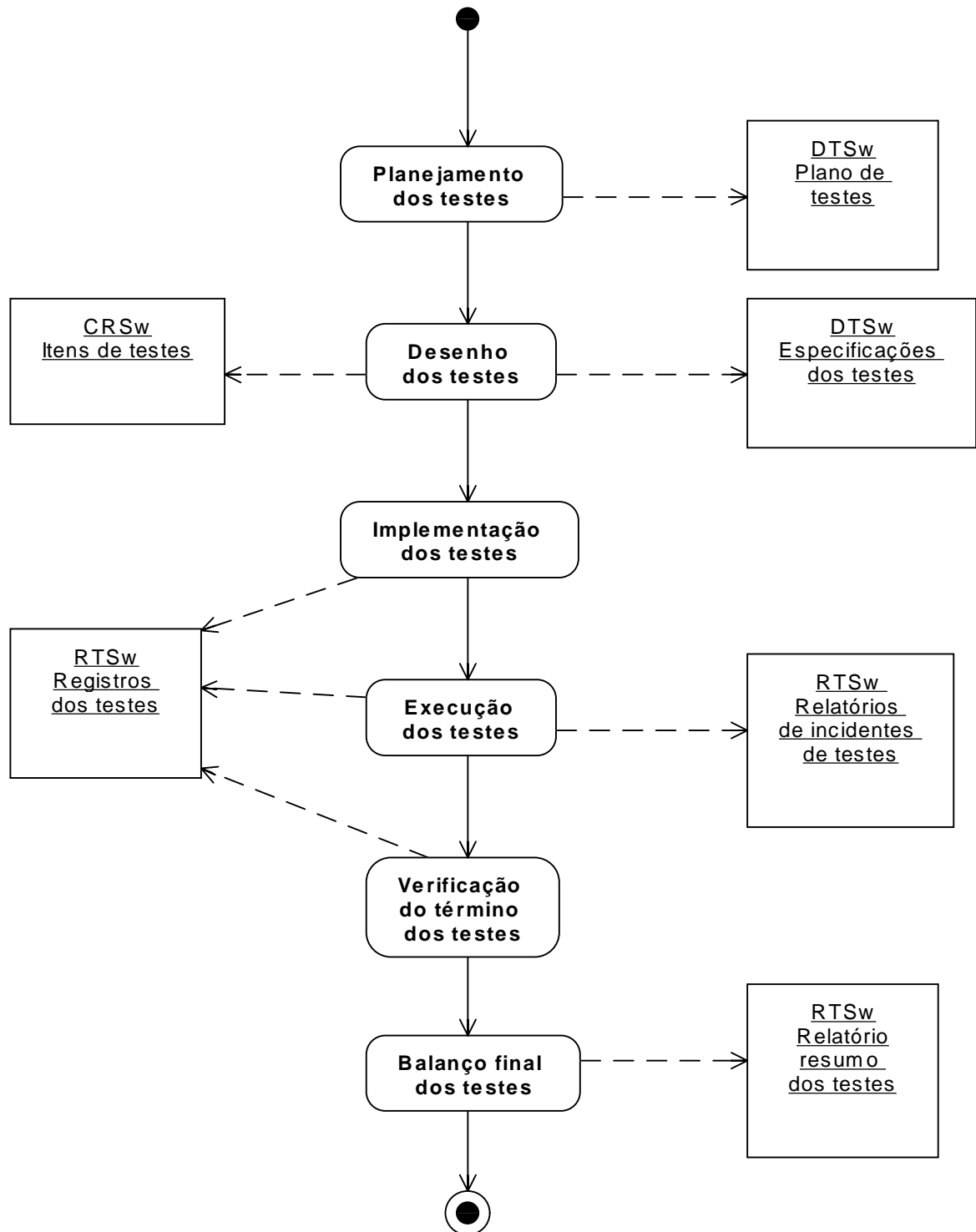


Figura 106 - Atividades e artefatos do fluxo de Testes

No caso dos testes de aceitação, os passos correspondentes à preparação serão executados durante a iteração de Desenho Inicial da fase de Construção. Os passos correspondentes à realização serão executados durante os Testes Alfa e repetidos durante os Testes Beta. Para os testes de integração, a preparação ocorre no início de cada Liberação, e a realização ocorre no final dessa Liberação.

Esse fluxo poderia ser usado também para os testes de unidade, e até deveria sê-lo, no caso de unidades críticas (cuja falha possa acarretar consequências graves, como risco de vida ou perdas materiais vultosas). Na prática, os testes de unidade são normalmente feitos pelos próprios desenvolvedores, e

geralmente não há necessidade de processos tão formais. Por isso, os testes de unidade serão tratados dentro do fluxo de Implementação, usando-se métodos simplificados.

2.2 Detalhes das atividades

2.2.1 *Planejamento dos testes*

2.2.1.1 Visão geral

O planejamento dos testes de produtos não triviais é complexo, envolvendo muitos aspectos técnicos e gerenciais. Para facilitar a descrição, a atividade de planejamento foi dividida em três tarefas separadas: planejamento inicial, identificação dos itens a testar e refinamento do plano de testes. Antes de descrevê-las, faz-se na próxima subseção uma descrição dos planos de testes.

2.2.1.2 Planos de testes

O plano de testes de aceitação é único e obrigatório dentro do Praxis. Ele será normalmente preenchido durante a iteração de Desenho Inicial. Sua base é a Especificação de Requisitos do Software; geralmente, cada caso de uso gera uma especificação de teste funcional. Esse plano inclui as tarefas dos Testes Alfa e dos Testes Beta.

Os planos de testes de integração são preenchidos no início de cada Liberação. Os testes escolhidos para cada Liberação partem normalmente de um subconjunto dos testes de aceitação, correspondente aos casos de uso implementados nessa Liberação.

Testes adicionais de caixa branca são usados para verificar se as interfaces¹⁵ dos componentes implementados nessa Liberação estão funcionando corretamente (entre si e com os componentes herdados das liberações anteriores). Estes testes adicionais podem ser derivados das realizações dos casos de uso de desenho, onde as interfaces são exercitadas através das colaborações entre componentes.

Cada plano de testes de unidade é preenchido durante a respectiva implementação. Os testes de unidade geralmente requerem a construção de componentes de teste. Normalmente, só são desenhados, registrados e guardados os testes das unidades críticas, para uso futuro em manutenção.

Tipicamente, uma organização começaria por planejar e desenhar os testes de aceitação. Na medida em que a cultura da organização absorvesse esses testes, passaria a planejar e a desenhar também os testes de integração, e finalmente os testes de unidades críticas.

O padrão aqui adotado para planos de testes prevê a estrutura mostrada na Tabela 68.

¹⁵ Neste capítulo, o termo interfaces se refere genericamente aos mecanismos de comunicação entre unidades, e não tem o sentido técnico específico definido pela UML.

Item	Descrição
Identificador do plano de testes	Identificador único para o plano.
Introdução	Objetivos, histórico, escopo, referências a documentos.
Itens a testar	Conjunto dos itens cobertos pelo plano.
Aspectos a testar	Conteúdo dos testes que serão feitos.
Aspectos que não serão testados	Aspectos significativos que não serão testados.
Abordagem	Opções metodológicas que são aplicáveis ao conjunto dos testes.
CrITÉRIOS de completEza e sucesso	Condições que devem ser satisfeitas e estado que deve ser atingido para que o conjunto dos testes seja considerado bem sucedido.
CrITÉRIOS de suspensão e retomada	Problemas graves que podem provocar a interrupção dos testes.
Resultados do teste	Artefatos que serão produzidos durante a realização da bateria de testes.
Tarefas de teste	Lista detalhada das tarefas que são cobertas por este plano.
Ambiente	Hardware e software das configurações usadas para o conjunto dos testes.
Responsabilidades	Responsabilidades de cada um dos participantes dos testes.
Agenda	Data de início e de fim de cada tarefa do plano.
Riscos e contingências	Riscos e contingências aplicáveis aos testes deste plano.
Aprovação	Nomes e assinaturas dos responsáveis pela aprovação do plano.

Tabela 68 - Estrutura do plano de testes

2.2.1.3 Planejamento inicial

A Tabela 69 mostra em detalhes o planejamento inicial dos testes, primeira tarefa dessa atividade. Os insumos são documentos de planejamento, como o Plano de Desenvolvimento do Software, no caso dos testes de aceitação, e a seção de Plano das Liberações Executáveis da Descrição do Desenho do Software, no caso de testes de integração.

São determinados as abordagens a adotar, os critérios de completEza e término e os requisitos de recursos, obtendo-se uma agenda de testes. Um item a testar, por exemplo, pode ser o produto inteiro ou um componente, versões parciais do produto, ou módulos de diversos níveis, dependendo do tipo de teste. São definidos os principais pontos do planejamento, de forma a se poder requisitar os recursos necessários.

Descrição	Planejamento inicial dos testes.
Insumos	Plano de Desenvolvimento do Software (testes de aceitação). Descrição do Desenho do Software – Plano das Liberações (teste de integração).
Atividades	<p>Escolher abordagens para os testes, identificando:</p> <ul style="list-style-type: none"> áreas de risco que devem ser testadas; restrições existentes aos testes; fontes existentes de casos de testes; métodos de validação dos casos de teste; técnicas para registro, coleta, redução e validação das saídas; necessidades de estruturas provisórias. <p>Especificar condições de completeza dos testes:</p> <ul style="list-style-type: none"> itens a serem testados; grau de cobertura por itens. <p>Especificar condições de término dos testes:</p> <ul style="list-style-type: none"> condições para término normal; condições de término anormal e procedimentos a adotar nestes casos. <p>Determinar requisitos de recursos:</p> <ul style="list-style-type: none"> peçoas; hardware; software de sistema; ferramentas de teste; arquivos de testes; formulários; suprimentos. <p>Especificar agenda dos testes.</p>
Resultados	<p>Informação geral de planejamento dos testes.</p> <p>Requisições de recursos para os testes.</p>

Tabela 69 - Planejamento inicial dos testes

2.2.1.4 Identificação dos itens a testar

A segunda tarefa do planejamento dos testes (Tabela 70) compreende a identificação dos itens a testar. Os insumos são a Especificação de Requisitos do Software, no caso dos testes de aceitação, e as partes apropriadas da Descrição do Desenho do Software, no caso dos demais testes. São identificados os requisitos a testar, é determinado o status dos itens sob teste (por exemplo, itens novos ou modificados) e são caracterizados os dados para os casos de teste. É produzida a lista dos itens e aspectos a testar.

Descrição	Caracterização dos itens a testar.
Insumos	Especificação de Requisitos do Software (testes de aceitação). Descrição do Desenho do Software – Desenho Arquitetônico e Plano das Liberações Executáveis (testes de integração). Descrição do Desenho do Software – Desenho Detalhado da unidade (testes de unidade).
Atividades	Identificar: requisitos funcionais dos itens a testar; requisitos não funcionais dos itens a testar; status dos itens a testar, dentro do projeto; características dos dados de entrada e saída.
Resultados	Lista de itens e aspectos a testar. Eventuais pedidos de esclarecimentos aos desenvolvedores, relativos aos itens a testar.

Tabela 70 - Caracterização dos itens a testar

2.2.1.5 Planejamento detalhado

Conhecidos os itens e aspectos a testar, é possível fazer um planejamento detalhado, na terceira tarefa (Tabela 71). Os detalhes restantes do plano de teste são completados.

Descrição	Refinamento do plano de testes.
Insumos	Lista de itens e aspectos a testar. Informação geral de planejamento dos testes.
Atividades	Determinar: detalhes da abordagem; requisitos de recursos específicos dos itens a testar; cronograma detalhado.
Resultados	Plano de testes completo.

Tabela 71 - Refinamento do plano de testes

2.2.2 Desenho dos testes

2.2.2.1 Tarefas de desenho dos testes

Nessa atividade (Tabela 72) é feito o desenho da bateria de testes. São completadas as especificações dos testes da bateria, desenhando-se os procedimentos e casos de teste. São selecionados para reutilização artefatos de teste de outros projetos que possam ser reaproveitados. É definida a ordem de execução dos casos de teste. Em certos casos, principalmente nos testes de unidade, pode ser necessário solicitar o redesenho de módulos, para melhorar a testabilidade destes.

Descrição	Desenho da bateria de testes.
Insumos	Especificação de Requisitos do Software (testes de aceitação). Descrição do Desenho do Software – Desenho Arquitetônico e Plano das Liberações Executáveis (testes de integração). Descrição do Desenho do Software – Desenho Detalhado da unidade (testes de unidade). Plano de testes. Especificações de testes anteriores (se houver).
Atividades	Desenhar bateria de testes, estabelecendo: objetivos dos testes; reutilização de especificações de testes existentes; ordenamento dos casos de teste. Especificar os procedimentos de teste. Especificar os casos de teste.
Resultados	Especificações dos testes. Solicitações de melhoria do desenho para a testabilidade, quando necessário.

Tabela 72 - Desenho da bateria de testes

2.2.2.2 Especificações de testes

As especificações de testes contêm os detalhes dos testes a serem realizados. A separação entre planos e especificações permite o reaproveitamento das especificações em diversos planos. Uma especificação de teste reflete o resultado do desenho deste, detalhando vários aspectos que são específicos em relação ao teste de um caso de uso ou outra característica.

Cada especificação de teste contém os seguintes itens subordinados:

- Os procedimentos de teste contêm uma seqüência de ações que devem ser executadas para realizar um grupo de testes semelhantes. Geralmente, cada procedimento de teste corresponde a um roteiro importante de um caso de uso. Um procedimento pode ser executado de forma manual ou automática. Neste último caso, deve ser codificado em linguagem de script da ferramenta de automação de testes.
- Os casos de teste contêm valores de entradas e valores de saídas esperados para cada instância de teste. Esses valores de entrada são escolhidos de acordo com critérios que maximizam a cobertura da especificação de teste.

Os casos de teste têm uma ordem especificada de execução, pois a execução correta de um caso pode depender de um estado de uma base de dados, que é produzido por um caso anterior. A separação entre procedimentos e casos permite a reutilização de procedimentos por casos que diferem apenas pelo conjunto de valores. Permite também que procedimentos complexos sejam divididos em procedimentos menores e possivelmente reaproveitáveis por vários casos de teste.

O padrão aqui adotado para especificações de testes prevê a estrutura mostrada na Tabela 73.

Item		Descrição
Identificador da especificação de teste		Identificador único para este teste.
Aspectos a serem testados		Aspectos a testar combinados neste teste.
Detalhes da abordagem		Detalhes de abordagem específicos deste.
Identificação dos testes	Procedimentos de teste	Procedimentos associados a este teste.
	Casos de teste	Casos de teste associados a este teste.
Critérios de completeza e sucesso		Critérios de completeza e sucesso específicos deste teste.
Procedimentos de teste		Descrição de cada um dos procedimentos de teste listados anteriormente.
Casos de teste		Descrição de cada um dos casos de teste listados anteriormente.

Tabela 73 - Estrutura de uma especificação de testes

2.2.2.3 Desenho dos procedimentos de teste

Os procedimentos de teste devem descrever a sequência de passos necessária para executar uma variação de teste. Nos testes funcionais, cada variação é tipicamente baseada em um roteiro do respectivo caso de uso. É conveniente prever procedimentos de teste para execução de seqüências erradas mas possíveis. O fluxo do procedimento representa os passos que devem ser executados por um testador humano ou automatizado, em termos das telas, campos e comandos envolvidos. Os valores efetivos dos campos serão determinados por cada caso de teste.

Objetivo	Verificar se a inclusão de um usuário é feita corretamente no Merci.
Fluxo	<ol style="list-style-type: none"> 1. Abrir interface Tela de Usuários. 2. Acionar <i>Novo</i>. 3. Inserir <u>Nome</u>, <u>Login</u>, <u>Senha</u>. 4. Acionar <i>Salvar</i>. 5. Inserir <u>Login</u>. 6. Acionar <i>Pesquisar</i>.

Figura 107 – Exemplo de procedimento de teste

2.2.2.4 Desenho dos casos de teste

Cada teste constante de uma bateria procura verificar uma característica dos itens sob teste. Tratando-se de uma bateria de testes de aceitação, cada teste geralmente verificará a implementação de um caso de uso do produto. Cada teste consta da execução de um ou mais procedimentos de teste, que são aplicáveis a vários conjuntos de entradas, chamados de **casos de teste**. Cada caso de teste, por sua vez, tem um ou mais procedimentos de teste associados.

Um bom caso de teste tem por objetivo detectar defeitos ainda não descobertos, e não demonstrar que o programa funciona corretamente. Ele deve obrigatoriamente incluir uma descrição das saídas esperadas, que será usada para comparação com as saídas reais obtidas em cada execução do caso de teste. Devem existir casos de teste que cubram tanto entradas válidas quanto inválidas.

Os casos de teste devem cobrir as combinações de entrada relevantes para dar ao teste uma cobertura adequada. Além disso, é conveniente prever casos de teste para execução de procedimentos errados mas possíveis. Cada caso de teste deve corresponder a um conjunto de entradas e saídas esperadas.

Entradas	Campo	Valor
	Nome	Caixeiro_01
	Login	Caixeiro_01
	Senha	www
Saídas esperadas	Campo	Valor
	Nome	Caixeiro_01
	Grupos do Usuário	Caixeiro

Figura 108 – Exemplo de caso de teste

2.2.2.5 Revisão do desenho dos testes

Um bom desenho de testes gera grande quantidade de informação. Para cada caso de uso, as respectivas especificações de testes geralmente ocupam várias páginas. Vê-se que mesmo em sistemas relativamente simples, com poucas dezenas de casos de uso, a Descrição dos Testes pode chegar a dezenas ou centenas de páginas.

Neste volume de informação, os desenhistas de testes facilmente introduzem defeitos. Por isso, é recomendável que os planos e especificações dos testes sejam submetidos a uma revisão técnica formal, pelo menos no caso dos testes de aceitação. No Praxis, essa revisão é feita juntamente com a revisão da Descrição do Desenho, no final da iteração de Desenho Inicial.

2.2.3 Implementação dos testes

Nessa atividade (Tabela 74) é realizada a implementação dos testes. O ambiente de teste é completamente preparado, tornando disponíveis todos os recursos necessários. Os itens a testar são instalados e configurados, conforme necessário, assim como as ferramentas e componentes de teste. Os componentes podem ser reaproveitados de testes anteriores, ou desenvolvidos especialmente para os testes em questão.

Descrição	Implementação dos testes.
Insumos	Plano dos testes. Especificações dos testes. Recursos para os testes. Itens a testar. Ferramentas de teste. Dados de atividades anteriores de teste, se houver. Estruturas provisórias de testes anteriores, se houver.
Atividades	Configurar ambiente de testes. Disponibilizar todos os recursos necessários. Instalar itens a testar, ferramentas e estruturas provisórias.
Resultados	Itens a testar instalados e configurados. Ferramentas e estruturas provisórias instaladas e configuradas.

Tabela 74 - Implementação dos testes

2.2.4 Execução dos testes

Essa atividade (Tabela 75) executa os testes da bateria, produzindo os relatórios correspondentes. Podem resultar, dos problemas encontrados neste passo, solicitações de correção dos itens sob teste, assim como alterações nos planos e especificações dos próprios testes.

Descrição	Execução dos testes.
Insumos	Itens a testar instalados e configurados. Ferramentas e estruturas provisórias instaladas e configuradas. Plano dos testes. Especificações dos testes. Recursos para os testes. Dados de atividades anteriores de teste, se houver.
Atividades	Executar os testes. Determinar e registrar os resultados. Analisar as falhas e tomar as providências adequadas: em caso de falhas do próprio teste; em caso de defeitos de implementação dos itens; em caso de defeitos de desenho dos itens.
Resultados	Relatórios dos testes. Especificações de testes revisadas, se for o caso. Casos de testes revisados, se for o caso. Solicitações de investigação e correção de defeitos, se for o caso.

Tabela 75 - Execução dos testes

2.2.5 Verificação do término dos testes

Essa atividade (Tabela 76) determina se estão satisfeitas as condições para completeza e sucesso dos testes. Caso necessário para garantir a cobertura desejada, testes suplementares podem ser desenhados, retornando-se ao passo de execução.

Descrição	Verificação do término dos testes.
Entradas	Plano dos testes. Especificações dos testes. Relatórios dos testes.
Atividades	Checar término normal dos testes: verificar se há necessidade de mais testes; se não houver, registrar o término normal. Checar término anormal dos testes, documentando o ocorrido. Suplementar o conjunto de testes, se necessário. Retornar à execução dos testes, se necessário.
Resultados	Relatórios dos testes verificados e completados. Especificações de testes revisadas, se for o caso. Casos de testes revisados, se for o caso.

Tabela 76 - Verificação do término dos testes

2.2.6 *Balanço final*

Essa atividade (Tabela 77) realiza o balanço final dos testes da bateria. São registradas as lições aprendidas, completando-se o relatório de resumo dos testes. Se possível, são identificados artefatos reutilizáveis em outros testes, inclusive procedimentos, casos e componentes de teste.

Descrição	Balanço dos testes.
Entradas	Relatórios dos testes verificados. Especificações de testes revisadas, se for o caso. Dados de testes revisados, se for o caso.
Atividades	Descrever o status dos testes, registrando: variações; avaliação dos términos anormais do teste; problemas não resolvidos. Descrever o status das unidades sob teste. Completar os relatórios dos testes. Colocar os artefatos reutilizáveis sob Gestão de Configurações.
Resultados	Relatório de resumo dos testes. Possivelmente, componentes de teste reutilizáveis.

Tabela 77 – Balanço final dos testes

3 *Técnicas*

3.1 *Visão geral*

Nesta seção são descritas técnicas específicas de cada bateria de testes. Os testes de integração são classificados de acordo com a abordagem escolhida para a integração. Os testes de aceitação são divididos em testes funcionais e testes de sistema. Em seguida, são tratados os testes de regressão.

O tratamento dos testes de unidade é deixado para o fluxo de Implementação. Esses testes geralmente são desenhados e realizados pelos próprios desenvolvedores, e não por uma equipe independente de testes.

3.2 *Testes de integração*

3.2.1 *Introdução*

O teste de integração sistematiza a atividade de integração dos módulos já submetidos ao teste de unidade, ao mesmo tempo em que as interfaces entre esses módulos são testadas. Alguns métodos de integração são discutidos a seguir. Eles diferem pela ordem em que são montadas as configurações de integração, ou construções (“*builds*”).¹⁶

Em alguns casos, testes constantes da bateria de aceitação podem ser usados como testes de integração. Em outros, há necessidade de utilizar cotos e controladores, possivelmente aproveitando os mesmos componentes usados para os testes de unidade.

3.2.2 *Abordagens*

A integração “*big-bang*” é, provavelmente, o método de integração mais freqüente na prática. As unidades são testadas individualmente e depois integradas de uma só vez. Esse tipo de integração é o

¹⁶ Não confundir este sentido do termo construção (“*build*”) com a fase de Construção (“*Construction*”).

que requer menos esforço, mas geralmente o resultado é desastroso. Diagnosticar um erro nessa situação é bastante complexo, pois é difícil isolar as causas. À medida que os erros são corrigidos, outros erros são introduzidos, por causa da desorganização do processo.

Na integração *bottom-up*, as unidades nos níveis mais subordinados são testadas individualmente, usando-se controladores para simular as funções das unidades superiores. À medida que se desenvolve o processo de integração, as unidades superiores substituem os controladores. Na Figura 109, as unidades U21 e U22 são testadas utilizando-se os controladores C1 e C2. No passo seguinte da integração, os controladores são substituídos pela unidade U1.

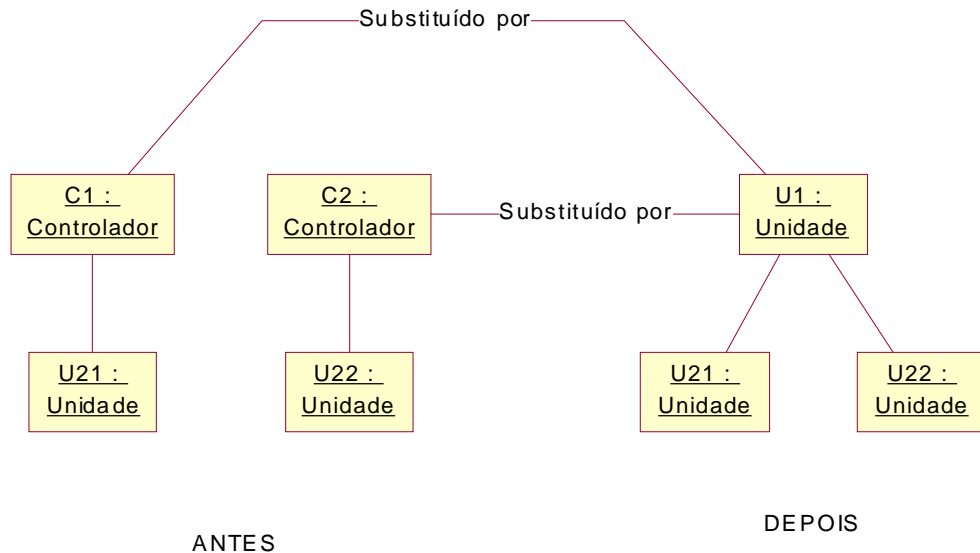


Figura 109 – Exemplo de integração *bottom-up*

Na de integração *top-down*, os testes iniciais são realizados com a unidade principal, e cada nova unidade introduzida adiciona funcionalidade ao sistema. As unidades subordinadas precisam ser simuladas por cotos (Figura 110). No início da integração, para que o teste seja eficaz, são necessários cotos muito sofisticados.,,

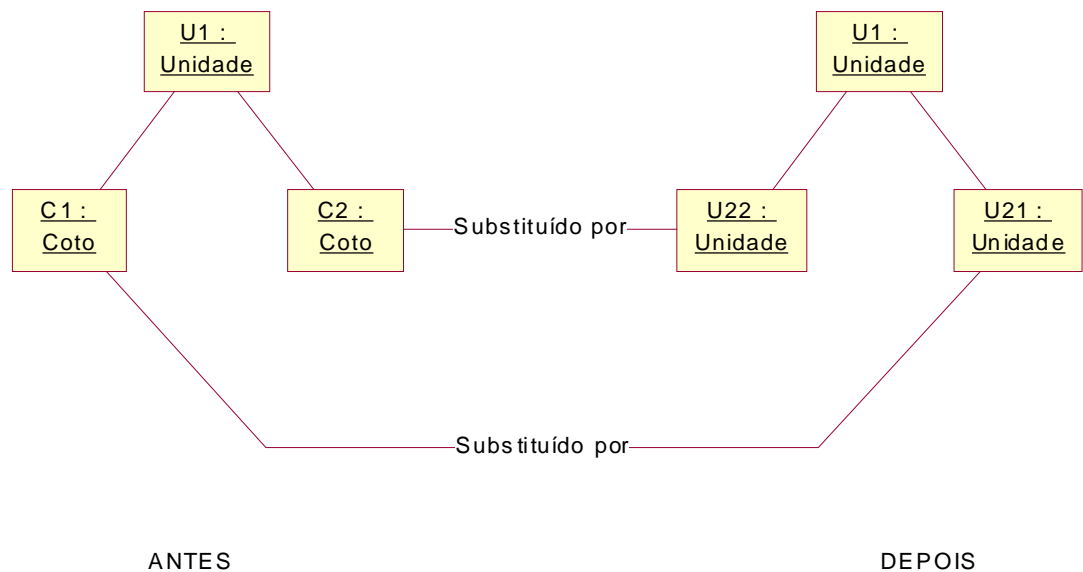


Figura 110 – Exemplo de integração *top-down*

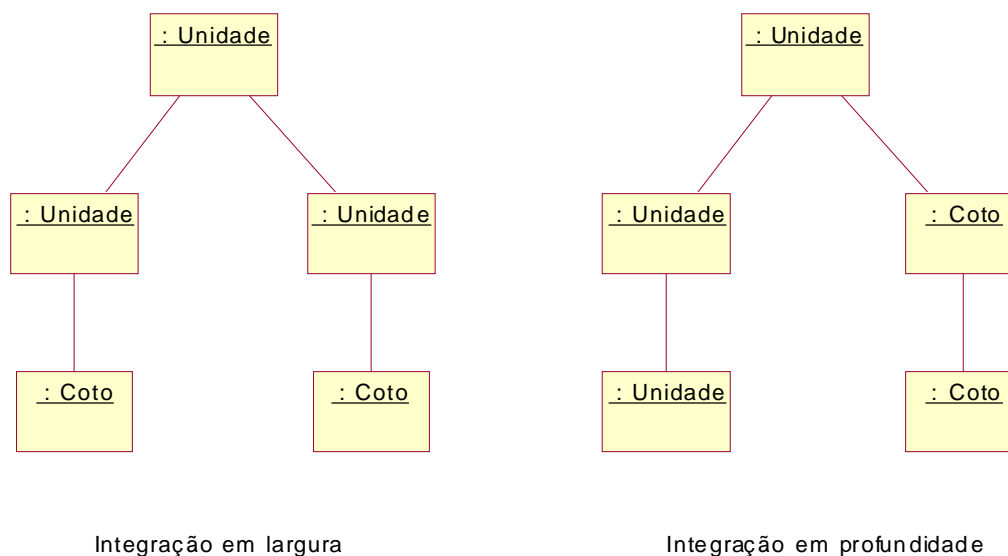


Figura 111 - Integração em largura × integração em profundidade

A integração *top-down* pode ser feita em largura ou em profundidade. Se a abordagem de integração adotada for em profundidade, os cotos são substituídos verticalmente ao longo da estrutura das unidades. Se a abordagem for em largura, os cotos são substituídos horizontalmente. Cada vez que novo módulo é integrado, o conjunto já integrado é testado. Os testes devem não só cobrir as funções adicionadas mas realizar uma regressão para confirmar que o restante dos módulos já integrados continua funcionando.

Principais características	Permite que módulos específicos críticos sejam testados mais cedo.
	Os módulos podem ser integrados em várias construções, de acordo com a necessidade e o planejamento do projeto.
	Maior ênfase na funcionalidade e desempenho dos módulos.
Vantagens	Não há necessidade de cotos.
	Maior facilidade para adaptar os testes aos recursos humanos disponíveis para executá-los.
	Erros em módulos críticos são detectados mais cedo.
	A produção de código avança mais rapidamente do que por meio da abordagem <i>top-down</i> .
	Em geral, esta abordagem é mais intuitiva
Desvantagens	Necessidade de controladores.
	Muitos módulos devem ser integrados antes de se obter um programa funcionando.
	Erros de interface são detectados mais tarde.

Tabela 78 – Análise da integração bottom-up

Principais características	O programa de controle é testado primeiro.
	Apenas um módulo é integrado a cada passo.
	Maior ênfase nas interfaces.
Vantagens	Não há necessidade de controladores.
	Com o programa de controle e alguns módulos se obtém um protótipo nas fases iniciais.
	Erros de interface são detectados mais cedo.
	Facilita a depuração das funções de cada módulo.
Desvantagens	Necessidade de cortes.
	Erros em módulos críticos nos níveis mais baixos da hierarquia são encontrados tardiamente.
	Como as fases iniciais são muito prolongadas, pode ser mais difícil determinar os recursos humanos necessários para a realização dos testes.
	Na prática, é muito difícil utilizar uma abordagem puramente <i>top-down</i> .
	A disponibilidade do protótipo nas fases iniciais facilita a demonstração dos resultados alcançados.

Tabela 79 – Análise da integração *top-down*

É possível realizar uma integração mista, integrando-se alguns módulos na forma *top-down* e outros na forma *bottom-up*. O Praxis determina que se realize a integração de modo que os maiores riscos identificados sejam atacados nas primeiras liberações e nos primeiros testes de integração.

É recomendável também que a ordem de integração complete a implementação de cada caso de uso, antes de prosseguir para os seguintes. No máximo, fluxos alternativos de implementação complexa podem ser deixados para liberações posteriores, documentando-se devidamente o que ficou para depois. Uma análise dos pontos fortes e fracos de cada forma de integração é apresentada na Tabela 78 e na Tabela 79.

3.2.3 Gestão de configurações

Como o processo de integração é incremental, é necessário controlar as várias configurações das liberações resultantes de cada passo da integração. Em se tratando de sistemas grandes, é recomendável que especialistas coordenem essa atividade, que abrange as seguintes tarefas:

- integração dos componentes em configurações de integração;
- manutenção das configurações (correções efetuadas durante o desenvolvimento);
- distribuição das construções para a equipe de desenvolvimento para a realização de testes.

As configurações de integração devem ser planejadas e controladas. Em projetos complexos, deve existir um Plano de Gestão de Configurações, que definirá os componentes de cada liberação e os componentes de teste associados a eles.

3.3 Testes de aceitação

3.3.1 Testes funcionais

3.3.1.1 Introdução

Os testes funcionais são desenhados para verificar a consistência entre o produto implementado e os respectivos requisitos funcionais. A completeza e a precisão da Especificação dos Requisitos do Software são fundamentais para assegurar a qualidade desses testes.

3.3.1.2 Partição de equivalência

A **partição de equivalência** é um método que divide o domínio de entrada em categorias de dados. Cada categoria revela uma classe de erros, permitindo que casos de teste na mesma categoria sejam eliminados sem que se prejudique a cobertura dos testes.

Para cada entrada do sistema, são identificados os conjuntos de valores válidos e inválidos associados que definem as classes¹⁷ de equivalência para essa entrada. Em um programa de gestão de pessoal, por exemplo, a idade de um funcionário pode variar entre 15 e 80 anos (de acordo com a especificação de requisitos do programa). As classes de equivalência para essa entrada são todos os valores inteiros menores que 15, valores inteiros entre 15 e 80, inclusive, e valores inteiros maiores que 80. Para cada uma dessas classes, qualquer valor tem, potencialmente, a mesma capacidade de detectar erros, sendo dispensável a execução de vários testes para valores pertencentes à mesma classe de equivalência.

A principal referência para a definição das classes de equivalência é a seção **3.1 Requisitos de interface externa** da Especificação dos Requisitos do Software. Cada definição de interface dessa subseção contém a definição das respectivas entradas. O detalhamento das entradas informa o respectivo tipo, limites inferior e superior ou lista de valores válidos. As classes de equivalência podem ser derivadas como se vê na Tabela 80:

Definição da entrada	Classes de equivalência
Intervalo válido	Uma válida, para os valores pertencentes ao intervalo; duas inválidas, para os valores menores e maiores que os limites inferior e superior, respectivamente.
Lista de valores válidos	Uma válida, para os valores incluídos na lista; uma inválida, para todos os outros valores.
Valor específico	Uma válida, que inclui o valor; duas inválidas, para os valores maiores e menores.
Lógica	Uma válida e uma inválida.

Tabela 80 - Desenho de classes de equivalência para teste funcional

3.3.1.3 Análise de valor limite

Em geral, erros nas fronteiras do domínio da entrada são mais frequentes do que nas regiões centrais. A **análise do valor limite** é uma técnica para a seleção de casos de teste que exercitam os limites. O emprego dessa técnica deve ser complementar ao emprego da partição de equivalência. Assim, em vez de se selecionar um elemento aleatório de cada classe de equivalência, selecionam-se os casos de teste nas extremidades de cada classe.

A seleção de casos de teste deve ser feita de acordo com as recomendações da Tabela 81

Entradas válidas	Casos de teste
Intervalo delimitado pelos valores a e b	Valores imediatamente abaixo de a , a , b e imediatamente acima de b
Série de valores	Valor imediatamente abaixo do mínimo, para o mínimo, para o máximo e imediatamente acima do máximo
Intervalo ou série de valores	Saídas máxima e mínima definidas
Estruturas de dados	Caso que exercite a estrutura em suas fronteiras

Tabela 81 - Desenho de casos de testes para análise de valor limite

¹⁷ Não confundir classes de equivalência com classes no sentido orientado a objetos.

3.3.1.4 Testes de comparação

Existem situações em que é necessário comparar as saídas de diferentes versões de um sistema quando submetidas às mesmas entradas. Esses testes se aplicam a situações como:

- uso de sistemas redundantes para aplicações críticas como controle de aeronaves;
- comparação de resultados de produtos em evolução.

Quando produtos são substituídos por versões mais novas que incluem mais funcionalidade, deve-se comparar os resultados das características que fazem parte de ambas as versões (não introduzem nova funcionalidade). Essas características já foram testadas pelos usuários com dados reais e tendem a estar mais estabilizadas. A comparação das saídas pode ser feita com o auxílio de uma ferramenta automatizada.

Nessa situação, casos de teste desenhados por meio de técnicas de caixa preta são usados para testar cada uma das versões existentes. Se as saídas forem consistentes, presume-se que todas as versões estejam corretas. Caso contrário, deve-se investigar em qual ou quais das versões se encontra o defeito.

3.3.1.5 Testes de tempo real

O desenho de testes para sistemas de tempo real não pode considerar apenas casos de teste de caixa preta e branca, mas também a temporização dos dados e o paralelismo das tarefas que os manipulam.

Os testes de tempo real podem ser executados através dos seguintes passos:

- **Teste de tarefas isoladas:** casos de testes de caixa preta e branca são desenhados para testar cada tarefa independentemente. Nessa etapa, não é possível detectar erros decorrentes de problemas de temporização. Apenas erros de lógica e funcionalidade são apontados.
- **Teste comportamental:** através de modelos executáveis por ferramentas de simulação é possível simular o comportamento de sistemas de tempo real e verificar como eles respondem a eventos externos. Cada um desses eventos identificados é testado individualmente, e o comportamento do sistema real é comparado ao comportamento apontado pelo modelo.
- **Testes de interação entre tarefas:** a fim de detectar erros de sincronização entre as tarefas, aquelas que se comunicam são testadas com diferentes taxas de dados e cargas de processamento. Devem ser desenhados casos de teste para avaliar situações de travamento, inanição e tamanho insuficiente de filas de mensagem.

3.3.2 Testes não funcionais

Os testes não funcionais procuram detectar se o comportamento do software ou sistema está consistente com a respectiva Especificação dos Requisitos quanto aos aspectos não funcionais. Esses testes cobrem, por exemplo:

Tipo de requisitos	Tipos de teste
Desempenho	Número de terminais suportados Número de usuários simultâneos Volume de informação que deve ser tratado
Dados persistentes	Frequência de uso Restrições de acesso Restrições de integridade Requisitos de guarda e retenção de dados
Outros atributos de qualidade	Funcionalidade Confiabilidade Usabilidade Manutenibilidade Portabilidade

Tabela 82 - Tipos de testes de sistema

3.4 Testes de regressão

As alterações feitas durante a correção dos erros detectados podem introduzir problemas em segmentos do código previamente testados. Os testes de regressão verificam novamente esses segmentos, para checar se eles continuam funcionando de maneira apropriada após a alteração de outras partes da aplicação.

Testes de regressão são particularmente importantes durante a manutenção, pois nesta fase é mais freqüente que as alterações realizadas afetem outras porções do código. São usados também durante a integração, para confirmar a estabilidade dos módulos já integrados anteriormente.

Essa técnica inclui a definição de uma bateria de testes de regressão, ou seja, um conjunto selecionado de testes de boa cobertura, que é executado periodicamente. Essa bateria geralmente é baseada nos testes de aceitação. Caso essa bateria seja muito grande, pode-se utilizar uma bateria menor com testes selecionados, para uso mais freqüente, testando-se a bateria completa em intervalos maiores.

Os seguintes aspectos também devem ser verificados durante os testes de regressão:

- se a documentação do sistema está consistente com o comportamento atual do sistema (por exemplo, se os documentos relevantes foram atualizados após as alterações realizadas);
- se os dados e as condições de teste continuam válidos (por exemplo, mudanças em uma interface podem requerer alteração de procedimentos e casos de teste).

Implementação

1 *Princípios*

1.1 **Objetivos**

O fluxo de Implementação realiza um desenho em termos de diversos tipos de componentes de código fonte e código binário, conforme as tecnologias escolhidas. A implementação inclui as seguintes tarefas:

- planejamento detalhado da implementação de cada liberação;
- implementação das classes e outros elementos do modelo de desenho, em unidades de implementação, geralmente constituídas de arquivos de código fonte;
- nos caso de sistemas distribuídos, alocação das unidades aos nodos do sistema;
- verificação das unidades, por meio de revisões, inspeções e testes de unidade;
- compilação e ligação das unidades;
- integração das unidades entre si;
- integração das unidades com componentes reutilizados, adquiridos de terceiros ou reaproveitados de projetos anteriores;
- integração das unidades com os componentes resultantes das liberações anteriores.

A verificação da integração é feita por meio de testes de integração, descritos no fluxo de Testes. Consideramos também que durante a implementação é gerada a documentação de uso do produto. Essa documentação inclui manuais de usuários, ajuda on-line, sítios Web integrados ao produto, material de treinamento, demonstrações e outros recursos.

1.2 **Artefatos**

Durante a implementação, os desenvolvedores lidam com os seguintes artefatos:

- o modelo de desenho, detalhado em nível suficiente para resolver todas as questões de implementação;
- os componentes produzidos e reutilizados, que incluem arquivos executáveis, arquivos de componentes, bibliotecas, tabelas de bancos de dados e documentos;
- as estruturas provisórias de testes, como cotos e controladores;
- o plano de desenvolvimento, detalhado em nível suficiente para gestão das liberações.

Alguns autores chamam o modelo detalhado de desenho de modelo de implementação. No Praxis, trata-se de um único modelo. Este modelo é criado na iteração de Desenho Inicial, contendo já todas as decisões arquitetônicas importantes. Até chegar ao código executável, uma série de decisões de desenho detalhado ainda precisa ser tomada.

Os componentes podem ser implementados por meio de muitos tipos diferentes de arquivos. Nos diagramas de componentes, esses tipos de arquivos podem ser destacados por meio de estereótipos, representados por meio de identificadores ou de ícones próprios. A Figura 112 mostra alguns tipos de arquivos fontes, com os respectivos estereótipos icônicos. A Figura 113 mostra alguns exemplos de componentes de código objeto comuns em ambientes Windows, mostrados com estereótipos textuais.

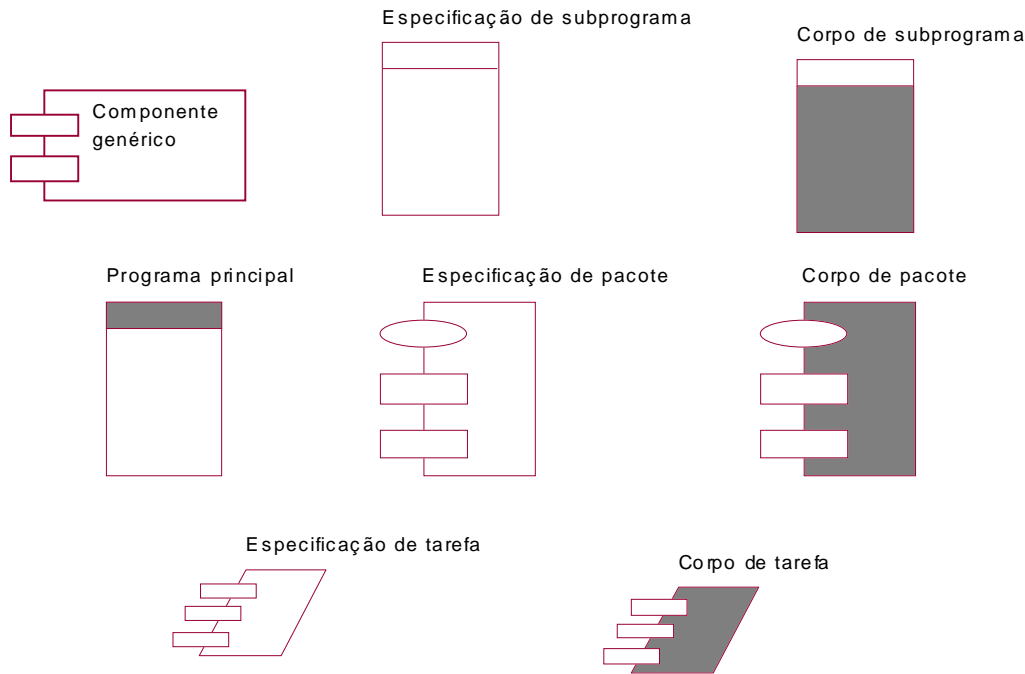


Figura 112 – Exemplo de componentes de código fonte

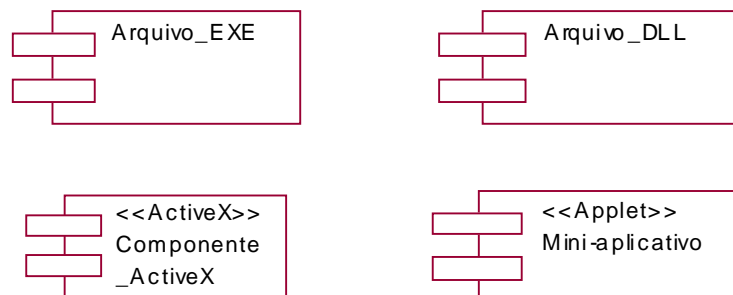


Figura 113 – Exemplo de componentes de código objeto

Na utilização de componentes, é importante considerar as **interfaces** que estes oferecem. As interfaces representam um subconjunto dos recursos implementados por um componente, que têm um tema comum. Por exemplo, na Figura 114, um componente ActiveX oferece duas interfaces distintas. Aplicativos que não tratem de vendas a prazo, por exemplo, possivelmente não utilizarão a interface *Gestao_de_Clientes*, utilizando apenas os recursos oferecidos através da interface *Operacao_de_Venda*.

O uso de interfaces facilita a evolução de um produto; por exemplo, seria possível, em uma primeira versão desse produto, utilizar uma versão mais barata do componente, que oferecesse apenas os serviços de *Operacao_de_Venda*. A interface representa um contrato entre clientes e fornecedor

desses serviços; ela garante que os serviços serão os mesmos, qualquer que seja a versão do fornecedor que os realiza.

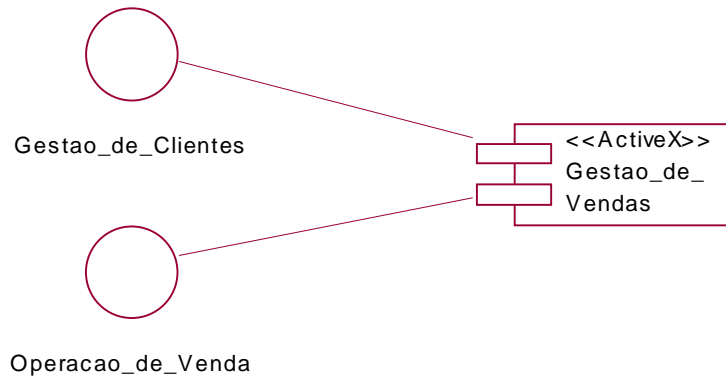


Figura 114 – Exemplo de interfaces de componente

2 Atividades

2.1 Visão geral

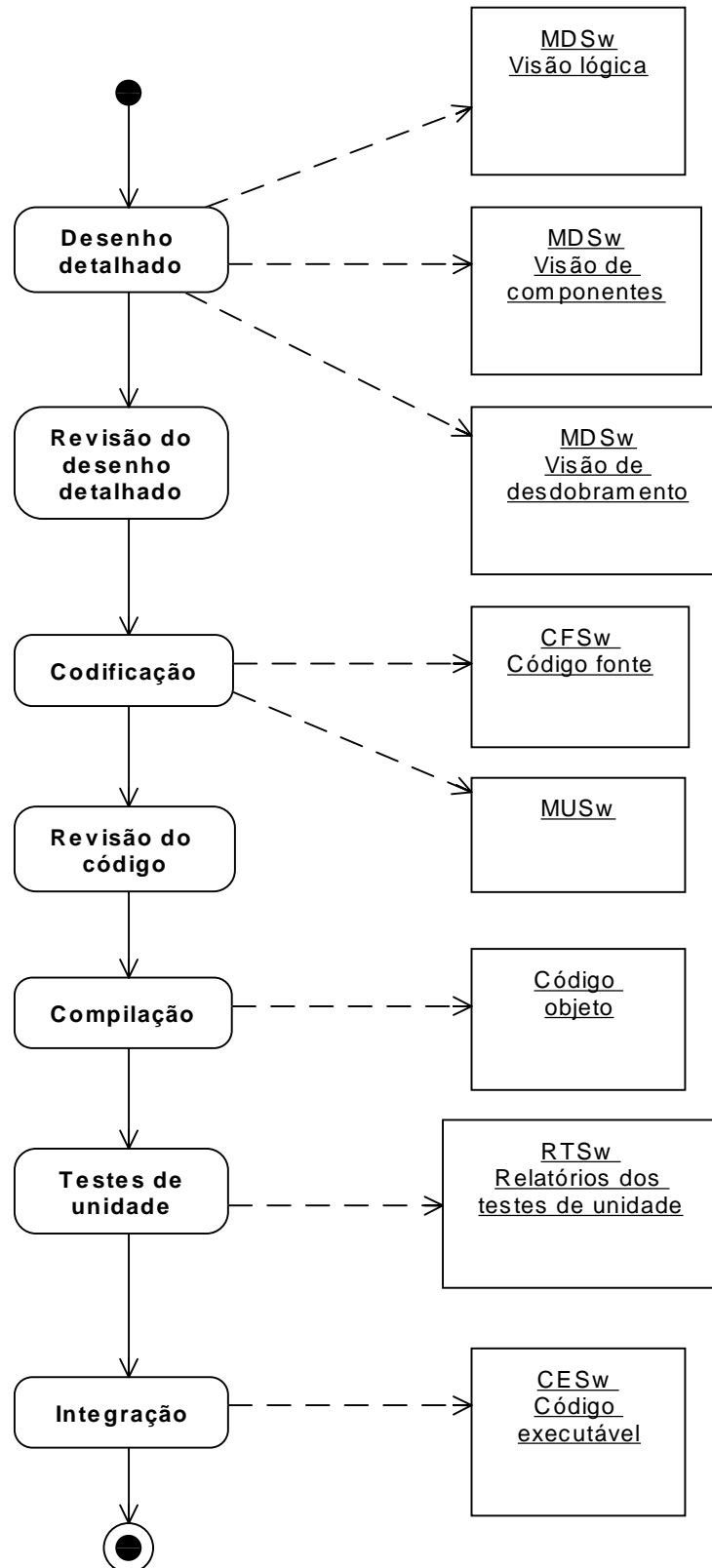


Figura 115 – Atividades e artefatos do fluxo de implementação

A implementação começa com o "**Desenho detalhado**", que preenche os detalhes restantes do Modelo de Desenho, no nível necessário para guiar a codificação. Essa atividade acrescenta detalhes às visões lógica e de desdobramento. A visão de componentes, que começou a ser esboçado no fluxo de Desenho, tem muitas vezes feita aqui sua maior parte. O modelo detalhado resultante é verificado por meio da "**Revisão do desenho detalhado**", na qual são conferidos os detalhes de interfaces, lógica e máquinas de estado das operações.

A tradução do desenho detalhado no código de uma ou mais linguagens de programação é feita na atividade de "**Codificação**". As unidades de código fonte produzidas são submetidas à "**Revisão do código**", para eliminar os defeitos que são introduzidos por erros de digitação ou de uso da linguagem.

As unidades de código fonte são transformadas em código objeto na "**Compilação**". Embora a maioria dos desenvolvedores provavelmente tenha o costume de compilar imediatamente após codificar, existem dados que mostram que a revisão do código é mais eficaz, em termos de remoção de defeitos, se feita antes da compilação [Humphrey95]. Essa é a ordem seguida no PSP. No TSP, os desenvolvedores fazem uma revisão pessoal antes da compilação e uma inspeção em grupo depois.

Os "**Testes de unidade**" vêm em seguida. Eles são feitos usando componentes de teste que devem também ter sido desenhados, revistos, codificados e compilados nas atividades anteriores. Embora bem menos formais que os testes de integração e aceitação, os resultados desses testes devem pelo menos ser registrados em relatórios simplificados.

A "**Integração**", finalmente, liga as unidades implementadas com os componentes construídos em liberações anteriores, reutilizados de projetos anteriores ou adquiridos comercialmente. O código executável resultante passa ao fluxo de Testes, para realização dos testes de integração.

2.2 Detalhes das atividades

2.2.1 *Desenho detalhado*

2.2.1.1 Tarefas

O desenho detalhado de uma rotina compreende as seguintes tarefas:

- checar os pré-requisitos, como as definições e especificações contidas no modelo de desenho;
- definir o problema a ser resolvido, inclusive informação a ser ocultada, entradas, saídas, globais e tratamento de erros;
- definir o desenho físico estático, atribuindo elementos do Modelo de Desenho a componentes (visão de componentes);
- no caso de sistemas distribuídos, completar o desenho físico dinâmico, definindo processos concorrentes e atribuindo unidades a nodos do sistema (visão de desdobramento);
- batizar as unidades de acordo com as regras para definição de nomes;
- decidir como testar as unidades, para que o desenho produzido seja testável;
- analisar os aspectos de eficiência, decidindo se a rotina é crítica do ponto de vista de especificações de desempenho, ou se esses aspectos podem ser tratados depois, em uma fase de otimização de código;
- pesquisar os algoritmos e as estruturas de dados que deverão ser usados;
- escrever um script em pseudolinguagem, possivelmente detalhando-o em refinamentos sucessivos;

- analisar e desenhar as estruturas de dados necessárias.

2.2.1.2 Desenho físico

2.2.1.2.1 Visão estática

A visão de componentes do modelo de desenho mostra os arquivos de código fonte e objeto utilizados na implementação, assim como componentes reutilizados do ambiente e de outros projetos. Mostra também as dependências existentes entre eles, que indicam o impacto das alterações realizadas em cada componente.

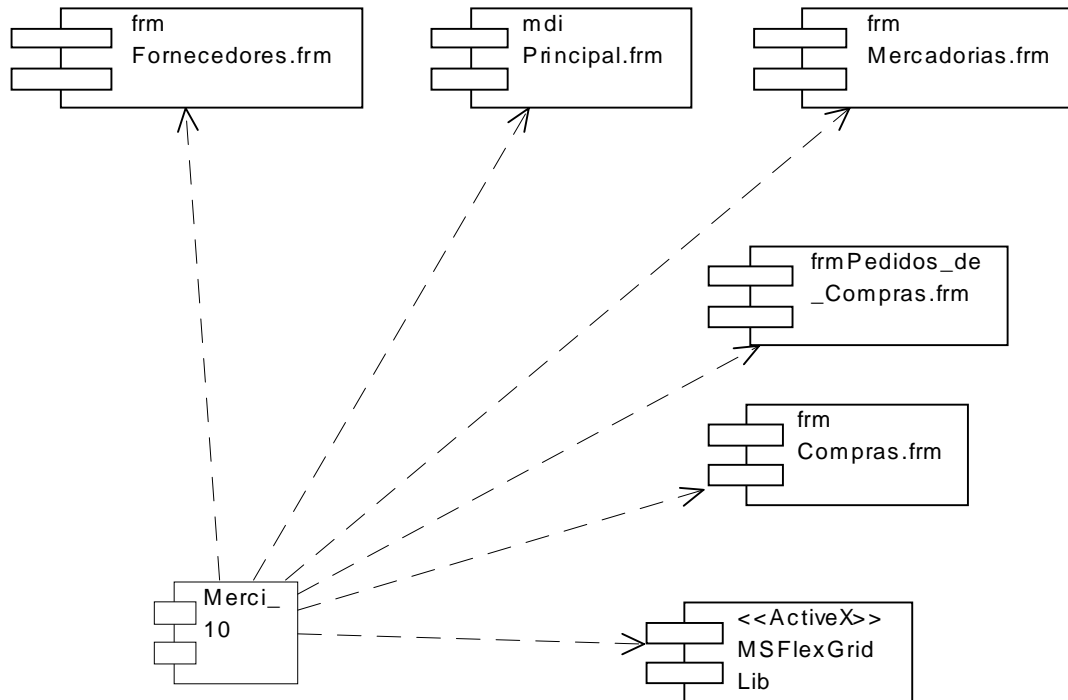


Figura 116 – Exemplo de diagrama de componentes (visão estática)

Os componentes podem ser de vários tipos dependentes de linguagem; os tipos aplicáveis a C++ são apresentados na Tabela 83.

Programa principal	Arquivo .cpp que contém o main ou equivalente.
Especificação de pacote	Arquivo .h que contém declarações e definições de classes.
Corpo de pacote	Arquivo .cpp que contém definições de operações das classes declaradas nas especificações de pacote correspondentes.
Especificação de subprograma	Arquivo .h que contém declarações de subprogramas livres.
Corpo de subprograma	Arquivo .cpp que contém definições de subprogramas livres.
Especificação de tarefa	Arquivo .h que contém declarações e definições de classes que implementam processos.
Corpo de tarefa	Arquivo .cpp que contém definições de operações das classes declaradas nas especificações de tarefa correspondentes.

Tabela 83 - Tipos de componentes em C++

2.2.1.2.2 *Modelo dinâmico*

O modelo dinâmico é descrito pelo diagrama de desdobramento, que mostra os processadores, dispositivos e conexões utilizados na implementação de um sistema. Em cada processador, pode-se mostrar os processos que nele executarão. Esse modelo é particularmente importante no caso de sistemas distribuídos. É comum a utilização de estereótipos para indicar diferentes tipos de processadores e dispositivos.

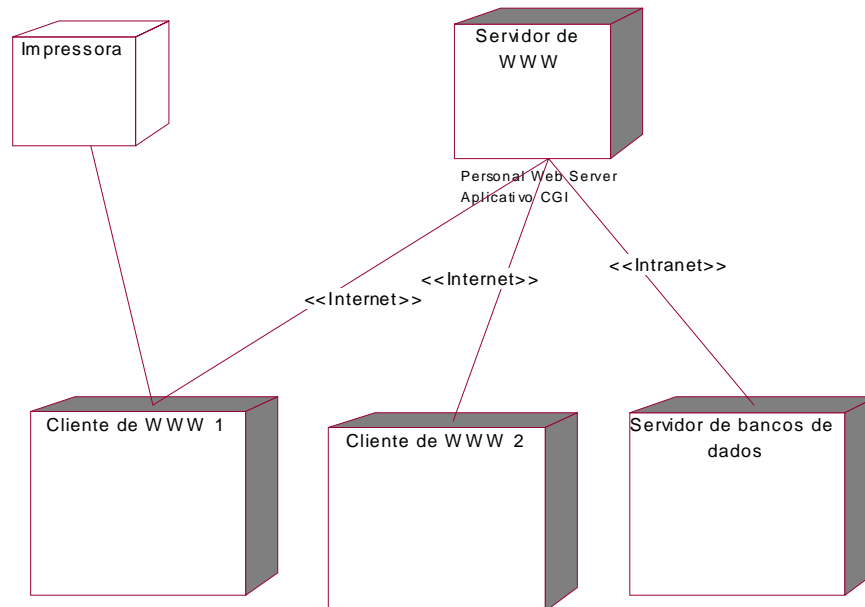


Figura 117 - Exemplo de diagrama de desdobramento

2.2.2 *Revisão do desenho detalhado*

O desenho detalhado deve ser cuidadosamente verificado. De preferência, deve-se fazer uma revisão informal, seguida de uma inspeção em grupo. Os pontos que devem ser verificados incluem os seguintes:

- correção da tradução dos mecanismos de desenho para os mecanismos da linguagem de programação, verificando-se, por exemplo, se relacionamentos, heranças e interfaces (no sentido da UML) foram implementados da maneira correta;
- correção das interfaces das unidades desenhadas, verificando-se se as assinaturas das operações estão corretas e documentadas corretamente;
- correção das máquinas de estado, verificando se estas não possuem estados inatingíveis nem armadilhas (ciclos de estados sem saída) e se os estados e transições são completos e ortogonais (dividem todas as situações possíveis em casos sem interseção);
- correção da lógica do pseudocódigo, verificando-se se as condições são logicamente corretas e otimizadas (por exemplo, analisando-se por meio de um mapa de Karnaugh), e se as malhas estão corretas, executando o número correto de vezes e mantendo os invariantes supostos em seu desenho.

O padrão para Desenho Detalhado e Codificação apresenta uma lista de conferência para o desenho detalhado.

2.2.3 Codificação

2.2.3.1 Tarefas

A codificação de uma rotina compreende as seguintes etapas:

- escrever a declaração das unidades, transformando o cabeçalho do pseudocódigo em um comentário de cabeçalho dentro das regras da linguagem;
- transformar o script em comentários de alto nível;
- preencher o código abaixo de cada comentário derivado do pseudocódigo.

2.2.3.2 Geração de código

Algumas ferramentas mais modernas de modelagem orientada a objetos oferecem o recurso de **geração de código**. Através desse recurso, o modelo de desenho detalhado é traduzido diretamente em um esqueleto do código executável. Esse recurso acelera a produção de código, e ajuda garantir a consistência entre código e modelo, principalmente no que diz respeito às interfaces. A tarefa de implementação com auxílio de geração automática de código é chamada de **engenharia direta** (*forward engineering*).

<<Class Module>> Fornecimento	
- mIClassDebugID : Long	
<<Get>> + ClassDebugID() : Long	
- Class_Initialize()	
- Class_Terminate()	
+ Mercadoria() : Mercadoria	

Figura 118 – Exemplo de classe em implementação

A Figura 118 mostra um exemplo de uma classe em implementação, sendo o correspondente código gerado em Visual Basic mostrado na Figura 119. A ferramenta gera, de modo controlável pelo desenvolvedor, muitos dos atributos e operações de apoio, que têm por objetivo tornar o código mais testável e mais seguro. Neste exemplo, foi inserido código para auxiliar na depuração, assim como operações para início e término do uso, e operações para acesso aos atributos. Foram também gerados os atributos necessários para implementação dos relacionamentos.

Marcadores especiais inseridos em comentários (no exemplo, “##”) facilitam a realização da **engenharia reversa** (*reverse engineering*). Graças a este recurso, o código gerado pode ser, por sua vez, refletido em atributos e operações do modelo, ajudando a manter a consistência.

```

VERSION 1.0 CLASS
BEGIN
    MultiUse = -1    'True
    Persistable = 0  'NotPersistable
    DataBindingBehavior = 0    'vbNone
    DataSourceBehavior  = 0    'vbNone
    MTSTransactionMode  = 0    'NotAnMTSObject
END
Attribute VB_Name = "Fornecimento"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Attribute VB_Ext_KEY = "RVB_UniqueID" , "380C6DAC0352"
'
Option Base 0
Option Explicit

'set this to 0 to disable debug code in this class
#Const DebugMode = 1
#If DebugMode Then
    'local variable to hold the serialized class ID that was created in
    'Class_Initialize
    '##ModelId=380CCE830168
    Private mlClassDebugID As Long
#End If

'##ModelId=380CCDA10047
Public aMercadoria As Mercadoria

'##ModelId=380CCDBF00AB
Public oFornecedor As Fornecedor

'##ModelId=380CCE830213
Private Sub Class_Terminate()
    #If DebugMode Then
        'the class is being destroyed
        Debug.Print "" & TypeName(Me) & " instance " & CStr(mlClassDebugID) & " is
terminating"
    #End If
End Sub

'##ModelId=380CCE830212
Private Sub Class_Initialize()
    #If DebugMode Then
        'get the next available class ID, and print out
        'that the class was created successfully
        mlClassDebugID = GetNextClassDebugID()
        Debug.Print "" & TypeName(Me) & " instance " & CStr(mlClassDebugID) & "
created"
    #End If
End Sub

#If DebugMode Then
    '##ModelId=380CCE830169
    Public Property Get ClassDebugID() As Long
        'if we are in debug mode, surface this property that consumers can query
        ClassDebugID = mlClassDebugID
    End Property
#End If

```

Figura 119 – Exemplo de código gerado automaticamente

A engenharia reversa é útil também para incorporar ao modelo elementos que são criados mais facilmente no ambiente de implementação, como as classes representativas das interfaces de usuário. A Figura 120 mostra um exemplo de classe que foi gerada a partir da interface mostrada na Figura 121. A engenharia reversa gerou as operações correspondentes aos cliques nos diversos botões.

<<Form>> frmFornecedores
- cmdFechar_Click() - cmdIncluir_Mercadoria_Click() - cmdNovo_Click() - cmdExcluir_Mercadoria_Click() - cmdPesquisar_Click() - cmdSalvar_Click() - cmdIncMercOK_Click() - txtNome_Change()

Figura 120 – Exemplo de classe criada por engenharia reversa

Figura 121 – Exemplo de interface submetida a engenharia reversa

Com esse tipo de ferramentas, a implementação é feita de forma conveniente através de muitas iterações entre o ambiente de modelagem e o ambiente de implementação. A automação parcial ajuda a manter a consistência, o que é particularmente difícil, dada a quantidade de detalhes que deve ser cuidada em uma implementação robusta e bem documentada. A combinação das engenharias direta e reversa forma a **engenharia iterativa** (*round-trip engineering*).

2.2.4 Revisão do código

O código deve ser cuidadosamente verificado. De preferência, deve-se fazer uma revisão informal, seguida de uma inspeção em grupo. Os pontos que devem ser verificados incluem:

- correção da digitação;

- correção sintática;
- correção lógica, que inclui a consistência com o desenho detalhado;
- obediência ao padrão de codificação;
- qualidade e consistência dos comentários.

O padrão para Desenho Detalhado e Codificação apresenta uma lista de conferência para o código.

2.2.5 *Compilação*

A compilação é uma tarefa quase completamente automatizada. Entretanto, o desenvolvedor deve verificar inicialmente se o compilador está configurado com as opções corretas e se os arquivos incluídos estão disponíveis nas versões corretas. A integração do compilador com uma ferramenta de gestão de configurações ajuda a eliminar esse tipo de problemas.

Após a compilação, todos os defeitos de compilação encontrados devem ser removidos, inclusive aqueles que são classificados apenas como advertências. Se os codificadores forem experientes e a revisão do código for bem feita, os erros de compilação serão muito raros. Por isso, a ocorrência de um número significativo de erros de compilação indica problemas mais sérios de codificação. Nesse caso, o código deve ser novamente inspecionado e possivelmente até refeito. Como mesmo os melhores compiladores deixam passar uma quantidade significativa de defeitos de codificação ([Humphrey95], [Humphrey99]), uma compilação com muitos erros provavelmente significa que existem também alguns defeitos não detectados pelo compilador.

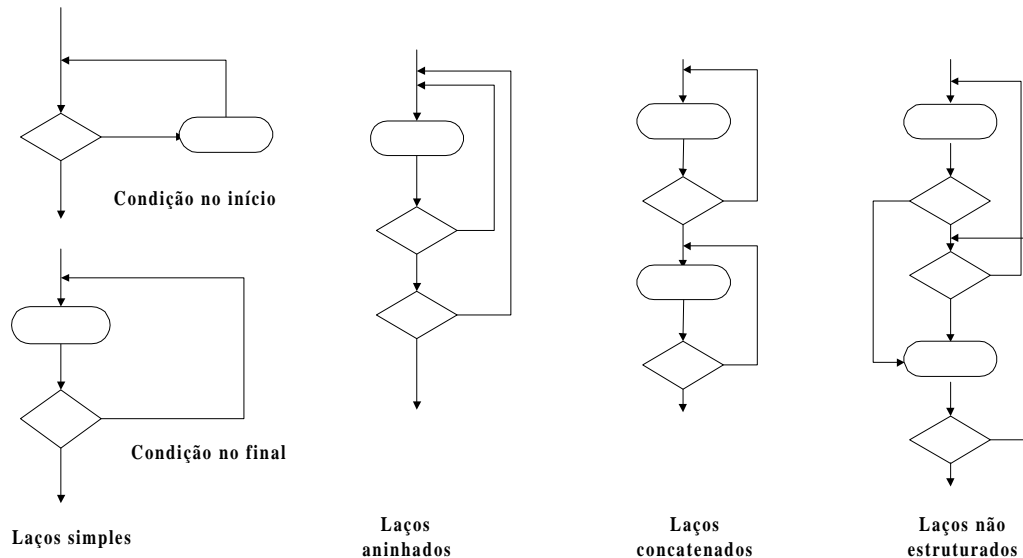
2.2.6 *Testes de unidade*

2.2.6.1 *Visão geral*

O teste de unidade é um tipo de teste de caixa branca; ele exercita detalhadamente uma unidade de código (por exemplo, uma classe ou grupo de classes correlatas). Um dos pontos fracos desse tipo de teste é que ele geralmente é desenhado pelos desenvolvedores, já que, na maioria das vezes, eles são os únicos que conhecem bem o desenho e o código das unidades a testar.

Entretanto, se forem bem planejados e executados, os testes de unidade poderão detectar cerca de 70% dos defeitos que seriam encontrados pelos usuários [McConnell96]. O Modelo do Desenho do Software e o próprio código da unidade sob teste devem ser as principais referências do desenho dos testes de unidade. Os testes de unidade devem exercitar os seguintes aspectos:

- As **interfaces** da unidade devem ser testadas para se assegurar que as informações (parâmetros de entrada e saída) fluam de forma adequada. Este deve ser o primeiro teste realizado.
- As **estruturas de dados** locais devem ser verificadas para se determinar se os dados armazenados temporariamente mantêm sua integridade durante todos os passos da execução do módulo sob teste.
- As **condições de limite** devem ser testadas para se assegurar que a unidade sob testes opera adequadamente nos limites estabelecidos, incluindo, entre outros aspectos, número de entradas e limites das entradas e saídas válidas.
- Os **caminhos independentes** devem ser verificados, de tal forma que todas as instruções da unidade sob teste sejam executadas pelo menos uma vez.
- Os **caminhos de tratamento de erros** devem ser verificados.



2.2.6.2 Tipos de teste de unidade

2.2.6.2.1 Teste de caminhos básicos

Os testes de caminhos básicos têm por objetivo verificar os **caminhos independentes** e os **caminhos de tratamento de erros**. É difícil selecionar um conjunto de caminhos que ofereça uma cobertura adequada dos possíveis defeitos. Entretanto, as diretrizes seguintes são geralmente úteis.

- Selecionar os caminhos que ligam o início ao fim da execução de cada procedimento ou método. Utilizar o Modelo de Desenho do Software e o Código Fonte do Software para encontrar esses caminhos.
- Selecionar caminhos que fazem pequenas variações, utilizando a lógica detalhada no Modelo do Desenho do Software.
- Selecionar caminhos sem significado funcional, para verificar se não existem combinações de entradas que possam provocar a ativação desses caminhos.

2.2.6.2.2 Teste de condições

O teste de condições tem por objetivo detectar erros nas condições contidas na unidade sob teste. Casos de teste são desenhados para que as expressões de condição assumam valores que exercitem o máximo de caminhos possíveis. Técnicas de lógica combinatória, como os mapas de Karnaugh, devem ser usadas para determinar todas as condições possíveis. Caso o número destas seja muito grande, é preciso verificar as combinações mais relevantes, para manter sob controle o número de casos de teste.

A referência mais importante para a definição dos testes de condição é o Modelo do Desenho do Software. Os diagramas de estado e a lógica do pseudocódigo servem de base para o desenho dos testes de condição, de forma semelhante à que foi empregada na revisão do desenho detalhado.

2.2.6.2.3 Teste de laços

O teste de laços verifica os laços da unidade sob teste. As técnicas para testar as categorias de laços apresentadas na Figura 122 estão descritas a seguir. Esse tipo de teste verifica também as condições de limite da unidade sob teste: um defeito comum nas malhas que percorrem um conjunto é errar, para cima ou para baixo, o número de elementos do conjunto.

Figura 122 - Categorias de laços

Para um laço simples cujo número máximo de iterações possíveis é n , casos de teste para as situações enumeradas a seguir devem ser desenhados:

- não executar o laço;
- executar apenas uma iteração do laço;
- executar duas iterações;
- executar m iterações pelo laço, onde $m < n$;
- executar $n-1$, n , $n+1$ iterações.

Estender a abordagem utilizada com laços simples a laços aninhados não é viável, pois o número de casos de teste possíveis pode ser proibitivo. Para reduzir o número de testes, deve-se iniciar o teste a partir do laço mais interno, aplicando-se a técnica de teste de laços simples. Nesse ponto, os outros laços devem realizar apenas uma iteração. Em seguida, partindo-se do laço mais interno para o mais externo, nessa ordem, deve-se testar cada um deles de tal forma que:

- os laços externos ao que está sendo testado realizem apenas uma iteração;
- os mais internos realizem um número de iterações provável de ocorrer na média.

Laços concatenados independentes entre si podem ser testados pelas mesmas técnicas que os laços simples. Entretanto, se, para dois laços concatenados, o contador do primeiro for usado como valor inicial do segundo, deve-se utilizar a técnica de teste de laços aninhados.

Os laços não estruturados não devem ser usados. Caso ocorra esse tipo de laço na unidade sob testes, ela deve ser redesenhada.

2.2.6.3 Componentes de teste

A execução dos testes só pode ser realizada ao se concluir a codificação da unidade sob teste. Entretanto, dependendo da ordem de integração, pode ser necessário realizar o teste de uma unidade sem que estejam codificadas unidades que a chamam ou que por ela são chamadas. Dessa forma, pode ser necessário desenhar e implementar componentes de teste, como os **controladores** e **cotos** (Figura 123).

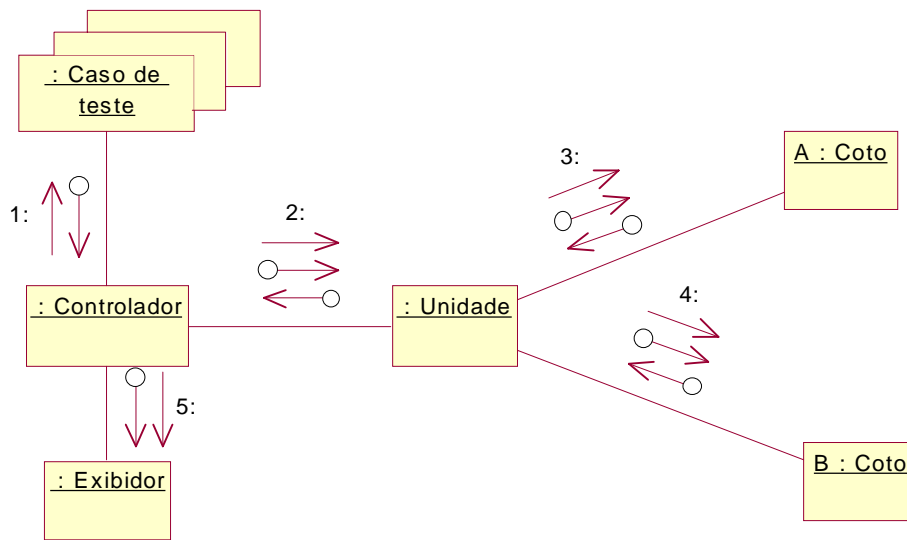


Figura 123 - Componentes de teste

Os controladores (*drivers*) realizam chamadas à unidade sob teste, possivelmente enviando dados e exibindo resultados. Os cotos (*stubs*) substituem as unidades subordinadas à unidade sob teste. O diagrama de colaboração da Figura 123 mostra uma seqüência típica de chamada (fluxo de controle, indicado pelas setas simples numeradas). O controlador recebe dados dos casos de teste e envia resultados para o exibidor. Entre o controlador, a unidade sob teste e os cotos os dados podem fluir em ambas as direções (fluxo de dados, indicado pelas setas iniciadas por círculos).

Em geral, o comportamento da unidade sob teste depende dos resultados retornados pelas unidades subordinadas, que, por sua vez, dependem dos valores recebidos da primeira. Um teste eficaz da unidade pode requerer o desenvolvimento de cotos sofisticados. Nesse caso, pode ser preferível ajustar a ordem de integração para fazer o teste já com as unidades subordinadas definitivas.

O desenvolvimento de componentes de teste representa um overhead para o projeto, já que eles não fazem parte do produto final. Os componentes de teste são simplificados quando as unidades apresentam menor acoplamento entre si e quando a ordem de integração foi bem escolhida. Entretanto, é comum que o código dos componentes de teste chegue à metade do volume do código útil. Por isso, componentes de teste devem ser mantidos e reutilizados sempre que possível.

3 Técnicas

3.1 Introdução

As técnicas de modularização, apresentadas a seguir, tratam da criação e organização de módulos e de rotinas. De maneira geral, essas técnicas dizem respeito às atividades de desenho detalhado, embora abranjam também aspectos de codificação. Grande parte dessas técnicas é independente de linguagens e mesmo de métodos de programação. Recomendações mais específicas fazem parte do padrão para Desenho Detalhado e Codificação de Software.

3.2 Modularização

3.2.1 Módulos

3.2.1.1 Níveis de modularização

Um módulo é uma unidade de armazenamento e manipulação de software. Módulos são coleções de rotinas, dados e tipos correlatos cuja interface com o restante de um sistema consiste em um conjunto bem definido de operações. Um módulo pode conter outros módulos, formando uma hierarquia (Tabela 84).

Nível	Definição
Subsistema	Partição de primeiro nível de um sistema
Pacote	Grupo de elementos correlatos de um modelo
Componente	Parte física substituível de um sistema, que encapsula a implementação de um conjunto de interfaces.

Tabela 84 – Níveis de modularização

Em linguagens orientadas a objetos, um componente pode encapsular uma classe, várias classes, ou ainda outros elementos de uma linguagem. Por exemplo, um utilitário é um módulo que agrupa variáveis globais e subprogramas livres, como uma biblioteca matemática. Se a linguagem usada para a implementação não suportar de forma direta um conceito que possa representar módulos, estes devem ser realizados por meio de convenções de codificação que permitam distinguir entre nomes importados, exportados e internos.

3.2.1.2 Organização física dos módulos

Cada módulo deve residir em somente um arquivo. Se um módulo tiver de ser partido em arquivos, por motivos de tamanho, deve constituir uma pasta separada. Os pacotes devem corresponder a diferentes pastas da estrutura física. É recomendável que haja correlação forte entre as estruturas lógica e física. Isso vale principalmente se a linguagem não suportar explicitamente a definição de módulos; nesse caso, a estrutura física pode ser usada para simular a estrutura lógica.

Por outro lado, pode ser conveniente colocar em um único arquivo um pequeno grupo de classes pouco complexas, coletivamente responsáveis pela execução de um mecanismo. O acesso a esse módulo deve ser feito apenas por meio de interfaces explicitadas no desenho do sistema e documentadas no desenho e no código. As revisões de código devem checar a obediência a essas interfaces.

3.2.1.3 Critérios de modularização

Módulos separados devem ser usados sempre que haja necessidade de isolar partes menos estáveis ou mais complexas de um produto. Isso aumenta a robustez do desenho: limitam-se os efeitos colaterais de mudanças e defeitos, tornando o produto mais extensível e de manutenção mais fácil. Devem ser isoladas em módulos, por exemplo, partes cuja implementação ou desenho apresentam dificuldades especiais, como uso de tecnologia nova ou necessidade de interfaces com plataformas complexas. As partes dependentes de aspectos específicos de uma plataforma devem ser isoladas, o que aumenta a portabilidade.

Em qualquer nível, um módulo deve exibir estas propriedades fundamentais:

- **alta coesão** - oferecer um grupo de serviços logicamente correlatos;
- **baixo acoplamento** - oferecer ao mundo exterior uma interface pequena e bem definida.

No desenho detalhado, é particularmente importante isolar as áreas propensas a mudanças, visando a reduzir os impactos de possíveis variações de requisitos ou de tecnologia. Tipicamente, essas áreas incluem:

- áreas com dependências de hardware;
- código responsável por operações de entrada e saída de dados;
- código que usa recursos de linguagem ou plataforma que estão fora do padrão;
- áreas de desenho particularmente difícil;
- áreas de implementação particularmente difícil;
- áreas que manipulam variáveis de status frequentemente atualizadas, como variáveis que indicam o estado de atualização de uma estrutura de dados;
- áreas dependentes de tamanho de dados, como declarações de constantes que definem tamanhos de estruturas de dados;
- áreas dependentes de regras de negócio, como lógica dependente de requisitos legais ou de políticas da organização.

3.2.1.4 Classes

Além de levar em conta os critérios anteriores, o desenho detalhado de classes de boa qualidade deve responder às seguintes questões:

- Como os objetos devem ser construídos?
- Como os objetos devem ser destruídos?
- Como funcionará a atribuição dos objetos?
- O que significa passar os objetos por valor?
- Quais os valores legais para os objetos?
- Que tipos de verificação de erros devem ser feitos nas funções?
- Que operações devem ser abstratas?
- Que conversões de tipo serão permitidas?
- Que operadores fazem sentido para os objetos?
- Que operadores e funções devem ser explicitamente proibidos?
- Quem tem que tipo de acesso aos membros?

3.2.2 Rotinas

3.2.2.1 Visão geral

A interface de um módulo deve ser feita apenas por meio de operações. Dados devem ser privados em relação ao módulo. Em uma classe, uma operação é implementada por meio de um ou mais métodos.

Usaremos o conceito de rotinas para abranger tanto métodos quanto subprogramas livres. Dependendo da linguagem, o nome "rotina" deve ser interpretado como sub-rotina, função, procedimento, método ou membro de função, podendo retornar ou não um valor.

3.2.2.2 Critérios para criação de rotinas

A criação de novas rotinas durante o desenho detalhado deve obedecer a critérios bem definidos. O critério mais importante é sempre a redução da complexidade dos programas, substituindo-se uma série de detalhes por uma abstração adequada. Alguns aspectos específicos de aplicação desse princípio incluem:

- isolar operações e expressões complexas, que são mais propensas a erros;
- aumentar a legibilidade do código, diminuindo o número de linhas de cada rotina;
- ocultar a ordem de execução de seqüências, colocando em rotinas separadas código cuja ordem de execução é independente;
- ocultar detalhes de implementação de estruturas de dados;
- ocultar acessos a dados globais;
- ocultar operações sobre apontadores;
- ocultar código não portátil;
- evitar duplicação de código, agrupando em uma única rotina seqüências similares de código;
- melhorar o desempenho, agrupando código otimizável;
- agrupar código que pode diferir entre os membros de uma família de programas;
- centralizar pontos de controle, como seqüências de acesso a arquivos ou dispositivos de hardware;
- promover a reutilização de código.

3.2.2.3 Formas de coesão

Os níveis de coesão de uma rotina são mostrados na Tabela 85, do nível mais alto para o mais baixo. Todas as rotinas devem exibir coesão pelo menos comunicacional. A coesão temporal pode ser usada ocasionalmente, e os níveis inferiores de coesão devem ser evitados.

Nível de coesão	Definição
Funcional	Rotina executa uma única tarefa (simples).
Seqüencial	Rotina executa uma seqüência de tarefas correlatas e encadeadas.
Comunicacional	Rotina executa seqüência de tarefas que usa um conjunto comum de dados.
Temporal	Rotina executa um conjunto de tarefas coincidentes no tempo (por exemplo, iniciação).
Procedimental	Rotina executa uma seqüência de tarefas ordenadas mas não correlatas nem encadeadas.
Lógico	Rotina executa uma dentre várias tarefas, de acordo com o valor de uma variável de estado (if ... else if ... else, switch).
Coincidente	Rotina executa tarefas não correlatas.

Tabela 85 – Níveis de coesão de rotinas

3.2.2.4 Formas de acoplamento

A acoplamento se refere à maneira de comunicação entre as rotinas. O acoplamento é frouxo se existe um bom compromisso entre as seguintes características (Tabela 86): baixa cardinalidade, baixa intimidade, alta visibilidade e alta flexibilidade. Compromissos de engenharia são necessários porque há contradições entre as características. Por exemplo, agrupar em estruturas os dados comunicados tende a melhorar a cardinalidade e a piorar a visibilidade e a flexibilidade.

O grau de acoplamento pode ser classificado na ordem mostrada na Tabela 87, do mais frouxo para o menos frouxo. O acoplamento deve ser o mais frouxo possível. O acoplamento de estrutura de dados é tanto mais aceitável quanto maior for a porção da estrutura que realmente é usada por ambas as rotinas. O acoplamento de dados globais é tolerável se o acesso for feito apenas para leitura.

Característica de comunicação	Definição
Cardinalidade	Número de objetos comunicados entre duas rotinas (por exemplo, número de parâmetros).
Intimidade	Grau de controle de acesso (por exemplo, as seguintes formas de comunicação apresentam graus crescentes de intimidade: listas de parâmetros, atributos de classe privados, atributos de classe públicos, variáveis globais e arquivos ou bancos de dados).
Visibilidade	Grau de visibilidade dos dados passados (por exemplo, dados passados como parâmetros separados são mais visíveis do que quando agrupados em uma estrutura).
Flexibilidade	Grau de liberdade que a rotina chamadora tem na montagem dos dados para a rotina chamada (por exemplo, a necessidade de montar uma estrutura complexa acarreta baixa flexibilidade).

Tabela 86 – Características de comunicação entre rotinas.

Nível de acoplamento	Definição
Acoplamento de dados simples	Todos os dados são passados como parâmetros não estruturados.
Acoplamento de estrutura de dados	Usam parâmetros que são estruturas de dados.
Acoplamento de controle	Uma rotina diz à outra o que fazer, passando por exemplo uma variável de estado.
Acoplamento de dados globais	As rotinas fazem uso dos mesmos dados globais.
Acoplamento patológico	Uma rotina usa código interno ou altera dados locais de outra.

Tabela 87 – Níveis de acoplamento entre rotinas

Rotinas não devem ser divididas artificialmente, apenas para limitar seu tamanho. Por outro lado, deve-se procurar transformar subsequências genéricas de código em rotinas separadas. Se uma subsequência for comum a métodos de uma classe, deve-se procurar transformá-la em operação privada dessa classe. Rotinas acima de 150-200 linhas de código devem exibir coesão funcional ou sequencial.

3.2.2.5 Nomes das rotinas

Deve-se usar nomes que descrevam de maneira eficaz o que a rotina faz. Normalmente, os nomes de rotinas devem ser verbos com significados fortes. Em linguagens não baseadas em objetos, o nome deve incluir também o objeto da rotina; em linguagens orientadas a objetos, a referência ao objeto será feita pelo nome da classe (por exemplo, `Imprimir_Relatorio` ou `Relatorio.Imprimir`). Rotinas cujo foco é o valor retornado podem usar uma descrição do valor de retorno (por exemplo, `Impressora_Pronta` ou `Impressora.Pronta`).

O nome deve ser tão longo quanto necessário para descrever completamente o que a rotina faz. Na maioria dos casos, ficará tipicamente entre 15 a 20 caracteres. Operações semelhantes devem corresponder a nomes semelhantes (por exemplo, `Obter`, `Consultar` etc.).

3.2.2.6 Parâmetros das rotinas

Devem-se usar listas de parâmetros que promovam o entendimento eficaz do uso da rotina. Isso significa obedecer às seguintes diretrizes.

- Casar os tipos dos parâmetros formais e reais, mesmo que a linguagem não o exija.
- Colocar os parâmetros na seguinte ordem: parâmetros de entrada, parâmetros modificados e parâmetros de saída.
- Colocar parâmetros de status ou códigos de erro em último lugar. Por outro lado, devem ser consideradas práticas usuais da linguagem: por exemplo, existe, em C, a prática de usar o valor de retorno para esse fim.
- Usar a mesma ordem para parâmetros similares em rotinas similares.
- Só passar parâmetros que venham realmente a ser usados. Uma exceção é o caso em que funções são passadas como parâmetros. Nesse caso, pode ser necessário passar parâmetros artificiais, para cobrir casos em que as funções-parâmetro possam ter listas de parâmetros de diferentes tamanhos.
- Não usar os parâmetros como variáveis de trabalho.
- Documentar suposições de interface a respeito dos parâmetros.
- Limitar o número de parâmetros a cerca de sete. Essa é uma aplicação da regra de Miller: na memória humana de curto prazo só cabem sete itens, mais ou menos dois.
- Passar apenas as partes de uma estrutura que são usadas pela rotina chamada. Por outro lado, pode ser adequado passar toda a estrutura quando esta representar um tipo abstrato de dados.
- Não utilizar suposições sobre o mecanismo de passagem de parâmetros utilizado pelo compilador, o que torna o código mais portátil e estável.

3.2.2.7 Pseudolinguagem

Ao lado das notações de máquinas de estado, os scripts em pseudolinguagem são uma importante notação de desenho detalhado. Ambos são, em geral, preferíveis aos fluxogramas. Os diagramas de atividade podem trazer alguma vantagem no caso de existir paralelismo interno à rotina.

As seguintes diretrizes devem ser usadas para a escrita de scripts em pseudolinguagem:

- usar frases da linguagem natural que descrevam com precisão operações específicas;
- evitar detalhes de sintaxe da linguagem alvo, pois isso abaixa o nível de abstração e torna a pseudolinguagem desnecessariamente restrita;
- descrever a intenção do desenho, e não o mecanismo de implementação na linguagem alvo;
- descer em nível de detalhe suficiente para tornar quase automática a tradução para a linguagem alvo.

Os scripts em pseudolinguagem trazem os seguintes benefícios:

- facilitar revisões do desenho detalhado e reduzir o esforço adicional de inspeção de código;
- dar suporte ao refinamento iterativo, permitindo aumentar o detalhe do desenho em passadas consecutivas;
- facilitar as mudanças do desenho detalhado, antes que esse tenha sido comprometido em código;
- diminuir o esforço de comentar, já que a maior parte de um bom pseudocódigo pode ser aproveitado como comentários.

3.2.2.8 Programação defensiva

O desenho detalhado de um módulo deve incluir diversos aspectos de prevenção de defeitos. O módulo deve ser desenhado e codificado considerando sempre a possibilidade de receber dados errados, por meio de suas interfaces. A prevenção de defeitos é baseada no conceito de programação defensiva.

Um exemplo de programação defensiva é o uso de **asserções**. Estas são funções que testam, em determinados pontos de código, a validade de determinadas hipóteses. Por exemplo, o desenho de uma rotina pode conter as precondições que devem ser válidas ao entrar-se e as pós-condições que devem ser válidas imediatamente após a saída da rotina. Uma malha de repetição pode ter uma condição invariante. Precondições, pós-condições e invariantes podem ser testadas por meio de asserções.

A programação defensiva inclui também as seguintes medidas:

- checar os valores de todos os dados de fontes externas;
- checar os valores de todos os parâmetros de entrada;
- tratar de forma consistente os erros em parâmetros;
- usar mecanismos de tratamento de exceções.

Normalmente, os mecanismos de depuração (asserções, mensagens de depuração etc.) devem ser retirados em versões de operação e podem ser novamente introduzidos em situações de manutenção. Deve-se desativar código que trata de erros menores ou provoca a parada do sistema, e deixar ativo código que trata de erros importantes ou permite a degradação gradual do sistema. Nesses casos, as mensagens devem indicar que se trata de um erro interno, e recomendar o contato com o suporte técnico.

Existem as seguintes maneiras de administrar a retirada e a inserção desses mecanismos, em ordem decrescente de conveniência:

- controle por um sistema de gestão de configurações;
- uso de pré-processadores e compilação condicional;
- uso de cotos (*stubs*) de depuração.

3.3 Documentação de usuário

3.3.1 Visão geral

Técnicas relativas à documentação de usuário são também tratadas aqui. A documentação de usuário, seja na forma de manuais ou de ajuda on-line, é muitas vezes tão necessária para o sucesso de um produto quanto o código executável. Em alguns tipos de produto, como sítios WWW, programas de

treinamento baseado em computador e outros produtos de multimídia, sequer existe uma fronteira nítida entre programa e documentação.

Todas as atividades do fluxo de implementação são aplicáveis à produção de documentação de usuário. Ela tem de ser desenhada, revisada e codificada. No caso de ajuda on-line e de sítios WWW, é evidente a necessidade de ambientes de desenvolvimento, mas mesmo os modernos processadores de textos são ferramentas sofisticadas de programação¹⁸.

A próxima subseção trata do desenho e da confecção dos manuais de usuário, impressos ou on-line. A subseção seguinte trata de questões específicas dos documentos on-line.

3.3.2 *Manuais de usuário*

3.3.2.1 **Introdução**

As recomendações desta seção se aplicam o documentos destinados a ajudar o usuário a instalar, configurar, operar e gerir produtos de software. Elas não se aplicam a documentos destinados à manutenção de software, à instalação e configuração de hardware, e nem aos seguintes outros tipos de documentação de usuário:

- material de treinamento;
- material de marketing;
- material de ajuda on-line.

3.3.2.2 **Desenho dos documentos de usuário**

Recomenda-se o seguinte procedimento para desenhar os documentos de usuário que deverão ser produzidos:

- Identificar o produto de software, suas interfaces de usuário e as tarefas onde será aplicado.
- Determinar o público dos documentos, determinando, para cada grupo de usuários, o respectivo perfil, tal como descrito na seção 3.2.3 Características dos usuários da Especificação dos Requisitos do Software. Os usuários podem ser divididos em públicos diferentes, e estilo e nível de detalhe da documentação devem ser escolhidos em função do respectivo público.
- Determinar o conjunto de documentos. A documentação pode ser dividida em múltiplos conjuntos, conforme o público, e cada conjunto pode conter um ou mais volumes, dependendo da quantidade de material. Quando um produto tiver mais de um público, devem-se usar conjuntos diferentes de documentos, ou pelo menos, caso se use um único conjunto, indicar claramente quais as partes de interesse de cada público.
- Determinar os modos de uso dos documentos. Vide descrição dos modos de uso na próxima subseção.
- Determinar a estrutura dos documentos, seguindo as recomendações aplicáveis contidas no padrão para Documentação para Usuários de Software.

3.3.2.3 **Modos de uso dos documentos de usuário**

3.3.2.3.1 ***Modo instrucional***

Os documentos de modo **instrucional** destinam-se a ajudar o usuário a **aprender** o que lhe for necessário a respeito de um produto. Este tipo de documento pode ser **orientado para informação**,

¹⁸ Vide vírus destas ferramentas.

quando tem por objetivo oferecer ao leitor informação necessária para entendimento do Produto e de suas funções, ou **orientado para tarefa**, quando tem por objetivo mostrar ao leitor como realizar uma tarefa ou atingir um objetivo.

A informação típica de documentos orientados para informação inclui:

- visão geral;
- teoria de operação;
- material tutorial.

A informação típica de documentos orientados para tarefa inclui:

- procedimentos de instalação;
- procedimentos de operação;
- procedimentos de diagnóstico.

3.3.2.3.2 Modo referencial

Os documentos de modo **referencial** destinam-se a armazenar e organizar informação necessária sobre um produto, facilitando a pesquisa por palavras-chaves. São exemplos de documentos referenciais típicos:

- manual de comandos;
- manual de mensagens de erro;
- manual de interface de programação;
- cartela de referência rápida.

3.3.2.3.3 Modos mistos

Um manual pode combinar os modos instrucional e referencial e as orientações para informação ou para tarefa, desde que cada um deles esteja em uma Seção separada do corpo do documento, com um título que identifique claramente o tipo de material.

3.3.2.4 Controle da qualidade

A documentação de usuário deve ser submetida aos seguintes controles:

- revisão da consistência com os documentos de desenvolvimento, principalmente com a Especificação de Requisitos de Software;
- revisão da clareza, completeza e conformidade com essas recomendações e outras normas aplicáveis;
- controle por meio dos sistemas de gestão de configurações usados para desenvolvimento e manutenção;
- revisão da consistência com o resultado das atividades de manutenção.

Sempre que possível, devem-se usar revisões técnicas para garantir alto nível da qualidade.

3.3.3 *Documentação on-line*

3.3.3.1 **Manuais on-line**

Os manuais on-line devem, de preferência, ser apresentados em um formato imagem de impressão, como o formato .PDF do Adobe Acrobat. Nesse caso, eles podem ser distribuídos em um título em CD-ROM, ou disponibilizados em um sítio Web de suporte. Deve-se fornecer também o aplicativo para leitura, ou indicar um sítio de onde esse possa ser baixado.

3.3.3.2 **Títulos e sítios de suporte**

Os títulos em CD-ROM e sítios Web de suporte são muito úteis para os seguintes propósitos:

- resumir informação importante sobre o produto;
- fornecer cópias dos manuais on-line;
- fornecer um ponto de contato para suporte e manutenção;
- fornecer informação de atualização do produto ou de sua documentação;
- distribuir correções urgentes para o produto;
- manter listas de perguntas freqüentes;
- distribuir demonstrações e outros materiais promocionais;
- fornecer ajuda on-line, quando o ambiente de desenvolvimento não dispuser de ferramentas para gerá-la.

É importante notar que a estrutura de hipermídia adotada em um título ou sítio é muito diferente da estrutura dos manuais impressos ou imprimíveis. Os seguintes pontos devem ser observados.

- A estrutura de hipermídia consiste em uma rede de nós contendo informações, conectados por hiperligações (*hyperlinks*), que ligam nós por meio de uma palavra chave. O tamanho dos tópicos não deverá exceder uma página e essa página deve conter hiperligações agrupadas por temas, para evitar que o usuário fique consultando diversas páginas sobre um mesmo tópico.
- Outro aspecto importante diz respeito à facilidade de localização da informação desejada. O título ou sítio deve permitir fácil navegação, o que pode ser obtido utilizando-se cadeias reduzidas de hiperligações para alcançar a informação, e um número reduzido de hiperligações por página.
- Devem-se levar em conta os navegadores (*browsers*) que poderão ser usados, informando o leitor sobre quais navegadores e quais versões permitem melhor leitura. Deve-se evitar usar recursos não padronizados e específicos de um navegador, embora não valha a pena preocupar-se com navegadores mais antigos ou raros.
- Imagens e recursos de multimídia devem ser usados com cuidado, considerando-se aspectos estéticos, de clareza visual e principalmente da capacidade das linhas utilizadas pelos leitores. Por exemplo, uma intranet normalmente permite velocidades de transmissão muito maiores que as ligações por modem. A distribuição em CD-ROM deve ser preferida no caso de material que contenha grandes imagens, som e animação. No caso de uso da Web, deve-se informar ao usuário qual a resolução gráfica mínima recomendada, quais suplementos do navegador são necessários e como obtê-los.

- Considerações de segurança e confidencialidade são particularmente importantes no caso de acesso via Internet. Os leitores devem ser devidamente informados das restrições aplicáveis. Os mecanismos de distribuição e atualização de senhas devem ser seguros, mas não devem prejudicar a produtividade dos usuários.

O assunto de desenho de títulos e sítios é extenso e ainda está em início de evolução. Recomenda-se consultar a literatura existente e recorrer a profissionais especializados.

3.3.3.3 Ajuda on-line

O desenho e a implementação de ajuda on-line devem ser feitos com ferramentas especializadas, embora seja tolerável utilizar-se hipertexto genérico quando o ambiente de desenvolvimento não dispuser de ferramenta de ajuda on-line. Recomenda-se adotar uma ferramenta que disponha de um guia de desenho de ajuda on-line, e seguir as recomendações desse guia.

3.3.3.4 Outros materiais

Outros materiais importantes para o suporte a um produto incluem materiais de treinamento e de marketing. Recomenda-se consultar a literatura existente e recorrer a profissionais especializados.

Padrões técnicos

Página em branco

Proposta de Especificação de Software

1 Introdução

Este padrão descreve uma estrutura de Proposta de Especificação de Software (PESw). Esse documento é tipicamente produzido através da execução das primeiras atividades do fluxo de Requisitos, dentro da iteração de Ativação, na fase de Concepção.

O objetivo principal dessa proposta é delimitar e dimensionar o esforço da fase de Elaboração. Isso é particularmente importante para justificar, perante o cliente, o preço dessa fase. Além disso, procura-se deixar claro o esforço que o próprio pessoal do cliente terá de fazer, participando das atividades de Requisitos e talvez de Análise, durante a Elaboração.

Esse padrão é apenas indicativo. Dependendo do tipo de relacionamento com o cliente, do produto e do projeto, pode ser necessário apresentar uma proposta muito mais detalhada. Nesse caso, recomenda-se anexar as seções pertinentes das partes iniciais da Especificação dos Requisitos do Software e do Plano de Desenvolvimento do Software.

2 Preenchimento da Proposta de Especificação de Software

2.1 Missão do produto (PESw-1)

Descrever os objetivos do produto que deverá ser desenvolvido no projeto. De preferência, usar um único parágrafo que sintetize a missão a ser desempenhada pelo produto: ou seja, que valor o produto acrescenta para o cliente e os usuários.

O Produto Merci 1.0 visa a oferecer apoio informatizado ao controle de vendas, de estoque, de compra e de fornecedores da mercearia Pereira & Pereira.

Tabela 88 - Exemplo de Missão do produto

A declaração de missão deve cumprir os seguintes objetivos:

- delimitar as responsabilidades do produto;
- delimitar o escopo do produto;
- sintetizar o comprometimento entre cliente e fornecedor.

2.2 Lista de funções (PESw-2)

Listar as funções básicas do produto. Descrever as necessidades que se pretende atender e os benefícios esperados, se possível desdobrados por função. Cada função deve sintetizar uma interação completa entre o usuário e o produto; portanto, funções parciais não devem ser listadas.

Salientar a prioridade relativa das funções, se possível classificando-as em essenciais, desejáveis e opcionais. No caso de nova versão de produto existente, listar tanto as funções existentes, modificadas ou não, quanto as funções que se pretende acrescentar. Ressaltar o que muda em relação à versão anterior.

Número de ordem	Nome da função	Necessidades	Benefícios
1	Cadastramento de mercadorias	Fornecimento de informações a outras funções. Identificação das mercadorias.	Agilidade na compra e venda de mercadorias. Melhoria do conhecimento dos produtos comercializados.
2	Controle da operação de venda	Registro de produtos e dos valores vendidos. Viabilização do controle de estoque. Emissão de tickets de caixa para o cliente.	Economia de mão-de-obra. Diminuição do tempo de venda. Diminuição de erros. Diminuição dos prejuízos.
3	Controle de estoque	Reposição das mercadorias. Controle efetivo de mercadorias em estoque.	Identificação de produtos mais e menos vendidos. Indicação de promoções. Diminuição de perdas. Otimização do estoque de cada produto.
4	Emissão de pedidos	Registro do pedido. Acompanhamento da recepção das mercadorias. Controle de cancelamento de pedidos.	Eliminação da duplicidade de pedidos. Informação ao setor de compras das mercadorias não entregues. Diminuição dos atrasos nas entregas.
5	Emissão de notas fiscais	Cumprimento de obrigação legal. Documentação legal da venda.	Qualidade na emissão da nota, em relação à emissão manual. Garantia para o cliente e a mercearia. Diminuição dos erros nas notas fiscais. Economia de mão-de-obra.
6	Controle de compras	Melhoria na análise das condições de compra. Acompanhamento do prazo de fornecimento de mercadorias. Controle do vencimento das faturas. Emissão do pedido de compras.	Apoio na avaliação das melhores condições de preço e menores prazos de entrega. Maior agilidade nas decisões de compra. Compra de mercadorias com melhor qualidade e menores preços. Diminuição do custo de estocagem.
7	Cadastramento de fornecedores	Atualização dos dados de cadastro. Atualização da lista de produtos comercializados por cada fornecedor.	Controle dos dados de fornecedores (ativos e potenciais). Conhecimento do mercado de fornecedores. Avaliação da qualidade de fornecimento. Conjugação da qualidade com o menor preço de fornecimento, prazo de entrega e aceitação do produto.

Tabela 89 - Exemplo de Lista de funções

2.3 Requisitos de qualidade (PESw-3)

Descrever os aspectos mais importantes das características não funcionais ou requisitos de qualidade do produto a ser entregue. Exemplos de requisitos de qualidade são requisitos de desempenho, ambientes de operação desejados etc.

Só devem ser incluídas características específicas, significativas e mensuráveis do produto proposto, que sejam imprescindíveis para sua aceitação. Evitar a menção a características genéricas de qualidade que qualquer produto de software deva ter (por exemplo, será fácil de usar, de manutenção barata, confiável etc.).

O produto deverá atender aos seguintes requisitos de qualidade:

- a utilização será feita através de interface gráfica;
- a operação de venda deverá gastar no máximo o tempo a ser definido na especificação de requisitos;
- deverá ser possível a expansão dos pontos de venda.

Tabela 90 - Exemplo de Requisitos de qualidade

2.4 Metas gerenciais (PESw-4)

Descrever as metas e limitações de ordem gerencial que se deseja atingir, tais como:

- prazos máximos;
- custos máximos;
- restrições legais;
- padrões que devam ser adotados.

O produto deverá atender às seguintes metas gerenciais do cliente:

- prazo máximo de desenvolvimento: 12 meses;
- custo máximo de desenvolvimento: R\$60.000,00.

Tabela 91 - Exemplo de Metas gerenciais

2.5 Outros aspectos (PESw-5)

Incluir outras informações de valor estratégico, tais como;

- limitações de escopo do produto;
- possíveis interfaces com outros produtos;
- questões pendentes que devam ser esclarecidas durante a especificação dos requisitos.

Será utilizado o mesmo sistema financeiro adotado em outras atividades do cliente.

Tabela 92 - Exemplo de Outros aspectos

2.6 Estimativa de custos e prazos para a especificação (PESw-6)

Estimar custos e prazos para a especificação do produto, indicando-se, com a melhor precisão possível, as tarefas que fazem parte da atividade de especificação, os recursos necessários e os custos envolvidos. Indicar clara e detalhadamente como será a participação do cliente no processo de especificação, prevendo-se as reuniões necessárias.

A especificação do produto obedecerá ao seguinte cronograma:

1. Reunião para levantamento inicial dos requisitos do Merci 1.0: 4 horas.
2. Análise e documentação inicial pela equipe da United Hackers: 1 dia útil.
3. Reunião para detalhamento dos requisitos: 1 dia útil.
4. Fechamento da análise e documentação da Especificação de Requisitos pela equipe da United Hackers: 3 dias úteis.
5. Elaboração dos Planos de Desenvolvimento e da Qualidade pela equipe da United Hackers: 1 dia útil.
6. Reunião para apresentação da Especificação de Requisitos e dos Planos de Desenvolvimento e da Qualidade: 2 horas.

A Pereira e Pereira Comercial Ltda. deverá indicar, para participação nas atividades 1, 3 e 6, um representante com poder de decisão e representantes de cada grupo de futuros usuários do Produto.

O preço e o prazo de entrega do produto serão determinados na atividade 6.

Tabela 93 - Exemplo de Estimativa de custos e prazos para a especificação

Especificação de Requisitos de Software

1 Visão geral

1.1 Introdução

A Especificação dos Requisitos do Software (ERSw) resulta do fluxo de Requisitos, parte do processo Praxis. Este fluxo começa com a definição de um problema, que é descrito em uma Proposta de Especificação do Software e termina com uma Especificação dos Requisitos do Software, que descreve, de forma detalhada, um conjunto dos requisitos que devem ser satisfeitos por uma solução implementável para o problema.

A atividade de Gestão de Requisitos continua durante toda a vida do produto, tanto para manter a consistência entre os requisitos e os demais documentos do produto como para incorporar de forma controlada possíveis modificações nos requisitos.

Este padrão não cobre aspectos de sistema externos ao software. Quando for o caso, esses aspectos devem ser especificados em documentos separados. Isso é particularmente importante nos projetos que envolvem bancos de dados. O povoamento e o controle da qualidade dos bancos de dados são considerados como fora do escopo do software, embora possam ser parte efetiva de um projeto de sistema. O mesmo vale para aspectos de hardware e redes de comunicação.

Define-se aqui a estrutura para a Especificação dos Requisitos do Software. Nenhuma seção deve ser omitida, mantendo-se a estrutura de numeração aqui indicada; seções não pertinentes ao projeto em questão devem ser indicadas com a expressão “Não aplicável”.

Define-se também um roteiro para revisão da Especificação dos Requisitos do Software. Este roteiro define os passos que devem ser seguidos na realização dessas revisões, e aplica-se principalmente à revisão técnica que o Praxis requer para o final da fase da Elaboração. Este roteiro inclui uma lista de conferência para a verificação do Modelo de Análise do Software.

1.2 Referências

A principal referência deste padrão é:

IEEE. *IEEE Std. 830 – 1993. IEEE Recommended Practice for Software Requirements Specifications*, in [IEEE94].

Outras referências importantes são [ABNT93], [ABNT94], [Booch+97], [Davis93], [Gause+89] e [Jacobson94].

1.3 Convenções de preenchimento

A página do título da ERSw deve incluir os seguintes elementos:

- nome do documento;
- identificação do projeto para o qual a documentação foi produzida;
- nomes dos autores e das organizações que produziram o documento;

- número de revisão do documento;
- data de aprovação;
- assinaturas de aprovação;
- lista dos números de revisão e datas de aprovação das revisões anteriores.

Recomenda-se incluir, logo após a página de título, um sumário (obrigatório para documentos de mais de oito páginas) e uma lista de ilustrações.

1.4 Organização dos requisitos específicos

A seção seguinte deste padrão tem uma organização na qual existe uma correspondência entre subseções do padrão (metadocumento) e subseções de cada Especificação de Requisitos (documento) particular. Entretanto, chama-se aqui atenção para alguns aspectos organizacionais da terceira seção da ERSw, na qual são detalhados os requisitos específicos.

Nessa seção, primeiramente, são listados todos os requisitos referentes a interfaces externas. Em seguida, os requisitos funcionais são listados na seguinte ordem:

- diagramas de casos de uso;
- casos de uso (fluxos).

Finalmente, são listados os requisitos não funcionais.

Recomenda-se o seguinte modelo de organização da seção 3 Requisitos específicos da ERSw:

1. Requisitos de interface externa

1.1. Interfaces de usuário

1.1.1. Interface de usuário 1

1.1.2. Interface de usuário 2

1.1.3. Interface de usuário ...

1.2. Interfaces de hardware

1.2.1. Interface de hardware 1

1.2.2. Interface de hardware ...

1.3. Interfaces de software

1.3.1. Interface de software 1

1.3.2. Interface de software ...

1.4. Interfaces de comunicações

1.4.1. Interface de comunicações 1

1.4.2. Interface de comunicações ...

2. Requisitos funcionais
3. Requisitos não funcionais
 - 3.1. Requisitos de desempenho
 - 3.2. Requisitos de dados persistentes
 - 3.3. Restrições ao desenho
 - 3.4. Atributos de qualidade
 - 3.5. Outros requisitos

Toda descrição de requisito pode conter uma subseção opcional de observações, que pode ser usada para indicar restrições, referências a documentos externos e interações com outros requisitos. As observações devem vir no final de cada requisito, e devem ser simplesmente omitidas quando não há o que informar.

2 *Preenchimento da Especificação dos Requisitos do Software*

2.1 *Introdução (ERSw-1)*

2.1.1 *Objetivos deste documento (ERSw-1.1)*

Descreve-se aqui o propósito da ERSw, especificando o público **deste documento**.

Descrever e especificar as necessidades da Pereira & Pereira Comercial Ltda. que devem ser atendidas em relação ao produto Mercí, bem como definir para os desenvolvedores o produto a ser feito.

Público-alvo: cliente, usuários e desenvolvedores do projeto Mercí.

Tabela 94 - Exemplo de Objetivos deste documento

2.1.2 *Escopo do produto (ERSw-1.2)*

Descreve-se aqui uma primeira visão sintética do escopo do produto especificado. Esta visão deve:

1. Identificar pelo nome o produto do software a ser desenvolvido. A ERSw pode especificar produtos com mais de um componente. Por exemplo, uma aplicação cliente-servidor consta de pelo menos um componente cliente e um componente servidor. Nesse caso, os componentes maiores devem ser enumerados. Resultados normais de um projeto, tais como documentos previstos no processo padrão, não são considerados componentes separados; por outro lado, seria cabível tratar um material de treinamento e demonstração como um componente separado.
2. Explicar o que o produto do software fará. Deve-se aqui reiterar a missão do produto, conforme a Proposta de Especificação do Software, modificada ou não pela iteração de Levantamento dos Requisitos.
3. Se necessário, esclarecer também os limites do produto, ou seja, o que o produto não fará. Por exemplo, isso pode ser conveniente para evitar falsas expectativas quanto a algum tópico, ou para ressaltar funções e atributos que serão implementados por outros componentes de um sistema maior, ou em versões futuras desse produto.
4. Identificar os benefícios que se espera obter com o produto e o valor destes para o cliente. Nesta subseção avalia-se o valor do produto como um todo. Caso se deseje desdobrar o valor pelas

funções propostas (por exemplo, para aplicação do método QFD, descrito em [Humphrey95] ou [Cheng+95]), isso deve ser feito na seção 4 Informação de Suporte. O valor pode ser descrito simplesmente pela importância atribuída na priorização dos requisitos, ou pode ser expresso de outras formas, inclusive quantitativas.

5. Ser consistente com outros possíveis documentos de nível mais alto, como documentos de definição de produto, modelos de processos de negócio ou especificações de requisitos de sistema. Embora a ERSw seja derivada da Proposta de Especificação de Software, não é necessário que seja completamente consistente com ela; modificações dos itens da proposta são aceitáveis, desde que feitas de acordo com o cliente. Nesse caso, não há necessidade de modificar a proposta original, já que esta não fará parte de nenhuma linha de base do projeto.

O Merci não fará vendas parceladas e só receberá dinheiro ou cheque.

O Merci só fará a Emissão de Nota Fiscal durante a Operação de Venda.

O Merci não fará um cadastro de clientes da mercearia Pereira & Pereira Comercial Ltda.

O preço de venda deverá ser calculado pela mercearia Pereira & Pereira Comercial Ltda. e informado ao Merci.

Atividades como backup e recuperação das bases de dados do sistema ficam a cargo da administração de dados e não serão providas no Merci.

O Merci não terá ajuda on-line.

Não haverá tolerância a falhas no Merci.

Tabela 95 - Exemplo de Limites do produto

Número de ordem	Benefício	Valor para o Cliente
1	Agilidade na compra e venda de mercadorias.	Essencial
2	Conhecimento do mercado de fornecedores visando a uma melhor conjugação de qualidade, preço e prazo.	Essencial
3	Diminuição de erros na compra e venda de mercadorias.	
4	Economia de .	Essencial
5	Eliminação da duplicidade de pedidos de compra.	Essencial
6	Qualidade na emissão da Nota Fiscal e Ticket de Venda em relação à emissão manual.	Essencial
7	Diminuição do custo de estocagem.	Desejável
8	Identificação de distorções entre o quantitativo vendido e o ainda existente no estoque.	Desejável
9	Maior agilidade nas decisões de compra.	Opcional

Tabela 96 - Exemplo de Benefícios do produto

2.1.3 Materiais de referência (ERSw-1.3)

Descreve-se aqui a informação necessária para que todas as fontes de dados citadas na ERSw possam ser recuperadas, caso necessário. Para isso, esta subseção deve:

- fornecer uma lista completa de todos os documentos referenciados na ERSw;
- identificar cada documento de acordo com os padrões bibliográficos usuais, indicando-lhes pelo menos o nome, número, data e organização que o publicou ou que pode fornecê-lo.

No caso de outras fontes, tais como atas de reunião e memorandos, a referência deve indicar como obtê-las.

Número de ordem	Tipo do material	Referência bibliográfica
1	Entrevistas	Ata das entrevistas que podem ser conseguidas com a secretária da mercearia Pereira & Pereira Comercial Ltda.
2	Manual	Manual de Usuário do Finance 98.
3	Relatório	Proposta de Projeto do Sistema de Gestão de Mercearia Merci Versão 1.0 - Revisão 1.

Tabela 97 - Exemplo de Materiais de referência

2.1.4 Definições e siglas (ERSw-1.4)

Descreve-se aqui a definição de todas as siglas, abreviações e termos usados na ERSw. Deve-se supor que a ERSw será lida tanto por desenvolvedores quanto por usuários, e por isso deve conter as definições relevantes, tanto de termos da área de aplicação, quanto de termos de informática usados na ERSw e que não sejam do conhecimento do público em geral.

Recomenda-se que esta subseção seja extraída de um glossário geral do projeto, de maneira que as definições e siglas sejam consistentes com aquelas usadas nos demais documentos desse projeto.

Número de ordem	Termo	Definição
1	Abertura do Caixa	Iniciação do caixa, autorizando o caixeiro a trabalhar. É informado o valor inicial no caixa.
2	Cadastro de Compras	Cadastro de pedido de compra de mercadoria da mercearia.
3	Cadastro de Fornecedores	Cadastro dos fornecedores de mercadorias da mercearia.
4	Cliente da Mercearia	Pessoa que procura a mercearia para efetuar suas compras.
5	Emissão de Nota Fiscal	Emissão de Nota Fiscal para o cliente da mercearia.
6	Emissão de Relatórios	Emissão de relatórios com as informações das bases de dados do Merci.
7	Fechamento do Caixa	Totalização das vendas do dia mais o valor inicial do caixa.
8	Merci	Nome do projeto de software; salvo se dito ao contrário, refere-se à versão 1.0.
9	Nota Fiscal	Documento exigido pela legislação fiscal para fins de fiscalização.
10	Ticket de Venda	Um relatório impresso pelo Merci que exibe e totaliza os itens referentes a uma venda efetuada.

Tabela 98 - Exemplo de Definições e siglas

2.1.5 Visão geral deste documento (ERSw-1.5)

Descreve-se aqui o que o restante da ERSw contém, indicando sua estrutura básica. Caso nesta ERSw particular alguma seção prevista neste padrão seja omitida ou alterada, a omissão ou alteração deve ser aqui justificada. Para que a numeração seja mantida consistente com o padrão, os títulos das seções e subseções devem continuar a aparecer no documento, indicando-se “Não aplicável.” no respectivo corpo.

De acordo com o Padrão para Especificação de Requisitos de Software, ou seja:
Parte 2: Descrição Geral do Produto Merci
Parte 3: Requisitos Específicos do Merci
Parte 4: Informação de Suporte

Tabela 99 - Exemplo de Visão geral desse documento

2.2 Descrição geral do produto (ERSw-2)

2.2.1 Perspectiva do produto (ERSw-2.1)

2.2.1.1 Diagrama de contexto (ERSw-2.1.1)

Inclui-se aqui um **diagrama de contexto**, isso é, um diagrama de blocos que mostre as interfaces do produto com seu ambiente de aplicação, inclusive os diversos tipos de usuários e outros sistemas do cliente com os quais o produto deve interagir (Figura 124). O diagrama de contexto deve indicar fontes e sorvedouros de dados. Se o produto fizer parte de um sistema maior, este diagrama deve estabelecer o relacionamento entre os requisitos do produto e os requisitos do sistema; deve também identificar as interfaces entre o produto e o restante do sistema.

Deve-se usar, como diagrama de contexto, um diagrama de casos de uso. Nesse diagrama, os usuários, sistemas externos e outros componentes de um sistema maior são representados por atores, enquanto os casos de uso representam as possíveis formas de interação do produto com os atores. Os casos de uso correspondem às funções principais do software ou sistema, que são detalhadas na seção 3 Requisitos Específicos da ERSw.

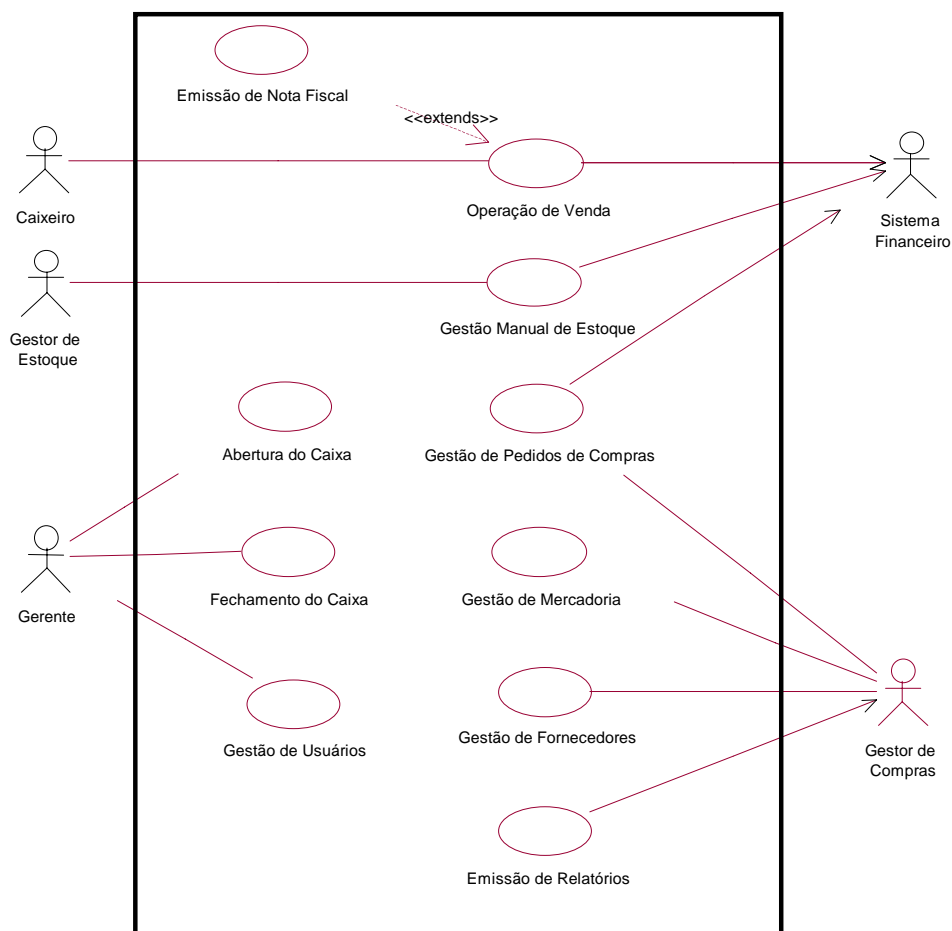


Figura 124 - Diagrama de contexto

2.2.1.2 Interfaces de usuário (ERSw-2.1.2)

Número de ordem	Nome	Ator	Caso de uso	Descrição
1	Tela de Abertura do Caixa	Gerente	Abertura do Caixa	Abertura do caixa, liberando a realização de Operações de Venda.
2	Tela de Compras	Gestor de Compras	Gestão de Pedidos de Compras	Emissão, consulta, baixa e exclusão de pedidos de compra para a mercearia.
3	Tela de Estoque	Gestor de Estoque	Gestão Manual de Estoque	Aumento ou diminuição da quantidade de uma mercadoria no banco de dados, para que este valor se adeque ao valor efetivamente em estoque.
4	Tela de Fechamento do Caixa	Gerente	Fechamento do Caixa	Fechamento do caixa, liberando assim a execução dos os casos de uso que operam em Modo de Gestão.
5	Tela de Fornecedores	Gestor de Compras	Gestão de Fornecedores	Inserção, alteração, consulta e exclusão de fornecedores no banco de dados.
6	Tela de Mercadorias	Gestor de Compras	Gestão de Mercadoria	Inserção, consulta, alteração e exclusão de mercadorias no banco de dados.
7	Tela de Nota Fiscal	Caixeiro	Emissão de Nota Fiscal	Emissão de nota fiscal de venda ao cliente da mercearia.
8	Tela de Pedidos de Compras	Gestor de Compras	Gestão de Pedidos de Compras	Visualização, inserção, exclusão e modificação de um pedido de compra específico.
9	Tela de Relatórios Gerenciais	Gestor de Compras	Emissão de Relatórios	Impressão de relatórios com as informações das bases de dados do Merci.
10	Tela de Usuários	Gerente	Gestão de Usuários	Inclusão, consulta, alteração e exclusão de usuários dos grupos definidos no Merci.
11	Tela de Vendas	Caixeiro	Operação de Venda	Vendas de mercadorias aos clientes da mercearia

Tabela 100 - Exemplo de lista de interfaces de usuário – telas

Identificam-se aqui as interfaces do produto com os seus usuários humanos. Para cada interface, detalhar o respectivo nome, caso de uso, ator e uma descrição sucinta do seu objetivo. Maiores detalhes devem ficar para a parte **3.1.1 Interfaces de usuário** da subseção **3.1 Requisitos de interface externa** da ERSw. O nome da interface deve ajudar a identificar a natureza requerida: tela, janela, relatório¹⁹ etc.

¹⁹ Relatórios podem ser on-line (telas e janelas informativas não interativas) ou impressos.

12	Relatório de Estoque Baixo	Gestor de Compras	Emissão de Relatório	Lista das mercadorias cujo estoque está abaixo do estoque mínimo.
13	Relatório de Fornecedores	Gestor de Compras	Emissão de Relatório	Lista dos fornecedores da mercearia.
14	Relatório de Mercadorias	Gestor de Compras	Emissão de Relatório	Lista das mercadorias comercializadas pela mercearia.
15	Nota Fiscal	Caixeiro	Emissão de Nota Fiscal	Nota Fiscal solicitada.
16	Pedido de Compra	Gestor de Estoque	Gestão de Pedidos de Compras	Emissão do Pedido de Compras solicitado.
17	Relação de Pedidos de Compra	Gestor de Compras	Emissão de Relatórios	Lista dos pedidos de compra da mercearia.
18	Ticket de Venda	Caixeiro	Operação de Venda	Ticket de caixa correspondente a uma Venda.

Tabela 101 - Exemplo de lista de interfaces de usuário - relatórios

2.2.1.3 Interfaces de hardware (ERSw-2.1.3)

Identificam-se aqui as características de hardware do sistema maior que sejam relevantes do ponto de vista da especificação do software, tais como dispositivos especiais. Não devem ser incluídos dispositivos suportados normalmente pelo ambiente operacional, mas apenas aqueles que requererão desenvolvimento especial de suporte.

Todo dispositivo aqui listado deve estar presente no diagrama de contexto. Detalhar nome e os respectivos ator e caso de uso, assim como uma descrição sucinta, que pode incluir uma referência a um documento externo de especificação. Maiores detalhes devem ficar para a seção 3 Requisitos Específicos da ERSw.

2.2.1.4 Interfaces de software (ERSw-2.1.4)

Identificam-se aqui as interfaces com outros produtos de software, tais como aplicativos que recebem dados do produto ou enviam dados para ele, seja on-line, através de arquivos ou através de bancos de dados. Não incluir componentes normais do ambiente operacional, como bibliotecas e plataformas.

Todo software aqui listado deve estar presente no diagrama de contexto. Detalhar nome, os respectivos ator, caso de uso, versão e fornecedor, assim como uma descrição sucinta, que pode incluir uma referência a um documento externo de especificação. Maiores detalhes devem ficar para a seção 3 Requisitos Específicos da ERSw.

Número de ordem	Nome	Atores	Casos de uso	Descrição
1	Conexão com Sistema Financeiro	Sistema Financeiro	Operação de Venda, Gestão Manual de Estoque, Gestão de Pedidos de Compras	Arquivo textual para interface com o sistema Xpto 98, da Xpto Inc., descrito no guia de referência Xpto 98. O propósito dessa interface é informar a um Gerenciador Financeiro as transações diárias, desonerando o Mercador de consultas e relatórios referentes à administração financeira.

Tabela 102 - Exemplo de descrição de interface de software

2.2.1.5 Interfaces de comunicação (ERSw-2.1.5)

Identificam-se aqui as características das redes de comunicação, tais como protocolos e padrões, que exijam tratamento especial por parte desse produto. Não incluir componentes normais de comunicação do ambiente operacional, tais como protocolos usuais de rede.

Todo recurso aqui listado deve estar presente no diagrama de contexto. Detalhar nome e os respectivos ator e caso de uso, assim como uma descrição sucinta, que resuma protocolos, padrões e outros aspectos relevantes, e que pode incluir uma referência a um documento externo de especificação. Maiores detalhes devem ficar para a seção 3 Requisitos Específicos da ERSw.

2.2.1.6 Restrições de memória (ERSw-2.1.6)

Identificam-se aqui os limites requeridos de memória primária e secundária. Estes limites só devem ser especificados quando isso for um requisito a ser exigido para a aceitação do produto.

Número de ordem	Tipo de memória	Limites aplicáveis
1	HD	O produto deve ocupar no máximo 100 MB (sem considerar as bases de dados).

Tabela 103 - Exemplo de descrição de restrições de memória

2.2.1.7 Modos de operação (ERSw-2.1.7)

Identificam-se aqui os modos requeridos de operação (normal e especiais), tais como:

- operação interativa;
- operação em lote;
- operação automática;
- realização de funções de suporte;
- realização de funções de backup e recuperação.

Número de ordem	Tipo de operação	Descrição da operação	Detalhes de operação
1	Interativa	Modo de Gestão	Modo de operação do Mercú no qual o sistema está disponível para a Gestão de Mercadorias, Gestão Manual de Estoque, Gestão de Pedidos de Compras, Gestão de Fornecedores, Emissão de Relatórios, Gestão de Usuários e Abertura do Caixa.
2	Interativa	Modo de Venda	Modo de operação do Mercú no qual o sistema está liberado apenas para a Operação de Venda, Emissão de Nota Fiscal e Fechamento do Caixa.

Tabela 104 - Exemplo de descrição de modos de operação

Operações realizadas em duas etapas (por exemplo, uma interativa e outra em lote) devem ser desdobradas como modos separados. Cada um dos tipos de operação da Tabela 104 pode ser desdobrado em modos diferentes, desde que esta diferença seja significativa do ponto de vista do desenvolvimento (por exemplo, tenha consequências relacionadas com concorrência ou segurança).

2.2.1.8 Requisitos de adaptação ao ambiente (ERSw-2.1.8)

Definem-se aqui possíveis requisitos de adaptação do produto aos ambientes particulares onde ele será implantado. Por exemplo, parâmetros e métodos de configuração requeridos para ambientes específicos devem ser aqui descritos.

Número de ordem	Requisito	Detalhes
1	Configuração da impressão do ticket de venda e da Nota Fiscal	Dimensões dos relatórios deverão ser configuráveis.

Tabela 105 - Exemplo de requisito de adaptação ao ambiente

2.2.2 Funções do produto (ERSw-2.2)

Identificam-se aqui as principais funções que o produto desempenhará, descrevendo de forma sintética o objetivo de cada uma. Cada função corresponde a um dos casos de uso presentes no diagrama de contexto.

Normalmente, uma função ou caso de uso corresponde a um único processamento completo, que gera algum valor para os usuários representados por um ator. Grupos de processamentos simples e correlatos (por exemplo, inclusão, exclusão e alteração dos mesmos itens) podem ser agrupados em um único caso de uso, desde que isso contribua para tornar a ERSw mais legível.

Número de ordem	Caso de uso	Descrição
1	Abertura do Caixa	Passagem para o Modo de Venda, liberando assim o caixa da mercearia para a Operação de Venda. O Gerente da mercearia deve informar o valor inicial desse caixa.
2	Emissão de Nota Fiscal	Emissão de Nota Fiscal para o cliente da mercearia (extensão da Operação de Venda).
3	Emissão de Relatórios	Emissão de relatórios com as informações das bases de dados do Merci.
4	Fechamento do Caixa	Totalização das vendas do dia e mudança para o Modo de Gestão.
5	Gestão de Fornecedores	Processamento de inclusão, exclusão e alteração de fornecedores.
6	Gestão de Mercadorias	Processamento de inclusão, exclusão e alteração de mercadorias.
7	Gestão de Pedidos de Compra	Processamento de inclusão, exclusão e alteração de pedidos de compra de mercadorias.
8	Gestão de Usuários	Controle de usuários que terão acesso ao Merci.
9	Gestão Manual de Estoque	Controle manual de entrada e saída de mercadorias.
10	Operação de Venda	Operação de venda ao cliente da mercearia.

Tabela 106 - Exemplo de lista de funções do produto

2.2.3 Características dos usuários (ERSw-2.3)

Descrevem-se aqui as principais características dos grupos de usuários esperados para o produto, tais como cargo ou função, permissão de acesso, frequência de uso, nível de instrução, proficiência no processo de negócio e proficiência em informática. Deve-se diferenciar entre grupos de usuários cujas atribuições ou permissões sejam distintas. Cada grupo distinto deve corresponder a um ator.

Número de ordem	Ator	Definição
1	Caixeiro	Funcionário operador comercial de caixa.
2	Gerente	Funcionário responsável pela abertura e fechamento do caixa, além do cadastro de usuários.
3	Gestor de Compras	Funcionário responsável por: <ul style="list-style-type: none"> • cadastrar as mercadorias pertencentes ao estoque; • manter os níveis do estoque em valores acima do mínimo permitido para cada mercadoria; • emitir os pedidos de compra da mercearia.
4	Gestor de Estoque	Funcionário responsável pela elaboração do inventário do estoque da mercearia e por manter esses níveis coerentes com as bases de dados do Merci.

Tabela 107 - Exemplo de descrição de atores

Número de ordem	Atores	Permissão de acesso	Frequência de uso	Nível de instrução	Proficiência na aplicação	Proficiência em informática
1	Caixeiro	Operação de Venda, Emissão de Nota Fiscal.	Diário em horário comercial	1º Grau	Operacional	Aplicação
2	Gerente	Abertura do Caixa, Fechamento do Caixa, Gestão de Usuários.	Diário	2º Grau	Completa	Aplicação Windows 95
3	Gestor de Compras	Gestão de Mercadorias, Emissão de Relatórios, Gestão de Fornecedores e Gestão de Pedidos de Compras.	Diária	3º grau	Completa	Aplicação Windows 95
4	Gestor de Estoque	Gestão Manual de Estoque.	Diário	1º Grau	Operacional	Aplicação

Tabela 108 - Exemplo de descrição de características dos usuários

2.2.4 Restrições (ERSw-2.4)

Descrevem-se aqui aspectos técnicos e gerenciais que possam limitar as opções dos desenvolvedores, tais como:

- restrições legais;
- limitações de hardware;
- restrições relativas a interfaces com outros produtos;
- restrições quanto a linguagens de programação;
- requisitos de auditoria;
- restrições de desempenho;
- restrições de confiabilidade;
- restrições de segurança.

Estas restrições podem gerar requisitos detalhados na subseção 3.3 Requisitos não funcionais da ERSw.

Número de ordem	Restrição	Descrição
1	Ambiente	O ambiente operacional a ser utilizado é o Windows 95 (ou compatível).
2	Ambiente	O sistema deverá executar em um Pentium 133 MHz, com impressora de tecnologia laser ou de jato de tinta, a ser usada para impressão de todos os relatórios, exceto os tickets de venda.
3	Ambiente	Será utilizada uma impressora específica para a emissão dos tickets de venda, configurável como impressora suportada pelo ambiente operacional.
4	Expansibilidade	O produto deve ser desenvolvido levando-se em consideração que poderá ser expandido para mais de um caixa.
5	Legal	O produto deverá estar em conformidade com as leis e regulamentos vigentes na época da aprovação da Especificação de Requisitos.
6	Segurança	O produto deverá restringir o acesso através de senhas individuais para cada usuário.

Tabela 109 - Exemplo de lista de restrições

2.2.5 Hipóteses de trabalho (ERSw-2.5)

Descrevem-se aqui fatores que não são restrições limitativas do desenho, como na subseção anterior, mas fatores cuja alteração requer modificações na ERSw, como, por exemplo, versão a ser utilizada do ambiente operacional ou plataforma de desenvolvimento. As hipóteses aqui arroladas correspondem a fatores de natureza técnica. Providências de ordem gerencial que devam ser tomadas pelo cliente devem ser listadas no Plano de Desenvolvimento do Software.

Número de ordem	Hipótese	De quem depende
1	Deve ser utilizado o sistema de gestão de bancos de dados < nome do sistema >.	Pereira & Pereira Comercial Ltda deve adquirir, instalar e povoar.

Tabela 110 - Exemplo de hipótese de trabalho

2.2.6 Requisitos adiados (ERSw-2.6)

Descrevem-se aqui os requisitos que foram identificados durante a elaboração dessa especificação, mas cujo atendimento se decidiu deixar para versões futuras. O preenchimento dessa subseção serve para registrar idéias no momento de seu aparecimento e para facilitar a engenharia de requisitos em novas versões.

Número de ordem	Referência ao requisito	Detalhes
1	Cadastro de Clientes	Gestão de informações a respeito dos clientes da mercearia.
2	Estorno no Caixa	Cancelamento de um ou mais itens de vendas concluídas.
3	Retirada no Caixa	Retirada de dinheiro no caixa durante o expediente (Modo de Vendas) da mercearia.

Tabela 111 - Exemplo de requisitos adiados

2.3 Requisitos específicos (ERSw-3)

2.3.1 Interfaces externas (ERSw-3.1)

2.3.1.1 Visão geral

Descreve-se aqui, de forma detalhada, todas as entradas e saídas do produto. As interfaces externas não incluem arquivos de trabalho usados apenas pelo produto, mas incluem qualquer tipo de dados partilhados com outros produtos e componentes de sistema.

3.1.3.1 Interface de software Sistema Financeiro

3.1.3.1.1 Fonte da entrada

Não aplicável.

3.1.3.1.2 Destino da saída

Arquivo texto para o Xpto 98.

3.1.3.1.3 Relacionamentos com outras interfaces

As interfaces de Estoque e de Venda geram lançamentos para a interface com o Sistema Financeiro.

Tabela 112 - Exemplo de requisitos para interface de software

Os campos que serão informados para o Sistema Financeiro são:

Data, Número, Tipo (Receita, Despesa, Prejuízo ou Ganho), Valor e Nome.

O campo Data é do tipo *DateTime*. Os campos Número e Valor são do tipo *Double*, e o campo Tipo é do tipo *varchar*. O campo Nome pode se referir ao nome do cliente, do fornecedor, ou ainda ser nulo, dependendo do tipo de operação que está sendo realizada. O campo Tipo tem as seguintes interpretações:

- Receita:
 1. A mercearia vende mercadoria para um cliente.
 2. A mercearia devolve uma mercadoria para o fornecedor.
- Despesa:
 1. A mercearia compra mercadoria de um fornecedor.
 2. O cliente da mercearia devolve uma mercadoria.
- Prejuízo: alguma mercadoria estragou ou foi roubada na mercearia.
- Ganho: o nível de estoque na prateleira é maior do que o registrado no Merc.

O formato do registro financeiro consiste nesses 5 campos, separados por uma vírgula. Cada linha do arquivo corresponde a um registro no Sistema Financeiro. Por exemplo, os seguintes registros são válidos:

```
"25/10/97" , "101" , "Ganho" , "1.000,00" , " "
"20/11/97" , "102" , "Despesa" , "1500,00" , "Fornecedor A"
"22/11/97" , "110" , "Prejuízo" , "50,00" , " "
"25/11/97" , "120" , "Receita" , "5000,00" , "Fornecedor B"
"25/11/97" , "122" , "Despesa" , "50,00" , "Cliente A"
```

Tabela 113 - Exemplo de formato de interface de software

O nome e a descrição sucinta da cada interface já devem fazer parte das 2.1.2 Interfaces de usuário da subseção 2.1 Perspectiva do produto da ERSw. Normalmente indicam-se aqui o conteúdo e o formato dos seguintes elementos de cada interface, quando aplicáveis (exemplos em Tabela 112 e Tabela 113):

- fonte da entrada;
- destino da saída;
- relacionamentos com outras interfaces;
- formato.

As interfaces gráficas de usuário recebem um tratamento especial, descrito a seguir.

2.3.1.2 Requisitos para interfaces gráficas de usuário

Sugere-se, no caso de interfaces gráficas de usuário, a inclusão dos seguintes elementos:

- um esboço do leiaute gráfico sugerido para a interface (Figura 125);
- uma descrição dos relacionamentos com outras interfaces (Tabela 114);
- um diagrama de estados, caso necessário para melhor entender-se o comportamento requerido da interface (Figura 126);
- uma lista dos campos de dados da interface (Tabela 115);
- uma lista dos comandos (botões de ação ou controles equivalentes) da interface (Tabela 116);
- observações.

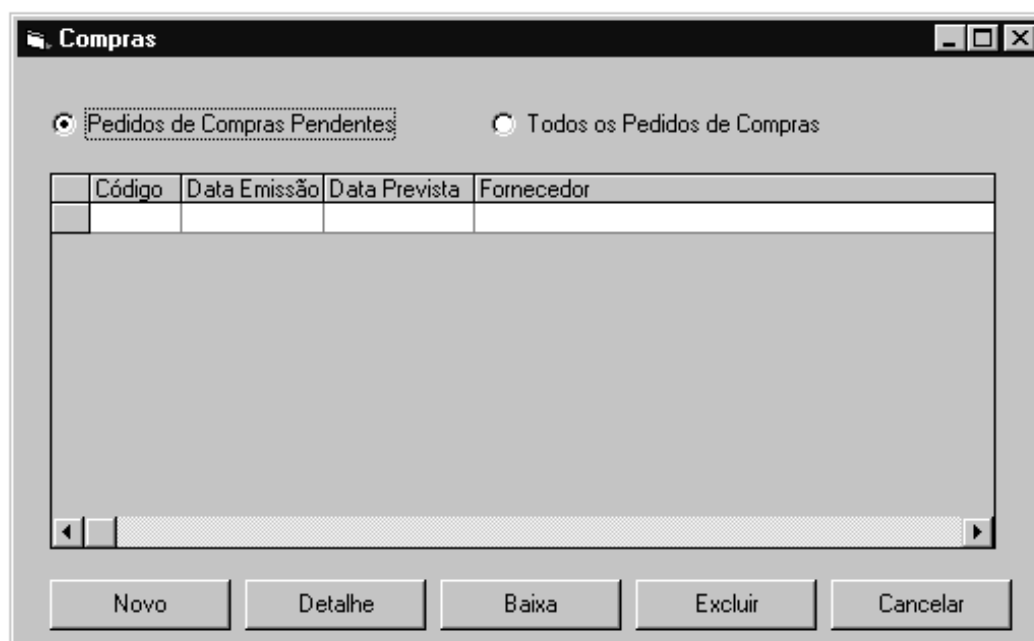


Figura 125 - Exemplo de esboço de leiaute de interface de usuário

<p>O botão Cancelar retorna à interface principal.</p> <p>O botão Emitir NF chama a interface Tela de Nota Fiscal.</p>
--

Tabela 114 - Exemplo de descrição de relacionamento com outras interfaces

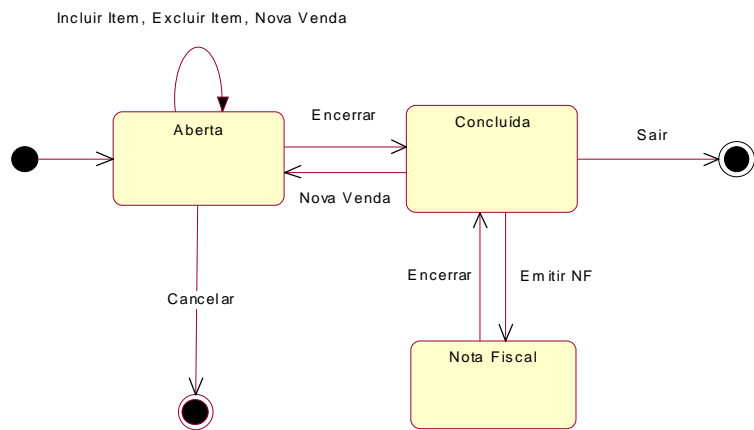


Figura 126 - Exemplo de diagrama de estados para interface de usuário

Diagramas de estado devem ser usados quando o comportamento da interface não for óbvio. Tipicamente, interfaces com três ou mais botões de ação requerem diagramas de estado. O diagrama de estado deve ser incluído no Modelo de Análise do Software, amarrado à respectiva classe de fronteira.

Número	Nome	Valores válidos	Formato	Tipo	Restrições
1	Código	Maior que 0.	Até 6 dígitos.	Número inteiro	Preenchido pelo Merci / não editável.
2	Data Emissão	Maior que data atual.	DD/MM/AAA A	Data	Preenchido pelo Merci / não editável.
3	Data Prevista	Maior que Data Emissão.	DD/MM/AAA A	Data	Preenchido pelo Merci / não editável.
4	Fornecedor	Não vazio.	Até 30 caracteres.	Texto	Preenchido pelo Merci / não editável.

Tabela 115 - Exemplo de lista de campos de uma interface

A lista dos campos deve detalhar todos os campos requeridos na interface. Fica entendido que, no desenho da interface definitiva, esses campos podem ser substituídos por soluções funcionalmente equivalentes, desde que isso contribua para facilitar o uso do produto. Para cada campo, devem constar:

- número;
- nome;
- valores válidos;
- formato (por exemplo, tamanho máximo, divisões do campo etc.);
- tipo (por exemplo, inteiro, real, moeda, data etc.);
- restrições (por exemplo, opcional, alterável, calculado etc.).

Para os comandos ou equivalentes (botões, itens de cardápio, hiperligações etc.), descrever:

- número;
- nome;
- ação (o que deve acontecer quando o comando é acionado);
- restrições (por exemplo, quanto o comando está habilitado).

Número	Nome	Ação	Restrições
1	Baixa	Faz a baixa do pedido de compra selecionado. Muda seu status para Atendido e automaticamente inclui os itens da compra no estoque da mercearia.	Sempre habilitado, com confirmação.
2	Cancelar	Retorna para a interface principal.	Sempre habilitado, com confirmação.
3	Detalhe	Aciona a interface Tela de Pedidos de Compras, para mostrar os detalhes do pedido de compra selecionado.	Sempre habilitado.
4	Excluir	Exclui um pedido de compra.	Sempre habilitado, com confirmação.
5	Novo	Cria novo pedido de compra e abre a interface Tela de Pedidos de Compras, para o preenchimento dos dados.	Sempre habilitado.

Tabela 116 - Exemplo de lista de comandos de uma interface

2.3.2 Requisitos funcionais (ERSw-3.2)

2.3.2.1 Visão geral

Os requisitos funcionais definem as ações fundamentais através das quais o produto aceita e processa as entradas especificadas, gerando as respectivas saídas. Nesta seção é feito o detalhamento desses requisitos, em nível suficiente para o desenho do produto, de seus testes de aceitação e de seu manual de usuário.

A forma de descrição funcional adotada nesse padrão é a Modelagem de Casos de Uso, baseada na notação UML. Os casos de uso descrevem o comportamento esperado do produto como um todo. Nessa subseção, eles são detalhados através de diagramas e de fluxos. Os diagramas de casos de uso descrevem os relacionamentos dos casos de uso entre si e com os atores, enquanto os fluxos descrevem os detalhes de cada caso de uso.

Deve-se notar que os passos dos fluxos dos casos de uso devem ter finalidade puramente explicativa, e não se constituir em hipóteses quanto ao desenho do produto. A existência de um passo na descrição de uma função não significa que deva existir um componente correspondente na arquitetura.

2.3.2.2 Diagramas de casos de uso

Recomenda-se incluir os seguintes diagramas:

- partições do diagrama de contexto, mostrando grupos correlatos de casos de uso primários e os atores;
- diagramas locais:

- um certo caso de uso e seus relacionamentos;
- todos os casos de uso para um certo ator;
- todos os casos de uso que se pretende implementar em uma liberação.

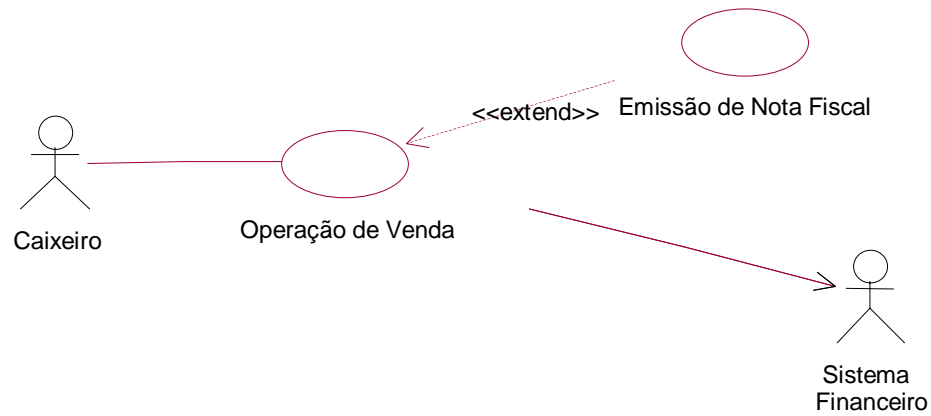


Figura 127 - Exemplo de diagrama local a um caso de uso

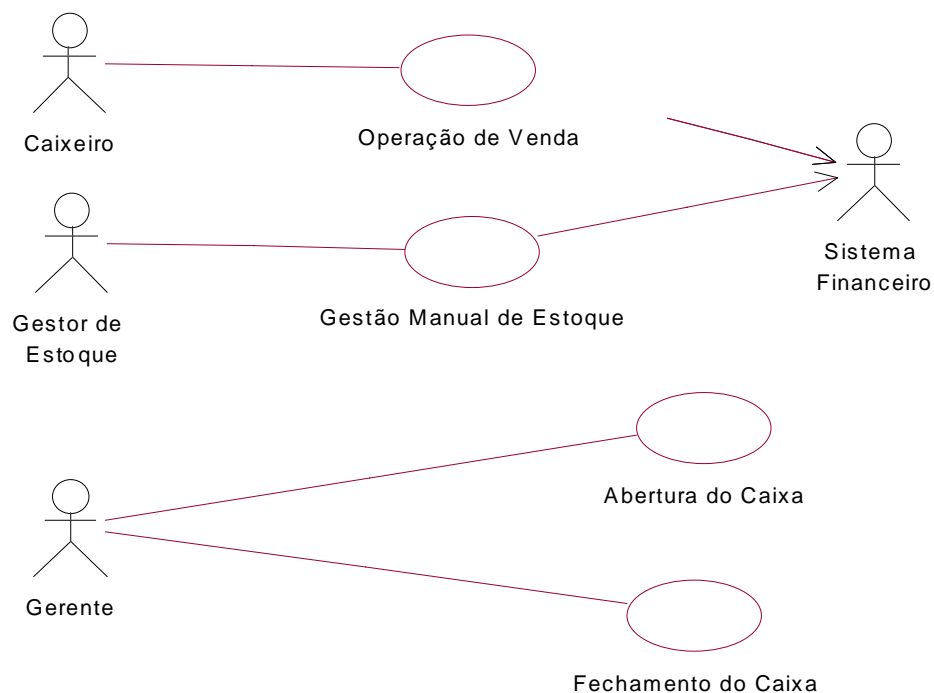


Figura 128 - Exemplo de proposta de casos de uso para uma liberação executável

É fundamental que esse diagrama seja legível; caso necessário, deve-se omitir os casos de uso e atores menos importantes, deixando-os para os diagramas detalhados da subseção [3.2.1 Diagramas de caso de uso](#) da ERSw. No caso de produtos com especificações muito complexas, pode ser útil particionar em grupos os casos de uso, atores e seus diagramas, através do uso de pacotes lógicos, para encapsular grupos de requisitos correlatos.

2.3.2.3 Fluxos dos casos de uso

Supõe-se que uma descrição sucinta do caso de uso já estará contida na subseção 2.2 Funções do produto da ERSw. Na seção 3 Requisitos específicos, deve-se descrever os seguintes detalhes de cada caso de uso:

- condições para a realização do caso de uso;
- fluxo principal do caso de uso, descrito na forma de uma sequência de passos;
- fluxos secundários do caso de uso (por exemplo, malhas de iteração e alternativas de condicionais);
- fluxos alternativos do caso de uso (por exemplo, tratamento de exceções e condições pouco frequentes);
- descrições mais formais, como diagramas de estado ou de atividade, se a complexidade do caso de uso o exigir;
- observações.

Toda mercadoria a ser vendida (item de venda) deve estar previamente cadastrada.
O Merci deve estar no Modo de Vendas.

Tabela 117 - Exemplo de condições do caso de uso Operação de Venda

O Caixeiro faz a abertura da venda.
O Merci gera o código da operação de venda.
Para cada item de venda, o Merci aciona o subfluxo Registro.
O Caixeiro registra a forma de pagamento.
O Caixeiro encerra a venda.
Para cada item de venda, o Merci aciona o subfluxo Impressão de Linha do Ticket.
O Merci notifica o Sistema Financeiro informando: Data, Número da Operação de Venda, “Receita”, Valor Total”, Nome do Cliente (caso tenha sido emitida a nota fiscal).

Tabela 118 - Exemplo de fluxo principal do caso de uso Operação de Venda

O Caixeiro registra o item de venda, informando a identificação e a quantidade.
O Merci totaliza a venda para o cliente da mercearia.

Tabela 119 - Exemplo de subfluxo: registro de item em Operação de Venda

Se o Gestor de Compras solicitar:
o Merci imprime o pedido de compra.

Tabela 120 - Exemplo de fluxo alternativo: Impressão de Pedido de Compras

As Observações podem ser usadas, por exemplo, para descrever condições posteriores, referências a requisitos de desempenho vinculados ao caso de uso e referências a informação externa relativa a esse caso de uso.

2.3.3 Requisitos não funcionais (ERSw-3.3)

2.3.3.1 Visão geral

Todo requisito não funcional tem um campo de descrição, onde o requisito deve ser sintetizado. Essa descrição deve ser sucinta e permitir a definição de um teste de aceitação, ou, no mínimo, de um item de revisão. Maiores detalhes podem ser relatados nas Observações.

2.3.3.2 Requisitos de desempenho

Todos os requisitos de desempenho devem ser especificados de forma quantitativa e mensurável. Por exemplo, “O produto deverá ter resposta rápida” não é um requisito aceitável; “90% das vezes o tempo de resposta do produto deverá ser inferior a 2 segundos” é um requisito aceitável.

A totalização da Operação de Venda não pode gastar mais do que 5 segundos, devendo ser realizada em 2 segundos, 80% das vezes.

Tabela 121 - Exemplo de requisito de desempenho

2.3.3.3 Requisitos de dados persistentes

Descrevem-se aqui estruturas lógicas de dados persistentes²⁰ que sejam usadas pelo produto. Cada estrutura de dados pode ser, por exemplo, um arquivo convencional, uma tabela em um banco de dados relacional ou uma classe persistente²¹.

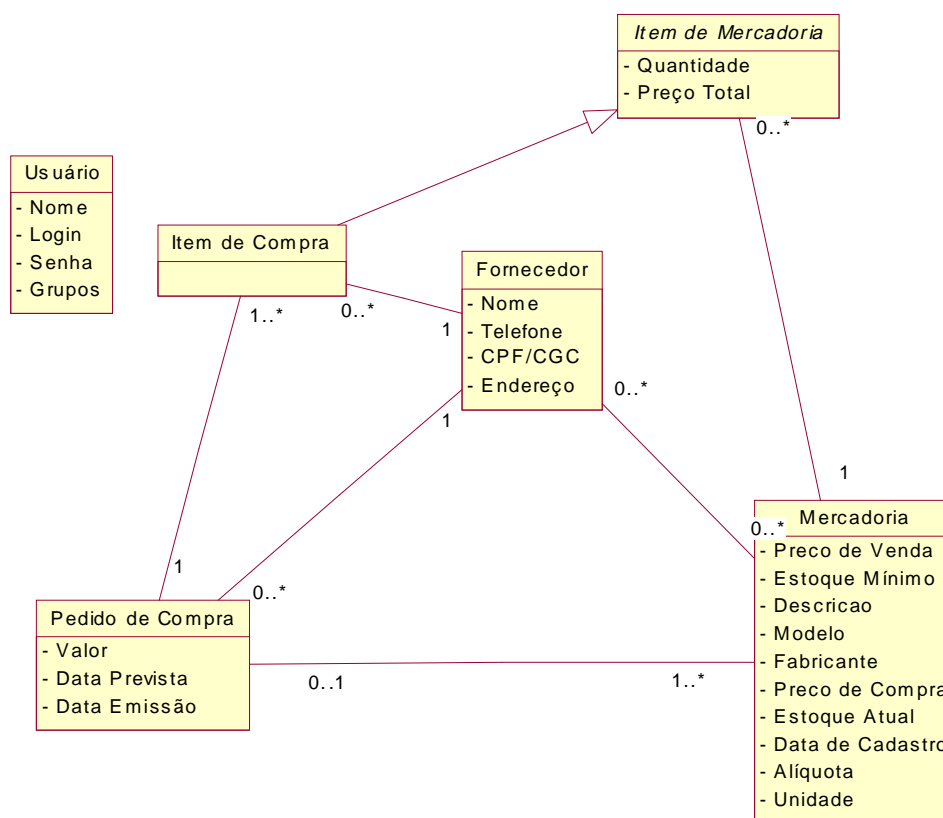


Tabela 122 - Exemplo de diagrama de dados persistentes (classes)

Deve-se apresentar um diagrama de entidades e relacionamentos ou um diagrama das classes persistentes, mostrando seus relacionamentos e atributos.

Além disso, devem ser apresentados os requisitos específicos aplicáveis a essas estruturas lógicas de dados persistentes. Eles podem incluir:

- tipos de informação que devam ser suportados;

²⁰ Isso é, que mantém seu valor após a execução do programa.

²¹ Grupo de objetos persistentes da mesma classe, armazenados em um banco de dados orientado a objetos ou de forma simulada, através de outro tipo de banco de dados.

- frequência de uso;
- restrições de acesso;
- restrições de integridade;
- requisitos de guarda e retenção de dados.

2.3.3.4 Restrições ao desenho

Essa subseção descreve restrições ao desenho que sejam impostas por padrões externos. Essas restrições devem ter sido listadas na subseção 2.4 Restrições, da ERSw, e são aqui detalhadas.

O leiaute da nota fiscal utilizada pela mercearia deve ter sido previamente aprovado pela Secretaria de Receita.

Tabela 123 - Exemplos de restrição ao desenho (legal)

2.3.3.5 Atributos de qualidade

Essa subseção indica os atributos de qualidade, seguindo as características e subcaracterísticas recomendadas pela norma ABNT ISO-9126. Quando possível, os atributos devem ser quantificados através de métricas adequadas.

Um operador de caixa proficiente em máquina registradora deverá ser capaz de aprender a operar o Mercú com um dia de treinamento.

Tabela 124 - Exemplos de atributos de qualidade (apreensibilidade)

2.4 Informação de suporte (ERSw-4)

A ERSw deve incluir informação de suporte adequada, tais como índices e apêndices. Possíveis apêndices incluem:

- diagramas e especificações extraídos do modelo de análise;
- cadastro de requisitos;
- amostras de formatos de entrada e saída;
- análises de custo e benefício, tais como matrizes QFD de requisito × benefício;
- informação auxiliar para os leitores da Especificação dos Requisitos do Software sobre tópicos de informática ou da aplicação;
- requisitos de embalagem, distribuição e instalação etc.

3 Revisão da Especificação dos Requisitos do Software

3.1 Introdução

Essa subseção apresenta um roteiro para revisão da Especificação dos Requisitos do Software (ERSw). Ao fim dessa revisão, deve-se assegurar que a ERSw:

- esteja conforme com esse padrão para Especificação de Requisitos de Software e com outros padrões aplicáveis ao projeto em questão;

- atenda aos critérios de qualidade dos requisitos;
- seja consistente com o Modelo de Análise do Software, tendo todos os seus casos de uso não triviais realizados através de colaborações entre objetos das classes constantes desse modelo;
- forneça informação suficiente para o desenho do produto, de seus testes de aceitação e do seu manual de usuário.

Nesse sentido, recomenda-se que, durante a revisão, todas as seções da ERSw sejam analisadas na ordem em que aparecem no documento, estabelecida pelo respectivo padrão, observando-se as questões descritas a seguir. As expressões grifadas se referem a seções e subseções do documento revisado.

3.2 Revisão da seção Página de Título

Verificar se os seguintes itens constam das páginas iniciais da ERSw:

- nome do documento;
- identificação do projeto para o qual a documentação foi produzida;
- número de revisão do documento;
- nomes dos autores e da organização que produziram o documento;
- data de emissão;
- datas de aprovação;
- assinaturas de aprovação;
- lista dos números de revisão e datas de aprovação das revisões anteriores.

Observar se a disposição desses itens na página de título está de acordo com as convenções estabelecidas pela organização.

3.3 Revisão da seção Sumário

Verificar se o sumário está consistente com o restante do documento. Essa seção é opcional para documentos com oito ou menos páginas.

3.4 Revisão da seção Lista de Ilustrações

Verificar se as listas de figuras e tabelas estão consistentes com o documento. Essa seção é opcional.

3.5 Revisão do corpo da Especificação dos Requisitos do Software

3.5.1 *Revisão da seção 1 Introdução*

3.5.1.1 Revisão da subseção 1.1 Objetivos

Avaliar se o público do documento de ERSw está claramente identificado e delimitado.

3.5.1.2 Revisão da subseção 1.2 Escopo do produto

Avaliar se:

- está claramente identificado pelo nome o produto do software a ser desenvolvido;
- está claramente explicado o que o produto do software fará e, caso necessário para evitar falsas expectativas, o que não fará;
- está claramente descrita a aplicação do produto especificado, inclusive suas metas e seus benefícios, se possível quantificados;
- esta subseção é consistente com outros possíveis documentos de nível mais alto.

3.5.1.3 Revisão da subseção 1.3 Materiais de referência

Esse item deve estar suficientemente detalhado para que os revisores possam avaliar se o contato com o cliente e usuários é completo. Verificar se foram incluídas referências à documentação das versões correntes (se houver), que permitam sua rápida localização.

3.5.1.4 Revisão da subseção 1.4 Definições e siglas

Deve-se averiguar, entre os revisores, se todos os termos foram compreendidos sem ambigüidades. Todas as definições apresentadas na ERSw devem ser agrupadas nessa seção. É importante verificar, entre os usuários, se a terminologia definida é a mais aceita e se é bem compreendida por todos.

3.5.1.5 Revisão da subseção 1.5 Visão geral deste documento

Verificar se a subseção reflete corretamente a estrutura do restante da ERSw.

3.5.2 Revisão da seção 2 Descrição geral do produto

3.5.2.1 Revisão da subseção 2.1 Perspectiva do produto

3.5.2.1.1 Diagrama de contexto (ERSw-2.1.1)

Observar se:

- foram definidas as interfaces importantes para todos os elementos relevantes do ambiente do produto;
- o diagrama apresentado é claro e pode ser compreendido sem textos complementares.

Verificar se bancos de dados compartilhados, arquivos compartilhados, interfaces com outros sistemas, interfaces com usuários e outras interfaces externas relevantes foram incluídas no diagrama.

3.5.2.1.2 Interfaces de usuário (ERSw-2.1.2)

Verificar se:

- são identificadas todas as interfaces do produto com os seus usuários humanos, com denominações adequadas;
- são identificados os atores e casos de uso associados com as interfaces;
- é apresentada uma descrição sucinta que resume de forma adequada o propósito da interface.

3.5.2.1.3 Interfaces de hardware (ERSw-2.1.3)

Verificar se:

- são identificados os dispositivos de hardware que exigirão algum tipo de tratamento especial por parte do produto;

- são identificados os atores e casos de uso associados com esses dispositivos;
- não são incluídos dispositivos cujo suporte seja rotineiro dentro do ambiente operacional;
- é apresentada uma descrição sucinta que resuma de forma adequada o propósito da interface.

3.5.2.1.4 Interfaces de software (ERSw-2.1.4)

Verificar se:

- são identificadas todas as interfaces com outros produtos de software, especialmente aplicativos que partilham dados com o produto;
- são identificados os atores e casos de uso associados com essas interfaces;
- não são incluídas interfaces rotineiras e transparentes para o produto, que sejam partes usuais do ambiente operacional;
- é apresentada uma descrição sucinta que resuma de forma adequada o propósito da interface;
- para cada produto requerido de software, foram indicados o nome, a sigla, a versão, o fornecedor e, se for o caso, documento de especificação.

3.5.2.1.5 Interfaces de comunicação (ERSw-2.1.5)

Verificar se:

- são identificados os recursos de comunicações que exigirão algum tipo de tratamento especial por parte do produto;
- são identificados os atores e casos de uso associados com esses recursos;
- não são incluídos recursos cujo suporte seja rotineiro dentro do ambiente operacional;
- é apresentada uma descrição sucinta que resuma de forma adequada o propósito da interface.

3.5.2.1.6 Restrições de memória (ERSw-2.1.6)

Verificar se foram descritos os requisitos de configurações de memória primária e secundária.

3.5.2.1.7 Modos de operação (ERSw-2.1.7)

Verificar se foram descritos os requisitos dos modos normal e especiais de operação. Observar a consistência entre os modos de operação especificados nessa Subseção e os requisitos de desempenho definidos na Subseção 3.3.1 Requisitos de desempenho da ERSw.

3.5.2.1.8 Requisitos de adaptação ao ambiente (ERSw-2.1.8)

Verificar se foram descritos os principais requisitos relativos a adaptações do produto aos locais onde será implantado.

3.5.2.2 Revisão da subseção 2.2 Funções do produto

Verificar se:

- a lista das funções é consistente com a subseção 1.2 Escopo do produto da ERSw;
- a lista de funções é consistente com o diagrama de contexto;
- todas as funções estão claramente descritas;

- todas as funções são consistentes com outros documentos relevantes.

3.5.2.3 Revisão da subseção 2.3 Características dos usuários

Verificar se:

- todos os usuários do produto foram definidos;
- a lista dos usuários é consistente com a subseção 1.2 Escopo do produto da ERSw;
- a caracterização da comunidade de usuários foi suficientemente detalhada;
- a lista de atores é consistente com o diagrama de contexto.

3.5.2.4 Revisão da subseção 2.4 Restrições

Verificar se essa seção descreve aspectos relevantes que possam limitar as opções dos desenvolvedores, e apenas esses aspectos.

3.5.2.5 Revisão da subseção 2.5 Hipóteses de trabalho

Verificar se essa seção descreve fatores que se supõem sejam atendidos para que o produto possa ser desenvolvido e implantado, e apenas esses aspectos.

3.5.2.6 Revisão da subseção 2.6 Requisitos adiados

Verificar se a lista do requisitos aqui arrolados é consistente com a subseção 1.2 Escopo do produto da ERSw.

3.5.3 *Revisão da seção 3. Requisitos específicos*

3.5.3.1 Revisão da subseção 3.1 Requisitos de interface externa

Verificar se:

- para cada comunicação entre caso de uso e os respectivos atores, são descritas as interfaces necessárias;
- foi especificada a quantidade necessária e suficiente de detalhes de todas as interfaces;
- a descrição das interfaces não avança em detalhes de desenho, não pertinentes a requisitos.

3.5.3.2 Revisão da subseção 3.2 Requisitos funcionais

3.5.3.2.1 *Visão geral*

Verificar se as funções estão organizadas de forma consistente com a organização recomendada nesse padrão. Deve ter sido seguida a organização por casos de uso, recomendada por este padrão, ou uma forma alternativa de organização, que seja recomendada por um processo personalizado adotado para o projeto.

Os seguintes princípios devem ser verificados:

- cada requisito deve atender às características de qualidade dos requisitos (precisão, consistência etc.);
- cada requisito relacionado com outros documentos deve conter referência a esses documentos;
- a organização dos requisitos deve ser orientada para facilitar a legibilidade e a compreensão.

3.5.3.2.2 Aspectos específicos de modelos de casos de uso

Verificar os casos de uso de acordo com a lista de conferência seguinte (Tabela 125).

Casos de uso	Foram todos descritos sucintamente na subseção <u>2.2 Funções do produto</u> .	
	Todos os casos de uso presentes nos diagramas e na subseção <u>2.2 Funções do produto</u> foram detalhados.	
Diagramas de casos de uso	São suficientes e adequados para o entendimento dos relacionamentos entre casos de uso e atores.	
	São consistentes entre si e com o diagrama de contexto.	
	Os relacionamentos entre casos de uso foram expressos corretamente.	
Fluxos dos casos de uso	São consistentes com a descrição sucinta do respectivo caso de uso.	
	São claros e bem apresentados.	
	São descritos com o nível de detalhe adequado.	
	Têm todas as interações com atores definidas e descritas de forma correta.	
	Todas as precondições dos casos de uso são definidas e descritas de forma correta.	
	Os subfluxos e fluxos alternativos estão suficientemente detalhados.	
	Diagramas de estado ou diagramas de atividade	São usados para esclarecer comportamentos complexos.
		São claros e bem apresentados.
		São consistentes com o fluxo e descrição dos casos de uso

Tabela 125 – Lista de conferência para casos de uso

3.5.3.3 Revisão da subseção 3.3 Requisitos não funcionais

3.5.3.3.1 Revisão da subseção 3.3.1 Requisitos de desempenho

Verificar se todos os requisitos de desempenho:

- são consistentes com as restrições contidas na subseção 2.4 Restrições da ERSw;
- são especificados de forma clara, quantitativa e mensurável.

3.5.3.3.2 Revisão da subseção 3.3.2 Requisitos de dados persistentes

Verificar se:

- foram especificados todos os requisitos lógicos de bancos de dados e arquivos compartilhados que sejam usados pelo produto;
- esses requisitos são consistentes com o diagrama de contexto, as interfaces de software e o modelo de análise.

3.5.3.3.3 Revisão da subseção 3.3.3 Restrições ao desenho

Verificar se:

- as restrições são consistentes com as restrições contidas na subseção 2.4 Restrições da ERSw;
- não foram incluídas decisões de desenho internas ao projeto.

3.5.3.3.4 *Revisão da subseção 3.3.4 Atributos de qualidade*

Verificar se foram incluídos os atributos relevantes de qualidade, seguindo as características e subcaracterísticas recomendadas pela norma ABNT ISO-9126, e se esses atributos estão quantificados através de métricas adequadas, quando necessário.

3.6 Revisão do Modelo de Análise

Inicialmente, verificar a consistência interna formal do Modelo de Análise. Essa verificação pode ser feita por inspeção de um registro (*log*) de checagem do modelo, gerado pela ferramenta de modelagem, que deve ser apresentado aos revisores.

Em seguida, verificar os detalhes do modelo usando a lista de conferência que se segue (Tabela 126).

Diagramas de classes	São claros e bem apresentados.	
	São organizados de forma adequada.	
	Relacionamentos	A apresentação dos relacionamentos é correta.
		Existem relacionamentos entre todas as classes cujos objetos trocam mensagens nas realizações dos casos de uso.
		A multiplicidade de cada papel é correta.
		As restrições aplicáveis aos relacionamentos, se existentes, foram anotadas.
		Os relacionamentos de agregação e composição são usados corretamente para exprimir relacionamentos de todo e parte.
		Os relacionamentos de herança, quando utilizados, descrevem adequadamente os aspectos de generalização e especialização.
Classes	As classes de fronteira são consistentes com as interfaces de usuário descritas na ERSw.	
	As classes de entidade representam entidades do domínio do problema.	
	As classes de controle representam lógica de casos de uso, algoritmos ou regras de negócio.	
	O campo de descrição descreve claramente o papel da classe dentro do domínio do problema.	
	Atributos	Fazem parte do domínio do problema, são campos de interfaces externas, ou são necessários para decidir sobre os relacionamentos de herança.
		A descrição de cada atributo esclarece o seu propósito.
	Operações	São suficientes para realizar os diagramas de interação dos quais os objetos da classe participam.
		O conjunto das operações de cada classe é completo em relação às responsabilidades dessas operações.
		Hierarquias de herança são usadas para extrair operações comuns a grupos de classes.
		As operações são documentadas de forma adequada.
		O nome de cada operação é significativo em relação à sua função.
		A descrição de cada operação esclarece o seu propósito.
		A assinatura da operação, caso presente, é relevante para o domínio do problema.
Realizações dos casos de uso	São adequadas para a validação dos casos de uso.	
	Os diagramas de interação são claros e bem apresentados.	
	Foi adequada a escolha entre diagramas de colaboração e de seqüência.	
	Todas as classes necessárias estão presentes e são documentadas na respectiva subseção.	

Tabela 126 – Lista de conferência para o Modelo de Análise

Página em branco

Revisões de Software

1 Visão geral

1.1 Objetivos

Este padrão tem por objetivo definir procedimentos para os seguintes aspectos das reuniões de revisão em grupo:

- preparação;
- condução;
- qualificação dos participantes;
- resultados.

A referência bibliográfica principal deste padrão é:

IEEE. *IEEE Std. 1028 – 1998. IEEE Standard for Software Reviews and Audits*, in [IEEE94].

Outras referências importantes são: [Freedman+93], [Humphrey90], [Humphrey95], [Paulk+95], [Pressman95] e [Weinberg93].

1.2 Alternativas

As **revisões técnicas** são o principal tipo de revisões de software, e as formalidades aqui definidas são essenciais para a sua eficácia na remoção de defeitos. Existem outros tipos de reuniões que podem funcionar como alternativas para as revisões técnicas, por razões de custo e prazo, ou por inadequação do material para esse tipo de revisão. Cabe ao gerente do projeto decidir, segundo as conveniências de cada projeto, sobre quando usar formas alternativas de revisão. Entre outras formas, destacam-se as seguintes:

- A **inspeção**, mais formal que a revisão técnica²², concentra-se na análise de aspectos selecionados, tratando de um aspecto de cada vez. É obrigatória a geração de uma lista de defeitos com classificação padronizada, requerendo-se a ação dos produtores para remoção desses defeitos. Inspeções removem de 60 a 90 por cento dos defeitos, e são 20 vezes mais eficazes que os testes ([McConnell96]). Normalmente, são mais aplicáveis na revisão do desenho detalhado e do código, no fluxo de Implementação.
- A **revisão de apresentação** (*walkthrough*) é uma revisão na qual o autor apresenta o material em ordem lógica, sem limite de tempo, a um grupo que checa o material à medida que ele vai sendo apresentado. Podem ser simulados os passos de um procedimento. Esse tipo de revisão não exige muita preparação prévia, e pode ser feito com maior número de participantes, por terem esses papel mais passivo. As revisões de apresentação têm eficácia média para a detecção de defeitos. Elas podem ser usadas nos marcos de projeto em que sejam necessárias apresentações ao cliente.
- A **revisão gerencial** é conduzida pelo gerente de um projeto, com o objetivo principal de avaliar os problemas técnicos e gerenciais do projeto, assim como o seu progresso em relação

²² Alguns autores ignoram a distinção entre revisões técnicas e inspeções.

aos planos. No Praxis, pelo menos uma revisão gerencial deve ser realizada ao final de cada iteração. Conforme a política adotada de controle de projetos, elas podem ser também realizadas por período (por exemplo, semana, quinzena ou mês).

- A **revisão informal** é realizada pelos autores e um grupo de pares, sem as formalidades requeridas pelas revisões técnicas. Esse tipo de revisão pode ser realizado dentro de oficinas de Requisitos, Análise, Desenho e preparação dos Testes.
- A **revisão individual** é realizada pelos autores, seguindo formalmente os roteiros pertinentes, eventualmente com a ajuda de pares.

2 Reuniões de revisão

2.1 Participantes

Recomenda-se que a revisão técnica seja feita por um grupo de 5 a 8 pessoas, assim distribuídas;

- 1 líder;
- 1 relator;
- 1 ou 2 autores;
- 2 a 4 revisores pares dos autores;
- 0 a 2 representantes dos usuários, dependendo do material em revisão.

Em casos excepcionais, é permitida a presença de assistentes, para fins de treinamento, avaliação ou transmissão de conhecimento. Esses assistentes não deverão se manifestar durante a realização da revisão. O líder da revisão pode abrir exceções para pedidos de esclarecimentos, tendo em vista eventuais objetivos didáticos.

2.2 Preparação

Os participantes da revisão técnica devem se apresentar para a reunião tendo lido, avaliado e preparado comentários a respeito do resultado do projeto a ser revisado. A leitura desse material não deve ser feita durante a reunião de revisão, em nenhuma hipótese.

O líder da revisão deve conferir se todo o material necessário foi recebido e analisado pelos revisores. Esse líder deve ser uma pessoa externa ao projeto, isto é, não deve ser um membro da equipe que produziu o material a ser revisado.

Recomenda-se que o grupo de participantes escolhido seja capaz de cobrir todo o escopo do material a ser revisado, para propiciar uma boa cobertura de defeitos. Isso não significa que cada membro do grupo deva ter conhecimento de todo o escopo, mas que a união dos membros deve cobrir os conhecimentos necessários para o trabalho de revisão.

2.3 Condução

No decorrer da reunião, é importante assegurar a participação de todos os integrantes da equipe. Um modo interessante de se fazer isso é determinar que cada membro faça pelo menos um comentário positivo e outro negativo a respeito do material revisado. Qualquer crítica ou sugestão de caráter individual deve ser sempre registrada. As recomendações finais devem refletir o ponto de vista mais pessimista entre os revisores.

Um outro ponto importante a ser lembrado é que o resultado do projeto é que está sendo revisado, e não seus produtores. Todas as críticas devem visar a melhoria da qualidade do resultado do projeto, e não a avaliação do desempenho dos produtores. Além disso, não é objetivo da revisão detalhar soluções, mas apenas dar sugestões sobre possíveis melhorias ao resultado do projeto em revisão.

A reunião de revisão não deve ultrapassar duas horas. Deve-se garantir que não sejam feitas interrupções externas à reunião e que os membros da revisão não sejam solicitados por telefonemas ou trabalhos externos.

A reunião deve ser iniciada pontualmente; nenhum participante poderá mais entrar, após o seu início. Se a reunião tiver que ser interrompida, ou se algum dos participantes estiver ausente, a revisão deverá ser cancelada, e nova data para a revisão deverá ser marcada pelo líder. Esse cancelamento é obrigatório nos seguintes casos:

- se só houver um autor do documento e este não comparecer;
- se o líder não comparecer e não houver outra pessoa qualificada para a posição;
- se o relator não comparecer e não houver outra pessoa qualificada para a posição;
- se após o remanejamento das funções dos membros, devido a ausências inesperadas, não sobra pelo menos dois revisores pares dos autores, além do líder e relator.

3 Perfil da equipe de revisão

3.1 Introdução

Entre os participantes da equipe de revisão devem incluir-se um líder, um relator, e pelo menos um autor. A seguir, apresenta-se o perfil do líder, do relator e dos participantes em geral.

3.2 Perfil do líder

O líder da revisão deve possuir as seguintes qualificações:

- compreender o propósito das revisões em geral, e entender seu funcionamento;
- compreender o propósito dessa revisão em particular;
- ter conhecimentos técnicos de alto nível sobre o material a ser revisado;
- ter participado de alguma outra revisão como revisor, e também, de preferência, como autor;
- não ter com qualquer um dos revisores alguma dificuldade pessoal que possa interferir em sua habilidade de liderar a revisão.

A Tabela 127 resume as responsabilidades do líder da revisão. É particularmente importante o balanço final da revisão, que deve considerar:

- se a revisão foi bem sucedida;
- se ela contribuiu efetivamente para a melhoria do produto;
- se algum dos participantes foi responsável por um eventual fracasso da revisão;
- se todos os participantes estão satisfeitos com os resultados da revisão;

- se o projeto sob revisão recebeu tratamento justo e adequado.

Antes da revisão	Conferir se todas as providências necessárias para a realização da reunião foram efetuadas pelo GGQSw.
Durante a revisão	Verificar se todos os participantes fizeram a preparação adequada.
	Suspender a reunião se não houver quorum de participantes com preparação adequada.
	Garantir que todos os participantes falem e contribuam.
	Garantir que todas as críticas sejam formuladas e registradas.
	Não deixar que o interesse diminua.
	Não permitir que a discussão perca a objetividade.
	Suspender a revisão em caso de perda de quorum, ou perturbação grave dos trabalhos.
	Procurar obter o consenso quanto aos resultados da revisão.
	Na ausência de consenso, garantir que sejam registradas como resultado as posições mais críticas dos revisores mais questionadores.
Ao final da revisão	Fazer um balanço da revisão.
	Garantir que o relatório esteja pronto e que seja preciso e conforme com o respectivo padrão.
	Analisar o que pode ser feito para tornar a próxima revisão ainda melhor, registrando na lista de tópicos de processo as possíveis sugestões.

Tabela 127 - Responsabilidades do líder da revisão técnica

3.3 Perfil do relator

O relator da revisão deve possuir as seguintes qualificações:

- compreender o propósito das revisões em geral, e entender seu funcionamento;
- compreender o propósito dessa revisão em particular;
- compreender o jargão e os formatos utilizados neste material;
- ser capaz de comunicar-se com as pessoas que estarão presentes na revisão;
- ter participado de alguma outra revisão como revisor ou como autor.

A Tabela 128 ilustra as responsabilidades do relator da revisão.

Antes da revisão	Saber identificar, pelo nome, todos os participantes da revisão.
	Reservar tempo para o trabalho que deverá fazer após a revisão.
	Disponer dos materiais necessários para manter um registro preciso, em formatos adequados.
Durante a revisão	Registrar todos os pontos discutidos.
	Fazer anotações que reflitam precisamente os comentários.
	Utilizar computadores portáteis ou outro registro visível dos pontos discutidos.
	Registrar os pontos numa linguagem neutra, de forma não ambígua.
	Ler em voz alta os pontos registrados.
	Classificar e totalizar os defeitos encontrados, segundo a classificação aqui adotada.
	Levantar junto aos revisores o total de horas gasto na preparação da revisão e registra-lo.
Ao final da revisão	Preparar o relatório de forma precisa.
	Distribuir o relatório para todos os participantes e demais interessados.
	Garantir que o relatório seja adequadamente revisado e assinado pelos revisores.
	Se tópicos de projeto ou processo foram levantados, redigir os respectivos relatórios.

Tabela 128 - Responsabilidades do relator da revisão técnica

3.4 Perfil dos revisores em geral

Os demais revisores devem levar em conta os seguintes aspectos de comportamento:

- estar preparado, lendo cuidadosamente o material antes da reunião;
- ter conhecimento técnico sobre parte do material da revisão;
- ser cooperativo;
- ser franco em relação ao material da revisão, mas polido em relação aos autores;
- compreender perfeitamente os pontos discutidos;
- usar um comentário positivo e outro negativo;
- apontar defeitos, mas não discutir como resolvê-los (a resolução não faz parte da revisão);
- evitar discussões sobre detalhes não pertinentes à qualidade do material em revisão;
- limitar-se aos assuntos técnicos;
- não avaliar os produtores, apenas o resultado do projeto.

Normalmente, os revisores são desenvolvedores, ou seja, pares dos autores. Em alguns casos, pode ser desejável a participação de usuários como revisores, por exemplo, em revisões das especificações de requisitos, dos desenhos das interfaces de usuário e da documentação de usuário. Nesse caso, os usuários devem estar cientes de que estarão analisando a qualidade do material sob revisão (por exemplo, quanto à limpeza, organização e clareza). O mérito das soluções contidas no material deve ter sido analisado anteriormente (por exemplo, em sessões de JAD).

4 Resultados da revisão

4.1 Relatórios da revisão

O resultado final da revisão é um Relatório de Revisão Técnica, redigido pelo relator da revisão e assinado pelos participantes, que apresenta os problemas encontrados no material revisado. Esse relatório apresenta um cabeçalho com um identificador único da revisão, tipo (revisão técnica ou uma das alternativas), data, hora de início e de término.

Quanto ao material sob revisão, deve-se informar:

- nome do projeto;
- tipo do material (especificação de requisitos, desenho dos testes, padrão etc.);
- tamanho (por exemplo, em páginas, servindo para estimar o esforço de revisões futuras);
- itens constituintes do material;
- autores do material.

Quanto aos participantes, o relatório deve conter os respectivos nomes, qualidade (líder, revisor etc.) e tempo de preparação. Esse dado é importante para a estimativa do custo das revisões. O relatório deverá incluir um registro e classificação dos defeitos encontrados quanto à natureza e gravidade. O material suplementar produzido pode incluir, por exemplo, as listas de tópicos descritas na próxima subseção. O resultado final deve ser um dos seguintes:

- **aceito sem modificações;**
- **aceito com pequenas modificações** (não será necessária nova revisão técnica para o resultado do projeto em pauta, colocando-se um dos membros da revisão técnica à disposição do gerente do projeto para uma revisão informal das modificações);
- **rejeitado para profundas modificações** (haverá necessidade de nova revisão após serem feitas as modificações sugeridas);
- **rejeitado para reconstrução** (será necessária nova confecção do material);
- **a revisão não foi completada** (foi necessário cancelar ou interromper a reunião de revisão, e uma nova revisão será marcada).

Identificação da revisão	Identificador	MERC-RR-05-99						
	Tipo	Revisão Técnica						
Data e hora	Data	12-03-1999						
	Hora de início	14:00						
	Hora de término	16:00						
Material sob revisão	Nome do projeto	Merci 1.0						
	Tipo	Especificação dos Requisitos do Software						
	Tamanho	71 páginas						
	Itens	Documento de Especificação dos Requisitos do Software do Projeto Merci Versão 1.0 Listagem do Modelo de Análise do Software do Projeto Merci Versão 1.0						
	Autores	Eudóxia Caxias Jovino Audax						
Participantes	Nome			Qualidade		Tempo de preparação (horas)		
	Eudóxia Caxias			Autora		2		
	Lúcia Malatesta			Líder		8		
	José Camões			Relator		6		
	Gérson Confúcio			Revisor		4		
	Guilherme Ockham			Revisor		5		
	Aristóteles Aquino			Revisor		4,5		
	Joaquim Pereira			Usuário		3		
Defeitos encontrados	Natureza	Omissão	Violação	Imprecisão	Estilo	Outros	Total	
		2	0	3	4	1	10	
	Gravidade	Críticos	Maiores		Menores		Total	
		0	1		9		10	
Material suplementar produzido	Lista de Tópicos do Projeto Lista de Tópicos do Processo							
Resultado da revisão	Aprovado com revisões menores							

Tabela 129 - Exemplo de relatório de revisão

De preferência, o relatório deve ser produzido durante a reunião, usando-se um processador de texto. Os defeitos encontrados devem ser registrados como anotações, no local onde se encontram no material. É conveniente o uso de um projetor, para permitir que todos vejam o que está sendo escrito. Ao final da reunião, deve-se produzir uma versão impressa, com listagem das anotações, que será conferida por todos os participantes e que receberá as assinaturas desses.

Os desenvolvedores devem trabalhar em cima do relatório de revisão sem alterar as anotações dos revisores. Para cada anotação, o desenvolvedor deve inserir seu próprio comentário, indicando que a proposta foi implementada, ou expondo as razões pelas quais discorda da proposta. Isso facilita a análise de pendências por terceiros, como os gerentes de projeto ou um grupo de garantia da qualidade (ou o instrutor, em uma situação de treinamento).

4.2 Classificação dos defeitos

Quanto à gravidade, os defeitos devem ser classificados em um dos grupos seguintes:

- **Críticos** - Defeito que reflete a ausência de um requisito essencial ou impede a utilização do produto. Não pode ser contornado, exigindo ação corretiva imediata.
- **Maiores** - Defeito que reflete a ausência de um requisito importante ou afeta, mas não impede, a utilização do produto. Pode ser contornado, por exemplo, combinando-se funções não atingidas pelo defeito. Exige ação corretiva tão logo possível.
- **Menores** - todos os outros problemas, como erros de documentação, erros nas mensagens do produto etc. Algumas correções, como a inclusão de facilidades menores solicitadas pelos usuários, podem ser deixadas para versões futuras. Em função desses aspectos, a determinação da prioridade de correção desses erros é de responsabilidade do gerente do projeto.

Quanto à natureza, os defeitos podem receber a seguinte classificação.

- **Omissão** - Ausência de uma seção, requisito, funcionalidade ou algum outro item qualquer no documento.
- **Violação** - Incoerência de um item do documento com um outro documento do projeto elaborado anteriormente, ou ainda alguma incorreção do documento levantada por um usuário participante da revisão técnica.
- **Imprecisão** - Requisito, seção ou funcionalidade do documento, especificada de forma ambígua ou incompleta.
- **Estilo** - Erros de português, de formatação etc.

Na ausência de consenso para a classificação dos defeitos, prevalece o pior status proposto pelos revisores. Recomendam-se as seguintes diretrizes:

- nas revisões de desenho e código, violações dos requisitos levam obrigatoriamente à rejeição;
- nas revisões de código, violações do desenho levam obrigatoriamente à rejeição;
- violações de padrões pertinentes ao material levam normalmente à rejeição;
- defeitos de forma (português, estética, falta de uniformidade de apresentação) levam normalmente à aceitação com pequenas modificações, exceto no caso de documentos de usuário, caso em que levam à rejeição.

Nas inspeções, deve-se usar uma classificação dos defeitos mais detalhada por tipo. A Tabela 130 mostra o padrão de classificação de defeitos do PSP ([Humphrey95]). Essa classificação detalhada permite identificar os tipos de erros mais frequentes e definir possíveis contramedidas. Por exemplo, podem-se introduzir nos padrões de desenho detalhado e codificação regras mais estritas que combatam os tipos de erros mais frequentes. O segundo dígito do código pode ser usado quando se quer uma classificação ainda mais detalhada.

Código	Tipo	Descrição
10	Documentação	Comentários, mensagens.
20	Sintaxe	Grafia, pontuação, digitação, formatos de instruções.
30	Construção	Gestão de configurações, ligação.
40	Atribuição	Declarações, nomes, escopo, limites.
50	Interface	Chamadas de métodos e procedimentos, entrada e saída.
60	Verificação	Mensagens de erro, falhas de verificação.
70	Dados	Estrutura, conteúdo.
80	Função	Lógica, apontadores, malhas, recursão, cálculos.
90	Sistema	Configuração, temporizações, memória.
100	Ambiente	Falhas nas ferramentas ou ambiente de desenvolvimento.

Tabela 130 – Classificação dos defeitos de desenho detalhado e código

4.3 Listas de tópicos

O Relatório de Revisão Técnica poderá apresentar os seguintes anexos opcionais:

- **Lista de Tópicos do Projeto**, que contém sugestões para melhorias no projeto. Estas sugestões não devem ser discutidas, mas simplesmente anotadas na ordem em que aparecerem, e podem ser aceitas ou não pelo gerente do projeto.
- **Lista de Tópicos de Processo**, contendo sugestões para melhorias nos processos de software utilizados, inclusive os processos de garantia da qualidade e revisão técnica. Os destinatários dessa lista são os grupos de suporte, como os grupos de Garantia da Qualidade e Engenharia de Processos.

Página em branco

Desenho de Interfaces de Usuário de Software

1 Diretrizes

1.1 Visão geral

Este padrão trata de diretrizes que devem ser seguidas no desenho das interfaces de usuário de um produto. Esta primeira seção trata de diretrizes genéricas, aplicáveis ao conjunto das interfaces. A seção seguinte apresenta diretrizes relacionadas com estilos específicos de interação.

1.2 Modelo mental

A interação com um produto se torna muito mais fácil quando os usuários conseguem formular um modelo mental consistente e íntegro. Por exemplo, no modelo de manipulação direta, cada ação consiste na seleção de um objeto e na aplicação de um dos comandos possíveis no estado corrente da interface. Idealmente, todas as interações devem ter a forma prevista no modelo, o que diminui a necessidade de memorização e a possibilidade de erros.

O modelo mental expressa a compreensão que os usuários têm do funcionamento de um sistema. Quando um modelo mental tem exceções e inconsistências, surgem dificuldades de interação. O desenho das interfaces deve ajudar na apreensão do modelo mental.

1.3 Consistência e simplicidade

O modelo mental íntegro, completo e simples promove a consistência da interação. A consistência ajuda os usuários a reaproveitar o conhecimento já adquirido, por meio de analogias. Com isso, eles aprenderão novas funções e novos mecanismos com maior rapidez. A consistência depende de vários atributos:

- terminologia adequada, consistente com o jargão da área de aplicação;
- aspectos visuais dos elementos das interfaces (cor, tamanho, leitura);
- comportamento das funções.

Alguns exemplos de aplicação do princípio da consistência são os seguintes:

- caracteres especiais de texto devem ter sempre a mesma função;
- comandos globais (Ajuda, Estado, Cancelar) devem ser invocáveis de qualquer estado;
- comandos genéricos (Recortar, Copiar, Colar) devem ter resultados análogos em qualquer situação.

Muitos sistemas interativos são difíceis de usar por causa da própria complexidade do domínio da aplicação. Um conjunto de interfaces mal desenhadas pode agravar muito a dificuldade de uso. Os desenhistas devem empenhar-se em simplificar as interfaces, dentro do que o orçamento e o prazo do projeto permitirem.

Os símbolos e ícones devem ser significativos dentro do domínio da aplicação. As tarefas complexas devem ser subdivididas até o nível necessário para um bom entendimento. Não deve ser oferecida aos usuários uma quantidade excessiva de campos e comandos. É preferível identificar um conjunto de

funções de uso freqüente, e investir em torná-las mais fáceis. As interfaces principais devem conter apenas os campos mais usados, colocando-se os demais em interfaces secundárias, acionadas somente quando necessário.

1.4 Questões de memorização

O armazenamento e a restauração de informação na memória humana são cansativos e sujeitos a falhas. Um resultado conhecido diz que a memória humana de curto prazo só é capaz de armazenar de cinco a nove itens de informação de cada vez. No desenho das interfaces de usuário, o tratamento das questões de memorização procura evitar que esse limite seja ultrapassado, em qualquer tipo de interação.

O fechamento freqüente das tarefas reduz parte importante da carga de memorização dos usuários. Quando uma tarefa com muitos passos tem de ser executada sem fechamento, corre-se o risco de que uma falha do sistema a deixe em estado inconsistente. Mesmo na ausência de falha, uma interrupção que o usuário sofra pode fazer com que este perca o contexto em que se encontra.

Fazer o usuário reconhecer objetos e funções, em lugar de lembrar-se, também reduz a memorização necessária. Linguagens de comando são um estilo de interação difícil porque exigem que o usuário memorize completamente a grafia do comando, os parâmetros e a ordem de chamada desses. Um cardápio, por outro lado, apresenta diretamente as opções possíveis: o usuário tem apenas que reconhecer aquela que lhe interessa.

Para os usuários novatos, uma estratégia baseada no reconhecimento dos objetos e funções facilita o aprendizado, pois não é necessário decorar previamente listas de comandos. Isso é conveniente também para os usuários casuais, que utilizam um produto a intervalos mais longos. Mesmo os usuários experientes geralmente só usam com freqüência um pequeno subconjunto de objetos e funções.

1.5 Questões cognitivas

Uma interface eficaz é transparente para os usuários. Ela permite a concentração na tarefa que deve ser realizada, e não nos mecanismos oferecidos pelo produto. Para isso, o desenho das interfaces deve minimizar as transformações mentais necessárias. Os mecanismos de associação mental podem ser estimulados através das seguintes técnicas:

- uso de mnemônicos significativos;
- atalhos e aceleradores usados através de uma seqüência significativa;
- desenho adequado dos ícones e outros elementos gráficos, se possível baseando-os em conceitos do domínio da aplicação.

É geralmente recomendável usar analogias com conceitos do mundo real, baseadas em elementos do domínio da aplicação, que sejam bem conhecidos pelos usuários. Muitas vezes os objetos e as funções das interfaces são metáforas de ferramentas não informatizadas do domínio da aplicação (Tabela 131).

Tarefa	Metáfora usual
Edição de texto	Máquina de escrever
Editoração de texto	Tipografia
Edição gráfica bidimensional	Ferramentas de desenho
Animção gráfica tridimensional	Estúdio de cinema

Tabela 131 - Exemplos de metáforas

Metáforas são úteis, mas, quando tomadas de forma muito literal, podem restringir desnecessariamente o desenho. Um exemplo absurdo seria exigir, para apagar um caractere em um editor de texto, que esse caractere fosse redigitado sobre uma "fita corretiva". Às vezes é preciso buscar um equilíbrio delicado entre os requisitos de consistência das interfaces e o fato de que certas operações são muito mais simples em um sistema informatizado do que em um ambiente físico.

1.6 Realimentação

Todo sistema interativo requer um bom uso de técnicas de realimentação (feedback) para o usuário. Através de mecanismos de realimentação os usuários são informados sobre o real efeito das suas ações, aumentando a segurança e portanto a produtividade. A realimentação sintática informa se o usuário selecionou funções e objetos corretos, enquanto a realimentação semântica confirma se a ação invocada teve o resultado desejado (Tabela 132).

Ação	Realimentação sintática	Realimentação semântica
Abrir arquivo	Item "Abrir" do cardápio "Arquivo" é destacado.	É exibida uma lista dos arquivos que podem ser abertos.
Mover arquivo	Ícone do arquivo é destacado e arrastado.	Ícone do arquivo desaparece na pasta de origem e surge na pasta de destino.
Excluir objeto	Item "Limpar" do cardápio "Editar" é destacado.	Representação do objeto desaparece (possivelmente, após confirmação).

Tabela 132 - Exemplos de realimentação

A tolerância quanto aos tempos de resposta varia muito com a complexidade percebida das tarefas. Tarefas simples e frequentes devem ser executadas em um segundo, no máximo. Tarefas normais têm tempo de execução típico entre dois e quatro segundos. Para tarefas complexas, tempos maiores são tolerados, desde que os usuários sejam devidamente informados.

A demora em tarefas complexas é mais bem tolerada se os usuários receberem indicadores apropriados de estado. Nesse caso, o produto fornece ao usuário algum tipo de indicador, gráfico ou textual, de que um processamento está em andamento. Evitam-se assim, por exemplo, dúvidas sobre se o sistema travou ou não.

O indicador é ainda melhor se for capaz de fornecer uma estimativa do percentual executado da tarefa, e de quanto tempo se estima que ainda falte para terminá-la. Terminada a tarefa, uma mensagem de término deve permanecer tempo suficiente para ser lida, sendo depois removida sem necessidade de confirmação por parte do usuário.

1.7 Mensagens do sistema

O fraseado das mensagens deve ser orientado ao usuário, considerando-se as tarefas da aplicação que estão sendo executadas. Não se deve incluir nas mensagens detalhes de processamento que não são de interesse da área de aplicação, como o nome do módulo que emitiu a mensagem e outras aberrações do mesmo gênero.

O fraseado deve ser positivo e não ameaçador. Para muitas pessoas, "erro fatal" ou "falha geral de proteção" dá a impressão de que aconteceram catástrofes, presumivelmente por culpa do usuário. Naturalmente, mensagens condescendentes, de humor duvidoso ou politicamente incorretas devem ser evitadas. Os termos devem ser corteses, mas precisos e impessoais. Não se devem usar referências antropomórficas ao computador. Geralmente, atribuir características humanas ao computador apenas confunde os usuários.

Os problemas ocorridos devem ser informados de maneira precisa. Quando possível, as mensagens devem sugerir procedimentos alternativos ou corretivos. Para evitar que as mensagens percam em concisão, essas sugestões podem ser oferecidas através do acionamento de um comando de "Ajuda" ou "Mais Informação".

Muitos produtos correntes no mercado são deficientes quanto às mensagens. A Tabela 133 apresenta uma coleção recente de mensagens encontradas em alguns dos aplicativos mais comuns do mercado.

Mensagem	Programa que emitiu
Houve um erro ao enviar um comando ao programa.	Planilha eletrônica
O <nome do programa> encontrou um erro que não pode ser corrigido. Você deve salvar as apresentações, sair e reinicializar o <nome do programa>.	Editor de apresentações
FAVOR REPETIR A OPERACAO (3 98 888).	Sistema de "Personal Banking"
Houve um problema com o registro	Ambiente de operação
A operação falhou. Não foi possível localizar um objeto.	Organizador pessoal
A interface para troca de mensagens retornou um erro desconhecido. Se o problema persistir, reinicie o <nome do programa>.	Organizador pessoal
A mensagem não foi enviada por causa de um erro interno. Salve seu trabalho.	Organizador pessoal
Houve um erro interno.	Navegador WWW
Impossível carregar "Hlink.dll"	Editor de apresentações
O <nome do programa> encontrou um problema que não pode ser diagnosticado nem solucionado.	Editor de apresentações
Erro de configuração do sistema.	Sistema de "Personal Banking"
Erro.	Controlador de impressora

Tabela 133 - Contra-exemplos de mensagens

1.8 Modalidade

Um modo é um estado de uma interface de usuário no qual uma ação tem um significado diferente do que tem em outro estado. Em consequência, ações aparentemente semelhantes podem ter resultados diferentes, dependendo do modo em que a interface está. De preferência, as interfaces não devem possuir modos. Quando isso for realmente necessário, os usuários devem receber realimentação que indique claramente o modo que está ativo. Por exemplo, a forma do cursor pode ser usada para sinalizar o modo corrente. A Tabela 134 mostra alguns dos formatos de cursor recomendados pelo manual de estilo do Windows 95.






Formato	Posição da tela	Ações aplicáveis
	sobre a maioria dos objetos	apontamento, seleção ou movimentação de objetos
	sobre o texto	seleção de texto
	dentro de uma janela	aproximação ou afastamento para visualização
	sobre uma margem redimensionável	redimensionamento horizontal da margem
	sobre uma margem redimensionável	redimensionamento vertical da margem

Tabela 134 - Cursores

Nas caixas de diálogo modais, o usuário é forçado a concluir uma tarefa antes de abandonar a caixa. Elas são utilizadas quando uma decisão do usuário é imprescindível para os próximos passos de uma interação, como por exemplo para confirmar a realização de ações potencialmente destrutivas. Nas caixas de diálogo não modais, o conjunto de comandos normais do aplicativo continua disponível ao

usuário. No estilo do Windows 95 caixas de diálogo não modais são usadas para operações de pesquisa e substituição de texto.

1.9 Reversibilidade

Quanto mais facilmente os usuários puderem reverter as suas ações, mais fácil de usar é o produto. A reversibilidade permite que o usuário explore diferentes cursos de ação, com menos receio de causar estragos. Por outro lado, a reversibilidade geralmente encarece o desenho e a implementação. Por isso, existem diversos graus de reversibilidade que devem ser considerados como alternativas de desenho.

O desenho da reversibilidade deve incluir as seguintes decisões:

- o número de ações que o usuário pode reverter, ou seja, o número de níveis de "Desfazer";
- a possibilidade de refazer ações desfeitas, através de mecanismos de "Refazer";
- quando existem múltiplos níveis de desfazer, a possibilidade de retornar diretamente a um nível profundo, através de mecanismos de "Histórico".

1.10 Atração da atenção

Para atrair a atenção do usuário aos pontos realmente importantes, esses devem ter o destaque necessário. Utilizar em excesso mecanismos de atração equivale a não utilizar nenhum, com a desvantagem de tornar a interface confusa. Algumas limitações importantes devem ser observadas:

- utilizar no máximo dois níveis de intensidade em cada interface;
- utilizar com cuidado sublinhado, negrito e inversão de vídeo;
- utilizar em uma mesma tela no máximo três tipos e quatro tamanhos de fonte;
- utilizar de preferência fontes com serifa, que são de leitura mais fácil;
- utilizar maiúsculas e minúsculas, conforme as regras da língua, e não apenas maiúsculas;
- utilizar piscamento apenas em situações muito importantes, para mensagens curtas.

Os estímulos de áudio são importantes por terem tempo de resposta muito curto. No seu uso, as seguintes regras devem ser observadas:

- reservá-los para situações mais importantes, ou situações em que há excesso de informação visual;
- reservar sons mais intensos ou estridentes para situações de emergência;
- considerar que na plataforma o áudio pode ser inexistente ou estar desabilitado.

As cores podem representar um recurso importante para chamar a atenção, desde que também não sejam usadas de forma excessiva. As seguintes diretrizes são recomendadas:

- utilizar no máximo quatro cores diferentes em cada interface, especialmente se a informação textual for predominante;
- não utilizar azul e cinza para detalhes, reservando-os para áreas de fundo de textos de cor clara;

- utilizar cores de fundo e de primeiro plano contrastantes, desde que não sejam de alta saturação;
- combinar a codificação em cores com a codificação em formas, para contemplar os usuários com deficiência de percepção de cores;
- utilizar cores para representar mudanças de estado de objetos da aplicação;
- considerar convenções culturais, como a que associa a cor vermelha ao perigo.

1.11 Exibição

De interface para interface, os elementos correspondentes devem ser mantidos nas mesmas posições, sempre que possível. Por exemplo, cardápios, botões e linhas de mensagem devem ser mantidos em posições consistentes entre si, nas diversas interfaces. As opções padrão devem também ser consistentes, mas há casos especiais em que essa regra deve ser quebrada. Por exemplo, na maioria das interfaces o botão padrão, acionado pela tecla de retorno, é o botão de OK, mas em interfaces potencialmente destruidoras de informação pode ser preferível colocar o botão de CANCELAR como padrão.

Para gerir a complexidade das interfaces, devem-se observar as seguintes diretrizes:

- comandos concisos;
- mensagens concisas;
- ícones de fácil reconhecimento;
- baixa densidade de informação no espaço das interfaces;
- distribuição balanceada de informação no espaço das interfaces;
- distribuição balanceada de áreas vazias no espaço das interfaces;
- agrupamento de informações correlatas;
- alinhamento de colunas e linhas.

1.12 Diferenças individuais

No desenho das interfaces, é preciso aceitar as diferenças e experiências individuais dos usuários. Um erro comum é aceitar a premissa de que as preferências do próprio desenhista são representativas das preferências da comunidade dos usuários reais. Na realidade, o comportamento dos usuários é influenciado por fatores como experiência na aplicação, idade e escolaridade.

Essas diferenças afetam as taxas de desempenho dos usuários. O perfil de erros mais frequentes varia também muito de um usuário para outro. Se possível, os usuários devem dispor de mecanismos para configurar suas próprias preferências. Por outro lado, o custo desses mecanismos pode ser elevado.

Um grupo importante de diferenças individuais decorre de diferenças de experiência entre os usuários. Por exemplo, os usuários novatos de um produto não têm conhecimento sintático, e podem ter deficiências de conhecimento semântico de computação e até da tarefa. Eles precisam trabalhar com conjuntos pequenos e poderosos de comandos, recebendo realimentação adequada e mensagens esclarecedoras.

Os usuários intermitentes geralmente têm conhecimento semântico apropriado. Entretanto, como usam o produto com pouca frequência, geralmente têm dificuldades com a sintaxe. Para ajudá-los, o desenhista de interfaces recorre a:

- estratégias de escolher em vez de lembrar-se, através do uso de cardápios e listas de opção;
- mecanismos de reversão de ações e recuperação de erros;
- lembretes e dicas, como linhas de mensagem que explicam o objeto sobre o qual o cursor está, ou mensagens de esclarecimento que aparecem quando o cursor se demora sobre um objeto.

Para os usuários experientes, o desenho das interfaces pode assumir o conhecimento sintático e semântico, enfatizando-se a produtividade. Deve-se usar recursos de aceleração, como atalhos, macros e linguagens de comando, e privilegiar desenhos internos que otimizem os tempos de resposta. Mecanismos de realimentação e de confirmação devem ser simplificados, evitando-se que contribuam para perda de tempo.

2 *Estilos de interação*

2.1 Janelas

2.1.1 *Visão geral*



Figura 129 - Exemplo de janela primária

As janelas são o principal canal de interação nos sistemas com interfaces gráficas de usuário. Elas permitem o trabalho simultâneo em várias tarefas. As janelas primárias (Figura 129) aparecem na abertura dos aplicativos, e através delas são invocadas as janelas secundárias. Janelas secundárias

(Figura 130) podem tomar formas especializadas, como caixas de diálogo. Janelas partidas (Figura 131) apresentam visões diferentes das mesmas estruturas: por exemplo, resumo e detalhes.

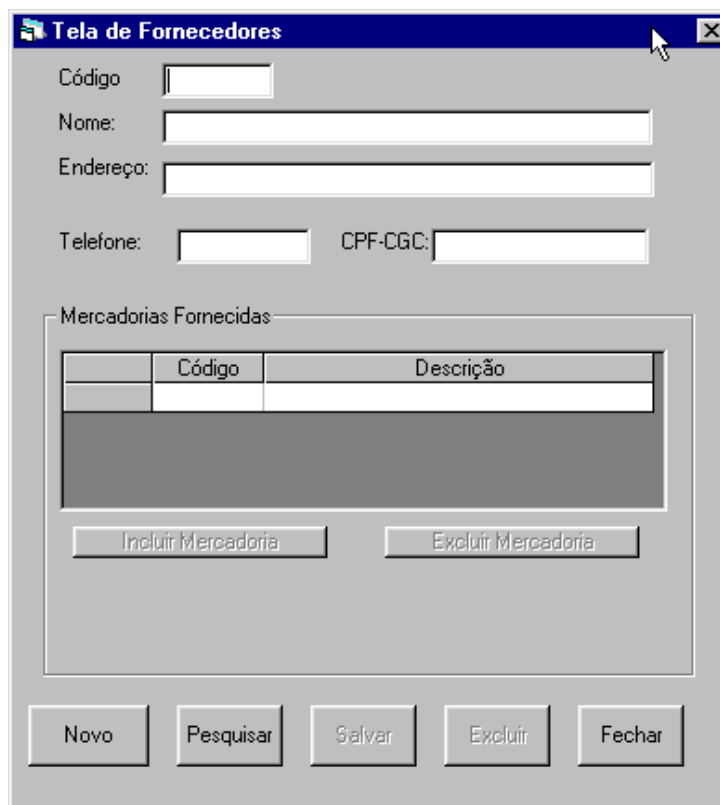


Figura 130 - Exemplo de janela secundária

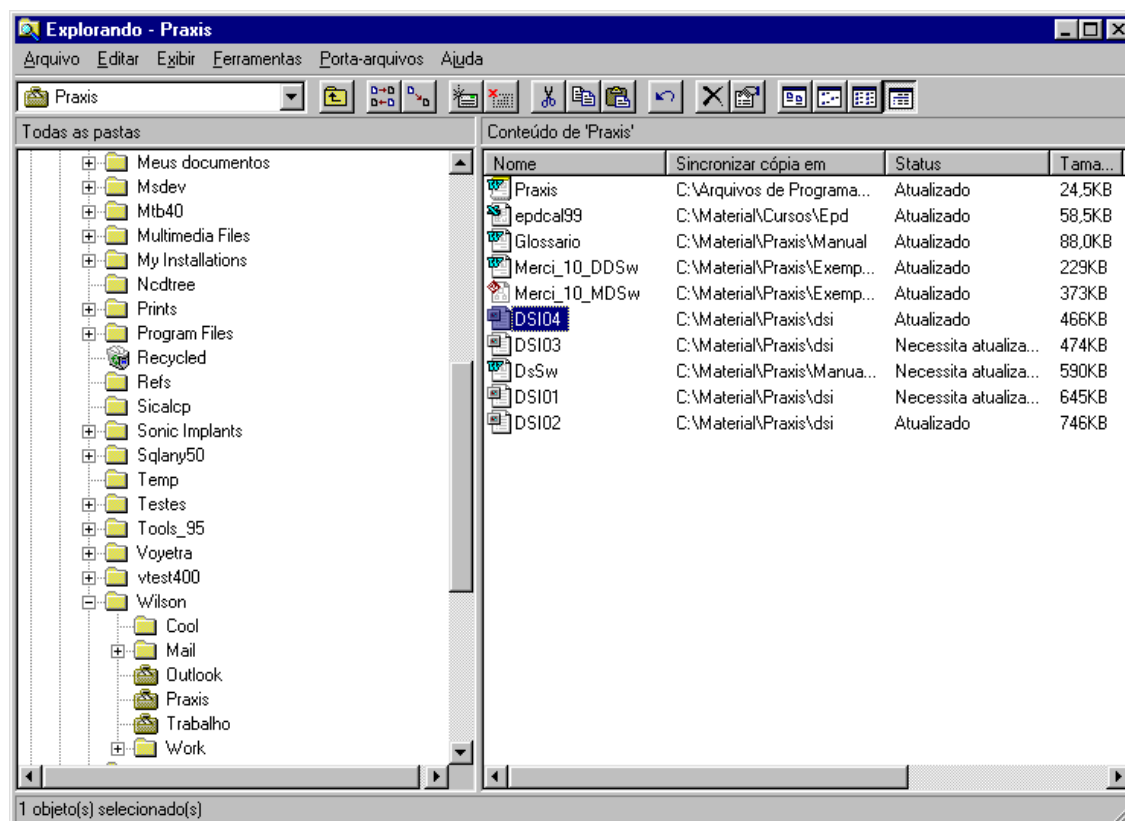


Figura 131 - Exemplo de janela partida

2.1.2 Diretrizes

As seguintes diretrizes são aplicáveis às janelas:

- evitar o excesso de janelas em cada aplicativo;
- salvo razões em contrário, permitir que as janelas sejam reposicionadas e redimensionadas;
- manter a consistência na aparência e no comportamento das janelas, principalmente das janelas primárias;
- usar janelas diferentes para permitir a realização de tarefas independentes diferentes;
- usar janelas diferentes para visões coordenadas diferentes da mesma tarefa (por exemplo, visualização dos mesmos dados sob a forma de tabelas e gráficos diferentes).

2.2 Cardápios

2.2.1 Visão geral

Um cardápio é uma lista de itens da qual o usuário pode fazer uma ou mais seleções. Eles reduzem o volume de memorização requerido; eliminam a digitação, reduzindo erros; e reduzem as necessidades de treinamento. O exagero no uso de cardápios, porém, é prejudicial para a produtividade dos usuários experientes.

2.2.2 Tipos de cardápios

2.2.2.1 Botões de apertar

Em um cardápio de botões de apertar (*push buttons*), as opções são selecionadas por meio de botões separados, que geralmente estão sempre visíveis. Estes botões tendem a ocupar muito espaço, devendo ser restritos a interfaces com poucos comandos.

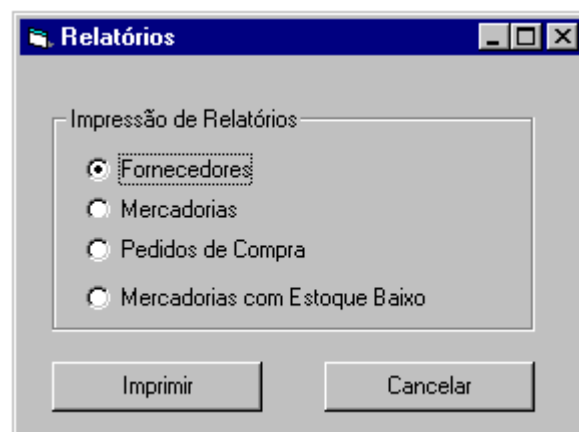


Figura 132 - Exemplo de interface com botões de apertar e botões de opção

As seguintes diretrizes são aplicáveis aos botões de apertar:

- os rótulos devem ser significativos e corresponder às ações que evocam (por exemplo, não usar "Cancelar" quando o objetivo é simplesmente fechar a interface);
- definir um dos botões de apertar como opção padrão, com aparência destacada, escolhendo o comportamento padrão mais adequado ao contexto (por exemplo, "OK" × "Cancelar");

- prover aceleradores e teclas de atalho para os usuários experientes.

2.2.2.2 Botões de opção

Botões de opção ("radio buttons"²³) apresentam um conjunto de opções que são mutuamente excludentes. A opção selecionada é graficamente destacada das demais (Figura 132). Como todas as opções são exibidas simultaneamente – o que ocupa muito espaço de uma janela – esse estilo de interação se aplica apenas a casos em que o número de opções é relativamente pequeno.

2.2.2.3 Botões de checar

Os botões de checar apresentam um conjunto de opções não mutuamente excludentes. As opções selecionadas são representadas visualmente por um sinal como X ou √. Na Figura 133, a interface da Figura 132 foi redesenhada para permitir a impressão conjunta de vários tipos de relatórios. Esse estilo é adequado apenas quando o número de opções é relativamente pequeno, pelo espaço que ocupa na interface.



Figura 133 - Exemplo com botões de checar

2.2.2.4 Cardápios permanentes



Figura 134 - Cardápio permanente

Os cardápios permanentes (*pull-down menus*) são sempre visíveis, sendo geralmente localizados na parte superior da interface (Figura 134). O cardápio é inteiramente exibido quando seu título é

²³ O nome em inglês vem do formato usual, que lembra os botões dos rádios antigos.

selecionado. A seleção é indicada pelo destaque de um item. Geralmente, os cardápios permanentes são usados para acesso aos fluxos principais dos casos de uso.

2.2.2.5 Cardápios instantâneos

Cardápios instantâneos (*pop-up menus*) aparecem na posição corrente do cursor quando o usuário pressiona um botão convencionado do mouse (Figura 135). Eles são geralmente usados para acionar uma operação aplicável ao objeto selecionado. Apenas as funções aplicáveis a esse objeto podem ser acionadas, ajudando o usuário a escolher ações válidas. Além disso, esses cardápios reduzem a movimentação do mouse, tornando a interação mais rápida e menos cansativa.

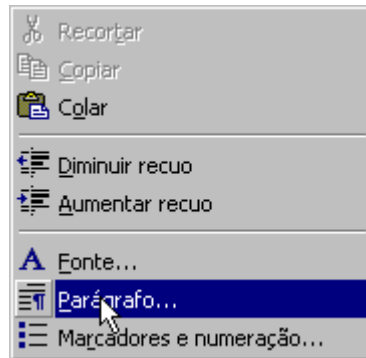


Figura 135 - Exemplo de cardápio instantâneo

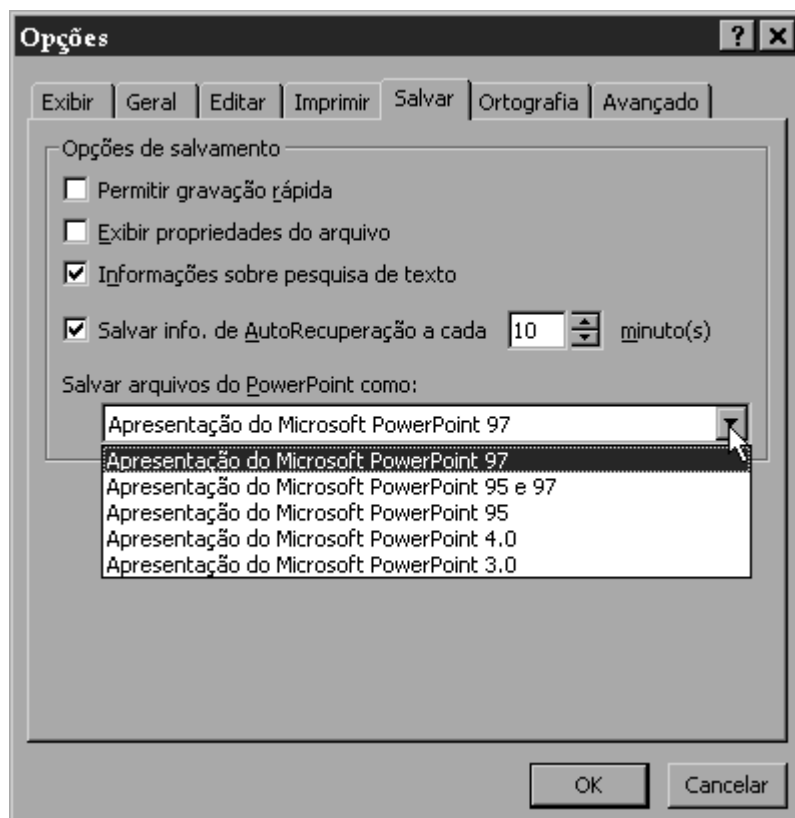


Figura 136 - Exemplo de interface com botões de checar e cardápio de opção

2.2.2.6 Cardápios de opção

Os cardápios de opção (Figura 136) mostram normalmente um único item. Os demais itens, que são mutuamente excludentes, aparecem quando o mouse é pressionado sobre um indicador de expansão. Uma vez selecionado um item dessa lista, ele passa a ser o único exibido. Esse tipo de cardápio ocupa

menos espaço que os botões de opção. Ele é indicado também para realizar campos de um formulário que só podem ser preenchidos por um conjunto limitado de valores.

2.2.2.7 Cardápios em cascata

Nos cardápios em cascata (Figura 137), a seleção de um dos itens faz aparecer um outro cardápio, com itens mais detalhados. Um indicador visual, como um pequeno triângulo, indica que um item representa um cardápio em cascata. Esse tipo de cardápio permite organizar de forma hierárquica cardápios com grande número de opções.

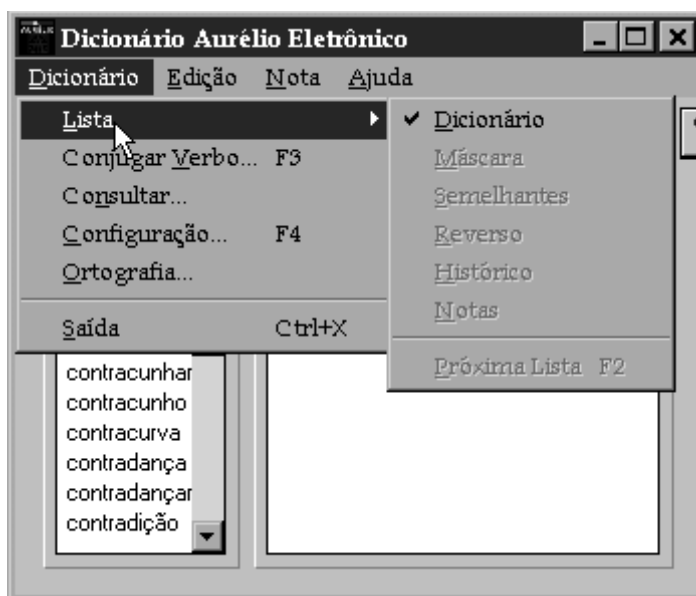


Figura 137 - Cardápio em cascata

2.2.2.8 Cardápios de paleta

Nos cardápios de paleta, as opções (mutuamente exclusivas) são representadas por ícones. Esses cardápios são muito usados em aplicativos gráficos. Os ícones podem representar atributos que se quer aplicar ao objeto corrente, como cores (Figura 138), ou ferramentas (Figura 139), que são arrastadas até a área de trabalho.



Figura 138 - Exemplo de cardápio de paleta de cores

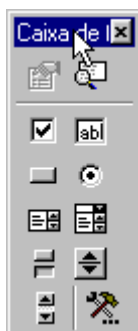


Figura 139 - Exemplo de cardápio de paleta usado como caixa de ferramentas

2.2.2.9 Cardápios de hiperligações

As hiperligações (*hyperlinks*), encontradas em documentos de hipertexto, e muito popularizadas pela WWW, podem ser usadas como itens de cardápio (Figura 140).

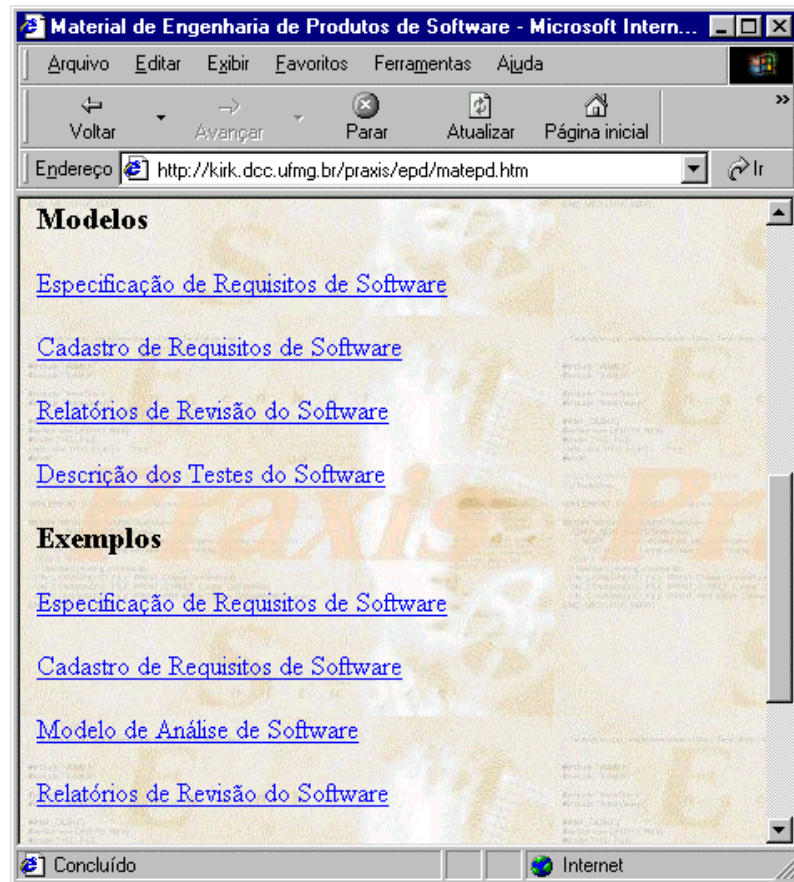


Figura 140 - Exemplo de cardápio de hiperligações

2.2.2.10 Cardápios dinâmicos

Os cardápios dinâmicos (lista de arquivos na parte inferior do cardápio da Figura 141) são alteráveis em tempo de execução. Opções podem ser acrescentadas ou retiradas, dependendo do contexto. Um uso comum é a inclusão de uma lista de documentos recentes.

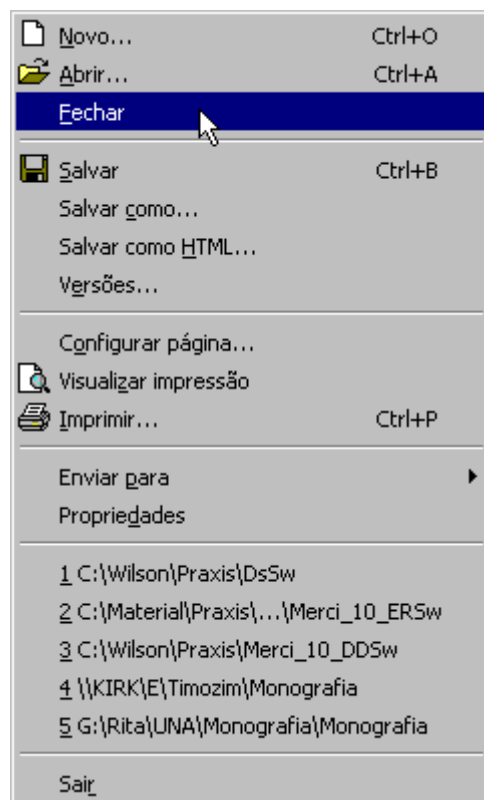


Figura 141 - Exemplo de cardápio com parte dinâmica

2.2.2.11 Mapas de imagem

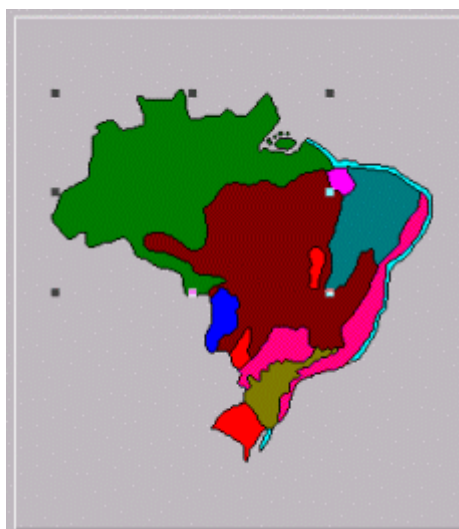


Figura 142 - Exemplo de mapa de imagem

Os mapas de imagem são cardápios nos quais os itens são constituídos por partes de uma imagem. Esses mapas são uma forma eficiente de interação em aplicativos gráficos, sendo também muito utilizados na WWW. Na Figura 142, cada região do mapa representa um bioma²⁴ diferente. Um clique sobre a representação de um bioma leva, nesse aplicativo, a uma página com informação sobre as respectivas aves.

²⁴ Conjunto dos seres vivos de uma área.

2.2.3 *Diretrizes*

As seguintes diretrizes são aplicáveis aos cardápios em geral.

- Obedecer às recomendações do manual de estilo do ambiente.
- Organizar hierarquias de cardápios com base nos casos de uso do produto ou nas tarefas dos usuários. Essas hierarquias devem ter no máximo três a quatro níveis de profundidade, formando-se em cada nível grupos de quatro a oito itens. Algumas exceções são aceitáveis, como doze itens para os meses do ano.
- Agrupar as opções do cardápio em grupos de itens mais relacionados entre si, usando, por exemplo, separadores ou espaços. Separadores devem ser usados com cuidado para não entulhar o cardápio.
- Ordenar os itens de forma consistente, usando uma regra adequada de ordenamento (por exemplo, alfabética, por frequência de uso ou por importância relativa).
- Usar para os itens do cardápio rótulos breves e significativos para o domínio de aplicação. A natureza gramatical desses rótulos deve ser consistente para todos os itens.
- Oferecer recursos aceleradores, como atalhos e macros. Pelo menos as funções de uso mais frequente devem ser acessíveis através de atalhos.

Os seguintes aspectos dos cardápios devem ser usados de forma consistente, de preferência com base no manual de estilo:

- separadores entre grupos;
- indicadores de cardápios em cascata (por exemplo, um pequeno triângulo);
- indicadores de itens que abrem caixas de diálogo (por exemplo, reticências);
- local para exibição de dicas sobre os itens;
- mensagens de erro (quanto ao fraseado e local de exibição).

2.3 **Formulários**

2.3.1 *Visão geral*

Os formulários são interfaces de uso geralmente fácil, que aproveitam analogias com o preenchimento de formulários de papel, usuais em muitos processos manuais do domínio de aplicação. Geralmente seus elementos são permanentemente visíveis, mas alguns formulários podem ter botões que permitem exibir mais detalhes.

Tipo de campo	Característica
Texto livre	Aceita qualquer sequência de um determinado conjunto de caracteres.
Texto validado	Obedece a uma sintaxe específica, cuja validade deve ser verificada pelo código da interface (por exemplo, datas ou CPF). É recomendável que a sintaxe seja indicada de forma visual.
Lista de escolha	Campo que aceita um conjunto discreto e limitado de valores.

Tabela 135 - Tipos de campos de formulário

Os campos (Tabela 135) devem ser preenchidos inicialmente, sempre que possível, com um valor padrão, principalmente quando se tratarem de campos obrigatórios. Os campos obrigatórios devem ser visualmente destacados em relação aos opcionais (por exemplo, através de asteriscos ou agrupamento). As dependências entre campos devem ser automatizadas; por exemplo, "Curso de graduação" pode só ser válido quando outro campo indica que a pessoa tem nível superior.

2.3.2 Diretrizes

As seguintes diretrizes devem ser usadas para se obterem leiaute e conteúdo consistentes e visualmente atraentes:

- título claro e significativo;
- campos ordenados e agrupados logicamente;
- agrupamentos delimitados visualmente;
- separadores e padrões de alinhamento consistentes em todos os formulários.

Quando um processo manual é informatizado, os formulários de papel são um bom ponto de partida para o desenho dos formulários on-line. Entretanto, a conversão entre eles nem sempre deve ser direta. Formulários on-line geralmente oferecem menos espaço útil, mas permitem várias opções não disponíveis em mídia de papel.

Durante a conversão, os formulários existentes devem ser analisados para verificar se:

- existem redundâncias a serem eliminadas;
- existem campos não utilizados que devam ser eliminados;
- todas as informações necessárias estão dispostas em campos próprios;
- há anotações extra nas margens ou no verso do formulário, que podem indicar a necessidade de novos campos;
- há necessidade de ter acesso a outros formulários ou registros para completar a tarefa, o que pode indicar a necessidade de um redesenho maior do conjunto de formulários.

Outras diretrizes para desenho de formulários são as seguintes:

- usar indicações visuais apropriadas para os campos dos formulários (por exemplo, separar subcampos de CEPs ou telefones);
- diferenciar visualmente os campos obrigatórios dos opcionais;

- usar rótulos e abreviações familiares dentro do domínio da aplicação e consistentes de um formulário para outro;
- usar navegação lógica entre campos, permitindo percorrê-los de forma cíclica, através de teclas de seta ou tabulação;
- usar navegação livre dentro de cada campo, através do mouse e de setas;
- suportar edição e correção de campos, destacando as partes incorretas e permitindo o uso de operações de copiar e colar;
- usar mensagens de erro consistentes para indicar campos preenchidos de forma inválida, procurando ser suficientemente específicas para indicar como deve ser o preenchimento correto;
- prover mensagens explicativas sobre as entradas esperadas em cada campo, principalmente naqueles cujo preenchimento não é óbvio para os usuários do produto;
- prover valores padrão nos campos, sempre que possível, preenchendo-os com o valor inicial mais freqüente.

Formulários não devem ser fechados automaticamente após o preenchimento do último campo. O fechamento deve ser feito por um comando explícito. Deve-se permitir que o usuário altere os campos que quiser antes do fechamento. Em alguns casos, pode ser necessário prover meios de salvamento de formulários incompletos, fora do banco de dados definitivo.

2.4 Caixas

2.4.1 *Visão geral*

Muitos tipos de caixas são usados como janelas secundárias ou parte de outras interfaces. Alguns usos comuns envolvem entrada de texto, acionamento de comandos e exibição de mensagens.

2.4.2 *Tipos de caixas*

2.4.2.1 **Caixas de lista**

Uma caixa de lista é uma janela que contém uma lista de opções (Figura 143) cujo número de itens pode ser muito grande. Tipicamente, seu conteúdo é variável de forma dinâmica. Geralmente, barras de rolagem ajudam a exibir grandes listas em área limitada. A pesquisa na lista pode ser feita por apontamento ou pela digitação dos primeiros caracteres do texto de um item.

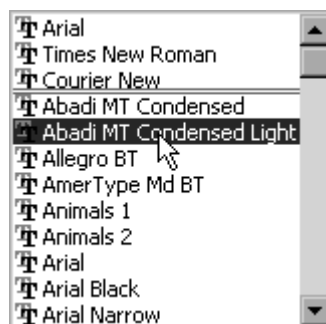


Figura 143 – Exemplo de caixa de lista

2.4.2.2 Caixas de entrada

Caixas de entrada são muito utilizadas para entrada e edição de texto (Figura 144). A caixa pode permitir uma única linha ou múltiplas linhas, e pode ter barras de rolagem para acomodar textos maiores.

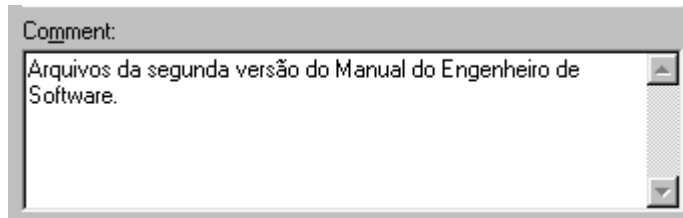


Figura 144 – Caixa de entrada

2.4.2.3 Caixas de mensagem

Caixas de mensagem são usadas para informar o estado de operações, mostrar avisos e apresentar opções para o usuário. Geralmente a interação é completada através da escolha de uma ação determinada por um conjunto muito pequeno de botões. Caixas de mensagem modais são usadas para exigir uma decisão crítica do usuário (Figura 145).

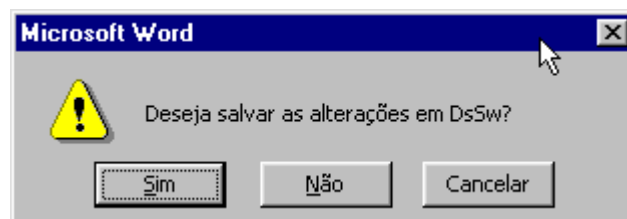


Figura 145 - Exemplo de caixa de mensagem modal

2.4.2.4 Caixas de diálogo

Caixas de diálogo agrupam vários controles para realizar um conjunto de operações correlatas. Geralmente, as caixas de diálogo são acessíveis através de determinados itens de cardápio, devidamente assinalados. Podem ser modais (Figura 146) ou não modais (Figura 147).

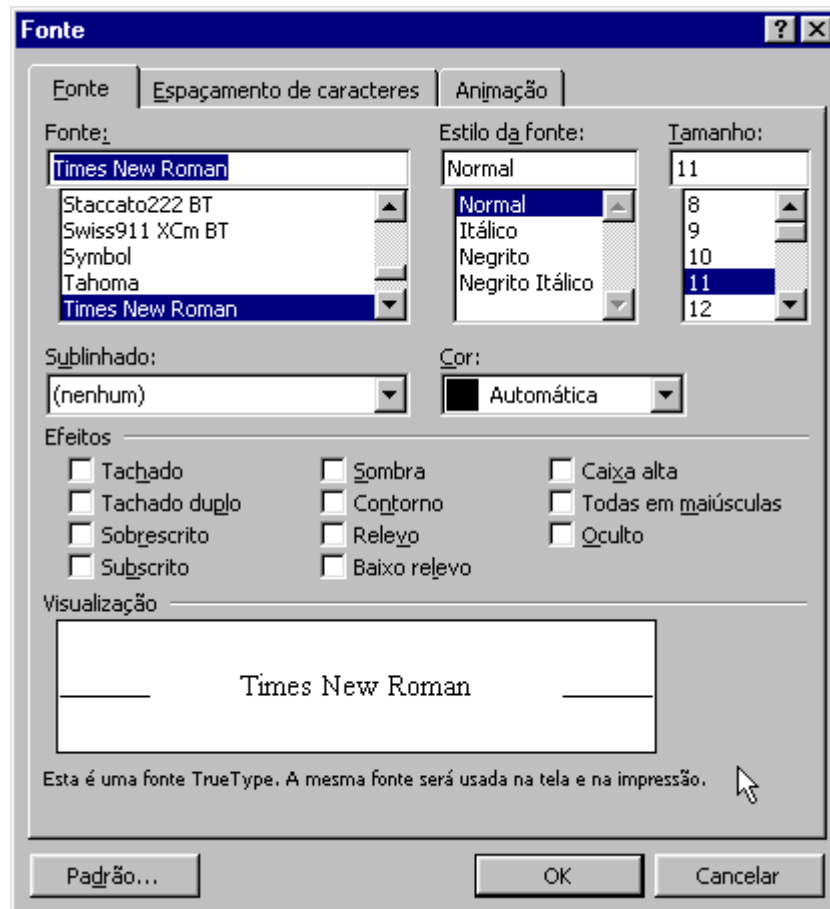


Figura 146 - Exemplo de caixa de diálogo modal

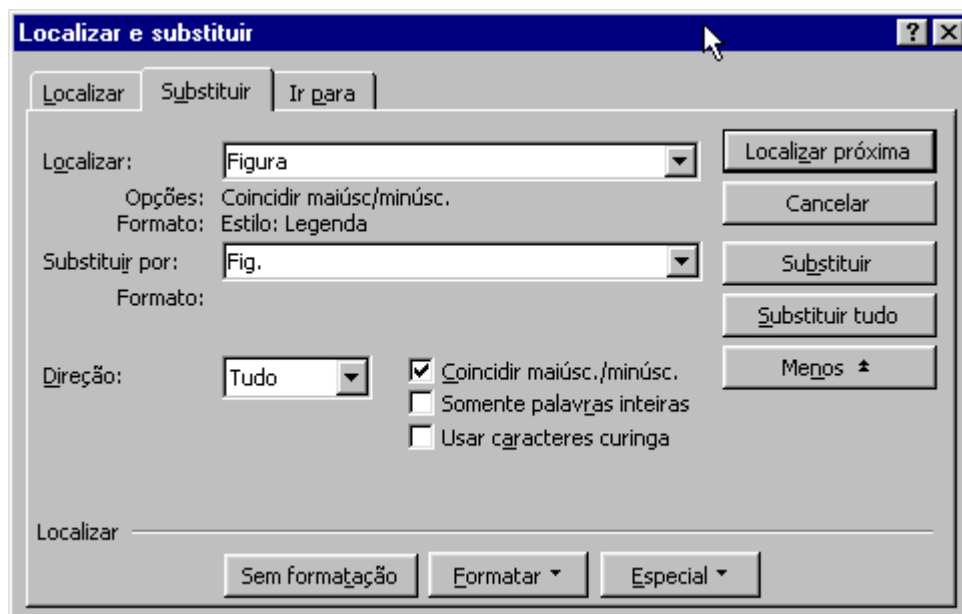


Figura 147 - Exemplo de caixa de diálogo não modal

2.4.3 Diretrizes

As seguintes diretrizes são aplicáveis às caixas:

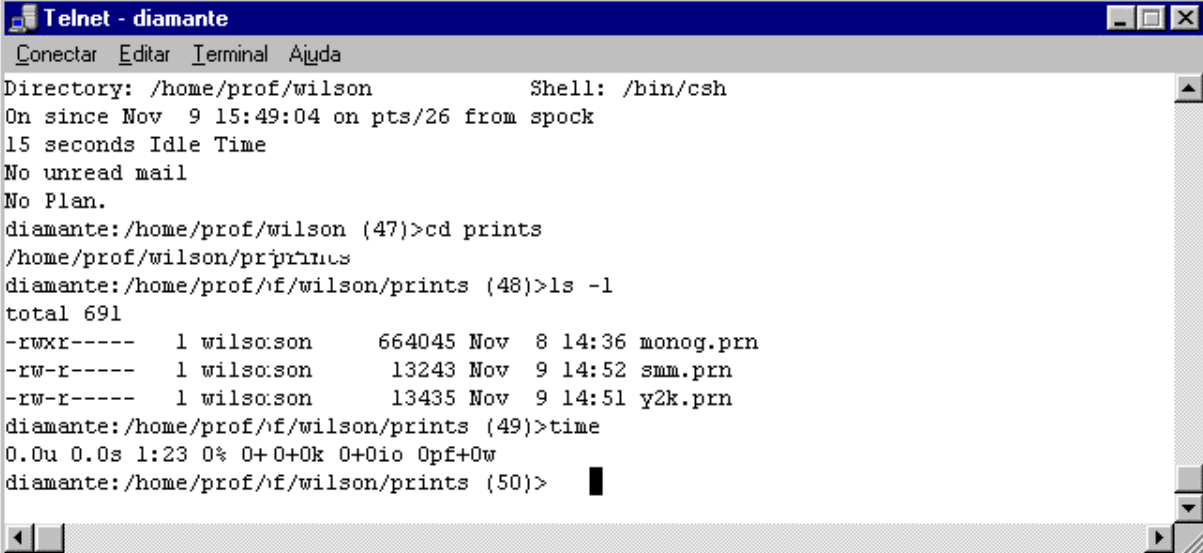
- usar instruções breves, simples e claras, especialmente para caixas de diálogo modais;

- usar mensagens concisas, claras, explicativas e expressas na linguagem dos usuários;
- usar agrupamentos lógicos e ordenamentos de objetos dentro das caixas, para indicar grupos de objetos correlatos;
- usar indicações visuais para delinear agrupamentos dentro das caixas, como separadores e espaços em branco;
- manter o leiaute consistente e visualmente atraente, conservando de uma caixa para outra as posições e alinhamentos dos elementos;
- destacar visualmente as opções padrão, permitindo que possam ser selecionadas de forma abreviada;
- indicar as opções de cardápio que levam a caixas de mensagem (geralmente, através de reticências);
- deixar a remoção das caixas sob controle do usuário, exceto caixas puramente informativas, que podem ser retiradas automaticamente depois de certo tempo.

2.5 Linguagens de comando

2.5.1 Visão geral

Linguagens de comando (Figura 148) usam seqüências alfanuméricas que representam comandos, parâmetros e opções. Elas são eficazes para os usuários freqüentes e experientes, mas exigem muito treinamento e apresentam maiores taxas de erros, pois o usuário tem que se lembrar precisamente da sintaxe empregada.



```

Telnet - diamante
Conectar Editar Terminal Ajuda
Directory: /home/prof/wilson      Shell: /bin/csh
On since Nov  9 15:49:04 on pts/26 from spock
15 seconds Idle Time
No unread mail
No Plan.
diamante:/home/prof/wilson (47)>cd prints
/home/prof/wilson/prints
diamante:/home/prof/wilson/prints (48)>ls -l
total 691
-rwxr----- 1 wilson      664045 Nov  8 14:36 monog.prn
-rw-r----- 1 wilson      13243 Nov  9 14:52 smm.prn
-rw-r----- 1 wilson      13435 Nov  9 14:51 y2k.prn
diamante:/home/prof/wilson/prints (49)>time
0.0u 0.0s 1:23 0% 0+0+0k 0+0io 0pf+0w
diamante:/home/prof/wilson/prints (50)>
  
```

Figura 148 - Exemplo de linguagem de comando

2.5.2 Diretrizes

As seguintes diretrizes são aplicáveis às linguagens de comando:

- as regras de formação dos comandos devem ter estilo consistente: por exemplo, iniciar o comando por um verbo, seguido de zero ou mais modificadores e zero ou mais objetos;

- se os parâmetros forem posicionais, a sua ordem deve ser consistente de um comando para outro;
- os nomes devem ser específicos, significativos e distinguíveis, baseados em terminologia consistente;
- a abreviação de comandos deve obedecer a regras consistentes;
- a correção de erros de digitação deve ser fácil e rápida, sem que seja preciso redigitar toda a linha;
- os usuários experientes devem poder escrever macros para abreviar seqüências de comandos de uso freqüente.

2.6 Interfaces pictóricas

2.6.1 *Visão geral*

Nos ambientes modernos, todas as interfaces de usuário são de natureza gráfica, mesmo que a informação exibida seja predominantemente textual. Nas interfaces pictóricas, a própria informação fornecida pelo sistema é de natureza intrinsecamente gráfica. Essas interfaces são particularmente adequadas para o tratamento de grande volume de informação. As aplicações de interfaces pictóricas incluem visualização de dados, bases de dados visuais e sistemas de multimídia e hipermídia. Um gráfico de planilha é um exemplo de uma das interfaces pictóricas mais comuns.

2.6.2 *Diretrizes*

As seguintes diretrizes são aplicáveis às interfaces pictóricas:

- usar analogias familiares com conceitos e objetos do mundo real;
- manter a representação visual o mais simples possível;
- mostrar visões complementares do mesmo objeto visual (por exemplo, tabelas e gráficos);
- usar a cor com cuidado e de forma significativa;
- usar com cuidado recursos de processamento pesado, como grandes volumes de áudio, vídeo e multimídia.

A codificação pictórica (Tabela 136) representa os elementos de informação através de vários indicadores visuais. É recomendável ficar dentro dos seguintes limites de codificação, dentro de cada interface pictórica:

- 6 tamanhos;
- 4 intensidades;
- 24 ângulos;
- 15 formas geométricas.

Tipo	Definição	Elementos usuais
Nominativa	Indica diferentes tipos de coisas.	Formas, fontes, estilos, hachuras.
Ordinal	Indica algum tipo de ordenação.	Tamanhos de texto, espessuras de linhas, densidade de hachuras.
Métrica	Indica algum tipo de medida.	Posição em escalas, comprimentos, ângulos, áreas, parâmetros de cor.

Tabela 136 - Tipos de codificação pictórica

2.7 Outros estilos de interação

Outros estilos de interação incluem os seguintes:

- **Telas de toque** - duráveis e simples de usar, são adequadas para sistemas de serviço ao público, como quiosques de informação.
- **Síntese de voz** - usada como alternativa para deficientes visuais ou para avisos ao público.
- **Teclados telefônicos** - usados para sistemas de atendimento telefônico, sendo combinado com síntese de voz.
- **Reconhecimento de voz** - usado geralmente para comandos em situações em que as mãos estão ocupadas. Com a evolução da tecnologia, tende a ser usado em muitas aplicações que atualmente são baseadas em teclados e janelas.

Descrição de Desenho de Software

1 *Visão geral*

1.1 Introdução

A Descrição do Desenho de Software (DDSw) resulta do fluxo de Desenho, parte do processo Praxis. Esse fluxo tem como insumo a Especificação de Requisitos do Software, que descreve, de forma detalhada, um conjunto de requisitos que define uma solução implementável para um problema. A Descrição do Desenho do Software (DDSw) estabelece a estrutura com que o produto deverá ser implementado para satisfazer aos requisitos. Essa descrição inclui as seguintes partes:

- o desenho das interfaces de usuário, que descreve a visão que os usuários terão do produto e a maneira como os usuário interagirão com o produto;
- o desenho interno, que descreve as partes lógicas e físicas do modelo, suas interconexões e a maneira como interação entre si e com sistemas externos;
- o desenho das liberações, que descreve como a fase de Construção é particionada em liberações a fase de Construção.

A Descrição do Desenho do Software tipicamente evolui durante as liberações, à medida em que se completa o desenho detalhado; não existe um modelo de desenho separado para cada liberação. Após a cada iteração, a versão correspondente da Descrição do Desenho do Software será guardada em uma linha de base da biblioteca de Gestão de Configurações.

Ao longo de sua evolução, a DDSw deve manter vários níveis de consistência:

- interna, entre as diferentes visões e níveis de detalhamento do desenho proposto;
- para trás, com cada entidade de desenho colaborando para a realização de um dos casos de uso do sistema;
- para a frente, com cada entidade de desenho sendo implementada através de uma construção de código.

Define-se aqui a estrutura para a Descrição do Desenho do Software. Nenhuma seção deve ser omitida, mantendo-se a estrutura de numeração aqui indicada; seções não pertinentes ao projeto em questão devem ser indicadas com a expressão “Não aplicável”.

Define-se também um roteiro para revisão da Descrição do Desenho do Software. Esse roteiro define os passos que devem ser seguidos na realização dessas revisões, e aplica-se principalmente à revisão técnica que o Praxis requer para o final da iteração de Desenho Inicial, na fase de Construção. Esse roteiro inclui listas de conferência para a verificação do Modelo de Desenho do Software e das interfaces de usuário.

1.2 Referências

As principais referências deste documento são:

IEEE. *IEEE Std. 1016 – 1987. IEEE Recommended Practice for Software Design Descriptions*, in [IEEE94].

IEEE. *IEEE Std. 1016 – 1987. IEEE Guide to Software Design Descriptions*, in [IEEE94].

Outras referências importantes são [Booch94], [Booch96], [Booch+97], [Booch+99], [Jacobson94], [Jacobson+94a], [Jacobson+99], [Love93], [Quatrani98], [Rumbaugh+99] e [White94].

1.3 Convenções de preenchimento

A página do título da DDSw deve incluir os seguintes elementos:

- nome do documento;
- identificação do projeto para o qual a documentação foi produzida;
- nomes dos autores e das organizações que produziram o documento;
- número de revisão do documento;
- data de aprovação;
- assinaturas de aprovação;
- lista dos números de revisão e datas de aprovação das revisões anteriores.

Recomenda-se incluir, logo após a página de título, um sumário (obrigatório para documentos de mais de oito páginas) e uma lista de ilustrações.

2 Preenchimento da Descrição do Desenho do Software

2.1 Introdução (DDSw-1)

2.1.1 *Objetivos deste documento*

Descreve-se aqui o propósito da DDSw, especificando o público **desse documento**.

Este documento tem por finalidade a descrição do desenho do software Merci versão 1.0, bem como o planejamento de suas liberações executáveis. O público alvo consiste dos desenvolvedores da United Hackers e do representante dos usuários, por parte do cliente.

Tabela 137 - Exemplo de Objetivos deste documento

2.1.2 *Escopo do produto*

Descreve-se aqui, de forma sintética, o escopo do produto desenhado. Geralmente, essa subseção resume os itens correspondentes da Especificação dos Requisitos do Software.

A DDSw deve ser consistente com a Especificação dos Requisitos do Software e outros possíveis documentos de nível mais alto.

Apoio informatizado ao controle de vendas, de compras, de fornecedores e de estoque da mercearia Pereira & Pereira Comercial Ltda.

Tabela 138 - Exemplo de Missão do produto

O Merci não fará vendas parceladas e só receberá dinheiro ou cheque.
 O Merci só fará a Emissão de Nota Fiscal durante a Operação de Venda.
 O Merci não fará um cadastro de clientes da mercearia Pereira & Pereira Comercial Ltda.
 O preço de venda deverá ser calculado pela mercearia Pereira & Pereira Comercial Ltda. e informado ao Merci.
 Atividades como backup e recuperação das bases de dados do sistema ficam a cargo da administração de dados e não serão providas no Merci.
 O Merci não terá ajuda on-line.
 Não haverá tolerância a falhas no Merci.

Tabela 139 - Exemplo de Limites do produto

Número de ordem	Benefício	Valor para o cliente
1	Agilidade na compra e venda de mercadorias.	Essencial
2	Conhecimento do mercado de fornecedores visando a uma melhor conjugação de qualidade, preço e prazo.	Essencial
3	Diminuição de erros na compra e venda de mercadorias.	Essencial
4	Economia de mão-de-obra.	Essencial
5	Eliminação da duplicidade de pedidos de compra.	Essencial
6	Qualidade na emissão da Nota Fiscal e Ticket de Venda em relação à emissão manual.	Essencial
7	Diminuição do custo de estocagem.	Desejável
8	Identificação de distorções entre o quantitativo vendido e o ainda existente no estoque.	Desejável
9	Maior agilidade nas decisões de compra.	Desejável

Tabela 140 - Exemplo de Benefícios do produto

2.1.3 *Materiais de referência*

Apresenta-se aqui a informação necessária para que todas as fontes de dados citadas na DDSw possam ser recuperadas, caso necessário. Para isso, essa subseção deve:

- fornecer uma lista completa de todos os documentos referenciados na DDSw;
- identificar cada documento de acordo com os padrões bibliográficos usuais, indicando-lhes pelo menos o nome, o número, a data e a organização que o publicou ou que pode fornecê-lo.

No caso de outras fontes, tais como atas de reunião, memorandos etc., a referência deve indicar como obtê-las.

Número de ordem	Tipo do material	Referência bibliográfica
1	Documentação de desenvolvimento	Especificação dos Requisitos do Software - Projeto Merci Versão 1.0 Revisão 1. RT 002-99, United Hackers Ltda.
2	Documentação de desenvolvimento	Plano de Desenvolvimento do Software - Projeto Merci Versão 1.0 Revisão 1. RT 003-99, United Hackers Ltda.
3	Livro	Ivar Jacobson, James Rumbaugh e Grady Booch. <i>Unified Software Development Process</i> . Addison-Wesley, Reading -MA, 1999.
4	Livro	Grady Booch, Ivar Jacobson e James Rumbaugh. <i>The Unified Modeling Language User Guide</i> . Addison-Wesley, Reading -MA, 1999.
5	Livro	W. S. Humphrey. <i>Managing the Software Process</i> . Addison-Wesley, Reading -MA, 1990.
6	Livro	IEEE. <i>IEEE Standards Collection - Software Engineering</i> . IEEE, New York - NY, 1994.

Tabela 141 - Exemplo de Materiais de referência

2.1.4 Definições e siglas

Inclui-se aqui a definição de todas as siglas, abreviações e termos usados na DDSw. Deve-se supor que a DDSw será lida tanto por desenvolvedores quanto por usuários, e por isso deve conter as definições relevantes, tanto de termos da área de aplicação quanto de termos de informática usados na DDSw e que não sejam do conhecimento do público em geral.

Número de ordem	Termo	Definição
1	Cadastro de Compras	Cadastro de pedido de compra de mercadoria da mercearia.
2	Cadastro de Fornecedores	Cadastro dos fornecedores de mercadorias da mercearia.
3	Cliente da Mercearia	Pessoa que procura a mercearia para efetuar suas compras.
4	Nota Fiscal	Documento exigido pela legislação fiscal para fins de fiscalização.
5	Ticket de Venda	Um relatório impresso pelo Merci que exhibe e totaliza os itens referentes a uma venda efetuada.

Tabela 142 - Exemplo de Definições e siglas

2.1.5 Visão geral deste documento

Descreve-se aqui o que o restante da DDSw contém, indicando sua estrutura básica. Caso seja omitida nesta DDSw particular alguma seção prevista neste padrão, a omissão deve ser aqui justificada.

Na parte 2 é detalhado o desenho das interfaces com os usuários, mostrando-se a realização dos casos de uso em termos dessas interfaces, assim como a estrutura dos componentes de interface.

Na parte 3 é detalhado o desenho interno de alto nível do produto, mostrando a estratégia de arquitetura, os diagramas lógicos de nível de desenho e os diagramas físicos.

Na parte 4 é descrito o plano das liberações executáveis. Estão previstas inicialmente duas liberações.

Na parte 5 são anexadas as especificações dos elementos do Modelo do Desenho do Software.

Tabela 143 - Exemplo de Visão geral deste documento

2.2 Desenho das interfaces de usuário (DDSw-2)

2.2.1 Aspectos gerais de processo

2.2.1.1 Caracterização dos usuários

Descrivem-se aqui os aspectos do perfil dos usuários que influenciaram o desenho das interfaces de usuário do produto. Essa informação deve ser consistente com a subseção 2.3 Características dos usuários da Especificação dos Requisitos do Software, mas tipicamente apresenta um nível maior de detalhes, resultante dos estudos de usabilidade do produto.

Número de ordem	Atores	Permissão de acesso	Frequência de uso	Nível de instrução	Proficiência na aplicação	Proficiência em informática
1	Caixeiro	Operação de Venda e Emissão de Nota Fiscal.	Diário em horário comercial	1º Grau	Operacional	Aplicação
2	Gerente	Abertura do Caixa, Fechamento do Caixa, Gestão de Usuários.	Diário	2º Grau	Completa	Aplicação Windows 95
3	Gestor de Compras	Gestão de Mercadorias, Emissão de Relatórios, Gestão de Fornecedores e Gestão de Compras.	Diária	3º grau	Completa	Aplicação Windows 95
4	Gestor de Estoque	Gestão Manual de Estoque.	Diário	1º Grau	Operacional	Aplicação

Tabela 144 – Exemplo de Caracterização dos usuários

2.2.1.2 Participação dos usuários no desenho das interfaces

Descreve-se aqui como os usuários participaram do desenho dessas interfaces.

Usuários de nível gerencial participaram de sessões de desenho participativo (JAD) das interfaces de usuário. Foram feitos testes de usabilidade com operadores de caixa, comparando-se a produtividade conseguida com um protótipo do produto com a produtividade da operação manual. Em versões futuras, deverão ser usados dados recolhidos na operação real.

Tabela 145 – Exemplo de Participação dos usuários no desenho da interface

2.2.2 Aspectos gerais de produto

2.2.2.1 Estrutura estática

Descrivem-se aqui as relações estáticas entre as classes que implementam as interfaces de usuário. A estrutura estática descreve, por exemplo, como interfaces mães contêm interfaces filhas, ou as estruturas de pastas de interfaces implementadas como páginas da Web.

Indicar quais componentes complexos são utilizados, dentro de quais interfaces. Em casos mais simples, essa estrutura pode ser descrita de forma textual. Em casos mais complexos, pode ser útil incluir um diagrama UML que mostre os relacionamentos de agregação existentes entre as classes de interface de usuário.

A Tela Principal é um quadro MDI. Todas as demais interfaces são implementadas como formulários Visual Basic, filhos da Tela Principal. Excetuam-se as seguintes interfaces, que são formulários independentes da Tela Principal:

- tela de Apresentação, que é um formulário do tipo Splash Screen;
- tela de Login.

Tabela 146 – Exemplo de Estrutura estática

2.2.2.2 Estrutura dinâmica

Descreve-se aqui como é possível navegar (transferir controle) entre os objetos da interface de usuário. Usar de preferência representações gráficas, como grafos, em que os nodos representam os componentes e os arcos representam os caminhos de navegação. Na UML, a estrutura dinâmica pode ser representada por um diagrama de colaboração entre os objetos representativos das interfaces de usuário, em que apareçam apenas as mensagens que implementam navegação entre interfaces.

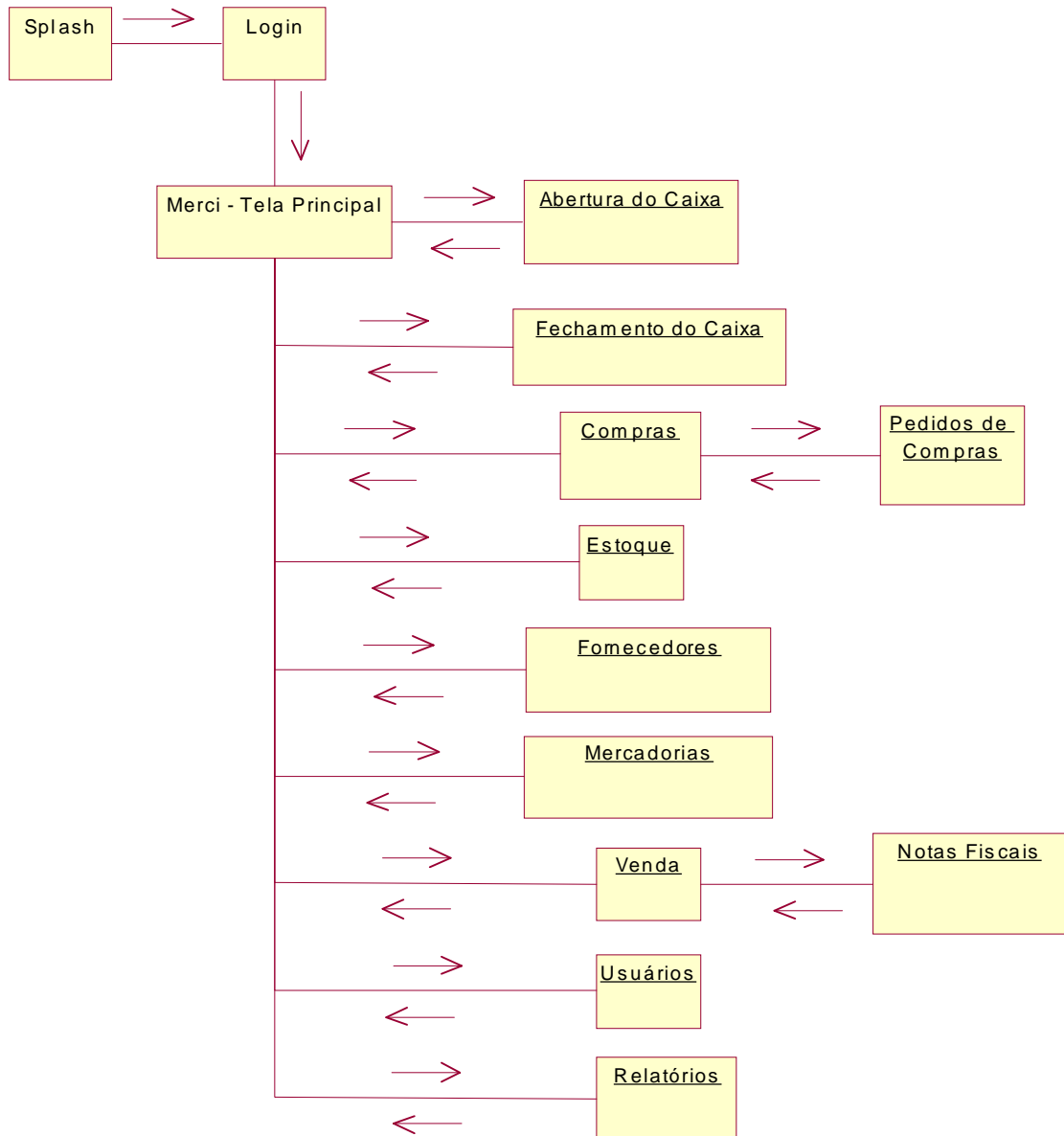


Figura 149 – Exemplo de Estrutura dinâmica

2.2.2.3 Funções do produto

Listam-se aqui os casos de uso implementados através desse desenho. Normalmente, essa lista é herdada da Especificação dos Requisitos, mas pode haver modificações.

Número de ordem	Caso de uso	Descrição
1	Abertura do Caixa	Passagem para o Modo de Venda, liberando assim o caixa da mercearia para a Operação de Venda. O Gerente da mercearia deve informar o valor inicial desse caixa.
2	Emissão de Nota Fiscal	Emissão de Nota Fiscal para o cliente da mercearia (extensão da Operação de Venda).
3	Emissão de Relatórios	Emissão de relatórios com as informações das bases de dados do Mercê.
4	Fechamento do Caixa	Totalização das vendas do dia e mudança para o Modo de Gestão.
5	Gestão de Fornecedores	Processamento de inclusão, exclusão e alteração de fornecedores.
6	Gestão de Mercadorias	Processamento de inclusão, exclusão e alteração de mercadorias.
7	Gestão de Pedidos de Compra	Processamento de inclusão, exclusão e alteração de pedidos de compra de mercadorias.
8	Gestão de Usuários	Controle de usuários que terão acesso ao Mercê.
9	Gestão Manual de Estoque	Controle manual de entrada e saída de mercadorias.
10	Operação de Venda	Operação de venda ao cliente da mercearia.

Tabela 147 - Exemplo de Funções do produto

2.2.2.4 Tratamento dos erros do usuário

Descreve-se aqui como serão tratados os erros dos usuários. A descrição deve incluir tanto a filosofia geral de tratamento quanto os casos específicos relevantes.

O tratamento dos erros cometidos pelo usuário e dos erros do sistema é feito através de mensagens explicativas. Alguns desses erros podem ser corrigidos em tempo de execução e não impedem que o usuário prossiga com o seu trabalho. Erros que não podem ser corrigidos pela aplicação (como falhas de sistema operacional) exigem que o sistema seja reiniciado. Toda ação potencialmente destrutiva exige confirmação do usuário.

Tabela 148 – Exemplo de Tratamento dos erros do usuário

2.2.2.5 Tratamento da ajuda ao usuário

Descreve-se aqui como será tratada a ajuda on-line ao usuário. A descrição deve incluir a filosofia geral de tratamento e a forma de organização da ajuda on-line.

Será disponibilizada ajuda on-line para o usuário. O acesso à ajuda on-line pode ser feito a qualquer momento durante a execução do sistema, através da tecla F1. A ajuda é dependente de contexto, isto é, serão mostradas explicações de acordo com a situação em que ela foi solicitada, permitindo-se navegação para tópicos relacionados ou não. A ajuda on-line descreverá as funções de todos os elementos da interface e o fluxo de execução das funções do produto.

Tabela 149 – Exemplo de Tratamento da ajuda ao usuário

2.2.2.6 Modelo mental do produto

Descreve-se aqui o modelo mental adotado para desenho das interfaces de usuário. Esse modelo explica as metáforas fundamentais escolhidas para o desenho das interfaces.

A maioria das interfaces de usuário usa metáforas de formulário. O formulário on-line geralmente lembra o formulário de papel usado nos processos de negócio manuais. Toda informação que possa vir a ser impressa é exibida on-line previamente.

Tabela 150 – Exemplo de Modelo mental do produto

2.2.2.7 Convenções gerais utilizadas

Descrever as convenções adotadas no desenho e que se aplicam a toda a interface de usuário. São exemplos típicos:

- regras para exibição de mensagens;
- mecanismos de navegação;
- formas de distinção entre campos opcionais e obrigatórios.

Número de ordem	Tipo de convenção	Descrição da convenção
1	Exibição de mensagens de erro	As mensagens de erro serão exibidas em caixas de mensagem.
2	Exibição de mensagens de sucesso	As mensagens de sucesso serão exibidas na barra de status.
3	Mecanismo de navegação	A navegação entre campos de um formulário é feita através da tecla TAB e SHIFT+TAB. A navegação se dará somente entre os campos que estiverem habilitados no momento.
4	Gráfica	Os campos que não estiverem disponíveis estarão na cor cinza e desabilitados, não sendo possível navegar até eles.

Tabela 151 – Exemplo de Convenções gerais utilizadas

2.2.3 Componentes das interfaces de usuário

São descritos aqui todos os componentes das interfaces de usuário. Essa lista deve ser consistente com as classes e objetos representativos desses componentes, presentes na seção de desenho interno. A descrição dos componentes é semelhante à descrição dos requisitos de interface de usuário, contidos na Especificação dos Requisitos do Software, mas a informação aqui apresentada corresponde ao que realmente será implementado. Componentes que são herdados do ambiente de desenvolvimento só devem ser mostrados se isso for relevante para o entendimento do desenho.

Normalmente, devem ser incluídos os seguintes elementos:

- uma imagem da interface, se possível capturada de uma execução real;
- um diagrama de estados, com pelo menos um estado para cada configuração possível de habilitação de comandos;
- uma descrição do relacionamento com outras interfaces, que deve ser consistente com a estrutura dinâmica das interfaces;
- uma lista dos campos de dados da interface;
- uma lista dos botões de ação ou controles equivalentes da interface;
- observações.

Tela de Fornecedores

Código:

Nome:

Endereço:

Telefone: CPF-CGC:

Mercadorias Fornecidas

Código	Descrição
<input type="text"/>	<input type="text"/>

Figura 150 - Exemplo de Imagem de interface de usuário

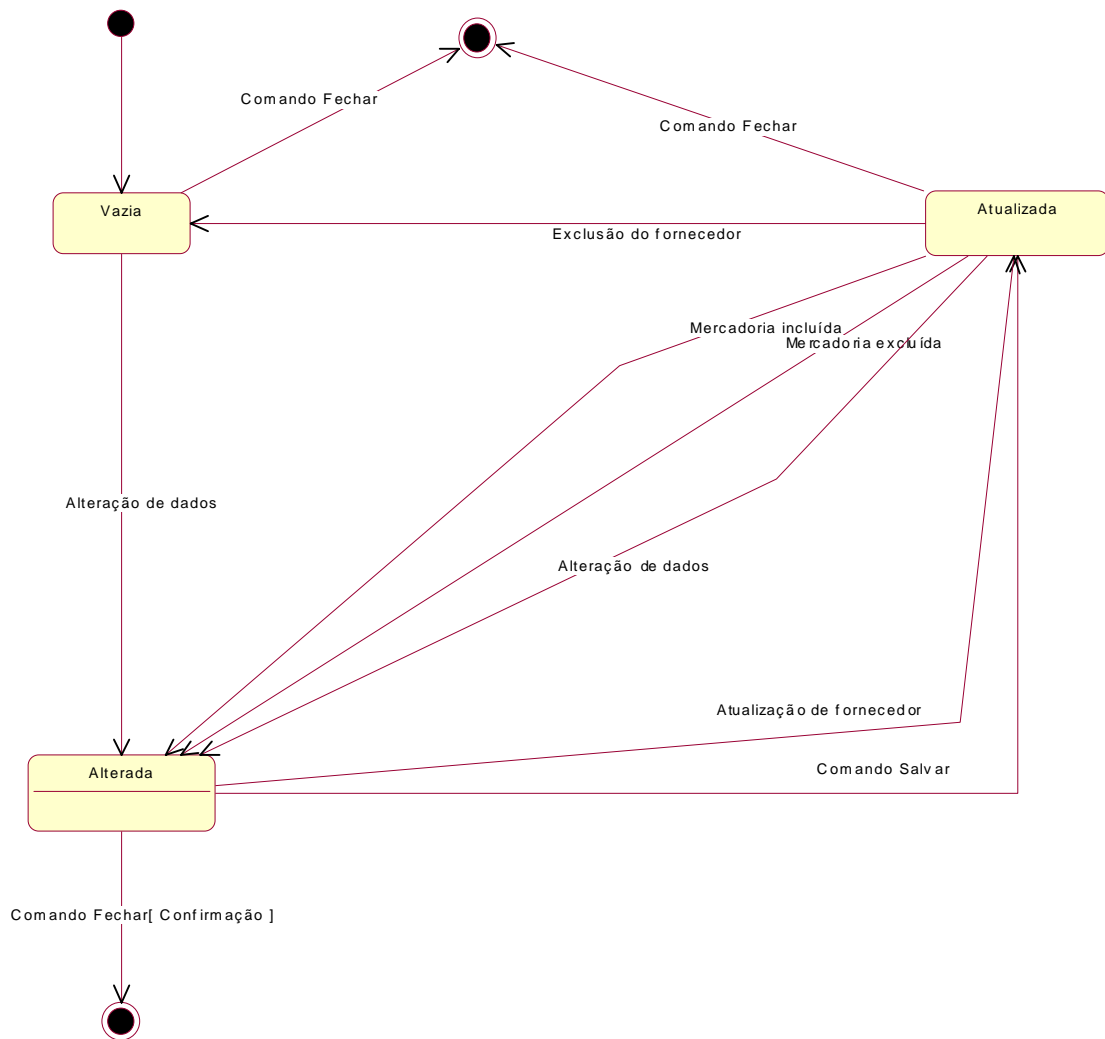


Figura 151 - Exemplo de Diagrama de estados de interface de usuário

Essa interface é ativada a partir da Tela Principal.
O botão Fechar retorna à Tela Principal.

Tabela 152 - Exemplo de Relacionamento com outras interfaces

Diagramas de estado devem ser usados quando o comportamento da interface não for óbvio. Tipicamente, interfaces com três ou mais botões de ação requerem diagramas de estado. O diagrama de estados deve obedecer normalmente às convenções da UML. O diagrama deve ser incluído no Modelo de Desenho do Software, amarrado à classe representativa da interface.

Número	Nome	Valores válidos	Formato	Tipo	Restrições
1	Código	Maior que 0.	Até 6 dígitos.	Número inteiro	Obrigatório / alterável.
2	Nome	-	Até 30 caracteres.	Texto	Obrigatório / alterável.
3	Endereço	-	Até 30 caracteres.	Texto	Obrigatório / alterável.
4	Telefone	-	Até 10 dígitos.	Número inteiro	Obrigatório / alterável.
5	CPF-CGC	CPF ou CGC válidos.	Até 15 dígitos.	Número inteiro	Obrigatório / alterável.
6	Código (quadro Mercadorias Fornecidas)	Maior que 0.	Até 6 dígitos.	Número inteiro	Preenchido pelo Merci.
7	Descrição (quadro Mercadorias Fornecidas)	-	Até 30 caracteres.	Texto	Preenchido pelo Merci.
8	Código da Nova Mercadoria	Maior que 0.	Até 6 dígitos.	Número inteiro	Obrigatório quando visível.

Tabela 153 – Exemplo de Campos

A lista dos campos deve detalhar todos os campos requeridos na interface. Essa lista é herdada da Especificação de Requisitos, mas podem acontecer alterações resultantes de detalhes de desenho. Só é necessário atualizar também a Especificação de Requisitos quando a alteração significar uma mudança de funcionalidade do produto.

Para cada campo, devem constar:

- **número;**
- **nome** para o usuário - nome exibido do campo;
- **valores válidos** - descrição da faixa de valores válidos, que deverá ser usada para validação do campo pelo código da classe da interface;
- **formato** - descrição do formato de entrada do campo, que será obrigatoriamente validado e interpretado pelo código da classe da interface;
- **tipo** para o usuário - tipo abstrato dos dados do campo, na visão do usuário;
- **restrições** - por exemplo, opcional, alterável, calculado etc.

Note-se que o nome para o usuário nem sempre coincide com o respectivo nome interno. Esse é o nome do atributo da classe correspondente, que é limitado pelas restrições da linguagem de implementação, e será documentado na respectiva especificação da classe. O tipo para o usuário também deverá ser mapeado em um tipo interno adequado, limitado pelas restrições da linguagem de implementação. As restrições devem ser consistentes com o diagrama de estados.

Quanto aos comandos, deve-se incluir:

- uma lista dos comandos disponíveis na interface;
- uma matriz de habilitação dos comandos.

Número	Nome	Estilo	Ação
1	Fechar	Botão VB	Fecha a interface.
2	Excluir	Botão VB	Exclui um fornecedor.
3	Excluir Mercadoria	Botão VB	Exclui uma nova mercadoria da relação de mercadorias fornecidas por um fornecedor.
4	Incluir Mercadoria	Botão VB	Insere uma nova mercadoria na relação de mercadorias fornecidas por um fornecedor.
5	Novo	Botão VB	Limpa interface para inserção de um novo fornecedor.
6	Pesquisar	Botão VB	Abre um fornecedor já cadastrado para consulta.
7	Salvar	Botão VB	Salva dados sobre um fornecedor.

Tabela 154 - Exemplo de Lista de comandos

A lista dos comandos é herdada da Especificação de Requisitos, mas normalmente ocorrem várias alterações decorrentes do desenho das interfaces de usuário. Para os comandos, deve-se descrever:

- número;
- nome;
- estilo do comando (botão, item de cardápio textual, hiperligação etc.);
- ação (o que deve acontecer quando o botão é acionado).

Número	Comando	Estado		
		Interface Vazia	Interface Alterada	Interface Atualizada
1	Fechar	Habilitado	Requer confirmação	Habilitado
2	Excluir	Desabilitado	Requer confirmação	Habilitado
3	Excluir Mercadoria	Desabilitado	Habilitado	Habilitado
4	Incluir Mercadoria	Desabilitado	Habilitado	Habilitado
5	Novo	Desabilitado	Requer confirmação	Habilitado
6	Pesquisar	Desabilitado	Requer confirmação	Habilitado
7	Salvar	Desabilitado	Habilitado	Desabilitado

Tabela 155 - Exemplo de Matriz de habilitação

A matriz de habilitação descreve o estado de habilitação dos comandos em cada estado da interface.

2.2.4 Funções do produto

Descrevem-se aqui as funções do produto, detalhando os casos de uso em termos das interfaces definitivas. Estes casos de uso podem ser herdados da Especificação de Requisitos do Software, mas são descritos em nível de detalhe muito maior, com referências aos elementos das interfaces definitivas.

Os seguintes elementos devem ser descritos:

- mecanismos de acesso;
- fluxo principal e subfluxos;
- diagramas de estado;

- condições de exceção;
- mensagens.

O acesso ao caso de uso Gestão de Fornecedores é sempre feito através da Tela de Fornecedores. Essa é mostrada e ativada quando o item Fornecedores do cardápio Gestão de Compras da Tela Principal é acionado. Inicialmente, todos os campos de texto estão vazios e habilitados; a interface está no estado "Vazia".

Tabela 156 – Exemplo de Mecanismo de acesso

A descrição dos mecanismos de acesso identifica como o caso de uso é ativado (por exemplo: cardápio permanente, cardápio instantâneo, duplo clique em um elemento da área de trabalho etc.). O mecanismo de acesso pode ser também uma sequência de elementos de interface (por exemplo, uma determinada sequência de caixas de diálogo). A descrição deve incluir o estado inicial dos elementos das interfaces relevantes.

O Gestor de Compras pode preencher e editar os campos Código, Nome, Endereço, Telefone e CPF-CGC.

Se o Gestor de Compras preencher ou editar pelo menos um dos campos de dados do fornecedor (Nome, Endereço, Telefone e CPF-CGC), o Merci coloca a interface no estado "Alterada".

Se o Gestor de Compras acionar o botão Pesquisar, o Merci executa o subfluxo **Pesquisa**.

Se o Gestor de Compras acionar o botão Excluir, o Merci executa o subfluxo **Exclusão**.

Se o Gestor de Compras acionar o botão Incluir Mercadoria, o Merci executa o subfluxo **Inclusão de Mercadoria**.

Se o Gestor de Compras acionar o botão Excluir Mercadoria, o Merci executa o subfluxo **Exclusão de Mercadoria**.

Se o Gestor de Compras acionar o botão Novo, o Merci executa o subfluxo **Inserção**.

Se o Gestor de Compras acionar o botão Fechar, o Merci executa o subfluxo **Fechamento**.

Tabela 157 – Exemplo de Fluxo principal

O Merci exibe e habilita o campo Código da Nova Mercadoria e o botão OK.

O Gestor de Compras preenche o campo Código da Nova Mercadoria e aciona o botão OK

O Merci esconde e desabilita o campo Código da Nova Mercadoria e o botão OK.

Se o Código preenchido não existir na tabela Mercadorias do banco de dados Merci (ou não estiver preenchido), o Merci executa a exceção EGF-03.

Se o Código preenchido existir na tabela Mercadorias do banco de dados Merci:

O Merci pesquisa se o par formado pelo Código da mercadoria e pelo Código do Fornecedor existe na tabela Fornecimentos do banco de dados Merci.

Se já existir, o Merci executa a exceção EGF-04.

Se não existir:

O Merci inclui na tabela Fornecimentos do banco de dados do Merci o par formado pelo Código da mercadoria e pelo Código do Fornecedor.

O Merci exibe mais uma linha na grade do quadro de Mercadorias Fornecidas, contendo os campos Código e Descrição da mercadoria incluída.

Tabela 158 – Exemplo de Subfluxo

Nos fluxos do caso de uso, descrevem-se os detalhes do fluxo principal e dos fluxos secundários do caso de uso, mostrando os mecanismos de interação adotados para sua execução. As referências aos

elementos das interfaces usam os nomes para o usuário, constantes das descrições dessas interfaces. Outros detalhes que devem ser descritos incluem tabelas de bancos de dados consultadas e atualizadas, assim como possíveis exceções.

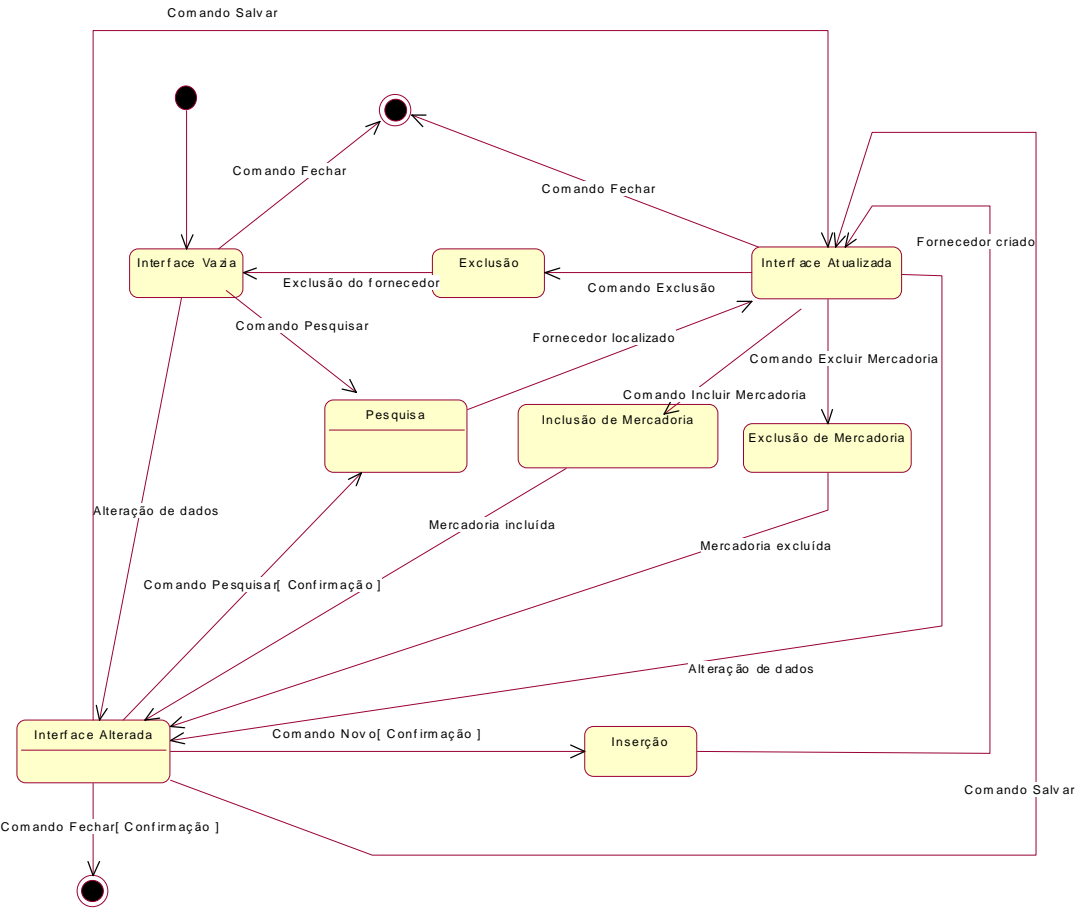


Figura 152 - Exemplo de Diagrama de estados de caso de uso

Os fluxos detalhados em nível de desenho são normalmente muito mais complexos que os apresentados na Especificação de Requisitos. Em muitos casos, é conveniente complementá-los com diagramas de estado. Estes diagramas devem ser consistentes com os diagramas das interfaces utilizadas, mas apresentam detalhes de processamento que não são refletidos diretamente na aparência da interface.

Número de ordem	Identificação da exceção	Descrição da exceção	Ação
1	EGF-01	Um código de fornecedor não corresponde a nenhum fornecedor cadastrado.	Emitir mensagem MGF-04.
2	EGF-02	Foi solicitada a exclusão de um fornecedor com pedidos de compra pendentes.	Emitir mensagem MGF-05.
3	EGF-03	Um código de mercadoria não corresponde a nenhuma mercadoria cadastrada.	Emitir mensagem MGF-06.
4	EGF-04	Foi solicitada a inclusão na lista de Mercadorias Fornecidas de uma mercadoria que já consta desta lista.	Emitir mensagem MGF-07.

Tabela 159 – Exemplo de Condições de exceção

Na lista de condições de exceção, descrevem-se as possíveis condições excepcionais que podem ocorrer durante a execução da função e a ação correspondente ao respectivo tratamento.

Número de ordem	Identificação da mensagem	Classe	Texto da mensagem
1	MGF-01	Requer confirmação	Favor confirmar se quer excluir esse fornecedor.
2	MGF-02	Informativa	O fornecedor foi excluído conforme solicitado.
3	MGF-03	Requer confirmação	Favor confirmar se quer excluir essa mercadoria.
4	MGF-04	Erro	O código informado não corresponde a fornecedor cadastrado.
5	MGF-05	Erro	Este fornecedor não pode ser excluído porque tem pedidos de compra pendentes.
6	MGF-06	Erro	O código informado não corresponde a uma mercadoria cadastrada.
7	MGF-07	Erro	Esta mercadoria já consta da lista das mercadorias fornecidas por este fornecedor.
8	MGF-08	Requer confirmação	Pode haver perda de dados. Favor confirmar se quer executar esta ação.

Tabela 160 – Exemplo de Mensagem

Na lista de mensagens, descrevem-se as possíveis mensagens, classificando-as quanto à natureza. Por exemplo, uma mensagem pode ser meramente informativa, indicar um erro do usuário, ou requerer confirmação para uma ação potencialmente destrutiva. O texto da mensagem é aqui incluído. A separação das mensagens em relação ao restante dos elementos do caso de uso facilita mudanças nos textos, decorrentes, por exemplo, de traduções, correções ou estudos de usabilidade.

2.3 Desenho interno (DDSw-3)

2.3.1 Estratégias de arquitetura

Descrevem-se aqui os principais aspectos estratégicos da arquitetura adotada, tais como:

- interfaces de usuário;
- interfaces com bancos de dados;
- interfaces de comunicação de dados;
- interfaces de dispositivo;
- uso dos serviços de sistema operacional.

É recomendável explicar os porquês da arquitetura adotada, comparando-se as decisões adotadas com outras alternativas que foram ou poderiam ter sido consideradas.

A interface de usuário seguirá o padrão Windows 9x para facilitar o aprendizado da utilização do produto por parte dos usuários.

O sistema de gerência de banco de dados utilizado será o Microsoft Access, devido à sua simplicidade de uso, difusão no mercado e compatibilidade com o paradigma relacional.

O ambiente de desenvolvimento será o Microsoft Visual Basic, pelos recursos de desenvolvimento rápido, difusão no mercado e suporte razoável para a tecnologia de orientação a objetos.

A comunicação de dados com o Sistema Financeiro será baseada na geração de arquivos de texto, dentro do formato Xpto 98.

A impressora de tickets de venda adotada será compatível com o modelo Bematech, por ser de baixo custo e de fácil manutenção.

Tabela 161 - Exemplo de Estratégias de arquitetura

2.3.2 Diagramas

2.3.2.1 Visão lógica

2.3.2.1.1 Diagramas de pacotes lógicos

Incluem-se aqui os diagramas de pacotes lógicos extraídos do Modelo de Desenho do Software.

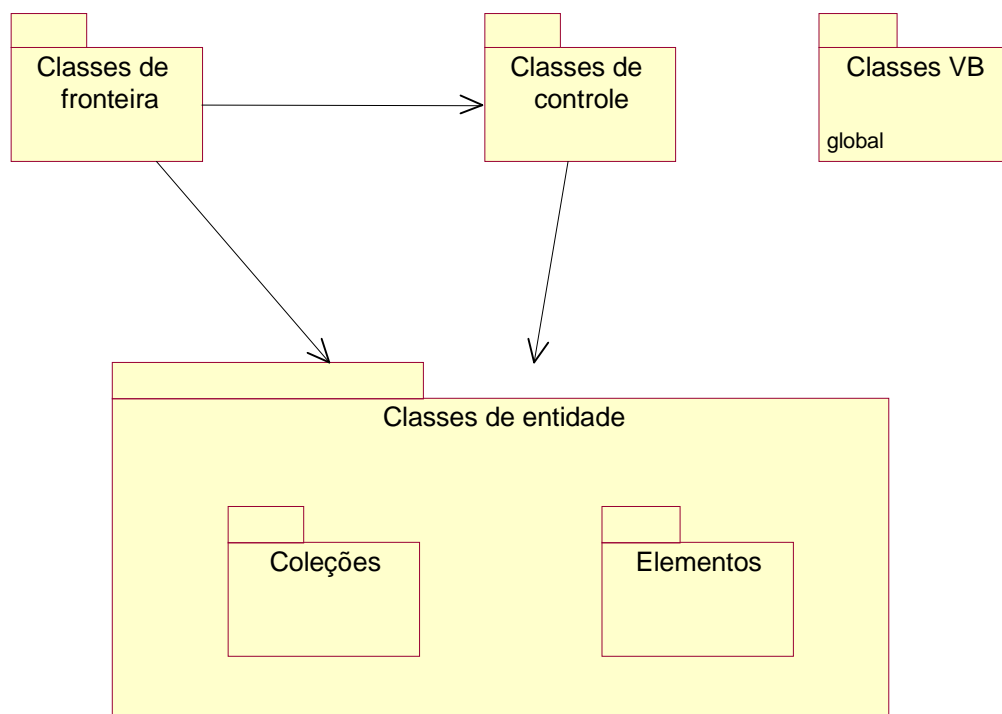


Figura 153 - Exemplo de diagrama de pacotes lógicos

2.3.2.1.2 Diagramas de classes

Incluem-se aqui os diagramas de pacotes lógicos extraídos do Modelo de Desenho do Software.

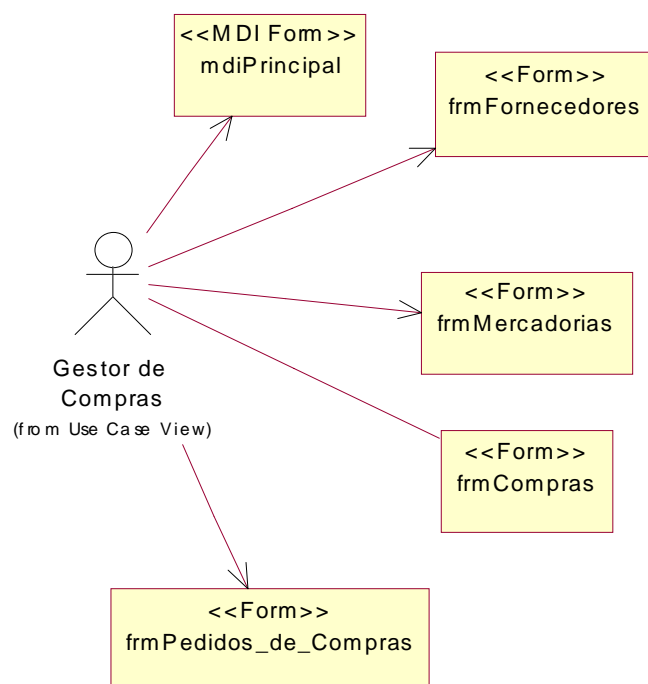


Figura 154 - Exemplo de diagrama de classes de interface do modelo de desenho

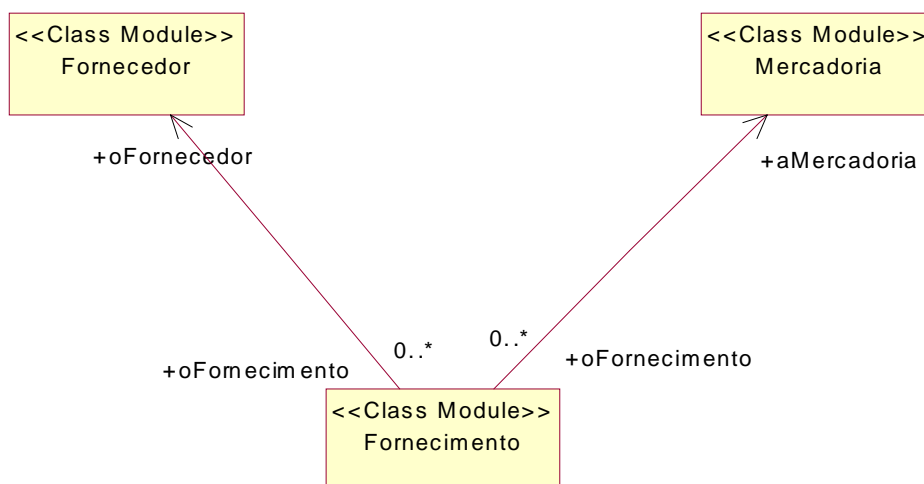


Figura 155 - Exemplo de diagrama de classes de elementos do modelo de desenho

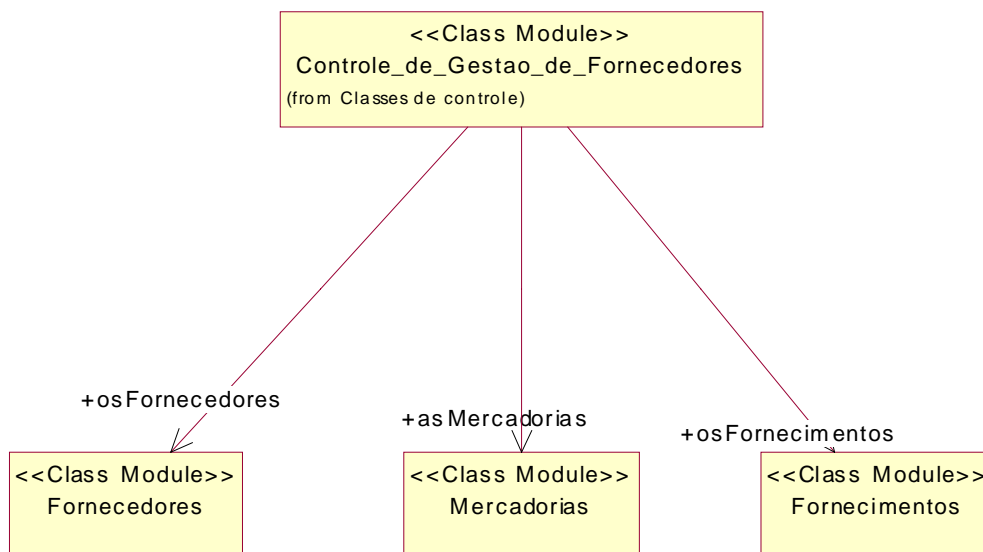


Figura 156 - Exemplo de diagramas de coleções do modelo de desenho

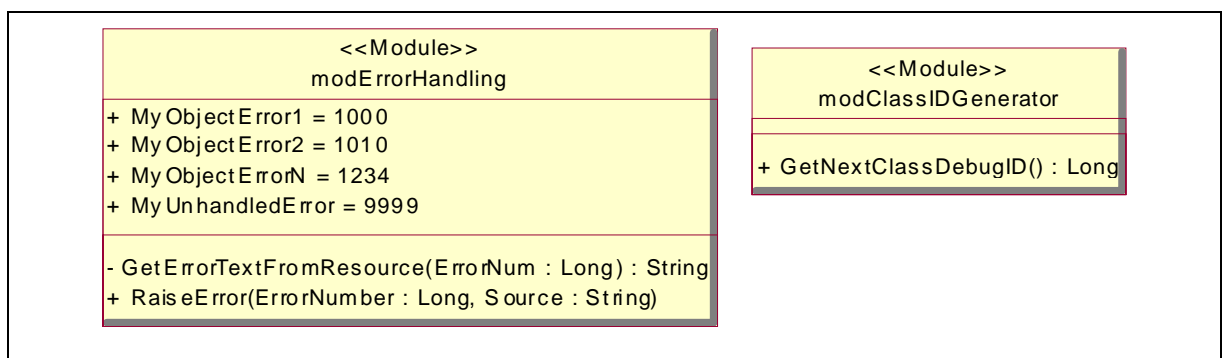


Figura 157 - Exemplo de diagrama de classes do ambiente de desenvolvimento

As classes do Modelo de Desenho do Software devem ser consistentes com as classes do Modelo de Análise e com a ERSw, observando-se que:

- as classes representativas de entidades do domínio do problema devem corresponder às classes de entidade do Modelo de Análise, ou estas devem ter sido rearranjadas de forma justificável;
- as classes representativas de interfaces de usuário devem ser consistentes com as interfaces de usuário descritas na seção de desenho da interface de usuário da DDSw e com as classes de fronteira do Modelo de Análise.

Os diagramas de classes de desenho devem ajudar a entender as principais decisões de desenho arquitetônico. Pode-se mostrar, por exemplo:

- classes representativas de interfaces de usuário, que interagem com determinado ator;
- classes representativas de coleções, que interagem com classes de controle de determinado caso de uso;
- classes representativas de elementos de coleções, que colaboram entre si para realizar determinados casos de uso;

- classes importadas do ambiente de desenvolvimento, cuja modelagem seja relevante para o desenho.

2.3.2.1.3 Diagramas de interação

Apresentam-se aqui os diagramas de interação constantes do Modelo de Desenho do Software. Usam-se diagramas de colaboração para enfatizar como as classes colaboram para realizar os casos de uso, e diagramas de seqüência para enfatizar o fluxo temporal das mensagens entre objetos.

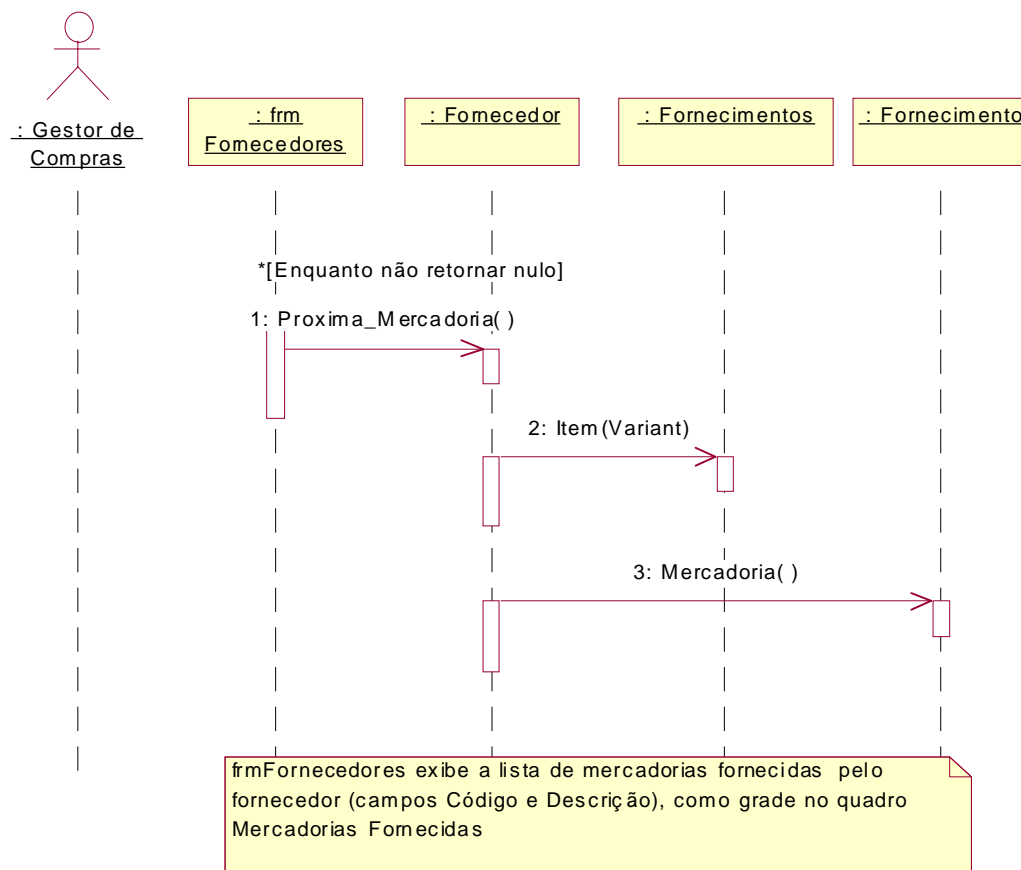


Figura 158 - Exemplo de diagrama de seqüência de desenho

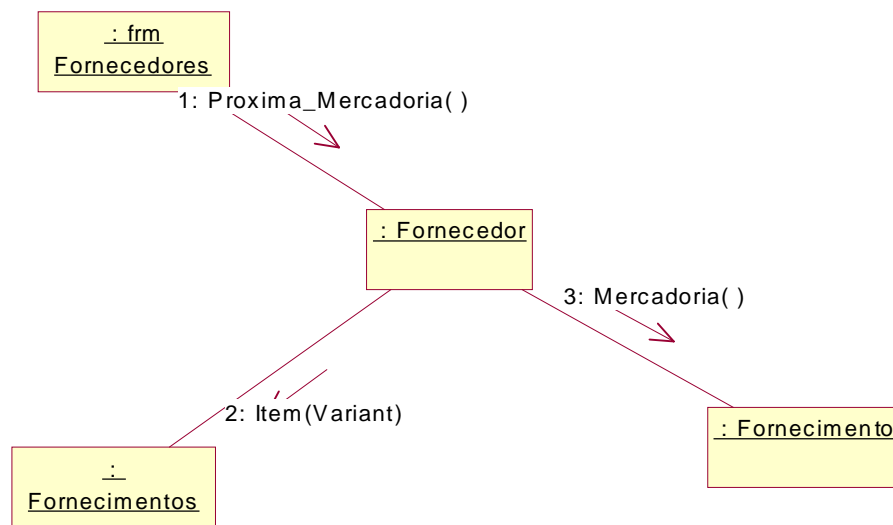


Figura 159 - Exemplo de diagrama de colaboração de desenho

Os diagramas de interação devem mostrar como as classes do modelo de desenho interagem entre si para realizar os casos de uso de desenho. Detalhes importantes de processamento interno das classes devem ser mostrados como anotações. Considerando que os casos de uso de desenho são bastante detalhados, pode ser conveniente partir um único fluxo em vários diagramas de interação, de modo que cada um destes caiba em uma página, sem prejudicar a leitura dos detalhes.

2.3.2.2 Visão física

2.3.2.2.1 Diagramas de componentes físicos

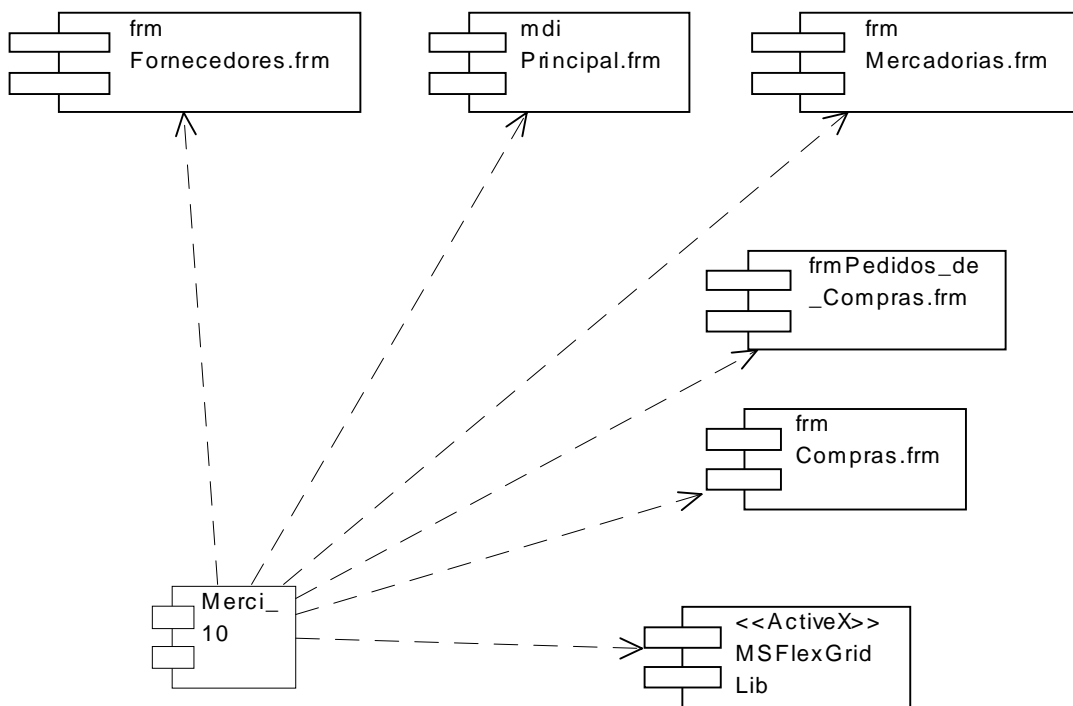


Figura 160 - Exemplo de diagrama de componentes físicos

Apresentam-se aqui os diagramas de componentes físicos constantes do Modelo de Desenho do Software. Eles indicam os elementos estáticos do produto, como arquivos de código fonte, código executável e de componentes reutilizados.

2.3.2.2.2 Diagrama de desdobramento

Apresenta-se aqui o diagrama de desdobramento(*deployment*) constante do Modelo de Desenho do Software, que indica a configuração física de processadores, dispositivos e conexões onde o produto será implementado. Esse diagrama é particularmente importante no caso de sistemas distribuídos.

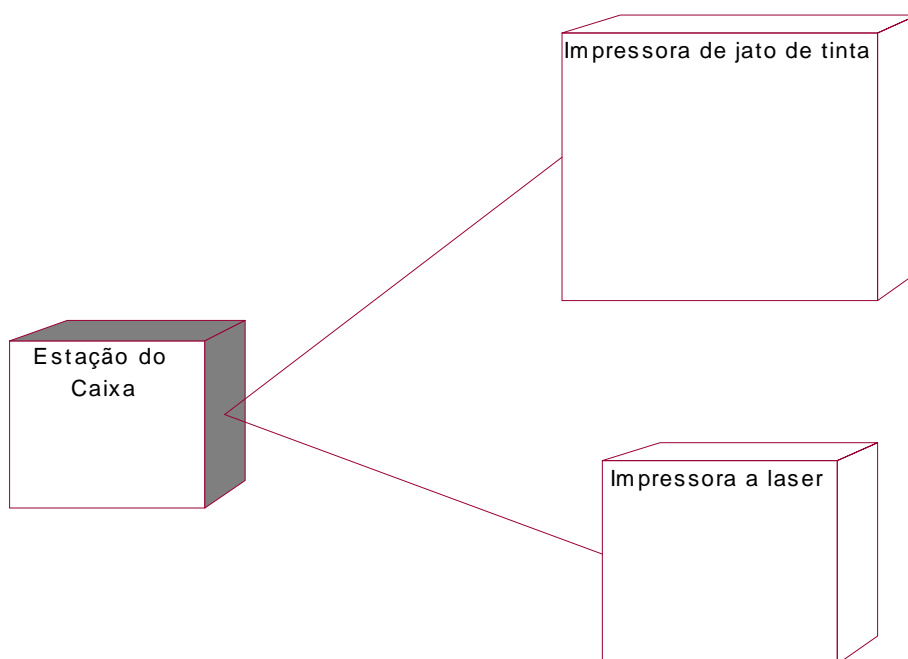


Figura 161 - Exemplo de diagrama de desdobramento

2.4 Desenho das liberações (DDSw-4)

2.4.1 Lista das liberações

Número de ordem	Nome da liberação	Objetivos
1	Compras	Implementar os casos de uso relacionados com a Gestão de Compras, permitindo o povoamento do banco de dados do Merci.
2	Vendas	Implementar os casos de uso relacionados com a Gestão de Vendas, completando as funções do Merci.

Tabela 162 - Exemplo de Lista das liberações

Apresenta-se aqui a lista das liberações planejadas, com a respectiva descrição de objetivos. O objetivo deve indicar quais aspectos do produto se pretenda avaliar.

2.4.2 Especificações das liberações

Classes a serem implementadas	Controle_Gestao_de_Mercadorias, Controle_de_Gestao_de_Pedidos_de_Compras, Controle_de_Gestao_Manual_de_Estoque, Mercadorias, Mercadoria, Fornecimentos, Fornecimento, Pedidos_de_Compras, Pedido_de_Compras, Compras, Compra, frmMercadorias, frmPedidos_de_Compras, frmCompras
Classes a serem alteradas	mdiPrincipal, frmFornecedores, Controle_Gestao_de_Fornecedores.
Casos de uso a serem implementados	Gestão de Mercadorias Gestão de Pedidos de Compra Gestão Manual de Estoque
Casos de uso a serem complementados	Gestão de Fornecedores - exclusão
Componentes de teste	Cotos para os casos de uso não implementados.
Componentes reutilizados	MS Flex Grid

Tabela 163 - Especificação das liberações

Descreve-se aqui a especificação de cada liberação, através dos seguintes campos:

- **Classes a serem implementadas** – classes que se pretende implementar nesta liberação.
- **Classes a serem alteradas** – classes que se pretende alterar nesta liberação, em relação às liberações anteriores (geralmente adicionando operações).
- **Casos de uso a serem implementados** – casos de uso que serão implementados pela colaboração entre as classes desta liberação e de liberações anteriores.
- **Casos de uso a serem complementados** – casos de uso em que alguns subfluxos e fluxos alternativos serão completados;
- **Componentes de teste** – estruturas provisórias (cotos e controladores) necessárias para os testes desta liberação.
- **Componentes reutilizados** – componentes que se pretende reutilizar, sejam comerciais ou resultantes de projetos anteriores.

2.5 Informação de suporte (DDSw-5)

2.5.1 Introdução

A DDSw deve incluir como anexos outros materiais que sejam necessários para a compreensão do desenho. Em particular, deve-se considerar as necessidades de revisores, testadores e implementadores. Maiores detalhes sobre o Modelo de Desenho do Software podem ser aqui colocados, como as especificações dos elementos desses modelos. Uma boa ferramenta de modelagem deve oferecer recursos para extração ou impressão dessas especificações em formato adequado.

2.5.2 Especificação dos relacionamentos

Muitos dos relacionamentos podem ser descritos através das suas especificações (Tabela 164).

Estereótipo	Metaclassificação do relacionamento, definida pelo desenhista ou por alguma metodologia (uma classificação do relacionamento, mais detalhada que as da UML).
Papel A	Primeira classe que participa do relacionamento (classe todo, nas agregações, e classe cliente, nas dependências).
Papel B	Segunda classe que participa do relacionamento (classe parte, nas agregações, e classe fornecedora, nas dependências).
Documentação	Detalhes não formalizados deste relacionamento.
Restrições	Restrições aplicáveis ao relacionamento como um todo.

Tabela 164 – Especificação de um relacionamento

Alguns detalhes são específicos de cada papel do relacionamento (Tabela 165).

Visibilidade	Controle de exportação (privado, protegido, público ou de implementação).
Documentação	Detalhes não formalizados deste papel, dentro do relacionamento.
Restrições	Restrições aplicáveis apenas ao papel.
Multiplicidade	Número de instâncias deste papel que podem ser associadas a uma instância do outro papel.
Navegável	Indica que o relacionamento é navegável na direção deste papel.
Agregado	Indica que o relacionamento é de agregação e este papel representa o “todo”.
Estático	Indica que um objeto da classe deste papel pertence à outra classe e não a um objeto desta. Por exemplo, em C++, seria implementado através de uma variável declarada na classe do outro papel como estática.
Amigo	Indica que a classe deste papel tem direito de acesso aos atributos e operações privadas da classe do outro papel.
Contido por	No caso de relacionamentos de agregação, indica se o papel parte é contido por valor ou referência.
Chaves qualificadoras	Atributos que identificam de forma única cada instância da classe do papel.

Tabela 165 – Especificação de um papel de um relacionamento

2.5.3 Especificação das classes

Muitos detalhes das classes podem ser definidos através das especificações destas classes (Tabela 166). Não é preciso preencher todos os campos das especificações, mas apenas aqueles que refletem decisões relevantes de desenho do produto. Cada atributo pode ter sua especificação própria (Tabela 167), assim como cada operação (Tabela 168).

Documentação	Definição, restrições, instruções para reutilização e outros detalhes não formalizados da classe.
Tipo	Tipo de classe: classe normal, classe parametrizada (<i>template</i> em C++), classe instanciada (instância de classe parametrizada), utilitário de classe (coleção de subprogramas livres ou de membros estáticos de dados e funções) ou utilitário de classe instanciado (combinação dos casos anteriores).
Estereótipo	Metaclassificação da classe, normalmente associada ao tipo de módulo de implementação no qual a classe será mapeada.
Visibilidade	Controle de exportação da classe em relação ao pacote lógico que a contém; a classe pode ser pública, protegida, privada ou de implementação.
Multiplicidade	Número esperado de instâncias da classe.
Componente	Unidade de código que contém a declaração da classe.
Espaço requerido	Espaço de memória principal ou secundária que se prevê seja requerido pelas instâncias da classe, se esse dado for relevante para a viabilidade do desenho ou o atendimento dos requisitos.
Persistência	Caráter persistente ou transiente da classe; classes transientes só podem ter instâncias transientes (que só existem em tempo de execução do programa), enquanto classes persistentes podem ter instâncias transientes ou persistentes (que ultrapassam os tempos de vidas dos programas, tais como registros de arquivos ou bancos de dados).
Concorrência	Caráter da semântica da classe quanto à concorrência: pode ser sequencial (semântica garantida apenas na existência de um único processo), ativa (classe tem seu próprio processo), guardada (classe pode ser partilhada por processos desde que esses garantam a exclusão mútua) e síncrona (classe pode ser partilhada por processos e ela mesma garante a exclusão mútua).
Abstração	Caráter concreto (pode ser instanciada) ou abstrato (não pode ser instanciada) da classe.

Tabela 166 – Especificação de uma classe

Documentação	Definição, restrições e outros detalhes não formalizados do atributo.
Tipo	Tipo primitivo da linguagem ou classe a que pertence.
Valor inicial	Valor inicial do atributo.
Contido por	No caso de atributos complexos (que são instâncias de classe), indica se o atributo é contido por valor (criado em tempo de compilação) ou referência (criado em tempo de execução).
Persistência	Indica se o atributo é estático (partilhado por todas as instâncias da classe), dinâmico (um para cada instância) ou derivado (existe no modelo, mas é implementado por cálculo, não gerando variável própria).
Visibilidade	Indica se o atributo é público, protegido, privado ou de implementação. O uso de atributos públicos não é recomendado.

Tabela 167 – Especificação de um atributo

Documentação	Definição, restrições e outros detalhes não formalizados da operação.
Assinatura	Lista de argumentos, com os respectivos tipos e valores padrão.
Tipo de retorno	Tipo do argumento de retorno.
Visibilidade	Indica se a operação é pública, protegida, privada ou de implementação.
Precondições	Condições que devem ser válidas para que a operação possa ser executada.
Semântica	Detalhes semânticos da operação, na forma de uma descrição em linguagem natural, pseudolinguagem algorítmica ou código em linguagem de programação.
Poscondições	Condições que devem ser válidas após a execução da operação.
Protocolo	Sequência preestabelecida de operações, dentro da qual a operação deve ser invocada.
Qualificação	Qualificações específicas da linguagem de implementação. Para uma implementação em C++, pode-se indicar se a operação é estática (tem acesso apenas a atributos estáticos), virtual (pode ser redefinida por subclasses), abstrata ou virtual pura (deve ser definida por subclasses), ou ainda amiga (implementada como operação de escopo global, amiga da classe).
Exceções	Exceções que podem ser causadas por esta operação.
Requisitos de espaço	Requisitos de memória principal ou secundária para esta operação, quando relevantes para o desenho.
Requisitos de tempo	Restrições de tempo que devem ser atendidas por esta operação.

Tabela 168 – Especificação de uma operação

2.5.4 Especificação dos componentes

Maiores detalhes sobre cada componente estático podem ser descritos através da especificação destes componentes (Tabela 169).

Documentação	Comentários de cabeçalho deste componente, tal como recomendado nos padrões de codificação.
Tipo	Tipo do componente dentro do ambiente de implementação.
Declarações	Declarações principais deste componente, tais como declarações de classes e de inclusões.
Classes	Classes implementadas (declaradas ou definidas) nesse componente.

Tabela 169 – Especificação de um componente

Maiores detalhes sobre componentes dinâmicos podem ser descritos através das especificações dos processadores (Tabela 170), dispositivos e conexões (Tabela 171).

Documentação	Detalhes não formalizados deste processador.
Características	Principais características técnicas, como, por exemplo, fabricante, modelo, quantidade de memória etc.
Processos	Lista dos processos associados a este processador.
Tipo de escalonamento	Regra para escalonamento dos processos neste processador. O escalonamento pode ser preemptivo, não preemptivo, cíclico, executivo (controlado por um algoritmo) ou manual.

Tabela 170 – Especificação de um processador

Documentação	Detalhes não formalizados deste dispositivo.
Características	Principais características técnicas, como, por exemplo, fabricante, modelo, protocolos etc.

Tabela 171 – Especificação de um dispositivo ou conexão

3 Revisão da Descrição do Desenho do Software

3.1 Introdução

Esta subseção apresenta um roteiro para revisão da Descrição do Desenho do Software (DDSw). Ao fim dessa revisão, deve-se assegurar que a DDSw:

- esteja conforme com esse padrão para Descrição do Desenho do Software, e com outros padrões aplicáveis ao projeto em questão;
- seja consistente com o Modelo de Desenho do Software, tendo todos os seus casos de uso não triviais realizados através de colaborações entre objetos das classes constantes desse modelo;
- forneça informação suficiente para a implementação do produto.

Nesse sentido, recomenda-se que, durante a revisão, todas as seções da DDSw sejam analisadas na ordem em que aparecem no documento, estabelecida neste padrão, observando-se as questões descritas a seguir. As expressões sublinhadas se referem a seções e subseções do documento revisado.

3.2 Revisão da seção Página de Título

Verificar se os seguintes itens constam das página iniciais da DDSw:

- nome do documento;
- identificação do projeto para o qual a documentação foi produzida;
- número de revisão do documento;
- nomes dos autores e da organização que produziram o documento;
- data de emissão;
- datas de aprovação;
- assinaturas de aprovação;

- lista dos números de revisão e datas de aprovação das revisões anteriores.

Observar se a disposição desses itens na página de título está consistente com as convenções estabelecidas pela organização.

3.3 Revisão da seção Sumário

Verificar se o sumário está consistente com o restante do documento. Essa seção é opcional para documentos com oito ou menos páginas.

3.4 Revisão da seção Lista de Ilustrações

Verificar se as listas de figuras e tabelas estão consistentes com o documento. Essa seção é opcional.

3.5 Revisão do corpo da Descrição do Desenho do Software

3.5.1 *Revisão da seção 1 Introdução*

3.5.1.1 Revisão da subseção 1.1 Objetivos deste documento

Verificar se o público **deste documento** é indicado corretamente.

3.5.1.2 Revisão da subseção 1.2 Escopo do produto

Verificar se essa subseção caracteriza adequadamente o produto que é objeto do desenho, e se é consistente com a respectiva Especificação dos Requisitos do Software.

3.5.1.3 Revisão da subseção 1.3 Materiais de referência

Verificar se essa subseção fornece a informação necessária para que todas as fontes de dados citadas na DDSw possam ser recuperadas, caso necessário.

3.5.1.4 Revisão da subseção 1.4 Definições e siglas

Verificar se essa subseção fornece a definição de todas as siglas, abreviações e termos usados na DDSw.

3.5.1.5 Revisão da subseção 1.5 Visão geral deste documento

Verificar se essa subseção descreve o que o restante da DDSw contém, indicando sua estrutura básica, e se as omissões são justificadas.

3.5.2 *Revisão da seção 2 Desenho das interfaces de usuário*

A revisão do desenho das interfaces de usuário deve ser feita de acordo com a lista de conferência apresentada nessa subseção.

A lista a seguir (Tabela 172) deve ser percorrida na ordem aqui apresentada. Alguns itens são aplicáveis ao conjunto das interfaces, e outros a interfaces específicas. Cada item específico deve ser verificado para cada uma das interfaces sob revisão. Só se deve passar ao item seguinte quando todas as interfaces tiverem sido checadas quanto ao item anterior.

Caracterização dos usuários	Os usuários foram completamente caracterizados?
	Foram determinadas as tarefas que se pretende automatizar?
	Foram avaliadas as versões correntes do produto?
Participação dos usuários no desenho das interfaces	Os usuários identificaram as tarefas que precisam executar?
	Os usuários identificaram com que frequência eles executam essas tarefas?
	Os usuários identificaram sob que condições eles executam tais tarefas?
	Os usuários identificaram os pontos fracos e fortes dos sistemas que se pretende substituir pelo produto?

Tabela 172 – Lista de conferência para a Descrição do processo

Orientação geral	O desenho como um todo pode ser considerado como centrado no usuário?
	As operações executadas pelos usuários são otimizadas?
Controle da interação pelo usuário	Nos formulários, os usuários podem movimentar-se livremente entre os campos?
	As ações são sempre iniciadas pelos usuários e não pelo sistema?
	As mensagens fazem com que os usuários sintam que o sistema está sempre pronto para responder às suas requisições?
	O usuário mantém o controle da interação com o produto?
Prevenção de erros	Apenas operações válidas são permitidas, em todas as situações?
	Em situações potencialmente destrutivas a confirmação dos usuários é solicitada?
	As ações dos usuários que podem provocar erros são previstas e inibidas?
Ajuda para usuários novatos	Essa ajuda informa as convenções utilizadas para os botões do mouse?
	Essa ajuda informa as áreas básicas da tela, como linhas de mensagens, barras de ferramentas etc.?
	Essa ajuda informa quais as funções básicas e os mecanismos para selecioná-las?
	Existe uma ajuda para usuários iniciantes?
Modelos do produto	É fornecido ao usuário um modelo mental do sistema, baseado nas tarefas desse usuário?
	A interação com os usuários é consistente?
Simplicidade	A terminologia é adequada e relacionada com as tarefas normais do usuário?
	Os símbolos e ícones são expressivos e associados ao domínio da aplicação?
	As tarefas são subdivididas de forma adequada?
	As funções de uso mais freqüente são particularmente fáceis de usar?
	As informações são bem distribuídas entre as telas?
	A interação com os usuários é simples, como um todo?
Questões de memorização	É permitido o fechamento freqüente das tarefas?
	O sistema promove o reconhecimento das funções e objetos que podem ser manipulados?
	O tratamento das questões de memória humana é adequado, como um todo?
Questões cognitivas	Os mnemônicos são significativos?
	Os ícones e outros elementos gráficos são significativos?
	As metáforas e analogias são usadas de forma adequada?
	O tratamento das questões cognitivas é adequado, como um todo?
Realimentação	A realimentação sintática é adequada?
	A realimentação semântica é adequada?

Tabela 173 - Lista de conferência para Diretrizes gerais - parte 1

Indicadores de estado	Para tarefas de média duração, é exibido um indicador de operação em progresso?
	Para tarefas de longa duração, é exibida uma estimativa do que falta para concluir a tarefa?
	O usuário recebe indicadores apropriados de estado, de maneira geral?
Mensagens	O fraseado das mensagens é orientado ao usuário?
	O fraseado das mensagens é positivo e não ameaçador?
	As mensagens de erro usam termos específicos e construtivos?
	Evitam-se referências antropomórficas ao sistema?
Modalidade	O modo ativo é sempre facilmente identificado?
	Interfaces não modais são usadas sempre que possível?
Reversibilidade	Existem mecanismos para cancelar tarefas?
	Existem facilidades para refazer as ações desfeitas?
	O número de níveis de ações reversíveis ²⁵ é adequado?
	No caso de reversão, volta-se sempre ao ponto adequado?
	Existem mecanismos para finalizar a aplicação em qualquer ponto?
Atração da atenção	Os níveis de intensidade são usados adequadamente?
	As fontes são usadas adequadamente?
	As maiúsculas e minúsculas são usadas adequadamente?
	Os piscamentos são usados apenas quando indispensáveis?
	As cores são usadas adequadamente?
	Os estímulos auditivos são usados adequadamente?
Questões de exibição	A posição dos elementos é mantida de tela para tela?
	As instruções e mensagens são concisas?
	Os ícones são fáceis de reconhecer?
	A densidade de informações na tela é minimizada, tanto global quanto localmente?
	A informação é distribuída de forma balanceada em cada tela?
	As áreas vazias são usadas para melhorar a legibilidade das informações?
	As informações relacionadas são agrupadas?
	As colunas e as linhas são alinhadas adequadamente?
Personalização	Existem recursos para que os usuários configurem a interface às suas preferências?
	Existe tratamento diferenciado para usuários com diferentes níveis de experiência?

Tabela 174 – Lista de conferência para Diretrizes gerais – parte 2

²⁵ Número de níveis de “undo”.

Janelas	O uso de janelas é adequado?	
	São mantidos a consistência na aparência e o comportamento da janela primária?	
	São usadas janelas diferentes para tarefas independentes diferentes?	
	São usadas janelas diferentes para visões diferentes do mesmo objeto?	
Cardápios	As hierarquias de cardápios são consistentes com as tarefas dos usuários e as funções do sistema?	
	As opções dos cardápios são agrupadas de forma significativa?	
	As opções dos cardápios são ordenadas de forma significativa?	
	As descrições das opções dos cardápios são breves?	
	O leiaute dos cardápios é consistente?	
	Os atalhos, aceleradores e recursos de macros fornecidos são adequados para os usuários experientes?	
Formulários	Leiaute e conteúdo	Os formulários têm títulos claros e significativos?
		Os campos dos formulários são ordenados e agrupados logicamente?
		Os agrupamentos de campos são delimitados visualmente de forma adequada?
		Os separadores e padrões de alinhamento são consistentes em todos os formulários?
	Conversão de formulários de papel	O formulário reflete o processo da aplicação tal como é usado atualmente?
		As redundâncias entre campos foram eliminadas?
		Os campos não utilizados no processo atual foram eliminados?
		Todas as informações foram colocados em campos apropriados, mesmo que não o tenham sido nos formulários de papel?
		Há necessidade de acesso a outros formulários ou registros?
	Os campos dos formulários são indicados de forma visual apropriada?	
	Os rótulos e abreviações são familiares e consistentes?	
	A navegação entre campos é lógica?	
	A navegação dentro dos campos é lógica?	
	É possível editar e corrigir os campos?	
	São exibidas mensagens de erro informativas e consistentes no caso de caracteres e valores inaceitáveis?	
	São exibidas mensagens explicativas para as entradas esperadas nos campos?	
	São providos valores padrão nos campos, sempre que possível?	
	É provido um indicador de completude para cada formulário?	

Tabela 175 - Lista de conferência para Estilos de interação – parte 1

Caixas	As caixas usam instruções breves mas compreensíveis?
	As caixas usam mensagens cuidadosamente fraseadas?
	As caixas usam agrupamentos lógicos e ordenamentos de objetos relacionados?
	As caixas usam indicações visuais para delinear agrupamentos?
	As caixas têm leiaute consistente e visualmente atraente?
	As opções padrões são ressaltadas?
	São indicadas visualmente as opções de cardápio que levam a caixas de mensagem?
	A remoção das caixas está sempre sob controle do usuário?
Linguagens de comando	As regras de formação são consistentes?
	Os nomes de comando são específicos, significativos e distinguíveis?
	As regras para abreviar comandos são consistentes?
	A correção de erros de digitação é fácil?
	Os usuários experientes podem desenvolver macros de comandos?
Interfaces pictóricas	As analogias com o mundo real são usadas adequadamente?
	A representação visual é simples?
	Os objetos visuais são apresentados em diferentes visões?
	A cor é usada com cuidado e de forma significativa?
	O vídeo é usado com cuidado?
Outros estilos	Outros estilos de interação (por exemplo, telas de toque, síntese e reconhecimento de voz) são usados de forma adequada?

Tabela 176 – Lista de conferência para Estilos de interação – parte 2

3.5.3 ***Revisão da seção 3 Desenho interno***

A revisão do desenho interno deve ser feita de acordo com a lista de conferência apresentada nessa subseção.

A lista a seguir (Tabela 177) deve ser percorrida na ordem aqui apresentada. Alguns itens são aplicáveis ao conjunto dos elementos do modelo, e outros a elementos específicos. Cada item específico deve ser verificado para cada um dos elementos do modelo sob revisão. Só se deve passar ao item seguinte quando todos os elementos tiverem sido checados quanto ao item anterior.

Aspectos gerais	O Modelo de Desenho de Software foi aprovado pela checagem da ferramenta de desenho do software?
	Foram descritos e justificados os principais aspectos estratégicos da arquitetura adotada?
	Os diagramas de pacotes lógicos refletem as decisões da arquitetura?
	Os pacotes lógicos apresentam alta coesão interna e baixo acoplamento externo?
Classes	As classes representativas de entidades do domínio do problema correspondem às classes de entidade do Modelo de Análise ou foram rearranjadas de forma justificável?
	As classes representativas de interfaces de usuário são consistentes com as interfaces de usuário?
	As especificações das classes são suficientes para a implementação destas classes?
	O campo de documentação reflete a definição, restrições, instruções para reutilização e outros detalhes pertinentes da classe?
	O estereótipo, caso indicado, é usado de forma consistente e documentada?
	A visibilidade é consistente com a estratégia de partição em pacotes lógicos?
	A cardinalidade das classes é correta?
	Foi indicado o componente físico que conterá a declaração da classe?
	O espaço requerido foi indicado, caso relevante?
	A persistência e a concorrência são consistentes com os objetivos da classe?
	A escolha entre classes abstratas e concretas é adequada?
	Todos os atributos são privados, a menos de muito boas razões em contrário?
	Os atributos são consistentes com os relacionamentos?
	As operações são suficientes para realizar os diagramas de interação dos quais os objetos da classe participam?
	O nome de cada operação é significativo em relação à sua função?
	A assinatura e o tipo de retorno de cada operação são consistentes com a respectiva função?
	A visibilidade de cada operação é adequada?
	Os detalhes especificados em cada operação são suficientes para a sua implementação?
	O uso de classes aninhadas, quando ocorrer, é justificável?
Diagramas de interação	Os diagramas de interação são claros e bem apresentados?
	Os diagramas de interação resolvem todos os casos de uso?
	A escolha entre diagramas de colaboração e de sequência é adequada?
Diagramas de estado	Os diagramas de estado são claros e bem apresentados?
	Os diagramas de estado foram usados de forma adequada?
	Os detalhes especificados de cada estado e transição são suficientes para a sua implementação?

Tabela 177 - Lista de conferência para o Modelo de Desenho – parte 1

Relacionamentos	A lista dos relacionamentos é consistente com os diagramas de classes?
	Os detalhes especificados de cada relacionamento são suficientes para a sua implementação?
	A navegabilidade dos relacionamentos de associação é consistente com o seu propósito e com os diagramas de interação?
	A escolha entre implementação das agregações por valor ou por referência é adequada?
	A multiplicidade de cada papel é correta?
	A visibilidade de cada papel é correta?
	As restrições aplicáveis a cada papel, se existentes, foram documentadas?
	Os relacionamentos complexos são descritos por classes de associação adequadas?
	A implementação dos relacionamentos entre classes persistentes é adequada, do ponto de vista dos sistemas de bancos de dados usados?
	As chaves qualificadoras, quando usadas, são adequadas?
	Os relacionamentos de generalização são usados de forma adequada?
Componentes estáticos	Todos os pacotes lógicos foram associados a pacotes de componentes?
	Os diagramas de componentes físicos indicam todos os arquivos fontes do produto, e, quando for o caso, a respectiva organização dentro do sistema de arquivos?
	O campo de documentação de cada componente físico contém os dados necessários para seu cabeçalho?
	O campo de declarações de cada componente físico contém as declarações pertinentes?
	O tipo de cada componente é consistente com as partes de classes e subprogramas que contém?
Componentes dinâmicos	No caso de sistemas distribuídos, foi feito um diagrama de distribuição, de forma adequada?
	O diagrama de distribuição é consistente com a arquitetura de sistema prevista para a instalação do produto?
	Para cada processador, processo, dispositivo e conexão foram documentados os aspectos relevantes para a instalação do produto?

Tabela 178 – Lista de conferência para o Modelo de Desenho – parte 2

3.5.4 ***Revisão da seção 4 Desenho das liberações***

3.5.4.1 **Revisão da subseção Lista das liberações**

Conferir a lista das liberações com os marcos previstos no Plano de Desenvolvimento do Software. Esse deverá ter sido atualizado para refletir as liberações previstas. Verificar se os prazos previstos são exeqüíveis, se a partição de funcionalidade entre as liberações é adequada, e se os maiores riscos do projeto são tratados pelas primeiras liberações.

3.5.4.2 **Revisão da subseção Especificações das liberações**

Para cada liberação indicada, conferir os seguintes aspectos:

- **Classes a serem implementadas** – verificar se são consistentes com o objetivo da liberação, e se estão presentes na seção 3 Desenho Interno da DDSw.

- **Classes a serem alteradas** – verificar se existem boas justificativas para alterar estas classes. Normalmente cada classe deve ser completamente implementada em uma única liberação. Verificar se estão presentes na seção 3 Desenho Interno da DDSw.
- **Casos de uso a serem implementados** – conferir se o conjunto dos casos de uso implementados por todas liberações cobre todos os casos de uso.
- **Casos de uso a serem complementados** – conferir se o conjunto dos subfluxos e fluxos alternativos implementados pelas liberações cobre todos os subfluxos e fluxos alternativos.
- **Componentes de testes** – conferir se são adequados para os testes desta liberação.
- **Componentes reutilizados** – verificar se a reutilização é consistente com as estratégias arquitetônicas descritas na subseção 3.1 Estratégias de Arquitetura da DDSw. Verificar se classes ou objetos que os representam estão presentes na Seção 3 Desenho Interno da DDSw.

Documentação de Testes de Software

1 Visão geral

No processo Praxis, a documentação de testes de software consiste nos seguintes documentos.

- A **Descrição dos Testes do Software** (DTSw) reúne os documentos redigidos antes da execução dos testes, contendo os planos e especificações desses. A DTSw pode ser alterada ao longo do projeto, fazendo parte das Linhas de Base desse.
- Os **Relatórios dos Testes do Software** (RTSw) reúnem todos os relatórios produzidos depois da execução dos testes de um projeto. Esse documento deve ser guardado sob controle, mas não faz parte da linha de base, pois cada novo relatório é simplesmente acrescentado à respectiva seção.

Este padrão contém diretrizes para a confecção da Descrição dos Testes do Software e dos Relatórios dos Testes do Software. Contém também um roteiro para revisão da Descrição dos Testes do Software.

2 Descrição dos Testes do Software

2.1 Introdução

O documento de Descrição dos Testes do Software (DTSw) é composto das seguintes partes:

- Os **Planos de Testes**, que registram os dados relativos ao planejamento de uma bateria de testes a ser executada.
- As **Especificações de Testes**, que descrevem os detalhes de cada tipo de teste que faz parte de uma bateria.

2.2 Referências

A principal referência deste padrão é:

IEEE. *IEEE Std. 829 – 1983. IEEE Standard for Software Test Documentation*, in [IEEE94].

2.3 Estrutura

O documento de Descrição dos Testes de Software deverá conter a seguinte estrutura:

7. Planos de testes
 - 7.1. Plano de testes de aceitação
 - 7.2. Planos de testes de integração
 - 7.2.1. Plano de testes de integração < nome da liberação >
 - 7.3. Planos de testes de unidade

7.3.1. Plano de testes da unidade < nome da unidade >

8. Especificações de testes

8.1. Especificação do teste < nome do teste >

2.4 Preenchimento da Descrição dos Testes do Software

2.4.1 Plano de testes

2.4.1.1 Identificador do plano de testes

Define-se aqui um identificador único para esse plano. Por exemplo, o identificador da Tabela 179 indica que esse é o primeiro plano de testes de aceitação do projeto Merci. Note-se que o projeto não tem número de versão, indicando que esse plano poderá ser reaproveitado no desenvolvimento de versões futuras do produto.

MERCI-PTA-01

Tabela 179 - Exemplo de Identificador do plano de testes.

2.4.1.2 Introdução

Resumem-se aqui aspectos importantes para caracterizar os itens que serão testados na bateria. Pode-se repetir informação presente em outros documentos, focalizando aqui os interesses dos testadores. São típicos os seguintes tópicos:

- objetivos dos testes (por exemplo, se são testes funcionais, de integração ou de unidade);
- histórico (aspectos do desenrolar do projeto que os testadores devam conhecer);
- escopo dos testes (abrangência e limites dos testes desse plano);
- referências a documentos relevantes.

Testar a funcionalidade, completeza e correção da implementação do produto Merci 1.0, comparando-o com a respectiva Especificação de Requisitos.
--

Tabela 180 - Exemplo de Objetivos dos testes

A United Hackers Informática especificou e desenvolverá o sistema de informatização de mercearias Merci 1.0, que visa ao suporte informatizado ao controle de vendas, de compras, de fornecedores e de estoque da mercearia Pereira & Pereira Comercial Ltda. A Especificação de Requisitos, o Plano de Desenvolvimento e o Plano da Qualidade desse produto foram apresentados pelo fornecedor em 19/3/99, e aceitos pelo cliente em 23/3/99.
--

Tabela 181 - Exemplo de Histórico

Todos os casos de uso e requisitos não funcionais levantados na Especificação de Requisitos serão testados.

Não será testado nenhum aspecto de instalação e configuração do Sistema Gerenciador de Banco de Dados Relacional utilizado e das impressoras utilizadas pelo produto.

Tabela 182 - Exemplo de Escopo dos testes

Número de ordem	Tipo do material	Referência bibliográfica
1	Documentação de desenvolvimento	Especificação dos Requisitos do Software - Projeto Merci Versão 1.0 Relatório Técnico 002-99, United Hackers Ltda.
2	Documentação de desenvolvimento	Plano de Desenvolvimento do Software - Projeto Merci Versão 1.0 Relatório Técnico 003-99, United Hackers Ltda.
4	Padrão	Wilson de Pádua Paula Filho e Cláudio Ricardo Guimarães Sant'Ana. <i>Manual de Engenharia de Processos de Software - Parte I: Políticas</i> . Relatório Técnico – DCC - 015/1998.
6	Livro	IEEE. <i>IEEE Standards Collection - Software Engineering</i> . IEEE, New York - NY, 1994.

Tabela 183 - Exemplo de Referências a documentos relevantes

2.4.1.3 Itens a testar

Define-se aqui o conjunto dos itens cobertos pelo plano. No caso dos testes de aceitação e integração, existe normalmente o único item, que é o produto total (respectivamente, completo ou incompleto). Componentes desenvolvidos no mesmo projeto podem ser itens separados. Nos testes de unidade, cada unidade sob teste é um item. Os comentários podem ser usados para definir estruturas provisórias de teste, que serão usadas junto com os itens.

Número de ordem	Item	Comentários
1	Produto Merci 1.0	O produto deverá estar instalado, juntamente com um banco de dados fictícios, usado exclusivamente para os testes de aceitação.

Tabela 184 - Exemplo de Itens a testar

2.4.1.4 Aspectos a testar

Define-se aqui o conteúdo dos testes que serão feitos. No caso dos testes de aceitação, cada caso de uso corresponde, normalmente, a um aspecto a testar. Requisitos não funcionais podem ser tratados como aspectos separados, correspondentes a testes de sistema, ou os respectivos testes podem ser combinados com os testes dos casos de uso (como os aspectos 9 e 10 na Tabela 185).

Número de ordem	Item	Referência à Especificação de Desenho de Teste
1	Caso de uso Abertura do Caixa	MERCI-ETF-01
2	Caso de uso Emissão de Nota Fiscal	MERCI-ETF-02
3	Caso de uso Emissão de Relatórios	MERCI-ETF-03
4	Caso de uso Fechamento do Caixa	MERCI-ETF-04
5	Caso de uso Gestão de Fornecedores	MERCI-ETF-05
6	Caso de uso Gestão de Mercadorias	MERCI-ETF-06
7	Caso de uso Gestão de Pedidos de Compra	MERCI-ETF-07
8	Caso de uso Gestão de Usuários	MERCI-ETF-08
9	Caso de uso Gestão Manual de Estoque	MERCI-ETF-09
10	Caso de uso Operação de Venda	MERCI-ETF-10
11	Requisito de desempenho Tempo de resposta	MERCI-ETF-10

Tabela 185 - Exemplo de Aspectos a testar

2.4.1.5 Aspectos que não serão testados

Definem-se aqui aspectos significativos que não serão testados explicitamente, o que normalmente significa que serão verificados por outros meios. Por exemplo, no plano dos testes de aceitação incluem-se aqui os requisitos que não serão objeto de testes específicos.

Número de ordem	Aspecto	Motivo
1	Restrição ao desenho Padrão de Nota Fiscal	Será checada através de inspeção da Notas Fiscais emitidas no teste do caso de uso Emissão de Nota Fiscal
2	Restrição ao desenho Expansibilidade	Será checada através de avaliação da adequação do desenho arquitetônico, em revisão técnica.
3	Atributo da qualidade Segurança do Acesso	Será checada, implicitamente, na realização dos testes funcionais.
4	Atributo da qualidade Apreensibilidade	Será checada durante o treinamento do operador de caixa, na atividade de Operação Piloto.

Tabela 186 - Exemplo de Aspectos que não serão testados

2.4.1.6 Abordagem

Definem-se aqui as opções metodológicas que são aplicáveis ao conjunto dos testes do plano. Por exemplo, deve-se tratar aqui decisões sobre automação dos testes, status de bancos de dados, ordem dos testes, estruturas provisórias de teste, reagrupamento de testes etc.

Os testes serão feitos manualmente. O banco de dados deverá estar inicialmente vazio. A ordem dos testes será escolhida de forma a povoar o banco de dados. A verificação do requisito de desempenho será realizada simultaneamente com o teste do caso de uso de Operação de Venda.

Tabela 187 - Exemplo de Abordagem

2.4.1.7 Critérios de completeza e sucesso

Definem-se aqui as condições que devem ser satisfeitas e o estado que deve ser atingido, para que o conjunto dos testes da bateria seja considerado bem sucedido.

Número de ordem	Critério
1	Para cada caso de teste, as saídas obtidas são completamente consistentes com as saídas previstas.
2	Todas as inclusões, alterações e exclusões de itens foram adequadamente refletidas no estado do banco de dados.
3	No teste do caso de uso Emissão de Nota Fiscal, o leiaute desta está de acordo com padrão aprovado pela Secretaria da Receita.
4	Os resultados de todas as operações aritméticas são consistentes

Tabela 188 - Exemplo de Critérios de completeza e sucesso

2.4.1.8 Critérios de suspensão e retomada

Os critérios de suspensão definem problemas graves que podem provocar a interrupção dos testes do plano. Os critérios de retomada definem o que deve ser feito para reiniciar os testes após a resolução desses problemas.

Ocorrência de Falha Geral de Proteção no Windows ou corrupção do banco de dados.
--

Tabela 189 - Exemplo de Critérios de suspensão dos testes

Correção do código e, se necessário, do desenho, para remoção da causa do erro. Ao ser reiniciado o teste de aceitação, deve ser realizada uma regressão, executando-se novamente todos os casos de teste. O banco de dados só será esvaziado se for corrompido.
--

Tabela 190 - Exemplo de Critérios de retomada dos testes

2.4.1.9 Resultados dos testes

Descrevem-se aqui os artefatos que serão produzidos durante a realização da bateria de testes. No mínimo, deve-se listar o conjunto de relatórios previsto no processo.

Serão produzidos os seguintes relatórios:

- | |
|---|
| <ul style="list-style-type: none"> um registro do teste; um registro de incidente para cada problema ocorrido; um relatório resumo do teste. |
|---|

Tabela 191 - Exemplo de Resultados dos testes

2.4.1.10 Tarefas de teste

Descreve-se aqui a lista detalhada das tarefas que são cobertas por esse plano de testes. Para cada tarefa, deve-se informar o esforço previsto (em pessoas-hora), as tarefas predecessoras e os responsáveis pela tarefa. Outras observações podem ser acrescentadas, descrevendo, por exemplo, proficiências necessárias do pessoal envolvido. A lista de tarefas define a ordem de execução das tarefas.

ID	Tarefa	Início	Fim	Esforço	Pred.	Pessoal	Observações
1	Testes de aceitação	03/05/99	29/11/99	154 h			
2	Desenho dos testes de aceitação	03/05/99	06/05/99	56 h			
3	Preparação do plano de testes de aceitação	03/05/99	03/05/99	8 h		D;T	
4	Preparação das espec. de desenho dos testes de aceitação	03/05/99	04/05/99	16 h	3	DS;T	
5	Preparação das espec. dos casos de testes de aceitação	04/05/99	06/05/99	32 h	4	DS;T	
6	Testes alfa	04/11/99	11/11/99	46 h	2		
7	Instalação e configuração do sistema para os testes alfa	04/11/99	04/11/99	4 h		DS	
8	Caso de uso Gestão de Usuários	04/11/99	04/11/99	4 h	7	T	
9	Caso de uso Gestão de Fornecedores	05/11/99	05/11/99	4 h	8	T	
10	Caso de uso Gestão de Mercadorias	05/11/99	05/11/99	4 h	9	T	
11	Caso de uso Gestão Manual de Estoque	08/11/99	08/11/99	4 h	10	T	
12	Caso de uso Abertura do Caixa	08/11/99	08/11/99	4 h	11	T	
13	Caso de uso Operação de Venda	09/11/99	09/11/99	4 h	12	T	
14	Caso de uso Emissão de Nota Fiscal	10/11/99	10/11/99	4 h	13	T	
15	Caso de uso Fechamento do Caixa	09/11/99	09/11/99	4 h	13	T	
16	Caso de uso Gestão de Pedidos de Compra	10/11/99	10/11/99	4 h	10	T	
17	Elaboração do relatório resumo dos testes alfa	11/11/99	11/11/99	6 h	16;15	T	
18	Testes beta	22/11/99	29/11/99	52 h	6		
19	Instalação e configuração do sistema para os testes beta	22/11/99	22/11/99	4 h		DS	
20	Caso de uso Gestão de Usuários	22/11/99	23/11/99	4 h	19	U	
21	Caso de uso Gestão de Fornecedores	23/11/99	23/11/99	4 h	20	U	
22	Caso de uso Gestão de Mercadorias	23/11/99	24/11/99	4 h	21	U	
23	Caso de uso Gestão Manual de Estoque	24/11/99	24/11/99	4 h	22	U	
24	Caso de uso Abertura do Caixa	24/11/99	25/11/99	4 h	23	U	
25	Caso de uso Operação de Venda	25/11/99	25/11/99	4 h	24	U	
26	Caso de uso Emissão de Nota Fiscal	26/11/99	29/11/99	4 h	25	U	
27	Caso de uso Fechamento do Caixa	26/11/99	26/11/99	4 h	25	U	
28	Caso de uso Gestão de Pedidos de Compra	26/11/99	29/11/99	4 h	22;27	U	
29	Elaboração do relatório resumo dos testes beta	29/11/99	29/11/99	12 h	28	DS;T;C	

Tabela 192 - Exemplo de Tarefas de teste

2.4.1.11 Ambiente

Definem-se aqui o hardware e software das configurações usadas para o conjunto dos testes. Incluem-se também as ferramentas que serão usadas, os componentes de testes que sejam necessários e os documentos que deverão estar disponíveis para os testadores.

Os testes deverão ser executados em um Pentium 133 MHz, com impressora de tecnologia laser ou de jato de tinta, a ser usada para impressão de todos os relatórios, exceto os tickets de venda. Será utilizada uma impressora específica para a emissão dos tickets de venda, configurável como impressora suportada pelo ambiente operacional.

Tabela 193 - Exemplo de Ambiente - Hardware

O ambiente operacional a ser utilizado é o Windows 95 (ou compatível). Deve ser utilizado o sistema de gerência de bancos de dados < nome do sistema >.

Tabela 194 - Exemplo de Ambiente - Software

< nome da ferramenta de testes >, a ser utilizada apenas em caso de testes de regressão.

Tabela 195 - Exemplo de Ambiente - Ferramentas de teste

Banco de dados fictício.

Tabela 196 - Exemplo de Ambiente - Componentes de teste

Número de ordem	Documento necessário
1	Especificação dos Requisitos do Software Projeto Merci Versão 1.0.
2	Descrição dos Testes do Software Projeto Merci Versão 1.0.

Tabela 197 - Exemplo de Ambiente - Documentos

2.4.1.12 Responsabilidades

Descrevem-se as responsabilidades de cada um dos participantes dos testes desse plano.

Número de ordem	Função	Responsabilidades
1	Testador	Planejamento dos testes de aceitação; especificação dos desenhos de teste e dos casos de teste de aceitação; realização dos testes alfa; redação do relatório resumo dos testes alfa; assessoria à elaboração do relatório resumo dos testes beta.
2	Desenvolvedor substituto	Instalação e configuração do sistema para os testes alfa e beta; assessoria ao testador na especificação dos desenhos de teste e dos casos de teste; assessoria à elaboração do relatório resumo dos testes beta.
3	Desenvolvedor	Assessoria ao planejamento dos testes de aceitação.
4	Usuário	Realização dos testes beta; redação do relatório resumo dos testes beta.

Tabela 198 - Exemplo de Responsabilidades

2.4.1.13 Agenda

Define-se a data de início e de fim de cada tarefa do plano. Essa informação também pode estar contida na lista das Tarefas de teste, como na Tabela 192.

2.4.1.14 Riscos e contingências

Descrever os riscos e contingências aplicáveis aos testes desse plano. Uma fonte inicial dessa informação pode ser o Plano de Desenvolvimento do Software.

Número	Risco	Gravidade	Probabilidade de ocorrência	Impacto previsto	Contramedidas previstas
1	Falta de usuários responsáveis por testes.	Alta	Baixa	Impossibilidade de realizar os testes beta	Cobrar providência do cliente
2	Falta dos equipamentos para testes beta.	Alta	Média	Impossibilidade de realizar os testes beta	Cobrar providência do cliente
3	Falta de inventário das mercadorias para o cadastramento.	Alta	Baixa	Impossibilidade de realizar os testes beta	Cobrar providência do cliente
4	Falta de povoamento inicial das bases de dados.	Alta	Baixa	Impossibilidade de realizar os testes beta	Cobrar providência do cliente

Tabela 199 - Exemplo de Riscos e contingências

2.4.1.15 Aprovação

Listam-se os nomes e assinaturas dos responsáveis pela aprovação do plano. Note-se que cada plano requer uma aprovação separada, pois os planos podem ser dirigidos a públicos diferentes. Por exemplo, os clientes podem participar da aprovação do plano dos testes de aceitação, mas não dos outros tipos de testes.

2.4.2 Especificações de testes

2.4.2.1.1 Identificador da especificação de teste

Define-se aqui um identificador único para esse teste. Por exemplo, o identificador da Tabela 200 indica que esta é a especificação do teste funcional número 08 do projeto Merci.

MERCI-ETF-08

Tabela 200 - Exemplo de Identificador da especificação de teste

2.4.2.2 Aspectos a serem testados

Listam-se aqui aspectos a testar que são combinados nesse teste.

Número	Requisito	Comentários
1	Caso de uso Operação de Venda	
2	Requisito de desempenho Tempo de resposta	

Tabela 201 - Exemplo de Aspectos a serem testados

2.4.2.3 Detalhes da abordagem

São descritos aqui detalhes de abordagem que sejam específicos desse teste. Essa subseção deve transmitir as razões para a escolha dos casos de teste e da seqüência destes.

Serão incluídos quatro usuários, um de cada grupo. Depois, dois usuários serão excluídos, e serão incluídos dois novos usuários, do mesmo grupo destes. Os casos de teste deverão ser executados na ordem em que são aqui apresentados.
--

Tabela 202 - Exemplo de Detalhes da abordagem

2.4.2.4 Identificação dos testes

2.4.2.4.1 Procedimentos de teste

Listam-se aqui os procedimentos associados a esse teste, com a respectiva identificação. Normalmente, cada procedimento corresponde a um roteiro importante do caso de uso.

Número	Procedimento de teste	Identificação do procedimento de teste
1	Inclusão de usuário	MERCI-ETF-08-PT-01
2	Alteração de usuário	MERCI-ETF-08-PT-02
3	Exclusão de usuário	MERCI-ETF-08-PT-03

Tabela 203 - Exemplo de Procedimentos de teste

2.4.2.4.2 Casos de teste

Listam-se os casos de teste associados a essa especificação. É recomendável listar os casos na ordem em que serão executados. O desenho dos casos de teste deve procurar maximizar a cobertura desse teste, enquanto o número de casos de teste será determinado pelos recursos e prazos disponíveis.

Número	Caso de teste	Identificação do caso de teste
1	Exclusão de Caixaero - 1	MERCI-ETF-08-CT-01
2	Inclusão de Gerente - 1	MERCI-ETF-08-CT-02
3	Inclusão de Gestor de Estoque - 1	MERCI-ETF-08-CT-03
4	Inclusão de Gestor de Compras	MERCI-ETF-08-CT-04
5	Inclusão de Caixaero - 1	MERCI-ETF-08-CT-05
6	Exclusão de Gestor de Estoque - 1	MERCI-ETF-08-CT-06
7	Exclusão de Caixaero - 2	MERCI-ETF-08-CT-07
8	Inclusão de Gestor de Estoque - 2	MERCI-ETF-08-CT-08
9	Inclusão de Caixaero - 2	MERCI-ETF-08-CT-09
10	Alteração de Gestor de Estoque - 1	MERCI-ETF-08-CT-10

Tabela 204 - Exemplo de Casos de teste

2.4.2.5 Critérios de completeza e sucesso

Descrever critérios de completeza e sucesso que sejam específicos desse teste. Estes critérios devem ser satisfeitos, em adição aos critérios gerais definidos no plano.

Número	Critério
1	Todos os usuários incluídos são localizados com nomes e grupos corretos, conforme mostrado por uma operação de pesquisa.
2	Todos os usuários excluídos são corretamente eliminados do banco de dados, conforme mostrado por uma operação de pesquisa.

Tabela 205 - Exemplo de Critérios de completeza e sucesso

2.4.2.6 Procedimentos de teste

Será descrito aqui cada um dos procedimentos de teste listados anteriormente. A descrição de cada procedimento deve conter os seguintes tópicos:

- identificação do procedimento;
- objetivo específico do procedimento;

- requisitos especiais para a execução do procedimento (por exemplo, montagens de volumes de dados);
- fluxo passo a passo do procedimento, detalhado o suficiente para que possa ser executado manualmente ou convertido em um script de teste.

O fluxo deve usar convenções tipográficas para distinguir interfaces, campos e comandos. No exemplo da Tabela 206 são usados, respectivamente, o negrito, o sublinhado e o itálico.

Identificação	MERCI-ETF-08-PT-01
Objetivo	Verificar se a inclusão de um usuário é feita corretamente no Merci.
Requisitos especiais	Nenhum.
Fluxo	<p>Abrir interface Tela de Usuários.</p> <p>Acionar <i>Novo</i></p> <p>Inserir <u>Nome</u>, <u>Login</u>, <u>Senha</u>.</p> <p>Acionar <i>Salvar</i>.</p> <p>Inserir <u>Login</u>.</p> <p>Acionar <i>Pesquisar</i>.</p>

Tabela 206 - Exemplo de descrição de procedimento de teste

2.4.2.7 Casos de teste

Será descrito aqui cada um dos casos de teste listados anteriormente. A descrição de cada caso deve conter os seguintes tópicos:

- identificação do caso de teste;
- itens a testar - aspectos do teste especificado que serão cobertos por esse caso;
- entradas - valores dos campos de entrada, deixando em branco os irrelevantes;
- saídas esperadas - valores esperados dos campos de saída;
- ambiente - necessidades de hardware, software, ferramentas e estruturas provisórias que sejam específicas do caso de teste;
- procedimentos - procedimentos de teste que serão usados para executar o caso;
- dependências - condições prévias para a execução desse caso, inclusive outros casos de teste que devam ser executados obrigatoriamente antes deste.

Identificação	MERCİ-ETF-08-CT-02	
Itens a testar	Processamento correto da inclusão de usuário no Mercı, com o banco de dados de usuários vazio.	
Entradas	Campo	Valor
	Nome	Gerente_01
	Login	Gerente_01
	Senha	xxx
Saídas esperadas	Campo	Valor
	Nome	Gerente_01
	Grupos do Usuário	Gerente
Ambiente	Banco de dados de teste.	
Procedimentos	Inclusão de Usuário - MERCİ-ETF-08-PT-01	
Dependências	Banco de dados de teste deve estar vazio.	

Tabela 207 - Exemplo de descrição de caso de teste

2.5 Revisão da Descrição dos Testes do Software

2.5.1 Introdução

Esta seção tem por objetivo prover diretrizes para a revisão da Descrição dos Testes do Software (DTSw). Ao fim dessa revisão, deve-se assegurar que:

- toda a funcionalidade do produto tenha adequadamente testada;
- os testes desenhados para o produto cubram aspectos e itens de teste suficientes para garantir a qualidade desejada;
- as redundâncias entre os aspectos e os itens de teste tenham sido minimizadas, resultando em uma bateria de testes econômica.

As expressões sublinhadas se referem a seções e subseções do documento revisado. Normalmente, uma revisão é feita para cada bateria de testes. Na revisão da primeira bateria (normalmente a bateria de testes de aceitação), são analisadas as páginas iniciais do documento.

2.5.2 Revisão da seção Página de Título

Verificar se os seguintes itens constam das páginas iniciais da DTSw:

- nome do documento;
- identificação do projeto para o qual a documentação foi produzida;
- número de revisão do documento;
- nomes dos autores e da organização que produziu o documento;
- data de emissão;

- datas de aprovação;
- assinaturas de aprovação;
- lista dos números de revisão e datas de aprovação das revisões anteriores.

Observar se a disposição desses itens na página de título está consistente com as convenções estabelecidas pela organização.

2.5.3 *Revisão da seção Sumário*

Verificar se o sumário está consistente com o restante do documento. Essa seção é opcional para documentos com oito ou menos páginas.

2.5.4 *Revisão da seção Lista de Ilustrações*

Verificar se as listas de figuras e tabelas estão consistentes com o documento. Essa seção é opcional.

2.5.5 *Revisão de um Plano dos Testes do Software*

2.5.5.1 *Revisão da seção 1 Identificador do plano de testes*

Verificar se foi criado um identificador único para esse plano de testes, de acordo com as convenções adotadas na organização.

2.5.5.2 *Revisão da seção 2 Introdução*

Verificar se:

- os objetivos dos testes estão claramente definidos e delimitados;
- o histórico contém a informação necessária para os testadores;
- está claramente explicado o que será e o que não será testado;
- todos os documentos relevantes são corretamente listados, de forma que possam ser localizados rapidamente.

2.5.5.3 *Revisão da seção 3 Itens a testar*

Verificar se:

- os itens a testar foram listados de forma clara e precisa;
- foram incluídas na listagem a versão e a preparação necessária a cada item a ser testado.

2.5.5.4 *Revisão da seção 4 Aspectos a testar*

Verificar se os aspectos listados correspondem a requisitos constantes da Especificação dos Requisitos do Software, principalmente com a lista dos casos de uso.

2.5.5.5 *Revisão da seção 5 Aspectos que não serão testados*

Verificar se:

- os aspectos que não serão testados foram listados de forma clara e precisa;
- a justificativa para cada um deles é plausível;

- foram indicados os métodos de verificação desses aspectos.

2.5.5.6 Revisão da seção 6 Abordagem

Verificar se a abordagem a ser utilizada está descrita de forma clara e objetiva.

2.5.5.7 Revisão da seção 7 Critérios de Completeza e Sucesso

Verificar se os critérios de completeza e sucesso foram clara e precisamente descritos, e se eles são suficientes e plausíveis.

2.5.5.8 Revisão da seção 8 Critérios de Suspensão e Retomada

Verificar se os critérios de suspensão e retomada foram descritos de forma clara e precisa, e se são plausíveis.

2.5.5.9 Revisão da seção 9 Resultados dos testes

Verificar se foram listados todos os artefatos que serão produzidos durante a realização da bateria de testes.

2.5.5.10 Revisão da seção 10 Tarefas de teste

Verificar se:

- a lista das tarefas é consistente com o processo adotado para os testes;
- a lista das tarefas é consistente com os aspectos a testar;
- foi estabelecida a ordem de precedência correta entre as tarefas.

2.5.5.11 Revisão da seção 11 Ambiente

Verificar se foram listados todos os recursos necessários, inclusive hardware, software, ferramentas e estruturas provisórias.

2.5.5.12 Revisão da seção 11 Responsabilidades

Verificar se:

- foi listada toda a equipe envolvida nos testes, definindo-se as respectivas responsabilidades;
- essa lista é consistente com as tarefas de teste.

2.5.5.13 Revisão da seção 13 Agenda

Verificar se essa seção está consistente com a seção de Tarefas de teste, ou se foi incorporada a ela.

2.5.5.14 Revisão da seção 14 Riscos e contingências

Verificar se foram previstos todos os riscos cabíveis, e se os planos de contingência citados podem realmente contornar a situação

2.5.5.15 Revisão da seção 15 Aprovações

Verificar se consta a assinatura de todo o pessoal citado na seção de Responsabilidades.

2.5.6 *Revisão de uma Especificação dos Testes do Software*

2.5.6.1 Revisão da subseção 1 Identificador da especificação de teste

Verificar se foi criado um identificador único para essa especificação de testes, de acordo com as convenções adotadas na organização.

2.5.6.2 Revisão da subseção 2 Aspectos a serem testados

Verifica se foram listados todos os aspectos que serão cobertos por esse teste.

2.5.6.3 Revisão da subseção 3 Detalhes da abordagem

Verificar se detalhes de abordagem específicos desse teste foram descritos de forma clara e precisa.

2.5.6.4 Revisão da subseção 4 Identificação dos testes

Verificar se foram listados todos os procedimentos e casos constantes desse teste.

2.5.6.5 Revisão da subseção 5 Critérios de completeza e sucesso

Verificar se os critérios de completeza e sucesso específicos desse teste foram clara e precisamente descritos, e se eles são suficientes e plausíveis.

2.5.6.6 Revisão da subseção 6 Procedimentos de teste

Verificar se a descrição de cada procedimento de teste contém:

- identificação do procedimento;
- objetivo específico do procedimento;
- requisitos especiais para a execução do procedimento;
- fluxo passo a passo do procedimento.

O fluxo deve ser detalhado o suficiente para que possa ser executado manualmente, ou convertido em um script de teste. Os nomes de interfaces, campos e comandos devem ter destaque tipográfico.

2.5.6.7 Revisão da subseção 7 Casos de teste

Verificar se a descrição de cada caso de teste contém:

- identificação do caso de teste;
- itens a testar;
- entradas;
- saídas esperadas;
- ambiente;
- procedimento;
- dependências.

3 *Relatórios dos Testes do Software*

3.1 Introdução

Os **Relatórios dos Testes do Software** registram os dados relativos à execução de um dos tipos de teste. Cada novo relatório deve ser adicionado sequencialmente aos Relatórios dos Testes do Software. Uma possível organização do documento de relatórios seria:

9. Relatórios dos testes de unidade

- 9.1. Relatórios dos testes de unidade da liberação 1
- 9.2. Relatórios dos testes de unidade da liberação 2
- 9.3. ...
- 9.4. Relatórios dos testes de integração

10. Relatórios dos testes de integração da liberação 1

- 10.1. Relatórios dos testes de integração da liberação 2
- 10.2. ...

11. Relatórios dos testes de aceitação

- 11.1. Relatórios dos testes alfa
- 11.2. Relatórios dos testes beta

Tipicamente, uma organização começaria a documentar os relatórios dos testes de aceitação. À medida que a cultura da organização absorvesse esses testes, passaria a documentar também os testes de integração, e, finalmente, os testes de unidade.

Cada relatório de testes segue a estrutura abaixo:

12. Relatórios dos testes de...

- 12.1. Relatório dos testes < nome do teste >
 - 12.1.1. Registro dos testes
 - 12.1.2. Relatórios de incidentes dos testes
 - 12.1.2.1. Relatório de incidente < nome do incidente do teste >
 - 12.1.3. Relatórios de resumo dos testes

Por exemplo, se a organização não documentasse os relatórios dos testes de unidade, realizasse e documentasse os testes de duas liberações, e realizasse e documentasse os resultados dos testes de aceitação, a estrutura do documento de relatórios seria a seguinte:

13. Relatórios dos testes de integração

- 13.1. Relatório dos testes de integração da liberação 1
 - 13.1.1. Registro dos testes de integração da liberação 1

13.1.2. Relatórios de incidentes dos testes de integração da liberação 1

13.1.3. Relatórios de resumo dos testes de integração da liberação 1

13.2. Relatório dos testes de integração da liberação 2

13.2.1. Registro dos testes de integração da liberação 2

13.2.2. Relatórios de incidentes dos testes de integração da liberação 2

13.2.3. Relatórios de resumo dos testes de integração da liberação 2

14. Relatórios dos testes de aceitação

14.1. Relatório dos testes alfa

14.1.1. Registro dos testes

14.1.2. Relatórios de incidentes dos testes alfa

14.1.3. Relatórios de resumo dos testes alfa

14.2. Relatório dos testes beta

14.2.1. Registro dos testes beta

14.2.2. Relatórios de incidentes dos testes beta

14.2.3. Relatórios de resumo dos testes beta

As subseções seguintes descrevem a estrutura de cada tipo de relatório.

3.2 Registros de testes

Durante a execução de um teste, os eventos relevantes são lançados no Registro do teste, em ordem cronológica. Cada registro tem um identificador próprio. Um registro tipicamente corresponde a uma execução completa ou parcial de uma bateria de testes. Por exemplo, haveria um registro de testes para os testes alfa, que são uma execução da bateria de testes de aceitação.

O campo de descrição do registro de testes contém dados que são comuns a todas as partes do teste, tais como:

- itens sob teste;
- ambiente de hardware e software;
- responsável pelos testes;
- outros participantes dos testes;
- outros recursos.

Dentro das entradas de atividades e eventos é registrada informação dos seguintes tipos:

- descrições de execução – início, interrupção, retomada, e término de atividades de teste, indicando os responsáveis e outros participantes;

- resultados de execução – registros do sucesso ou falha dos testes, deixando-se detalhes de falhas para a descrição dos incidentes;
- alterações ambientais – por exemplo, mudanças no hardware ou na configuração do ambiente;
- anormalidades – eventos inesperados em relação ao planejamento e especificação dos testes.

Um exemplo de registro de testes é apresentado na Tabela 208.

Identificador		MERCİ-RT-05	
Descrição dos testes		<p>Esse registro se refere aos Teste Alfa do Mercı 1.0, realizado de acordo com o plano de testes MERCİ-PTA-01.</p> <p>Configuração de testes: Pentium 133 MHz, com 64 MB de RAM e 300 MB de espaço livre em HD; impressoras laser e de ticket; Windows 95; Microsoft Access 97; Microsoft VB 6.0.</p> <p>Participantes do teste: Lúcia Malatesta (responsável); Jovino Audax.</p>	
Data	Hora	Atividades e eventos	Incidente
4-11-99	8:30	Jovino começou instalação da configuração para testes na máquina Romulus.	
	12:30	Jovino terminou checagem da configuração de testes na máquina Romulus.	
	13:00	Lúcia começou a executar testes do caso de uso Gestão de Usuários (MERCİ-ETF-08).	
	15:00	Lúcia interrompeu testes para reunião de acompanhamento do projeto Mercı 1.0.	
	17:30	Lúcia continuou a executar testes do caso de uso Gestão de Usuários (MERCİ-ETF-08).	
	19:30	Lúcia terminou testes do caso de uso Gestão de Usuários (MERCİ-ETF-08).	
5-11-99	9:00	Lúcia começou a executar testes do caso de uso Gestão de Fornecedores (MERCİ-ETF-06).	
	9:40	Defeito descoberto no caso de uso Gestão de Fornecedores (MERCİ-ETF-06): números de telefonemas interurbanos estão sendo exibidos truncados.	MERCİ-RT-05-IT-01
	13:00	Lúcia terminou testes do caso de uso Gestão de Fornecedores (MERCİ-ETF-06).	
	13:30	Jovino alterou configurações de rede do Windows 95, conforme registrado no Diário de Bordo da máquina Romulus.	
	14:00	Lúcia começou a executar testes do caso de uso Gestão de Mercadorias (MERCİ-ETF-06).	
	14:40	Falha geral de proteção na execução do caso de teste Exclusão de Merceria com Pedido Pendente (MERCİ-ETF-06-CT-04)	MERCİ-RT-05-IT-02
	14:50	Após consultar Eudóxia, Lúcia suspende os testes para pesquisa do defeito.	

Tabela 208 – Exemplo de Registro de testes

3.3 Relatórios de incidentes dos testes

Esses relatórios têm por objetivo documentar qualquer evento ocorrido durante a realização de testes, que mereça investigação. A investigação do incidente pode revelar ou não um defeito. Esse relatório tem os seguintes campos:

- identificador – identificador único do incidente;
- contexto – identificação do procedimento e caso de teste durante cuja execução aconteceu o incidente;
- descrição do incidente – entradas, saídas esperadas e reais, anomalias observadas, data, hora, passo do procedimento de teste, testador, observadores e particularidades do ambiente;
- impacto – consequências do incidente, se conhecidas.

Um exemplo de Relatório de incidente de teste é apresentado na Tabela 209.

Identificador	MERCI-RT-05-IT-01
Contexto	Caso de teste Inclusão de Fornecedor 3 (MERCI-ETF-05-CT-03), executando o procedimento de teste Inclusão de Fornecedor (MERCI-ETF-05-PT-01).
Descrição	Campos de entrada: Código = “167716”, Nome = “Metralha & Metralha Ltda.”, Endereço = “Av. Carandiru, 176 – São Paulo – SP”, Fone = “0xx11-67761761”, CGC = “6667771111/0001-76”, Mercadorias fornecidas - Código = “776611”, Descrição “Very Old Scotch Whisky”. Resultado esperado após comando Pesquisar com Código = “167716”: valores acima. Resultado real: valores acima, exceto Fone = “0xx11-677617”
Impacto	Revisão da classe Fornecedor: erro provável no atributo Telefone ou na operação Incluir.

Tabela 209 – Exemplo de Relatório de incidente de teste

3.4 Relatórios de resumo de testes

Os relatórios de resumo de testes fecham cada etapa de realização de testes, permitindo a avaliação da eficácia destes testes. Os relatórios de resumo fornecem também indicações sobre o nível de qualidade do produto, eventualmente indicando a necessidade de testes adicionais ou até de reconstrução de alguns itens sob teste.

Esses relatórios apresentam a seguinte estrutura:

- identificador – identificador único do relatório de resumo;
- contexto – itens objetos dos testes, com respectivos números de versão e revisão; plano de testes executado;
- variações – descrever possíveis variações dos testes em relação ao previsto nos respectivos plano e nas respectivas especificações;
- abrangência – avaliar se o grau conseguido de cobertura foi suficiente, conforme planejado, ou identificar possíveis deficiências dos testes, caso existam;
- sumário dos resultados – resumir os incidentes ocorridos, resolvidos ou não;

- avaliação – fornecer uma avaliação global da eficácia dos testes, se possível estimando a probabilidade de ocorrência futura de defeitos;
- sumário das atividades – comparar tempo e recursos previstos e efetivamente gastos para as tarefas de teste;
- aprovações.

Identificador	MERCİ-RRT-04		
Contexto	Relatório dos testes alfa do Mercı 1 revisão 1.		
Variações	Foram introduzidos novos procedimentos e casos de teste nos testes do caso de uso Gestão de Fornecedores (MERCİ-ETF-05) e do caso de uso Gestão Manual de Estoque (MERCİ-ETF-09), para cobrir roteiros adicionais que foram considerados importantes. Foram registradas as alterações na Especificação dos Requisitos e na Descrição dos Testes.		
Abrangência	A cobertura pode ser considerada satisfatória dentro do prazo e orçamento alocados para esses testes, considerados os novos procedimentos e casos de teste introduzidos conforme descrito anteriormente.		
Sumário dos resultados	Foram detectados e corrigidos 1 defeito maior e 5 defeitos menores. O número foi considerado satisfatório levando-se em conta a cobertura da bateria de testes e os defeitos detectados em revisões técnicas.		
Avaliação	A bateria é eficaz, dentro do contexto do projeto. Com base na experiência de outros projetos, estima-se que os testes beta provavelmente detectarão nenhum defeito maior e 3 defeitos menores.		
Sumário das atividades	Item	Previsto	Realizado
	Esforço (pessoas hora)	46	50,5
	Tempo de calendário (dias)	8	10
Aprovações	Nome	Assinatura	
	Lúcia Malatesta		
	Jovino Audax		
	Eudóxia Caxias		
	Metódio Prudente		

Tabela 210 – Exemplo de Relatório resumo dos testes

Página em branco

Desenho Detalhado e Codificação de Software

1 *Visão geral*

1.1 Introdução

Este padrão descreve um conjunto de diretrizes que devem ser seguidas na implementação de software, cobrindo-se as atividades de desenho detalhado e codificação. O principal objetivo destas diretrizes é facilitar a manutenção do código, promovendo sua legibilidade e desestimulando as construções mais propensas a erros. As principais referências deste padrão são [Booch+97], [Booch94], [Humphrey95], [Hyde97] e [McConnell93].

A segunda seção deste padrão apresenta diretrizes genéricas, que são aplicáveis à maioria das linguagens procedimentais de programação. A terceira seção apresenta algumas diretrizes específicas para a linguagem C++; diretrizes mais completas podem ser encontradas em [Meyers92] e [Meyers96].. Estas podem servir de exemplo para a definição de diretrizes para outras linguagens orientadas a objetos.

2 *Diretrizes genéricas*

2.1 Aspectos abordados

As recomendações genéricas incluem o tratamento dos seguintes temas:

- modularização;
- rotinas;
- tipos de dados;
- nomes;
- tipos abstratos de dados;
- estruturas de controle;
- leiaute de código;
- comentários;
- testabilidade.

2.2 Dados

2.2.1 *Tipos de dados*

2.2.1.1 Portabilidade de tipos

Regra: Usar definições portáveis de tipos.

Quando o código tiver que ser portátil, evitar o uso de tipos predefinidos cujo formato e tamanho variam com a arquitetura de máquina (por exemplo, `int`, `float` etc.). Se os dados dependerem de formatos específicos, definir tipos como `int16`, `int32`, `real64` etc.

2.2.1.2 Redefinição de tipos

Regra: Nunca redefinir um tipo predefinido do ambiente.

Isso vale para tipos predefinidos na linguagem, em bibliotecas padrão etc.

2.2.1.3 Denominação de tipos

Regra: Usar nomes de tipos funcionalmente orientados.

Os nomes de tipos devem ser orientados para as espécies de dados do Domínio do problema que representam (coordenadas, moedas, idades etc.), e não para aspectos de Implementação (`InteiroGrande`, `TextoLongo`).

2.2.1.4 Tipos simples

Algumas diretrizes para o uso de tipos numéricos em geral:

- usar constantes no lugar de literais (exceto no caso de 0 e 1);
- verificar se o valor dos divisores não pode ser 0;
- explicitar todas as conversões de tipo;
- evitar comparações de tipos mistos.

No caso de inteiros, considerar:

- tratar possibilidades de truncamento;
- tratar possibilidade de estouro (*overflow*), inclusive em resultados intermediários.

No caso de números de ponto flutuante, considerar:

- evitar somas e subtrações de números de magnitudes muito diferentes;
- evitar comparações de igualdade;
- tratar possibilidades de *overflow* e *underflow*;
- tratar possibilidades de erro de arredondamento.

No caso de caracteres e texto, considerar:

- usar constantes em lugar de literais;
- agrupar as constantes de textos em um arquivo de recursos, para facilitar mudanças e traduções.

No caso de variáveis booleanas, observar:

- fazer com que as variáveis booleanas ajudem a documentar o programa;
- substituir testes complicados por variáveis booleanas intermediárias;
- criar seu próprio tipo booleano, se não existir na linguagem.

2.2.1.5 Tipos complexos

No caso de variáveis enumeradas, observar:

- usar enumerações para facilitar a leitura;
- usar enumerações para aumentar a confiabilidade, promovendo checagens em tempo de compilação;
- usar enumerações para facilitar modificações;
- usar enumerações para generalizar variáveis booleanas para mais de dois resultados possíveis;
- testar valores inválidos;
- reservar uma entrada para representar valores inválidos;
- se a linguagem não tiver enumerações, simulá-las por meio de uma convenção de denominação.

No caso de arranjos, observar:

- usar arranjos apenas para acesso sequencial, e não randômico, de preferência;
- checar os pontos extremos dos arranjos;
- em arranjos multidimensionais, checar a ordem dos subscritos;
- em malhas aninhadas, cuidado com as confusões entre índices;
- por via das dúvidas, colocar um elemento extra no final do arranjo.

No caso de apontadores, considerar:

- isolar os apontadores em rotinas especializadas;
- checar os apontadores antes de usá-los;
- checar as variáveis apontadas antes de usá-las;
- usar campos de verificação de erros, em variáveis apontadas²⁶;
- anular o valor dos apontadores, depois de liberá-los;
- usar apontadores extra, quando isso contribuir para tornar o código mais legível;
- simplificar expressões com apontadores;
- procurar formas seguras de controlar a alocação e liberação de apontadores;
- documentar operações sobre apontadores, de preferência em formato gráfico;
- checar, na liberação de listas, se a ordem de liberação dos apontadores é correta.

²⁶ Campos que são preenchidos com um padrão preestabelecido e que nunca são modificados pelo código. Assim, modificações são sinal de que a variável está corrompida. Uma variante é duplicar campos e compará-los entre si.

Se o ambiente de depuração não dispuser de suporte para a exibição de apontadores, escrever uma rotina para imprimi-los. Onde for possível evitar apontadores de forma razoável, deve-se fazê-lo.

Tipos estruturados de dados (registros e estruturas) devem ser usados para:

- esclarecer relacionamentos de dados;
- simplificar operações sobre blocos de dados;
- simplificar listas de parâmetros, desde que o agrupamento seja lógico;
- reduzir encargos de manutenção.

O caso mais avançado de estruturação de tipos é formado pelos tipos abstratos de dados. Este padrão não trata desses tipos, por considerar sua utilização faz parte da análise e desenho inicial.

2.2.2 Variáveis

2.2.2.1 Declaração de variáveis - obrigatoriedade

Regra: Declarar sempre todas as variáveis.

Se a linguagem tiver a opção de permitir declarações implícitas, desligar esta opção. A declaração deve ficar próxima do primeiro uso da variável, tanto quanto a linguagem e as outras recomendações deste padrão permitirem.

2.2.2.2 Declaração de variáveis - comentários

Regra: As declarações devem ser acompanhadas de comentários descritivos.

Os comentários devem descrever as restrições e hipóteses relativas a essas variáveis, assim como os nomes completos, caso se use alguma abreviação. Cada declaração deve ocupar uma linha. Os tipos, os nomes das variáveis e os comentários devem ser alinhados em colunas.

2.2.2.3 Iniciação de variáveis

Regra: Iniciar todas as variáveis em sua declaração.

Deve-se tomar cuidado especial com contadores e acumuladores, e checar onde pode haver necessidade de reiniciação.

Em algumas situações, pode ser útil ligar mecanismos de iniciação padrão do compilador e preencher a memória de trabalho com um padrão determinado, no início do programa.

2.2.2.4 Escopo

Regra: Minimizar o escopo das variáveis.

Variáveis locais a uma rotina são preferíveis a variáveis locais a um módulo, que são preferíveis a variáveis globais. As referências a uma mesma variável devem ser localizadas o mais próximo possível.

Se um grupo de linhas de código de uma rotina gira em torno de uma variável, e outro grupo gira em torno de outra variável, esses grupos devem ser claramente formados e marcados no leiaute da rotina.

2.2.2.5 Finalidade

Regra: Usar cada variável com uma única finalidade.

Não reutilizar nomes como Temp para propósitos diferentes. Evitar variáveis com dois significados (por exemplo, a variável representa uma contagem se for positiva, e um código de status se for negativa). Conferir se todas as variáveis declaradas são usadas.

2.2.2.6 Variáveis globais

Regra: Minimizar o uso de variáveis globais.

Variáveis globais tendem a introduzir sérios problemas, como efeitos colaterais, pseudonímia (uso de diferentes nomes para a mesma coisa), impedimentos à reutilização de código, impedimentos à reentrância e quebra da modularidade, em geral. Apenas em alguns casos pode haver razões para usar variáveis globais:

- preservação de valores globais ao programa inteiro;
- simulação de constantes com nome;
- dados de uso extremamente comum;
- dados que são passados a rotinas de nível profundo, sem serem manipulados por rotinas de nível intermediário.

Nesses casos, as seguintes precauções devem ser tomadas:

- usar variáveis globais só em último caso;
- se possível, usar variáveis de módulo;
- usar uma convenção de nome que torne óbvios os nomes globais;
- manter uma lista anotada das variáveis globais;
- controlar o acessos às variáveis globais;
- não esconder as variáveis globais em uma estrutura monstro, para fingir que não as está usando.

Em uma linguagem pelo menos baseada em objetos, as variáveis globais deverão, na maioria dos casos, ser substituídas por objetos de classes adequadamente desenhadas.

2.2.3 *Nomes*

2.2.3.1 **Representação**

Regra: Todos os nomes devem descrever de forma completa e acurada a entidade que representam.

Além disso, nomes orientados para o problema são melhores que aqueles orientados para a implementação.

2.2.3.2 **Declaração**

Regra: Todos os identificadores devem representar palavras ou frases do idioma utilizado.

Existem estudos que indicam que a qualidade dos identificadores é mais importante até que a qualidade dos comentários. O nome deve descrever o conteúdo e/ou uso do objeto.

Em código que se destinar a eventual distribuição internacional, os nomes devem ser colocados em inglês (os comentários também). No Brasil, a maioria das pessoas prefere colocar normalmente os identificadores em português, costume que será adotado neste texto²⁷. Nesse caso, observar as seguintes regras.

- Ser gramaticalmente consistente em situações em que o português tem maior variedade de formas que o inglês. Por exemplo, nos nomes de funções, usar consistentemente o infinitivo (de preferência), o imperativo ou o indicativo presente.

²⁷ O resultado talvez não seja esteticamente agradável, por causa da mistura com as palavras reservadas e nomes predefinidos do ambiente, que estarão em inglês.

- Evitar o uso de caracteres acentuados nos identificadores, para evitar problemas de compilação. Nos comentários, usar a grafia correta, se o compilador o permitir.

2.2.3.3 Nomes compostos

Regra: Separar as palavras constituintes dos nomes longos.

Normalmente, os sublinhados (`_`) são usados para separar as partes de um nome composto. O uso de maiúsculas e minúsculas deve ajudar a captar o significado do nome.

Deve-se evitar usar identificadores totalmente maiúsculos. Uma possível exceção são os `defines` de C, já que essa é uma convenção habitual. No caso de C++, na maioria das situações os `defines` devem ser evitados.

2.2.3.4 Abreviações

Regra: Abreviações devem ser evitadas, se a linguagem o permitir.

Se necessário, padronizar a forma de abreviação. Colocar o nome completo nos comentários, de preferência na forma de uma tabela de abreviações.

2.2.3.5 Pronúncia

Regra: Os nomes devem ser pronunciáveis.

Evitar nomes que é preciso soletrar. Deve-se poder ler o código ao telefone, sem dificuldades.

2.2.3.6 Diferenciação

Regra: Os nomes devem ser diferenciáveis pelo seus primeiros caracteres.

Isso torna o código mais fácil de ler. Evitar diferenciações através de sufixos numéricos (exemplo: `temp`, `temp2`).

2.2.3.7 Notação húngara

Regra: Evitar o uso da notação húngara e de outras notações que denotem informação de baixo nível.

A notação húngara é baseada no uso de prefixos indicadores de tipos. Embora seja largamente utilizada no ambiente Windows, ela traz vários problemas:

- uso de tipos básicos de máquina, em lugar de tipos abstratos de dados;
- mistura de significado com representação;
- dificuldade de leitura.

Geralmente, o uso adequado da orientação a objetos torna a notação húngara desnecessária. Se houver necessidade de diferenciar entre identificadores de constantes, definições de tipos e variáveis, é preferível usar sufixos (ex.: `_c`, `_t`, `_v`). O sufixo não deve ser a única diferença entre dois identificadores.

2.2.3.8 Confusão de nomes

Regra: Evitar a ocorrência de identificadores com nomes que possam ser confundidos.

Exemplos de situações que devem ser evitadas:

- nomes diferenciados por caracteres que podem ser confundidos, como os grupo `{1, I, l}`, `{0, O}`, `{2, Z}`, `{5, S}` e `{6, G}`;
- abreviações enganosas (por exemplo, `int` para “`Indice_da_Nova_Tabela`”);
- nomes com significados similares (por exemplo, “`Proximo_Carater`” e “`ProxCar`” para dois objetos diferentes);

- nomes semelhantes com significados diferentes (por exemplo, “Numero_de_Linhas” e “Numero_da_Linha”);
- nomes homófonos;
- nomes com erros de ortografia;
- nomes que muitas pessoas grafam de forma errada.

2.2.3.9 Nomes predefinidos

Regra: Evitar o uso de nomes predefinidos no ambiente de desenvolvimento, a não ser que estejam sendo redefinidos.

Isso vale especialmente para bibliotecas-padrão e interfaces de programação de aplicativos.

2.2.3.10 Nomes de tipos específicos de dados.

Alguns tipos de dados requerem considerações específicas de denominação.

- **Índices de malha** - usar os tradicionais *i*, *j* e *k* apenas em casos muito simples. Preferir nomes significativos, como `Contador_de_Registros`. O uso de *i*, *j* e *k* é especialmente perigoso no caso de malhas aninhadas.
- **Variáveis de status** - devem indicar o significado do estado (`DadoPronto`, `Tipo_de_Relatorio`); não usar palavras que podem significar qualquer coisa, como `flag`.
- **Variáveis temporárias** - usar nomes que indiquem o papel ou significado da variável, evitando o uso de `temp`.
- **Variáveis booleanas** - usar nomes significativos, como `Feito`, `Erro`, `Achou`, `Sucesso`, que implicam um valor `Verdadeiro` ou `Falso`. Evitar nomes que não indicam um valor lógico, como `Status`, e nomes negativos, como `NaoAchou`.
- **Tipos enumerados** - usar um prefixo comum para nomes pertencentes ao mesmo grupo (`CorAzul`, `CorVermelha`, `CorVerde`).
- **Constantes** - usar nomes indicativos do significado da constante (`Total_de_Ciclos`) e não de seu valor (`Cinco`).

2.3 Estruturas de controle

2.3.1 Estruturas de seqüência

2.3.1.1 Seqüências obrigatórias

Regra: Quando existe uma seqüência correta de execução das instruções, ela deve ser facilmente reconhecível pela leitura do código.

A seqüência de execução pode ser realçada:

- pela forma de divisão em funções;
- pelos nomes das funções;
- pelos nomes dos parâmetros;
- pela documentação.

2.3.1.2 Sequência de leitura

Regra: O código deve ser organizado de forma que possa ser lido de cima para baixo.

Essa regra é contrariada pela presença de *gotos*, testes com aninhamento complexo, *breaks* mal colocados e outras construções não estruturadas.

A leitura é facilitada se forem observadas as seguintes regras:

- minimizar o alcance (*span*) das variáveis, isso é, a distância entre instruções em que a variável é referenciada;
- minimizar o tempo vivo da variável, isso é, a distância entre a primeira e a última referência a uma variável.

2.3.1.3 Agrupamento

Regra: Instruções correlatas devem ser agrupadas, e os grupos devem ser separados por linhas em branco e linhas de comentário.

Tipos comuns de correlação existem em grupos de instruções que correspondem a uma caixa de um fluxograma, uma fase de processamento, uma malha etc. Caso a coesão entre essas instruções seja particularmente forte, e o acoplamento com as demais particularmente fraco, elas devem ser colocadas em uma função separada.

2.3.2 Estruturas de seleção

2.3.2.1 Ifs Simples

No caso de instruções do tipo `if-then-else` simples, deve-se observar as seguintes regras:

- escrever primeiro o caso normal, e depois as exceções;
- prestar atenção especial no tratamento do caso de igualdade;
- colocar o caso normal depois do `if`, e não depois do `else`;
- evitar cláusulas vazias de `if`;
- quando não houver uma cláusula `else`, comentar por que não é necessária, a não ser, opcionalmente, quando o `if-then` é usado para tratamento de exceções;
- checar se ambas as cláusulas estão corretas, e no lugar correto.

2.3.2.2 Cadeias de ifs

Regra: Não utilizar o `else` final de uma cadeia de `ifs` para tratar um caso restante; reservá-lo para tratar condições de exceção.

Observar também as seguintes diretrizes:

- simplificar testes complicados através de chamadas a funções booleanas;
- checar se todos os casos estão cobertos.

Finalmente, se a linguagem dispuser de construções de seleção múltipla, elas são preferíveis às cadeias de `ifs`.

A Tabela 211 mostra um exemplo de cadeia de `ifs`.

```

if ( <caso 1> )
{
    <ação 1>;
}
else if ( <caso 2> )
{
    <ação 2>;
}
...
else if ( <caso n> )
{
    <ação n>;
}
else
{
    < ação de exceção>;
}

```

Tabela 211 – Exemplo de cadeia de ifs

2.3.2.3 Seleções múltiplas

Regra: Ao utilizar seleções múltiplas (**case/switch**), classificar os casos por ordem alfabética/numérica ou por frequência de ocorrência, mantendo entretanto juntos os casos cujo tratamento usa o mesmo código.

Essa recomendação visa a facilitar a leitura dos casos. As seleções múltiplas geralmente são usadas quando a ordem dos casos dentro da seleção é irrelevante.

Outras diretrizes aplicáveis às seleções múltiplas:

- manter cada caso simples;
- se não for possível usar uma variável de seleção simples, preferir as cadeias de ifs;
- usar o caso default apenas para tratar situações default legítimas, como erros, e não para apanhar casos remanescentes.

Em C, não esquecer o "break" no fim de cada caso. Se a omissão desse for intencional, como na situação em que vários casos são agrupados para partilhar o mesmo código, deve-se documentar o fato.

2.3.3 Estruturas de iteração

2.3.3.1 Tipos de malhas

Regra: Usar sempre o tipo de malha mais adequado à lógica da iteração.

Os tipos de malha diferem pelo teste de terminação no início (while (...) {...}), no final (do {...} while (...)) ou no meio (while (TRUE) {...; if (...) break; ...}). Usar a forma de terminação no meio para representar uma terminação no início ou no fim significa que espera-se adicionar novas instruções, respectivamente, antes ou depois do teste de saída.

A entrada em uma malha deve ocorrer em um único ponto. O código de iniciação da malha deve ser colocado imediatamente antes dela. O miolo da malha deve ser cercado por begin - end ou por { }. As variáveis de controle das malhas devem de preferência ser tipos ordinais ou enumerados, sempre com nomes significativos.

2.3.3.2 Malhas tipo for

Regra: Malhas tipo **for** devem ser usadas quando se sabe que a malha deverá ser executada um número específico de vezes.

Na maioria dos casos, o incremento do contador do for deve ser 1. Formulações mais complexas, mesmo quando permitidas pela linguagem, são prejudiciais à legibilidade.

Em C, as malhas tipo `for` são particularmente convenientes, por concentrarem em um único lugar o código de controle da malha. Entretanto, não devem ser usadas artificialmente em situações em que a malha `while` é mais apropriada.

2.3.3.3 Malhas eternas

Regra: Usar um formato padronizado e facilmente reconhecível para malhas eternas.

Por exemplo, o teste `while (TRUE)`, ou, em C, a macro:

```
#define FOREVER for ( ; ; )
```

2.3.3.4 Pontos de saída

Regra: A condição de saída da malha deve ser facilmente identificável.

De preferência, deve haver um único ponto de saída da malha. A condição de saída não deve ser provocada pela manipulação do índice de um `for`. Deve-se evitar código que dependa do valor terminal do índice.

2.3.3.5 Malhas vazias

Regra: Evitar malhas vazias.

Quando o único objetivo da malha é produzido por um efeito colateral do teste de terminação, mover o efeito colateral para dentro da malha, ou pelo menos colocar um comentário dentro do corpo dessa (`//Faz nada.`).

2.3.3.6 Efeitos colaterais

Regra: O uso de efeitos colaterais dentro do teste de terminação das malhas deve ser evitado ou pelo menos documentado com destaque.

No caso de C/C++, em que esse uso é de praxe em muitas situações, indicar em um comentário o propósito do efeito colateral.

2.3.3.7 Funções das malhas

Regra: Cada malha deve executar apenas uma função.

Por exemplo, o uso da mesma malha para iniciar dois vetores diferentes impede que um destes venha a ter posteriormente suas dimensões alteradas.

2.3.3.8 Comprimento das malhas

Algumas diretrizes para o comprimento das malhas:

- ser curtas o bastante para caber em uma tela ou uma folha de listagem;
- usar no máximo três níveis de aninhamento;
- em malhas longas, redobrar o cuidado com a clareza e a documentação.

2.3.4 Estruturas especiais de controle

2.3.4.1 GOTOS

Regra: **Gotos** só devem ser usados em casos extremos em que o código ficaria mais ilegível sem o **goto** do que com ele.

Essa decisão deve ser revista por pelo menos dois pares, e só deve ser aplicada no caso de tratamento de exceções ou depois de várias tentativas de escrever código mais claro sem o `goto`. Deve estar provado que o `goto`, no caso em questão, contribui para a legibilidade, a manutenibilidade e a eficiência.

O `goto` deve ser limitado a um rótulo por rotina (exceto quando usado para emulação de construções estruturadas), e deve apontar para diante. Deve-se verificar se todos os códigos são usados, e se o `goto` não cria código inatingível.

2.3.4.2 Return

Regra: Só usar o **return** para melhorar a legibilidade.

O número de `returns` deve ser minimizado. Geralmente, um `return` é usado para provocar um retorno imediato assim que o resultado de uma função é conseguido.

2.3.4.3 Recursão

Regra: Só usar a recursão quando puder provar que ela tem limite.

Contadores de segurança devem ser usados para evitar a recursão infinita, e o tamanho da pilha deve ser controlado. A recursão deve ser limitada a uma única rotina. Não usar a recursão para cálculos simples em que existem fórmulas abertas mais eficientes, como os fatoriais.

2.3.5 Aspectos gerais de controle

2.3.5.1 Expressões de Teste

Regra: Simplificar as expressões de teste usadas para controle.

Deve-se usar termos como `Verdadeiro` e `Falso` para tornar mais legíveis os testes booleanos. Se eles não existirem na linguagem, devem ser criados através de definições de pré-processador, constantes ou variáveis globais. Em C, por exemplo, pode-se definir:

```
#define VERDADEIRO ( 1 == 1 )
#define FALSO ( !VERDADEIRO )
```

As comparações com `Falso` devem ser implícitas (`!Pronto` em vez de `Pronto == Falso`). Os testes dos `ifs` devem ser positivos (o resultado positivo executa o `then`, não o `else`).

As seguinte medidas devem ser usadas para simplificar testes complexos:

- parti-los, se necessário introduzindo novas variáveis booleanas;
- usar funções booleanas;
- empregar tabelas de decisão;
- usar parênteses.

São especialmente úteis as regras de De Morgan:

```
!( A || B ) == !A && !B
!( A && B ) == !A || !B
```

2.3.5.2 Testes numéricos

Regra: Organizar os testes numéricos na direção dos pontos de uma reta.

Por exemplo, para selecionar os valores dentro de uma intervalo, é preferível usar a forma

```
min < i && i < max // ----- min iiii min max -----
```

enquanto para valores fora do intervalo deve-se usar

```
i < min || max < i // iiii min ----- max iiii
```

Para comparações com 0, usar sempre o 0 apropriado: 0 para números, `'\0'` para caracteres, `NULL` para apontadores etc.

2.3.5.3 Aninhamento de testes

Regra: Minimizar os níveis de aninhamento.

Isso pode ser conseguido através das seguintes medidas:

- retestar condições, quando isso contribuir para reduzir o aninhamento;
- converter conjuntos de `if s` aninhados em cadeias de `if then else`;
- converter conjuntos de `if s` aninhados em seleções múltiplas;
- passar código profundamente aninhado para uma rotina independente;
- redesenhar código profundamente aninhado.

2.4 Expressões

2.4.1 Operadores

2.4.1.1 Precedência

Regra: Deve ser assumida a seguinte ordem de precedência, da mais alta para a mais baixa – operandos, operadores unários, operadores multiplicativos, operadores aditivos, operadores relacionais e operadores lógicos. Se a precedência correta não puder ser estabelecida através dessa regra, deve-se usar parênteses.

Se dois operadores adjacentes em uma expressão pertencem a duas classes diferentes de operadores, não é preciso usar parênteses. Pode-se assumir que os operadores multiplicativos (`*`, `/`, `%`) e aditivos (`+`, `-`) são associativos à esquerda; nos outros casos, deve-se usar parênteses para estabelecer a ordem de associação.

2.4.1.2 Avaliação abreviada

Regra: Se o valor correto de uma expressão depende de avaliação abreviada, anotar o fato em um comentário.

Um exemplo de avaliação abreviada (*short-circuit evaluation*): em `(a && b)`, se a avaliação é feita da esquerda para a direita e `a` é falso, `b` não precisa ser avaliada.

Esse comentário não é necessário no caso de se usarem linguagens em que a avaliação abreviada é uma expressão idiomática (caso de Perl). Por outro lado, a avaliação abreviada não deve ser usada quando não faz parte dos usos e costumes da linguagem (caso de C).

2.4.2 Efeitos colaterais

2.4.2.1 Produção

Regra: Evitar o uso de efeitos colaterais.

Efeitos colaterais podem ser produzidos, por exemplo, pela chamada de funções com parâmetros passados por referência, ou que modifiquem variáveis globais.

2.4.2.2 Resultados

Regra: Não usar valores de variáveis modificadas por efeitos colaterais dentro da mesma expressão.

Em alguns casos, como em C/C++, operadores têm por objetivo produzir efeitos colaterais (`++`, `--`, `=`), ou os efeitos colaterais são de praxe (ler e testar fim de arquivo na mesma expressão). Uma expressão nunca deve ser executada apenas pelos efeitos colaterais que produz.

2.5 Leiaute de programa

2.5.1 Princípios

[McConnell93] propõe os seguintes princípios básicos de leiaute de programa.

- O leiaute deve refletir a estrutura lógica do programa (“Princípio Fundamental da Formatação”). O espaço em branco é a principal ferramenta de leiaute.
- A representação da estrutura deve ser consistente. Por exemplo, em C/C++, regras a respeito da colocação de { } devem ser consistentes entre si, em funções, seleções e repetições.
- O objetivo principal é a legibilidade, e não necessariamente a estética.
- O leiaute deve ser resistente a modificações. Por exemplo, usar { } depois de um if, mesmo que seja para conter uma única instrução, facilita a posterior adição de outras instruções, caso venham a ser necessárias.

2.5.2 *Espaçamento em expressões*

2.5.2.1 Operadores unários

Regra: Evitar espaços entre operadores unários e seus operandos.

Exemplos em C/C++:

```
-a
```

```
!x
```

```
*p
```

2.5.2.2 Operadores binários

Regra: Deve haver pelo menos um espaço de cada lado de um operador binário.

Exemplo em C/C++:

```
x = a / *p < -b;
```

2.5.2.3 Operadores de seleção

Regra: Operadores de seleção devem ser adjacentes a seus operandos.

Exemplos em C/C++:

```
registro.campo
```

```
estrutura->campo
```

```
vetor[ indice ]
```

2.5.2.4 Separadores

Regra: Separadores de itens devem seguir imediatamente o objeto anterior, e devem ser seguidos por pelo menos um espaço.

Separadores de itens são, por exemplo, vírgula e ponto-e-vírgula.

2.5.2.5 Símbolos parentéticos

Regra: Símbolos parentéticos ((), [], { }) devem ter um espaço depois do símbolo de abertura e antes do símbolo de fechamento.

Exemplo em C/C++: `func(fix + var[linha, coluna])`

2.5.3 *Endentação*

2.5.3.1 Tamanho

Regra: A endentação das estruturas de controle deve ocupar de dois a quatro espaços.

O mínimo recomendável é de dois espaços. Usar seis espaços pode melhorar o aspecto, mas constatou-se que dificulta a leitura.

2.5.3.2 Tabulações

Regra: Se a endentação for feita com tabulações, indicar o fato em um comentário colocado no início do programa, que indica o número de espaços por tabulação.

Por exemplo:

```
// Esse programa é endentado com tabulações de quatro espaços.
```

2.5.3.3 Endentação de chaves

Regra: A endentação das chaves de estruturas de controle deve ser consistente.

Existem várias alternativas de endentação para chaves (`begin end`, `{ }`) em estruturas de controle; a alternativa escolhida deve ser usada de forma consistente. [McConnell93] recomenda a alternativa:

```
while ( expressão )
{
    instrução1;
    instrução2;
}
```

argumentando que ela expressa melhor a estrutura do código, por mostrar as chaves como parte integrante da estrutura de controle. Entretanto, prefere-se aqui o estilo implementado automaticamente pelo Microsoft Visual C++:

```
while ( expressão )
{
    instrução1;
    instrução2;
}
```

Deve-se evitar o estilo com dupla endentação:

```
while ( expressão )
{
    {
        instrução1;
        instrução2;
    }
}
```

Os estilos para blocos de uma instrução devem ser consistentes, usando-se as chaves para maior robustez em relação a modificações, mesmo sabendo-se que nesse caso elas não são necessárias. Por exemplo, o estilo consistente com o aqui adotado, para um `if` sem `else`, seria:

```
if ( expressão )
{
    instrução;
}
```

2.5.3.4 Alinhamento de declarações

Regra: Declarações de dados devem ser colocadas uma por linha, com alinhamento do tipo e do nome.

Exemplos:

```
int          id_linha;
```

```
int      id_coluna;
Ponto    inferior_esquerdo;
Ponto    superior_direito;
```

E não:

```
int      id_linha,
         id_coluna;
Ponto    inferior_esquerdo,
         superior_direito;
```

Ou:

```
int      id_linha, id_coluna;
Ponto    inferior_esquerdo, superior_direito;
```

Para melhor estética, recomenda-se agrupar as declarações de acordo com o tamanho do identificador de tipo.

Em C, os asteriscos devem aparecer próximos ao tipo (contrariamente à prática). É melhor ainda definir um tipo apontador. Exemplos:

```
FILE *    arquivo_de_entrada; Ou
FILE_PTR  arquivo_de_entrada;
```

Mas não:

```
FILE      *arquivo_de_entrada;
```

2.5.4 *Linhas em branco*

2.5.4.1 **Espaços entre instruções**

Regra: Instruções correlatas devem ser agrupadas, sem linhas em branco ou endentação desnecessária.

Em particular, declarações devem ser separadas do código executável. Uma exceção é o caso de variáveis de curtíssima duração, como contadores de malha, que devem, de preferência, ser declarados junto à respectiva malha, se a linguagem o permitir.

2.5.4.2 **Espaços para comentários**

Regra: Pelo menos uma linha em branco deve separar um comentário do código precedente.

Blocos de comentário devem ser separados de blocos de código.

2.5.5 *Tamanho das linhas*

2.5.5.1 **Tamanho máximo**

Regra: Linhas de código não devem exceder 80 caracteres.

Esse limite facilita a leitura tanto em papel quando em monitores menores.

2.5.5.2 **Quebra de linhas**

Regra: Quando a instrução não cabe em uma linha de 80 caracteres, parti-las em duas ou mais linhas nos pontos de menor impacto sobre a legibilidade.

Esses pontos geralmente são situados logo após vírgulas ou operadores de baixa precedência, fora de parênteses. O operador ou a vírgula tornam evidente que a linha é incompleta e que a que se segue é uma continuação.

Se, em uma estrutura de controle, a expressão de teste for longa, os parênteses de abertura e fechamento devem ser colocados em linhas separadas e endentados, para que não sejam confundidos com o código a executar. Por exemplo:

```
if
(
    ( ..... ) &&
    ( ..... )
)
{
    ..... ;
    ..... ;
}
```

2.5.5.3 Listas de parâmetros

Regra: Se uma lista de parâmetros for longa demais, de tal modo que a respectiva definição ou chamada de função não caiba em uma linha, cada linha de continuação deve ser alinhada e começada por um parâmetro.

Os parênteses devem ser endentados, e as linhas de parâmetros, por sua vez, endentadas em relação aos parâmetros. Por exemplo:

```
void func
(
    int par_curto_1, float par_curto_2,
    int primeiro_parametro_de_nome_longo,
    float segundo_parametro_de_nome_longo,
    float &terceiro_parametro_de_nome_longo
)
```

Outra opção consiste em colocar um parâmetro por linha, mesmo se for curto. Essa opção facilita a localização do fim do grupo, mas ocupa muito espaço. A opção escolhida deve ser usada de forma consistente.

2.6 Comentários

2.6.1 *Qualidade dos comentários*

2.6.1.1 Maus comentários

Seguem-se alguns exemplos de maus comentários, em ordem crescente de ruindade:

- comentário obsoleto (não atualizado em relação a modificações do código);
- nenhum comentário;
- comentário irrelevante (por exemplo: piadas);
- comentário que explica o que a instrução faz (por exemplo: `A = 1; // Faz A igual a 1.`);

- comentário errado (por exemplo: `A = 1; // Faz A igual a 2.).`

2.6.1.2 Bons comentários

Seguem-se algumas recomendações para a elaboração de bons comentários:

- evitar estilos de comentário difíceis de modificar (por exemplo, delimitados por asteriscos alinhados);
- comentar enquanto se codifica;
- evitar comentários autopromocionais, insultuosos, obscenos, politicamente incorretos e similares;
- evitar comentários no final de linhas de instruções, exceto para declarações e notas de manutenção (tais como índices para uma base de dados de defeitos);
- comentar blocos de instruções, de preferência a instruções individuais;
- focalizar o porquê, de preferência ao como;
- colocar os comentários antes do código ao qual se aplicam;
- evitar comentários de pouca importância;
- documentar surpresas e truques (se não puder evitá-los);
- evitar abreviações;
- manter os comentários próximos ao código que comentam.

2.6.2 Casos especiais

2.6.2.1 Unidades de programa

Regra: Os prólogos de unidades devem descrever as entradas, saídas e hipóteses sobre essa unidade, e não o seu conteúdo.

Exemplos de prólogos:

- módulos - nome, descrição dos parâmetros, descrição funcional;
- funções - descrição dos parâmetros (inclusive direção de passagem), asserções, limitações, restrições, referências a algoritmos e efeitos colaterais.

Informações constantes da gestão de configurações (autor, número e data da revisão etc.) não devem ser duplicadas nos prólogos. Pode-se, entretanto, incluir referências a esses dados, caso a ferramenta de gestão de configurações faça sua manutenção automática. Para fins de revisão de código, essas informações devem ser apresentadas junto com o respectivo código.

Os prólogos devem incluir um sumário do conteúdo da listagem, indicando-se inclusões de código, declarações de classes, declarações de funções e outras partes, na ordem em que aparecem. Os prólogos podem também incluir, quando apropriado:

- instruções para reutilização, indicando-se, para as funções reutilizáveis, o respectivo formato, assim como valores, tipos e limites dos parâmetros;
- instruções para modificação;

- instruções para compilação.

2.6.2.2 Seções de programa

Regra: Seções mais importantes de programa devem ser precedidas por um comentário de bloco.

Esse comentário descreve o processamento que deve ser feito na seção, focalizando seu objetivo.

2.6.2.3 Declarações de dados

Regra: Em declarações de dados, usar os comentários para descrever aspectos da variável que não são descritos por seu nome.

Por exemplo:

- unidades de dados numéricos;
- faixa de valores aceitáveis;
- significados codificados;
- limitações sobre dados de entrada;
- indicadores a nível de bit.

O nome da variável no comentário deve ser sempre consistente com seu nome no código. Dados globais devem ser sempre comentados.

2.6.2.4 Comentários de linha

Regra: Usar com cuidado os comentários de linha.

Os comentários de linha são aqueles que aparecem no final das linhas de código:

```
instrução // comentário
```

Eles devem ser evitados:

- em linhas isoladas (tendem a repetir o que a linha faz);
- quando se aplicam a múltiplas linhas.

Eles podem ser usados:

- para comentar declarações de dados;
- para anotações de manutenção;
- para marcar o fim de blocos (exemplo: `end // if`).

2.6.2.5 Manutenção

Regra: Todos os comentários devem estar sempre atualizados.

Todo programador que altera o código é também responsável por alterar os respectivos comentários e outras formas de documentação. Deve-se evitar o uso de estilos de comentário que desencorajem as modificações (por exemplo, com *leiaute complexo*).

2.6.3 Código inacabado

2.6.3.1 Classificação

A permanência de código não definitivo é indesejável, mas às vezes não pode ser evitada. O código inacabado tem vários graus de severidade:

- código não funcional;
- código parcialmente funcional;
- código suspeito;
- código que precisa de aperfeiçoamento.

Destes tipos, o código parcialmente funcional é o mais perigoso, pela tendência que tem a se tornar definitivo. Todos os tipos, entretanto, devem ser marcados, de forma que seja possível localizá-los com facilidade. Além disso, devem-se marcar os trechos de código cuja documentação correspondente ainda não foi atualizada.

2.6.3.2 Marcação

Regra: Todo código não definitivo deve ser marcado através de um tipo de comentário padronizado.

Exemplos em C++:

```
// #defeito# funcional
// #defeito# semi-funcional
// #defeito# suspeito
// #defeito# melhoria
// #defeito# documentação
```

2.6.4 Referências a outros documentos

Regra: Toda referência a outros documentos, embutida em comentários, deve ser marcada de forma padronizada.

Exemplo em C++

```
// #referencia# nome do item referenciado
```

O nome do item referenciado deve identificar o documento e o item específico associado ao código em questão. Para documentação on-line, o uso de hiperligações é recomendável.

3 Diretrizes para C++

3.1 Diretrizes aplicáveis a C

3.1.1 Unidades de programa

3.1.1.1 Modularização

Regra: Toda comunicação entre módulos deve ser feita através de arquivos de cabeçalho (.h).

Todas as definições de variáveis externas, funções exportadas, classes públicas, tipos públicos e constantes públicas devem ser colocadas nos arquivos de cabeçalho.

3.1.1.2 Protótipos de função

Regra: Todas as funções que aparecerem em um arquivo fonte devem ter um protótipo associado.

As funções estáticas internas devem ser definidas no início do arquivo fonte, e as funções exportadas devem ser definidas no arquivo de cabeçalho. A ordem dos protótipos deve casar com a ordem de aparecimento das funções.

3.1.1.3 Retornos de função

Regra: As funções não devem retornar valores com significado diferente através de um único parâmetro ou resultado de retorno.

Por exemplo, pode-se retornar um código de êxito como resultado de uma função, enquanto o resultado efetivo do processamento é devolvido em um parâmetro por referência. Essa regra não é obedecida por muitas funções de bibliotecas-padrão de C.

Por outro lado, em C++, é preferível usar os recursos de tratamento de exceção da linguagem do que recorrer a códigos de retorno.

3.1.1.4 Reutilização

Regra: Os arquivos incluídos reutilizáveis devem diferenciar a primeira inclusão das demais.

Isso é feito através de instruções de pré-processamento, do tipo:

```
#ifndef NOME_DO_MODULO

// Instruções que só são executadas na primeira inclusão.

#endif

#define NOME_DO_MODULO

// Instruções executadas em todas as inclusões.
```

3.1.2 Testabilidade

3.1.2.1 Asserções

Regra: Deve-se usar asserções para checar os parâmetros de entrada para as funções, assim como todos os casos degenerados e condições supostamente impossíveis.

As asserções são testadas através da função `assert`, presente no cabeçalho padrão `assert.h`. Quando a asserção falha, essa função imprime literalmente a expressão que lhe for passada como parâmetro em uma mensagem de erro ou caixa de diálogo.

Essa função é ativada quando o identificador `DEBUG` é definido. Esse identificador deve ser sempre definido nas versões de depuração e não definido nas versões de produção.

3.1.2.2 Código de depuração

Regra: Todo código de depuração deve ser desativado quando `DEBUG` é não definido.

Isso pode ser feito delimitando-se todo código de depuração com:

```
#ifdef DEBUG

...

#endif
```

O código de depuração geralmente pertence a um dos seguintes tipos:

- código que compensa deficiências do ambiente de depuração quanto a localizar determinados tipos de defeito;
- código que registra resultados para análise posterior, e não faz parte da versão de produção;
- código que pode ser ativado para fins de manutenção.

Pode-se ativar subconjuntos de instruções de depuração, encapsulando-se essas em um segundo nível de `#ifdef` (Tabela 212).

```

#ifdef DEBUG

#ifdef Testar_Rekursividade

// Código específico para testar rotinas recursivas
...
// Fim do teste de recursividade

#endif

#endif

```

Tabela 212 – Exemplo de encapsulamento de subconjuntos de instruções de depuração

3.1.3 Tipos em C

3.1.3.1 Textos

No caso de textos em C:

- prestar atenção nas diferenças entre apontadores para textos e arranjos de caracteres;
- declarar o tamanho dos textos no formato `CONSTANTE+1` (reduz erros de tamanho da alocação);
- iniciar os textos com o caráter nulo;
- preferir arranjos de caracteres a apontadores a textos;
- usar `strncpy()` e `strncmp()` no lugar de `strcpy()` e `strcmp()` (versões mais seguras, por não dependerem de terminação correta).

3.1.3.2 Apontadores

No caso de apontadores em C:

- evitar apontadores `void`;
- minimizar o uso de moldes (`cast`);
- regra do asterisco: lembrar-se de que parâmetros só podem ser alterados em uma rotina se possuírem um asterisco do lado esquerdo de uma atribuição;
- usar `sizeof()` para determinar o tamanho na alocação de memória.

3.2 Transição de C a C++

3.2.1 Constantes

Regra: Usar **const** e **inline** em lugar de **#define**.

Por exemplo, em lugar de:

```
#define PI 3.141592
```

deve-se preferir

```
const double PI = 3.141592
```

No caso de declaração de apontadores constantes, tanto o apontador quanto o objeto apontado devem ser `const`:

```
const char * const companhia = "United_Hackers";
```

No caso de macros, em lugar de

```
#define MAX( a, b ) ( ( a ) > ( b ) ? ( a ) : ( b ) )
```

deve-se preferir

```
template <class T>
inline T& MAX (T& a, T& b)
{
    return a > b ? a : b;
}
```

Uma versão mais simples para um tipo específico (no caso, ints) é:

```
inline int MAX (int a, int b) { return a > b ? a : b; }
```

3.2.2 *Entrada e saída*

Regra: Preferir **iostream** a **stdio**.

As funções da Tabela 213 compõem a biblioteca de entrada e saída do C++.

<pre>// Substituem printf/scanf istream, ostream, iostream // Substituem fprintf/fscanf ifstream, ofstream, iofstream //Substituem sprintf/sscanf istrstream, ostrstream, strstream</pre>

Tabela 213 – Biblioteca de entrada e saída em C++

```

#include <iostream>
using namespace std;

class ComplexInt
{
private:
    int    r, i;
    // Partes real e imaginária
public:
    ComplexInt( int realPart = 0, int imagPart = 0 );
    friend ostream&
        operator <<( ostream& s, const ComplexInt& c );
};

// Emite o valor ( x, y ) como x + yi ou x - yi
ostream& operator <<( ostream& s, const ComplexInt& c )
{
    // Emite a parte real
    s << c.r;

    // Se a parte imaginária for diferente de 0, emite-a com o sinal correto
    //
    if ( c.i != 0 )
    {
        if ( c.i > 0 )
            s << '+';
        else
            s << '-';
        s << c.i << 'i';
    }

    // Retorna a ostream para permitir encadeamento
    return s;
}

```

Tabela 214 – Exemplo de saída de números complexos em C++

Classes de objetos persistentes, a serem guardados em arquivos, devem ter os operadores de inserção (<<) e extração (>>) definidos, para que possam se beneficiar dessas bibliotecas. A Tabela 214 ilustra um exemplo de saída de números complexos.

3.2.3 Comentários

Regra: Usar comentários estilo C++.

Os comentários estilo C++ (//) são superiores aos comentários C (/*...*/) em termos de legibilidade. Além disso são menos sujeitos a erros no caso de código temporariamente desativado.

3.3 Gestão de memória

3.3.1 Formas de alocação de memória

Regra: Usar **new/delete** em lugar de **malloc/free**.

New e delete são melhores porque interagem corretamente com construtoras e destruidoras. O pior caso é misturar new/delete com malloc/free e suas variantes.

3.3.2 Construtoras

3.3.2.1 Construtoras de cópia

Regra: Deve-se definir um operador de atribuição e uma construtora de cópia para todas as classes que contenham objetos alocados dinamicamente.

A construtora de cópia é uma construtora na qual é passado, como único parâmetro, um objeto da respectiva classe. Por exemplo:

```
class Qualquer
```

```
{
public:
    Qualquer( const Qualquer& x );
}
```

Normalmente, a construtora de cópia é usada para copiar o objeto argumento em outro objeto, através do operador de atribuição (=):

```
Qualquer y = Qualquer( x );
```

Na ausência de uma redefinição do operador de atribuição, é gerada pelo compilador uma cópia membro a membro. Isso significa, no caso dos membros alocados dinamicamente, que seus apontadores serão copiados, mas não eles próprios. Essa situação ocorre também na passagem de parâmetros por valor, em que há uma atribuição implícita.

Uma consequência será o fato de que a destruição de um dos objetos destruirá os objetos internos apontados. Por exemplo, se x for destruído, seus objetos dinâmicos internos serão destruídos, deixando os apontadores internos de y pendentes (apontando para lugar nenhum). No mínimo, isso leva ao vazamento de memória.

A implementação do operador de atribuição personalizado pode copiar cada objeto apontado, ou pode manter algum esquema de contagem de referências a esses objetos. Essa última solução é muito mais complexa, e deve ser usada apenas quando for realmente necessário.

3.3.2.2 Iniciação

Regra: Nas construtoras deve-se usar a iniciação dos atributos, de preferência à atribuição.

Por exemplo, seja a classe da Tabela 215. Deve-se preferir a construtora da Tabela 216 à alternativa da Tabela 217.

```
class Dados_com_nome
{
private:
    String  nome;
    void    *dados;

public:
    Dados_com_nome( const String& nome_inicial, void *aptdados );
}
```

Tabela 215 – Exemplo de classe com construtora

```
Dados_com_nome::Dados_com_nome( const String& nome_inicial,
    void *aptdados )
: nome( nome_inicial ), dados( aptdados )
{ }
```

Tabela 216 – Exemplo de construtora com iniciação dos atributos

```
Dados_com_nome::Dados_com_nome( const String& nome_inicial,
    void *aptdados )
{
    nome = nome_inicial;
    dados = aptdados;
}
```

Tabela 217 – Exemplo de construtora com atribuição dos atributos

As razões são as seguintes:

- atributos const e referência podem sofrer iniciação, mas não atribuição;

- nos outros casos, a iniciação é mais eficiente, porque evita pelo menos uma chamada adicional à construtora de cada atributo.

3.3.2.3 Lista de iniciação

Regra: Os atributos devem aparecer na lista de iniciação na mesma ordem em que são declarados.

Os atributos sempre são iniciados em sua ordem de declaração, qualquer que seja a ordem de aparição na lista.

Suponha-se que um atributo x depende de outro atributo y para sua iniciação. Se colocarmos x depois de y na lista de iniciação, o leitor do código terá a impressão de que a ordem de iniciação está correta. Na realidade, se x for declarado antes de y , a ordem estará errada, pois a ordem que importa é a ordem das declarações, e não a ordem de aparecimento na lista de iniciação. Para maior clareza, essas duas ordens devem ser consistentes entre si.

3.3.3 Destruidoras

3.3.3.1 delete em destruidoras

Regra: Chamar **delete** sobre membros apontadores em destruidoras.

É preciso chamar **delete** para todos os membros apontadores para os quais foi chamada **new** em alguma construtora. Caso contrário, introduz-se um vazamento de memória. A memória correspondente não é liberada, e, como não é mais apontada por nada, não pode mais ser utilizada.

3.3.3.2 Virtualidade

Regra: Destruidoras em superclasses devem ser virtuais.

Isso faz com que, ao ser destruído um objeto de uma classe derivada, sejam chamadas as destruidoras desta classe e de todas as classes superiores no grafo de herança.

Por outro lado, toda superclasse deve conter pelo menos uma outra função virtual. Uma classe que não tenha funções virtuais não foi desenhada para servir de superclasse. Tornar a destruidora virtual apenas introduzirá gratuitamente o overhead do mecanismo de virtualidade.

3.4 Desenho e declaração de classes e funções

3.4.1 Interfaces

3.4.1.1 Completeza e minimalismo

Regra: As interfaces das classes devem ser completas e mínimas.

Uma interface completa é aquela que oferece aos clientes²⁸ todas as operações de que possam razoavelmente precisar. Uma interface mínima é aquela na qual não há superposição de funcionalidade entre os métodos.

Interfaces com número grande demais de métodos públicos apresentam várias desvantagens:

- dificuldade de entendimento por parte dos clientes;
- dificuldade de manutenção;
- compilação mais demorada devido ao tamanho dos arquivos de cabeçalho.

Por outro lado, pode ser o caso de introduzir alguns métodos com redundância para:

- tornar mais eficientes tarefas de realização freqüente;

²⁸ Clientes, neste contexto, são os programadores que podem vir a utilizar uma classe. A interface é o conjunto das operações públicas, e não a interface no sentido da UML.

- facilitar o uso da classe;
- evitar erros por parte dos clientes.

3.4.1.2 Diferenciação de funções

Regra: Distinguir entre métodos, funções globais e funções amigas.

Funções que têm de ser amarradas dinamicamente têm de ser virtuais, e portanto têm de ser métodos. A amarração dinâmica ocorre quando um apontador declarado para uma superclasse passa a apontar para um objeto de uma classe derivada. Quando uma função da superclasse é redefinida pela classe derivada, a amarração dinâmica permite que o tipo do objeto apontado determine qual função deve ser executada.

Mesmo que a função não tenha que ser virtual, em geral a realização das operações como método permite melhor encapsulamento e, por isso, está mais de acordo com a filosofia orientada a objetos.

Em alguns casos, principalmente de operadores binários, as funções globais podem ser mais adequadas. Isso ocorre quando se quer permitir a realização de conversões de tipo sobre o lado esquerdo do operador. Por exemplo, suponha-se que o operador de multiplicação `*` seja definido para uma classe `Racional`. Suponha-se o seguinte código:

```
int    a;

Racional  b, c, d;

c = b * a;

d = a * b;
```

O código da terceira linha é correto se `operator*` for um método da forma:

```
Racional operator*( const Racional& ldir ) const;
```

Nesse caso, a variável `a` será convertida corretamente de `int` para `Racional`. Por outro lado, o código da quarta linha será incorreto, porque o compilador não saberá como fazer a conversão. Para isso, `operator*` deve ser implementado como global, embora faça parte da interface de `Racional`, do ponto de vista lógico:

```
Racional operator*( const Racional& lesq, const Racional& ldir );
```

A função global deverá ser declarada como amiga apenas se precisar de acesso aos métodos privados de uma classe. Isso é o que normalmente acontece com os operadores de extração (`operator >>`) e inserção (`operator <<`) de classes persistentes. Esses operadores têm de ser globais para poderem ser associativos²⁹, mas têm de ter acesso aos atributos (que devem ser privados, conforme a Subseção seguinte), para realizar o salvamento e a recuperação dos objetos nos fluxos de entrada e saída.

3.4.1.3 Atributos

Regra: Atributos não devem ser públicos.

Os acessos aos atributos devem sempre ser intermediados por métodos. Isso traz várias vantagens:

- facilidade de uso: todos os acessos são feitos por métodos;
- controle mais preciso de acesso: por exemplo, pode-se fazer com que os dados sejam acessíveis para leitura, mas não para atualização;
- abstração funcional: por exemplo, em versões diferentes de uma classe, métodos do mesmo nome podem devolver um valor de atributo ou um valor calculado.

²⁹ E portanto usáveis de forma encadeada, como é normal na entrada e saída estilo C++.

3.4.2 *Passagem de parâmetros*

3.4.2.1 **Uso de `const`**

Regra: **`const`** deve ser usado sempre que for possível.

Usar `const` permite que o compilador vigie determinados objetos que **não podem** ser modificados. Deve-se notar que existem alguns casos particularmente importantes de uso de `const`.

No caso de apontadores, tanto o objeto apontado quanto o apontador podem ser `const`:

```
char * p;           // Apontador não-const, objeto não-const.
const char * p;     // Apontador não-const, objeto const.
char * const p;     // Apontador const, objeto não-const.
const char * const p; // Apontador const, objeto const.
```

Para interpretar essas declarações, basta lembrar que:

- Quando `const` está à **esquerda** do asterisco, o **objeto apontado** é `const`.
- Quando `const` está à **direita** do asterisco, o **apontador** é `const`.

No caso de funções, pode-se ter:

- parâmetros `const`;
- valores de retorno `const`;
- métodos `const`, que só podem ser invocados sobre objetos `const`.

3.4.2.2 **Passagem de objetos**

Regra: Passar e retornar objetos por referência, exceto no caso de retorno de objetos construídos dentro da função.

A passagem de objetos por referência é mais eficiente; toda passagem de objeto por valor significa uma chamada a uma construtora de cópia. Além disso, se objetos de uma classe herdeira forem passados por valor, como parâmetros de uma superclasse (o que é permitido), será construído internamente à função um objeto cópia da superclasse, perdendo-se portanto a informação adicional contida na classe herdeira.

Entretanto, se o resultado de um método for construído dentro dele, ele deve ser retornado por valor. Por exemplo, vide um operador de adição de números complexos (Tabela 218).

```

Class Complex
{
private:
    double    r, i;
...

public:
    Complex( double parteReal = 0, parteImag = 0 );
    friend complex operator+( const Complex& lesq,
                              const Complex& ldir );
};

inline complex operator+( const Complex& lesq,
                          const Complex& ldir )
{
    return Complex( lesq.r + ldir.r, lesq.i + ldir.i );
}

```

Tabela 218 – Exemplo de passagem de objetos

O retorno do operador + é feito por valor, envolvendo explicitamente uma chamada a uma construtora. Construir uma variável temporária interna e devolver uma referência a essa, além de continuar exigindo uma chamada a uma construtora, é muito menos seguro.

3.4.3 Retorno

3.4.3.1 Retorno de informação de acesso

Regra: Evitar a devolução de informação de acesso a dados internos de objetos, através de métodos **const**.

Só métodos **const** podem ser invocados sobre objetos **const**. Entretanto, a constância desses objetos pode ser violada se o método devolver uma informação de acesso (apontador ou referência) a um dado interno do objeto.

3.4.3.2 Retorno de apontadores

Regra: Evitar que métodos retornem referências ou apontadores a métodos e atributos menos acessíveis que eles mesmos.

Através de tais referências ou apontadores pode-se anular a segurança que se pretendeu embutir no desenho da classe, através do uso de métodos e atributos privados e protegidos. Se for indispensável fazer esse tipo de acesso, por razões prementes de desempenho, o método ou atributo apontado deve pelo menos ser declarado **const**.

3.4.3.3 Retorno de referências

Regra: Evitar o retorno de referências a objetos locais ou objetos internos alocados dinamicamente.

Objetos locais são destruídos ao sair-se do método, e portanto já não existem quando a suposta referência a eles é retornada.

No caso de objetos alocados dinamicamente, ocorre outro tipo de problema. Valores de retorno são freqüentemente atribuídos a variáveis temporárias geradas pelo compilador, quando o método é invocado de dentro de uma expressão. Como estas variáveis são anônimas, não há como destruir os objetos apontados, e ocorre vazamento de memória.

3.5 Herança

3.5.1 Modelagem e herança

3.5.1.1 Relacionamentos “é”

Regra: Usar a herança pública para modelar relacionamentos “é”.

Relacionamentos “é” devem ser modelados através da herança pública, e a herança pública só deve modelar relacionamentos “é”. Se a classe D deriva publicamente da classe B, isso deve refletir o seguinte desenho: todo objeto D é um objeto B, mas nem todo objeto B é um objeto D.

3.5.1.2 Relacionamentos “tem”

Regra: Usar camadas de objetos para modelar relacionamentos “tem”.

A **inclusão** (*embedding*), **continência** (*containment*) ou a **estratificação** (*layering*) entre as classes A e B ocorre quando um objeto A contém (por valor ou referência) um objeto B como atributo. É a forma natural de modelar os relacionamentos “tem”.

3.5.1.3 Relacionamentos de implementação

Regra: Usar a inclusão ou a herança privada para modelar relacionamentos de implementação.

Existe um relacionamento de implementação entre as classes A e B quando A é implementada através de B; é um tipo particularmente forte de relacionamento de dependência. Por exemplo, uma classe `Conjunto` pode ser implementada através de uma classe `Lista_Encadeada`. Não existe relacionamento “é” entre estas duas classes; por exemplo, `Conjunto` é uma coleção não ordenada e a `Lista_Encadeada` é uma coleção ordenada. Operações que supõem ordenação, como o operador de indexação `[]`, não fazem sentido para conjuntos.

Em uma implementação por **inclusão**, um objeto `Lista_Encadeada` é um atributo de `Conjunto`. As operações de `Conjunto` invocam as operações dessa `Lista_Encadeada` interna.

A inclusão, na maioria dos casos, deve ser a maneira preferida para reutilização de implementações. Entretanto, ela não permite reutilizar funções protegidas, ou redefinir funções virtuais. A alternativa é usar a **herança privada**. Esta difere da herança pública porque:

- apontadores para objetos de classe derivada **não** são convertidos em apontadores para objetos de superclasse;
- membros públicos e protegidos de superclasse tornam-se membros **privados** da classe derivada.

A classe derivada privadamente, portanto, não **repassa** a seus clientes a interface da superclasse, como ocorreria se houvesse um relacionamento “é”.

3.5.1.4 Herança versus gabaritos

Regra: Distinguir entre herança e gabaritos.

Gabaritos (*templates*) devem ser usados para gerar uma coleção de classes, quando o tipo dos objetos não afeta o comportamento das classes da coleção. Por exemplo, as operações `Empilhar` e `Desempilhar` funcionam do mesmo modo para uma classe de inteiros, de Complexos, de Formas etc.

```

class Forma
{
public:
    virtual void desenhar( ) = 0;           //Função virtual pura
    ...
};

class Circulo: public Forma
{
public:
    void desenhar( );
    ...
};

class Retangulo: public Forma
{
public:
    void desenhar( );
    ...
};

```

Tabela 219 – Exemplo de redefinição de operações virtuais em rede de herança

Por outro lado, a herança deve ser usada quando o tipo de objeto afeta seu comportamento. Por exemplo, suponha-se a rede de herança da Tabela 219. Nesse caso, `desenhar()` é uma operação realizada de maneira diferente para cada subclasse de `Forma`.

3.5.2 Tipos de herança

3.5.2.1 Herança de interface e de implementação

Regra: Distinguir entre heranças de interface e de implementação.

As interfaces dos métodos são sempre herdadas. O que ocorre com as implementações depende do grau de virtualidade do método.

- O objetivo dos métodos virtuais puros é definir apenas uma interface. Cabe sempre às classes derivadas definir a implementação.
- O objetivo dos métodos virtuais simples é definir uma interface e uma implementação padrão. Esta implementação pode ser ou não redefinida pelas classes derivadas.
- O objetivo dos métodos não virtuais é definir uma interface e uma implementação obrigatória (isto é, o método da classe básica não pode ser redefinido).

3.5.2.2 Herança múltipla

Regra: Usar com cuidado a herança múltipla.

O uso da herança múltipla pode acarretar a ambigüidade. Se duas superclasses têm métodos com o mesmo nome, a invocação destes métodos em objetos da classe derivada deve explicitar a classe da qual se deseja herdar (Tabela 220).

```

class A
{
public:
    virtual void func( ... );
    ...
};

class B
{
public:
    virtual void func( ... );
    ...
};

class C: public A, public B
{
    ...                //Não re-declara func
};

A *apa = new C;
apa->func(...);        // Erro - ambigüidade.
apa->A::func(...);     // Correto - invoca func de A.
apa->B::func(...);     // Correto - invoca func de B.

```

Tabela 220 – Exemplo de herança múltipla ambígua

```

class A { ... };
class B: public A { ... };
class C: public A { ... };
class D: public B, public C { ... };

```

Tabela 221 – Heranças não virtuais em losango

```

class A { ... };
class B: virtual public A { ... };
class C: virtual public A { ... };
class D: public B, public C { ... };

```

Tabela 222 – Heranças virtuais em losango

Outro tipo de problema acontece com a herança em losango. Se as heranças de A não forem virtuais (Tabela 221), cada objeto de D conterá uma cópia duplicada de A. Por outro lado, se as heranças de A forem virtuais (Tabela 222), haverá apenas uma cópia de A, mas introduz-se um overhead de apontamento. A maior dificuldade é que A pode fazer parte de uma biblioteca cujo desenhista não sabe, a priori, de que forma a classe será derivada.

Outros problemas com a herança múltipla aparecem na passagem de argumentos de construtoras a superclasses virtuais, nas relações de dominância entre funções virtuais, e em restrições aos moldes (*casts*); vide detalhes em [Meyers92].

Por outro lado, a herança múltipla pode ser muito útil quando se deseja modelar objetos que têm algumas características em comum com um grupo de objetos (uma classe), e outras tantas características em comum com outros tipos de objetos (outra classe) [Booch94]. Se essas superclasses, por sua vez, não tiverem uma ancestral comum, evita-se o diagrama de herança em losango e a necessidade de herança virtual.

Além disso, uma herança múltipla pode ser usada para modelar um relacionamento “é” (herança pública) que ocorre junto com um relacionamento de implementação (herança privada):

```

class C: public A, private B { ... };

```

3.5.3 *Redefinição*

3.5.3.1 Métodos não virtuais

Regra: Nunca redefinir um método não virtual herdado.

Métodos não virtuais definem uma implementação obrigatória. Se uma classe derivada redefine essa implementação, está violando o modelo. Na prática, isso leva a situações bastante confusas (Tabela 223). O comportamento do objeto X passa a ser diferente, para o mesmo método, conforme o apontador utilizado.

```
class A
{
public:
    void func( ... );
};

class B : public A
public:
    void func( ... );
};

B *X;                // Cria objeto X da classe B.
A *apa = &X;         // Cria um apontador para X.
B *apb = &X;         // Cria outro apontador para X.
apa->func( ... );     // Chama A::func( ... )
apb->func( ... );     // Chama B::func( ... )
```

Tabela 223 – Redefinição de método não virtual

3.5.3.2 Valores padrão de parâmetros

Regra: Nunca redefinir um parâmetro herdado com valor padrão.

Métodos virtuais exibem **amarração dinâmica**: o comportamento é determinado pelo tipo de objeto apontado, e não do apontador. Valores padrão exibem **amarração estática**: o valor atribuído é determinado pelo tipo do **apontador**.

Assim, se B é uma superclasse, D uma classe derivada de B, e `func(...)` uma função virtual de B que é redefinida por D, então teremos o seguinte comportamento:

```
B *apb;

D *apd = new D;

apb = apd;

apb->func( ... );
```

A última linha invoca, sobre um objeto da classe D, a função `func(...)` definida pela classe D. Os valores-padrão usados nos parâmetros de `func(...)`, entretanto, serão aqueles definidos na `func(...)` da classe B, ignorando-se os valores-padrão definidos na `func(...)` da classe D.

3.5.4 Conclusões

Várias regras estabelecem como o código deve refletir o modelo, e devem ser sempre observadas:

- superclasses modelam características comuns a diferentes classes derivadas;
- herança pública modela relacionamentos “é”;
- herança privada ou inclusão modelam relacionamentos de implementação;
- inclusão modela relacionamentos “tem”.

3.6 Compilação

3.6.1 Erros de compilação e de ligação

Regra: Preferir erros de compilação e ligação a erros de execução.

Erros de compilação e ligação são, obviamente, muito mais fáceis de corrigir. Em certas situações, modificações no desenho detalhado fazem com que uma checagem seja transferida do tempo de execução para o tempo de compilação ou ligação.

Por exemplo, considere-se que um parâmetro de uma função pode representar um número de mês. Se for usado um inteiro, será necessária uma checagem de tempo de execução, para verificar se cada parâmetro passado está no intervalo [1,12]. Essa checagem pode ser dispensada se o mês, em lugar de um inteiro, for representado por uma classe na qual os possíveis valores do mês são definidos como constantes estáticas.

3.6.2 Avisos do compilador

Regra: Prestar atenção aos avisos de compilação.

Geralmente, todo aviso indica um problema potencial. Uma categoria de avisos só pode ser ignorada quando suas conseqüências são completamente entendidas.

3.7 Tópicos avançados

Vide [Meyers96] para um conjunto de recomendações a respeito de tópicos avançados em C++, que incluem:

- práticas recomendáveis, como:
 - distinguir entre referências e apontadores;
 - evitar construtoras sem iniciação;
 - quando definir o operador `op`, definir também o operador `op=`;
 - tornar abstratas as classes não terminais da estrutura de herança.
- recursos de novas versões da linguagem, como:
 - moldes estilo C++;
 - exceções;
 - identificação de tipo em tempo de execução (RTTI);
- técnicas avançadas de programação, como:
 - avaliação preguiçosa (*lazy evaluation*);
 - limitação do número de objetos de uma classe;
 - apontadores inteligentes (*smart pointers*);
 - contagem de referências (*reference counting*);
 - classes procuradoras (*proxy classes*).

4 Revisões da Implementação

4.1 Lista de conferência para o desenho detalhado

A lista a seguir, baseada em [Humphrey95] e [McConnell93], deve ser percorrida na ordem aqui apresentada. Cada item deve ser verificado para cada uma das unidades sob inspeção. Só se deve passar ao item seguinte quando todas as unidades tiverem sido checadas quanto ao item anterior. Um item com divisões só deve ser considerado checado quando todas as suas subdivisões estiverem checadas.

As questões de completeza e modularização devem ser respondidas para cada módulo. As demais devem ser respondidas para cada rotina.

Completeza	Todos os requisitos e o desenho de alto nível são completamente cobertos pelo desenho detalhado.
	Todas as saídas especificadas são produzidas.
	Todas as entradas necessárias são fornecidas.
	Todas as inclusões necessárias são declaradas.
Modularização	Tem um propósito central.
	Tem um conjunto comum de dados.
	Oferece um conjunto coerente de serviços.
	Tem alta coesão interna.
	Tem baixo acoplamento com outros módulos.
	Esconde os detalhes de implementação.
	Implementa as convenções adotadas para modularização.

Tabela 224 - Lista de conferência para desenho detalhado – parte 1

Rotinas – aspectos gerais	A rotina é necessária.	
	A rotina é suficiente.	
	A coesão interna é forte.	
	O acoplamento com outras rotinas é fraco.	
	Os pré-requisitos estão satisfeitos.	
	O nome é um verbo (com objeto) ou descritor do retorno.	
	O nome descreve a rotina de forma completa.	
	O teste de unidade foi planejado.	
	O nível adequado de eficiência foi considerado.	
	Há referências às fontes dos algoritmos usados.	
	O tamanho da rotina é natural.	
	O tratamento de exceções é adequado.	
Pseudolinguagem	É fácil de entender.	
	É consistente com o desenho de alto nível.	
	Foi acuradamente traduzido para o código.	
	Foi transformado em comentários adequados.	
Interfaces das rotinas	A lista de parâmetros tem ordem adequada.	
	A lista de parâmetros tem tamanho adequado.	
	As suposições de interface são documentadas.	
	A rotina se defende contra dados errados.	
	A rotina checa os valores de retorno.	
	Todos os parâmetros são usados.	
	A decisão entre passar variável estruturada como um parâmetro, ou os campos pertinentes como parâmetros separados, foi tomada de forma correta.	
	Se houver um valor de retorno, esse é definido em todos os casos possíveis.	
	O uso de variáveis globais é minimizado.	
Lógica	Todas as estruturas de dados são percorridas na ordem correta.	
	Todas as malhas são corretamente iniciadas, incrementadas e terminadas.	
	Todas as recursões são corretamente terminadas.	
Casos especiais	Correção para valores vazio, cheio, mínimo, máximo, negativo e zero, quando aplicáveis, para todas as variáveis.	
	Condições de ultrapassagem de limites, <i>overflow</i> e <i>underflow</i> .	
	Condições supostamente impossíveis são realmente impossíveis.	
	Funcionalidade	Todas as funções e classes são perfeitamente entendidas e usadas da forma correta.
		Todas as referências a abstrações externas são definidas precisamente.
	Nomes	Todos os nomes e tipos especiais são definidos de forma clara e específica.
		Os escopos de todas as variáveis e parâmetros são definidos ou evidentes.
		Todos os identificadores são usados dentro dos escopos declarados.

Tabela 225 - Lista de conferência para desenho detalhado – parte 2

4.2 Lista de conferência para código

A lista a seguir, na Tabela 226, também baseada em [Humphrey95] e [McConnell93], é orientada para C++, mas a maioria dos itens é aplicável a outras linguagens. Ela deve ser percorrida na ordem aqui apresentada. Tal como na revisão do desenho detalhado, cada item deve ser verificado para cada uma das unidades sob inspeção, e só se deve passar ao item seguinte quando todos as unidades tiverem sido verificadas quanto ao item anterior. Um item com divisões só deve ser considerado checado quando todas as suas subdivisões estiverem checadas.

Completeza	O código cobre todo o desenho detalhado.
Inclusões	As inclusões necessárias estão completas.
Iniciação (de parâmetros e variáveis)	No início dos programas.
	No começo de cada malha.
	Na entrada de todas as funções.
	Em contadores e acumuladores.
	Reiniciação em código repetido.
Chamadas	Uso dos parâmetros.
	Uso de apontadores.
	Uso de referências.

Tabela 226 – Lista de conferência para código – parte 1

Tipos em geral	Os tipos de dados são suficientemente robustos em relação a possíveis mudanças.
	Os nomes dos tipos são orientados para entidades do mundo real.
	Os nomes dos tipos são suficientemente descritivos.
	Tipos predefinidos da linguagem não foram redefinidos.
Tipos numéricos em geral	São evitados literais diferentes de 0 e 1.
	Não é possível que ocorram divisões por 0.
	As conversões de tipo são evidentes.
	São evitadas comparações com tipos mistos.
Inteiros	São evitados erros de truncamento.
	São evitados erros de estouro (<i>overflow</i>).
Números de ponto flutuante	São evitadas somas e subtrações de números de magnitude muito diferente.
	São considerados os erros de arredondamento.
	São evitadas as comparações de igualdade.
Textos e caracteres	São usadas constantes de texto no lugar de literais.
	As constantes de texto são agrupadas em arquivos de recursos.
Textos em C	Recebem o caráter nulo como valor inicial.
	São terminados corretamente com o caráter nulo.
	As declarações de textos usam a forma [CONSTATANTE + 1].
	Apontadores de texto e arranjos de caracteres são tratados de forma diferente.
	Arranjos de caracteres são utilizados preferencialmente.
	Usam a forma mais segura das funções de biblioteca de cópia.
Booleanos	São usados de forma autodocumentada.
	São usados para simplificar expressões complexas.
Tipos enumerados	São usados para tornar o código mais legível, confiável e modificável.
	Os valores inválidos são testados.
	Reserva-se uma entrada para valores inválidos.
Arranjos	Os índices caem sempre dentro dos limites do arranjo.
	Os subscritos de arranjos multidimensionais estão na ordem correta.
	Os subscritos em laços aninhados estão na ordem correta.
Apontadores	São isolados preferencialmente em funções especializadas.
	Têm valor inicial nulo.
	São checados antes de usados.
	As variáveis apontadas são checadas antes de usadas.
	Só são liberados depois de alocados.
	Sempre são liberados depois de alocados.
	É previsto o tratamento da falta de memória.
	Todos os apontadores de listas são liberados na ordem correta.

Tabela 227 – Lista de conferência para código – parte 2 (tipos)

Nomes	São consistentes.	
	Descrevem as entidades representadas de forma completa e acurada.	
	São orientados para o domínio do problema.	
	São abreviados apenas quando necessário, e, nesse caso, de forma adequada e consistente.	
	São pronunciáveis.	
	Não se enquadram nas seguintes situações que induzem à confusão.	São enganosos.
		Têm significados semelhantes.
		Diferem apenas por um ou dois caracteres, principalmente aqueles que são fáceis de confundir.
		Têm pronúncia igual ou semelhante.
		Usam numerais.
		Usam grafia errada.
		São grafados erradamente por muitas pessoas.
		São arbitrários.
		Conflitam com nomes padronizados do ambiente.
		Usam caracteres acentuados ou especiais.
	Casos específicos	Mesmo para índices e variáveis temporárias, os nomes são significativos.
		Para variáveis booleanas, o significado de Verdadeiro e Falso é claro.
		Para enumerações, o nome indica o grupo.
		Para constantes, o nome indica o significado, e não o valor.
		A denominação de membros de classes e estruturas é correta.
Variáveis	Têm o menor escopo possível.	
	As referências às variáveis são agrupadas por proximidade.	
	Cada variável tem uma e apenas uma finalidade.	
	Nenhuma variável tem significado múltiplo.	
	Todas as variáveis declaradas são usadas.	
	As variáveis globais, quando indispensáveis, são claramente marcadas:	
	Através de convenções de denominação.	
	Através de documentação.	

Tabela 228 - Lista de conferência para código – parte 3 (nomes e variáveis)

Estruturas sequenciais	Sequências obrigatórias são refletidas na divisão entre funções, denominação das funções e parâmetros ou documentação.
	O código é legível de cima para baixo.
	Instruções correlatas são agrupadas.
	O alcance e o tempo vivo das variáveis são minimizados.
Estruturas condicionais if – then - else simples	O caminho normal está claramente indicado.
	Os casos de igualdade são tratados de forma correta.
	A cláusula else está presente, documentada e correta.
	As cláusulas if e else não estão invertidas.
	O caso normal segue o if e não o else.
	O miolo da malha está cercado por begin - end ou { }.
Estruturas condicionais em cadeias de if – then - else	Os testes complexos estão encapsulados em chamadas de funções booleanas.
	Os casos mais comuns são testados primeiro.
	Todos os casos estão cobertos.
	A implementação if – then - else é mais apropriada que a seleção múltipla.
	O miolo da malha está cercado por begin - end ou { }.
Estruturas condicionais de seleção múltipla	A ordem dos casos é significativa.
	As ações de cada caso são simples.
	Os casos são baseados em uma variável simples e legítima.
	A cláusula default é legítima.
	Erros e casos inesperados são cobertos pela cláusula default.
	A implementação por seleção múltipla é mais apropriada que por if - then.
Estruturas em malha	Só se pode entrar na malha pelo topo.
	O código de iniciação está imediatamente antes da malha.
	Malhas eternas são destacadas explicitamente.
	O miolo da malha está cercado por begin - end ou { }.
	Malhas vazias, se existirem, são comentadas explicitamente.
	As funções de "arrumação da casa" são concentradas no início ou no fim da malha.
	A malha executa uma e somente uma função.
	A malha sempre termina.
	A condição de terminação é óbvia.
	A malha pode ser visualizada em uma única tela ou página.
	O máximo nível de aninhamento é de três.
	Malhas longas são especialmente documentadas.
Estruturas em malha do tipo for	O cabeçalho da malha é reservado para código de controle dessa.
	Evita-se manipular o contador da malha.
	Valores importantes são salvos em variáveis próprias e não no contador da malha.
	O índice da malha é ordinal ou enumerado.
	O nome do índice é significativo.
	Não existem confusões entre índices de malhas aninhadas.

Tabela 229 – Lista de conferência para código – parte 4 (estruturas comuns de controle)

Estruturas especiais	O goto é usado de forma correta, quando indispensável
	O return é usado de forma correta.
	A recursão é usada de forma correta.
Testes booleanos de controle	São claramente inteligíveis.
	São de formulação positiva.
	Estão na forma mais simplificada possível.
	Os testes numéricos obedecem à ordem dos pontos em uma linha.
	As estrutura aninhadas de controle são minimizadas.

Tabela 230 - Lista de conferência para código – parte 5 (estruturas especiais de controle)

Leiaute	O leiaute reflete a estrutura lógica do código.	
	O leiaute obedece a regras consistentes.	
	O leiaute é resistente a modificações.	
	O tamanho das endentações é adequado e consistente.	
	Caso as endentações sejam feitas com tabulações, o fato é comentado.	
	O espaçamento nas expressões é claro, consistente e conforme com o padrão.	
	As declarações de dados são colocadas uma por linha, com alinhamento do tipo e do nome.	
	Estruturas de controle	A endentação das estruturas de controle é legível e consistente.
		Os pares { } estão corretos e casados.
		Os pares { } estão alinhados e endentados corretamente.
	Os blocos de uma instrução têm leiaute consistente.	
	Os grupos de instruções correlatas são separados por linhas em branco.	
	As linhas obedecem a um limite de tamanho.	
	Linhas com continuação	A divisão é feita nos pontos de maior legibilidade.
		Em listas de parâmetros, as linhas de continuação são alinhadas e começadas por um parâmetro.
Comentários	Indicam a intenção do código, e não simplesmente o repetem (focalizam o porquê e não o como).	
	São atualizados e pertinentes.	
	São claros e corretos.	
	São facilmente modificáveis.	
	Preparam o leitor para o código subsequente.	
	No caso de comentários de linha, só são usados nos casos permitidos.	
	No caso de declarações de dados, são usados para acrescentar informação relevante em relação à respectiva variável.	
	No caso de rotinas, descrevem suas entradas, saídas e hipóteses de trabalho.	
Operadores lógicos	O uso de ==, =, etc. está correto.	
	Para cada função, se o uso de () está correto.	
Formatos de saída	As mudanças de linha estão corretas.	
	O espaçamento está correto.	
Arquivos	Cada arquivo é declarado corretamente.	
	Cada arquivo é aberto corretamente.	
	Cada arquivo é fechado corretamente.	
Linhas	Em cada linha, a sintaxe das instruções está correta.	
	Em cada linha, a pontuação está correta.	

Tabela 231 - Lista de conferência para código – parte 6

Página em branco

Documentação para Usuários de Software

1 Visão geral

1.1 Introdução

Este padrão tem o objetivo de fornecer as regras mínimas que devem ser seguidas pela estrutura e conteúdo informativo de um manual de usuário. São abordados aspectos relacionados a manuais impressos e on-line.

Este conjunto de recomendações e regras deve ser suplementado com Manuais de Estilo específicos para determinados ambientes de desenvolvimento e tipos de produto.

1.2 Referências

A referência básica deste padrão é:

IEEE. *IEEE Std. 1063 – 1987. IEEE Standard for Software User Documentation*, in [IEEE94].

1.3 Padronização de formato

Deve-se adotar formatos padronizados para cada família de produtos correlatos (por exemplo, produzidos para o mesmo cliente, ou para determinada área de aplicação). Essa padronização deve ser feita através de modelos (*templates*) colocados à disposição dos autores de manual de usuário, de preferência on-line.

Neste padrão, o modelo adotado para os manuais de determinada família de produtos será chamado de Modelo de Manual de Usuário de Software. Os formatos de itens do documento, tais como tamanho da página, estilos de títulos e numeração de página, devem estar definidos nesse modelo.

O estilo de linguagem será especificado pelo Modelo de Manual de Usuário de Software. Ele deverá ser adequado ao perfil do público, evitando-se tanto utilizar termos e construções com os quais o público não esteja familiarizado quanto usar estilos condescendentes em relação ao nível cultural e técnico do público.

Esse modelo também definirá os títulos exatos das seções e subseções. Os títulos da seção Preenchimento do Manual do Usuário do Software, deste padrão, são indicativos para o autor do manual, e podem ser seguidos ou não, no manual em si. Os estilos de título deverão ser consistentes com o estilo de linguagem adotado no modelo.

1.4 Requisitos de apresentação

As seguintes considerações de apresentação devem ser observadas.

- Todo material de especial importância, tal como precauções, deve ser destacado tipograficamente. O método de destaque deve ser descrito na Introdução.

- A terminologia e as convenções tipográficas e de estilo devem ser usadas de forma consistente em todo o conjunto de documentos de um produto. Desvios em relação às convenções devem ser identificados na primeira vez em que apareçam.
- Todos os termos que constam do glossário, acrônimos e abreviações devem ser definidos quando são usados pela primeira vez em um documento.
- Deve-se indicar a existência de material relacionado em outras partes de um conjunto de documentos.

2 Preenchimento do Manual do Usuário do Software

2.1 Página de título

A página do título do Manual do Usuário do Software deve incluir os seguintes elementos:

- nome do manual;
- identificação do produto que é documentado;
- nomes dos autores e das organizações que produziram o documento;
- número de versão do documento;
- data de aprovação;
- assinaturas de aprovação;
- lista dos números de versão e datas de aprovação das versões anteriores;
- restrições quanto ao uso ou cópia do manual ou do produto.

Em caso de revisões do Manual do Usuário do Software, recomenda-se destacar no texto as partes alteradas, assim como indicar em folha à parte as seções e subseções em que ocorreram alterações.

O aspecto gráfico da capa deve ser padronizado através do Modelo de Manual do Usuário do Software, inclusive estilos de texto, leiaute e elementos gráficos tais como logomarcas.

2.2 Garantias e obrigações contratuais

A especificação de quaisquer garantias ou obrigações contratuais, quando houver, deve vir imediatamente após a página de título. O Modelo de Manual do Usuário do Software deve incluir um esqueleto de texto para essa subseção.

2.3 Sumário

Deve-se incluir um sumário em todo manual com mais de oito páginas, sendo opcional para manuais com um número menor de páginas.

No caso de conjuntos com múltiplos documentos, o primeiro volume deve incluir um sumário para todo o conjunto. Em cada volume subsequente, deve-se incluir o sumário do volume. Quando as seções forem extensas, recomenda-se que o sumário do volume seja resumido e que seja incluído um sumário por seção. O sumário deve ser, de preferência, construído por método automatizado.

2.4 Índice de ilustrações

É opcional incluir uma ou mais listas de títulos e localização de todas as ilustrações no documento. Pode-se usar índices separados para figuras e tabelas, ou incluir todos os tipos de ilustrações em uma única lista. O índice de ilustrações deve ser, de preferência, construído por método automatizado.

2.5 Introdução

Deve-se incluir as seguintes informações na introdução:

- **Descrição e pré-requisitos do público alvo:** se houver seções diferentes para determinados públicos, descrever quais, indicando o nível de experiência e treinamento prévio esperado para cada público.
- **Aplicabilidade:** descrever a versão do sistema que está sendo documentada e o ambiente de hardware e software no qual o sistema é executado.
- **Missão do produto:** descrever a missão do produto, indicando as aplicações esperadas.
- **Descrição do uso do documento:** descrever o conteúdo de cada seção do documento, sua utilização esperada e o relacionamento entre seções, assim como quaisquer outras instruções necessárias para leitura do documento.
- **Lista de documentos relacionados:** listar os documentos relacionados e fornecer o relacionamento entre o manual e esses documentos.
- **Descrição de convenções:** indicar as convenções de símbolos e estilo utilizadas no documento, assim como as convenções usadas para descrever a sintaxe dos comandos do produto.
- **Instruções para registro de problemas:** informar como os usuários podem registrar problemas na documentação ou no produto, ou fazer sugestões, fornecendo o nome e o contato das organizações responsáveis pela manutenção e pela melhoria do produto.

2.6 Corpo do documento

2.6.1 *Material instrucional*

Manuais instrucionais orientados para informação devem começar por uma seção de Escopo, que informa ao leitor o escopo do material a ser discutido. O restante do documento deve ser organizado por tópicos da descrição do produto, tais como:

- teoria e conceitos básicos do produto;
- telas principais e navegação do produto;
- funções do produto;
- exemplos de aplicação do produto;
- arquitetura do produto.

Manuais instrucionais orientados para tarefa devem ser organizados de forma que cada seção descreva um grupo de tarefas correlatas ou sequenciais. Cada tarefa normalmente corresponde a um processo de negócio (ou caso de uso do modelo de negócio).

Em produtos mais simples, essa seção pode ser única. Cada seção deve ter a seguinte estrutura:

- **Escopo:** informar o escopo do material a ser discutido.
- **Materiais:** descrever quaisquer materiais de que o usuário precisará para completar a tarefa (por exemplo, manuais, senhas, computadores, periféricos, cabos, controladores de software, interfaces e protocolos).
- **Preparação:** descrever ações, técnicas ou administrativas, que devam ser executadas antes de iniciar a tarefa (por exemplo, obter senha, autorização de acesso, espaço em disco etc.).
- **Precauções:** informar que determinada ação poderá levar a consequências que são indesejáveis ou indefinidas. A precaução deve ser colocada imediatamente antes da ação que a requer, e na mesma página.
- **Método:** indicar, para cada tarefa, o que se deve fazer, que funções se deve invocar, quais os resultados esperados e quais os erros possíveis.
- **Informação relacionada:** fornecer informações úteis que não foram cobertas nos itens anteriores, tais com relacionamentos entre tarefas, notas e limitações.

2.6.2 *Material referencial*

Manuais referenciais devem ser organizados de forma a que cada seção descreva uma maneira de usar as funções do produto. Cada função normalmente corresponde a um caso de uso do produto, ou a um roteiro que percorra variantes importantes do fluxo de um caso de uso.

Cada seção deve ter a seguinte estrutura:.

- **Objetivo:** informar o objetivo dessa seção.
- **Materiais:** descrever materiais de que o usuário precisará para completar a função.
- **Preparação:** descrever ações, técnicas ou administrativas, que devem ser executadas antes de iniciar a função.
- **Entrada(s):** identificar e descrever os dados necessários para o processamento de cada função.
- **Precauções:** informar que determinada função poderá levar a consequências que são indesejáveis ou indefinidas; a precaução deve ser colocada imediatamente antes da função que a requer, e na mesma página.
- **Chamada:** fornecer as informações necessárias para usar e controlar a função, como parâmetros obrigatórios, parâmetros opcionais, opções padrões e sintaxe.
- **Suspensão de operações:** descrever como interromper a função durante sua execução e como reiniciá-la.
- **Encerramento de operações:** informar como reconhecer o término de uma função, inclusive terminações anormais.
- **Saída(s):** descrever os resultados da execução da função, como saída na tela, efeito em arquivos ou dados, valores de parâmetros de saída e saídas que disparam outras ações (como ações mecânicas em aplicações de controle de processos).
- **Condições de erro:** descrever os erros que podem resultar da execução da função (por exemplo, listando as possíveis mensagens de erro).

- **Informação relacionada:** fornecer informações úteis que não foram cobertas nos itens anteriores, tais como relacionamentos entre tarefas, notas e limitações.

2.7 Mensagens de erro, problemas conhecidos e recuperação de erros

As mensagens de erro devem ser colocadas de modo que sua localização seja fácil, em uma seção ou apêndice separados. Para cada mensagem de erro, deve-se descrever em detalhes o erro que a provocou e os procedimentos necessários para recuperação. Deve-se incluir também uma descrição dos problemas conhecidos do produto e de como resolvê-los.

2.8 Apêndices

Incluir nos apêndices qualquer informação que sirva de suporte para a utilização do Sistema, como:

- dados de entrada/saída e formatos utilizados em diversas funções;
- códigos³⁰ usados para entrada ou saída;
- interação entre tarefas e funções;
- limitações globais de processamento;
- descrição de formato de dados e estrutura de arquivos;
- arquivos, relatórios ou programas de exemplo.

2.9 Bibliografia

Incluir a lista das publicações citadas no texto e publicações que contêm informações relacionadas.

2.10 Glossário

Listar, em ordem alfabética, definições de todos os termos, acrônimos e abreviaturas usados no documento que possam ser desconhecidos do usuário, ou que sejam utilizados de modo não familiar a ele.

2.11 Índice remissivo

O índice remissivo deverá ser baseado em palavras-chave ou conceitos e será obrigatório para documentos com mais de oito páginas. Ele deve ser, de preferência, construído por método automatizado.

³⁰ No sentido de código do equipamento, número da solicitação etc.

Página em branco

A notação UML

1 Introdução

Este apêndice apresenta os principais conceitos da UML (Unified Modeling Language) que são utilizados neste livro. Não é objetivo deste texto a descrição completa de cada conceito, e muitos são apresentados de forma simplificada; os interessados nas descrições completas devem consultar [Rumbaugh+99] ou as especificações oficiais da UML, que podem ser obtidas no sítio do OMG (Object Management Group).

Por outro lado, utilizam-se aqui algumas convenções de notação que o processo Praxis adota, em situações nas quais a UML permite o uso de alternativas. Por exemplo, as convenções de descrição textual dos casos de uso não são estipuladas pela UML, mas são aqui incluídas por estarem intimamente associadas à maneira de utilização dos casos de uso dentro do Praxis.

2 Modelagem funcional

2.1 Atores

Os papéis dos usuários de um produto são modelados através dos atores (Figura 20). Cada ator representa uma classe de usuários. Os atores modelam os papéis e não as pessoas dos usuários; por exemplo, o mesmo usuário físico pode agir como “Gerente”, “Gestor de Estoque” ou “Gestor de Compras”. Pode-se também definir atores não humanos, para modelar outros sistemas que devam interagir com o produto em questão: por exemplo, o “Sistema Financeiro”.



Figura 162 - Exemplos de atores

Caso exista grande número de atores, deve-se procurar agrupá-los em atores genéricos, que representem características comuns a vários grupos de usuários de comportamento semelhante em relação ao produto. Atores genéricos e específicos são ligados por relacionamentos de herança. Na Figura 23, indica-se que “Gerente de Vendas” e “Gerente de Compras” têm alguns aspectos em comum, que são abstraídos através do ator “Gerente”.

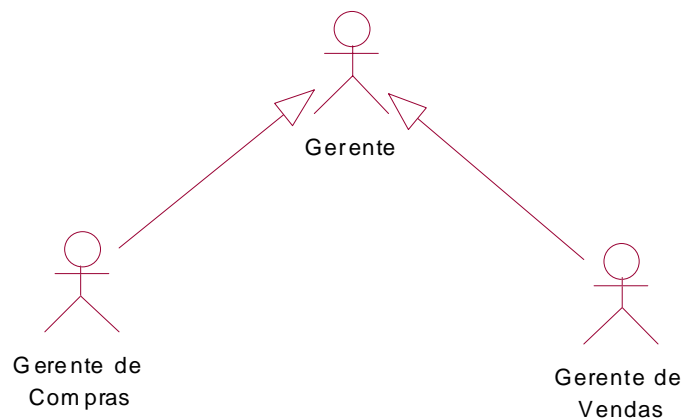


Figura 163 – Herança entre atores

2.2 Casos de uso

Os **casos de uso** representam funções completas do produto. Um caso de uso realiza um aspecto maior da funcionalidade do produto: deve gerar um ou mais benefícios para o cliente ou os usuários. Na Figura 19 são mostrados os casos de uso que representam a funcionalidade de um produto de informatização de uma mercearia. O conjunto dos casos de uso cobre toda a funcionalidade do produto, e cada caso de uso representa uma fatia independente de funcionalidade.



Figura 164 - Casos de uso

2.3 Diagramas de casos de uso

2.3.1 Relacionamentos entre atores e casos de uso

Diagramas de casos de uso podem especificar os relacionamentos entre casos de uso e atores (Figura 21). Os relacionamentos indicam a existência de comunicação entre atores e casos de uso. Um caso de uso pode estar associado a mais de um ator, quando a sua execução requer a participação de diferentes atores.

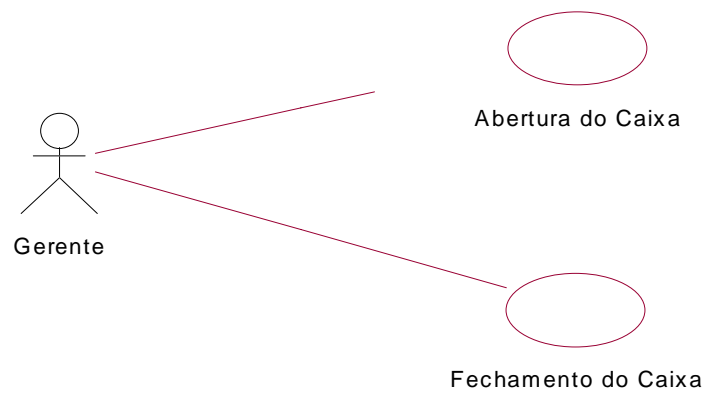


Figura 165 - Exemplo de casos de uso de um ator

Normalmente, a comunicação será representada como ligação sem direção; convencionou-se, nesse caso, que a iniciativa de comunicação parte do ator. Quando a iniciativa parte do caso de uso (por exemplo, alarmes, mensagens, dados enviados para outros sistemas etc.), a comunicação deve ser direcionada para o ator (Figura 22). Nesse exemplo, o caso de uso “Gestão Manual de Estoque”, acionado pelo ator “Gestor de Estoque”, envia dados para o “Sistema Financeiro”.

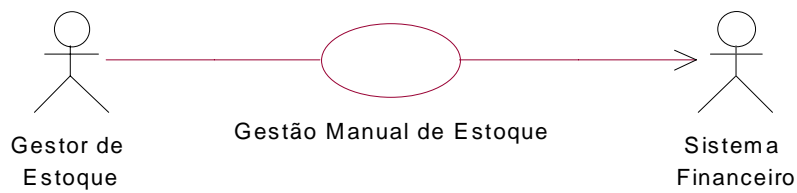


Figura 166 - Caso de uso com mais de um ator

Os diagramas de casos de uso podem ser simplificados por meio da herança entre atores. Neste caso, mostra-se um caso de uso comum aos atores específicos, que se comunicam apenas com o ator genérico (Figura 24). Neste exemplo, tanto o “Gerente de Compras” quanto o “Gerente de Vendas” podem executar o caso de uso “Emissão de relatórios”, porque eles são herdeiros (especializações) de “Gerente”.

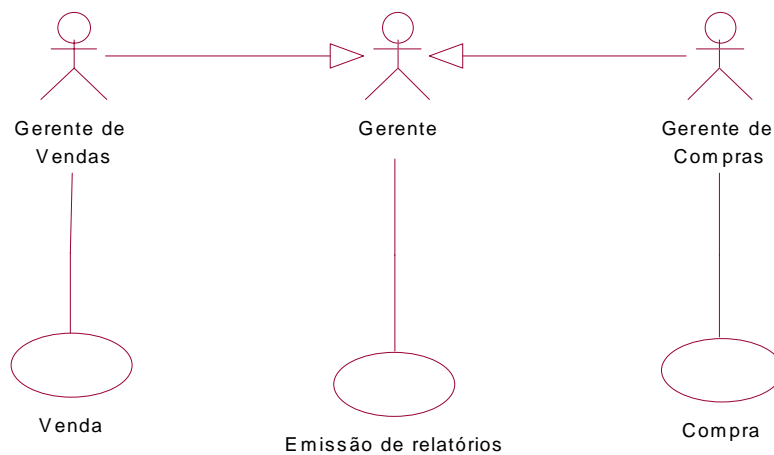


Figura 167 – Uso de atores genéricos

O diagrama de contexto (Figura 25) é um diagrama de casos de uso que mostra as interfaces do produto com seu ambiente de aplicação. Os diversos tipos de usuários e outros sistemas com os quais o produto deva interagir são representados por atores situados fora de um retângulo que marca a fronteira do produto.

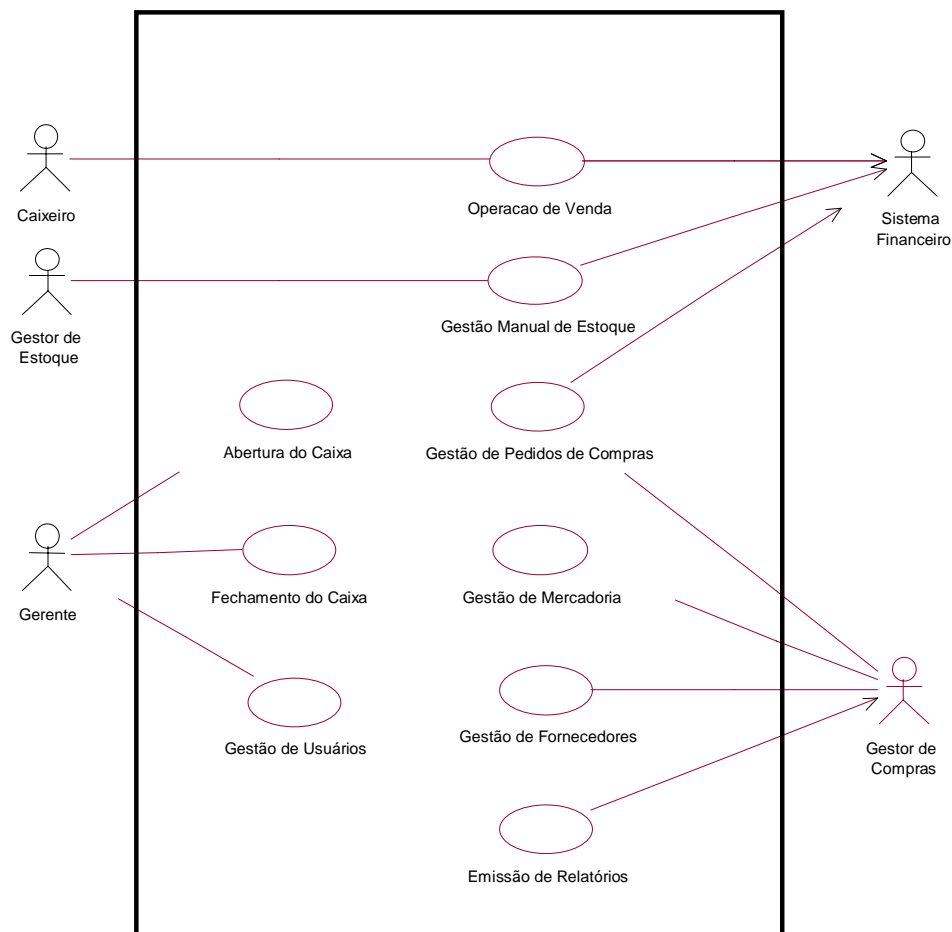


Figura 168 - Diagrama de contexto

2.3.2 Relacionamentos entre casos de uso

Notações especiais são utilizadas para facilitar a descrição de funcionalidade mais complexa. Entre estas notações, destacam-se os casos de usos secundários, que simplificam o comportamento dos casos de uso primários através dos mecanismos de **extensão** e **inclusão**. No Praxis, casos de uso primários são aqueles que são invocados por iniciativa direta de um ator; casos de uso secundários são invocados em um passo de outro caso de uso. Os termos “primário” e “secundário”, quando referentes a casos de uso, não fazem parte da UML.

O caso de uso B estende o caso de uso A quando B representa uma situação opcional ou de exceção, que normalmente não ocorre durante a execução de A. Essa notação pode ser usada para representar fluxos complexos opcionais ou anormais. O caso de uso “estendido” é referenciado nas precondições do caso de uso “extensor”. As precondições são a primeira parte dos fluxos dos casos de uso. Na Figura 169, a “Emissão de Nota Fiscal” é uma funcionalidade que pode ser invocada ou não, durante a “Operação de Venda”.

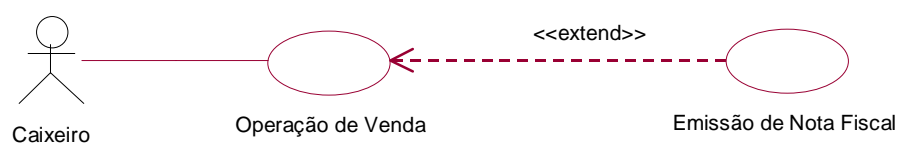


Figura 169 - Caso de uso de extensão

O caso de uso A inclui o caso de uso B quando B representa uma atividade complexa, comum a vários casos de uso. Essa notação pode ser usada para representar subfluxos complexos e comuns a vários casos de uso. O caso de uso “incluído” é referenciado no fluxo do caso de uso “includor”. Na Figura 170, “Baixa no Estoque” representa um comportamento comum a “Gestão Manual de Estoque” e “Operação de Venda”.

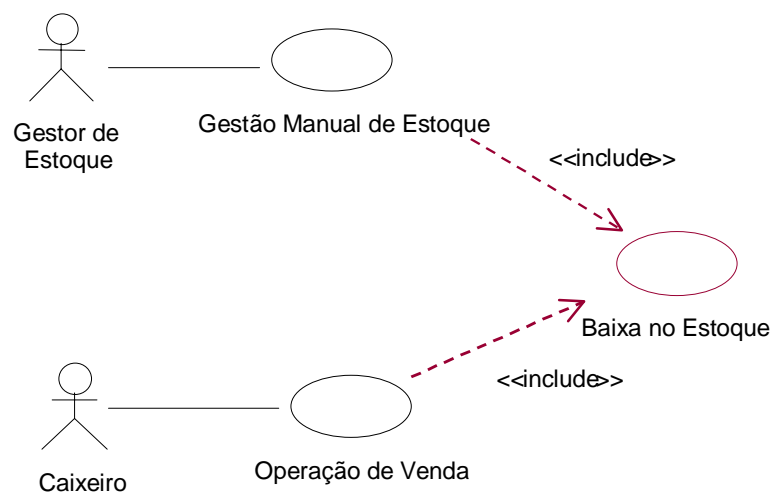


Figura 170 - Caso de uso de inclusão

2.4 Fluxos dos casos de uso

No Praxis, os fluxos dos casos de uso são detalhados por meio de descrições textuais. A UML não impõe formatos obrigatórios para as descrições dos fluxos, mas a forma de descrição textual aqui apresenta é semelhante às formas usadas pela maioria dos autores que utilizam os casos de uso.

O detalhamento dos fluxos dos casos inclui as suas precondições, ou seja, as condições que supõem estejam satisfeitas, ao iniciar a execução de um caso de uso (Tabela 43); o fluxo principal, que representa a execução mais normal da função; e os subfluxos e fluxos alternativos, que representam variantes que são executadas sob certas condições. A UML permite que diversas notações sejam utilizadas para descrever os detalhes dos casos de uso.

Toda mercadoria a ser vendida (item de venda) deve estar previamente cadastrada.
--

O Merci deve estar no Modo de Vendas.

Tabela 232 - Precondições de um caso de uso

Os fluxos são comumente descritos em linguagem natural, na forma de uma sequência de passos (Tabela 44). Cada passo corresponde a uma ação de um ator ou do produto; estes devem aparecer explicitamente como sujeitos da frase. Outros atores podem aparecer como objetos verbais de uma ação. Condições e iterações podem aparecer, mas os detalhes destas devem ser descritos em subfluxos, de preferência. Isso ajuda a manter a legibilidade do fluxo, que é essencial para garantir o bom entendimento de todas as partes.

<p>O <u>Caixeiro</u> faz a abertura da venda.</p> <p>O <u>Merci</u> gera o código da operação de venda.</p> <p>Para cada item de venda, o <u>Merci</u> aciona o subfluxo Registro.</p> <p>O <u>Caixeiro</u> registra a forma de pagamento.</p> <p>O <u>Caixeiro</u> encerra a venda.</p> <p>Para cada item de venda, o <u>Merci</u> aciona o subfluxo Impressão de Linha do Ticket.</p> <p>O <u>Merci</u> notifica o <u>Sistema Financeiro</u> informando: Data, Número da Operação de Venda, “Receita”, Valor Total”, Nome do Cliente (caso tenha sido emitida a nota fiscal).</p>

Tabela 233 - Fluxo principal

Quando uma condição ou iteração for composta por uma sequência muito curta de passos, elas podem ser representadas por meio de endentação.

<p>O <u>Gestor de Compras</u> seleciona a mercadoria.</p> <p>O <u>Merci</u> verifica se existe algum pedido pendente que contenha esta mercadoria.</p> <p>Se não houver pedido pendente contendo a mercadoria a ser excluída:</p> <p style="padding-left: 40px;">o <u>Merci</u> desvincula a mercadoria dos fornecedores (os fornecedores não mais fornecerão a mercadoria que esta sendo excluída).</p> <p style="padding-left: 40px;">o <u>Merci</u> faz a remoção da mercadoria.</p> <p>Se houver pedido pendente contendo a mercadoria a ser excluída</p> <p style="padding-left: 40px;">o <u>Merci</u> emite uma mensagem de erro.</p>

Tabela 234 – Fluxo principal com condicionais endentados

Os subfluxos descrevem geralmente detalhes de iterações, ou de condições executadas com frequência (Tabela 45).

<p>O <u>Caixeiro</u> registra o item de venda, informando a identificação e a quantidade.</p> <p>O <u>Merci</u> totaliza a venda para o cliente da mercearia.</p>

Tabela 235 - Exemplo de subfluxo: Registro de item em Operação de Venda

Os fluxos alternativos descrevem condições pouco habituais ou exceções (Tabela 46).

<p>Se o <u>Gestor de Compras</u> solicitar:</p> <p style="padding-left: 40px;">o <u>Merci</u> imprime o pedido de compra.</p>

Tabela 236 - Exemplo de fluxo alternativo: Impressão de Pedido de Compras

2.5 Diagramas de estados

Para casos de uso complexos, pode-se descrever seu comportamento de maneira mais formal, através de um diagrama de estados ou diagrama de atividade da UML. Diagramas de estados são utilizados para descrever fluxos de lógica complexa ou com muitos detalhes, como costuma acontecer nos casos de uso de desenho (Figura 91). As principais convenções são mostradas na Tabela 237.

Símbolo	Descrição
Retângulo com cantos arredondados	Estado.
Seta cheia	Transição entre estados diferentes ou de um estado para si mesmo (transições reflexivas).
Pequeno círculo cheio	Estado inicial.
Círculo cheio dentro de círculo vazio	Estado final.

Tabela 237 - Símbolos dos diagramas de estados

Os eventos que provocam as transições aparecem como rótulos destas. Se a transição depender de uma condição, esta pode ser representada por uma guarda (texto entre colchetes). Anotações são utilizadas para complementar o modelo; por exemplo, indica-se, na Figura 91, que se pode invocar a tela de nota fiscal no estado “Atualizada”.

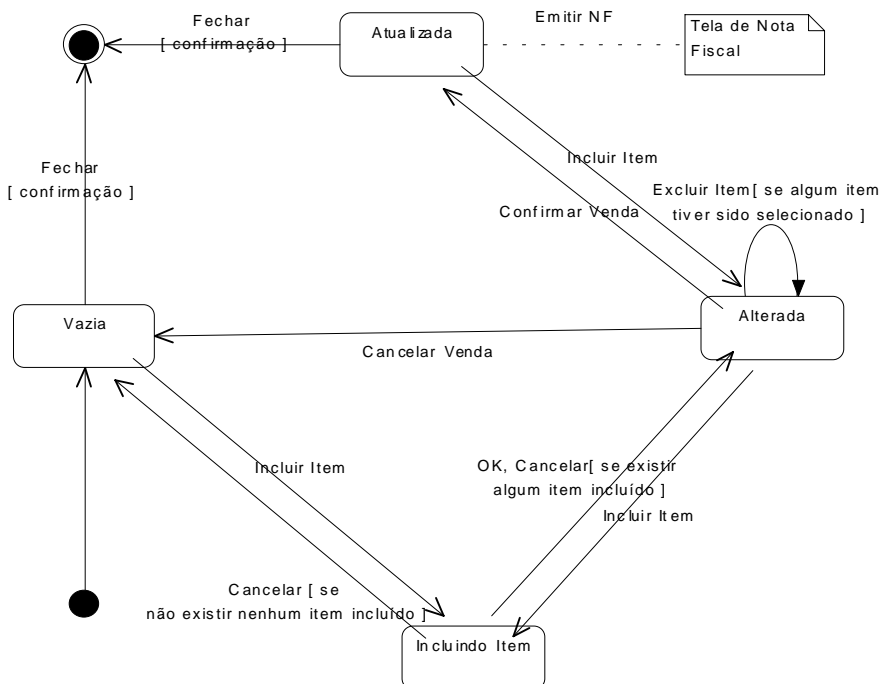


Figura 171 – Diagrama de estado de caso de uso

Diagramas de estado podem ser usados para mostrar lógica bastante complexa. Na Figura 172, os estados complexos X e Y tem subestados, que são usados para exprimir a seguinte lógica:

- O estado inicial conduz a X, dentro qual existe um subestado inicial que conduz a X1. Ao entrar em X1, a ação A0 é executada. O evento E1 leva ao subestado X2.
- No subestado X2, se o evento E2 ocorrer com a condição C1, é executada a ação A1 e continua-se no subestado X2. Se E2 ocorrer com a condição C1 falsa, executa-se a ação A2 e passa-se para o subestado Y1 do estado Y.

- Durante a permanência no subestado Y1 é executada a ação A4. Se no subestado Y1 ocorrer o evento E3, executa-se a ação A3 e passa-se ao subestado Y2.
- Se no subestado Y2 ocorrer o evento E4, volta-se ao estado X; na saída de Y2 envia-se o evento E5, que pode provocar alguma transição não mostrada nesse diagrama.
- Se no estado Y (em qualquer subestado) ocorrer o evento E6, vai-se para um estado final.

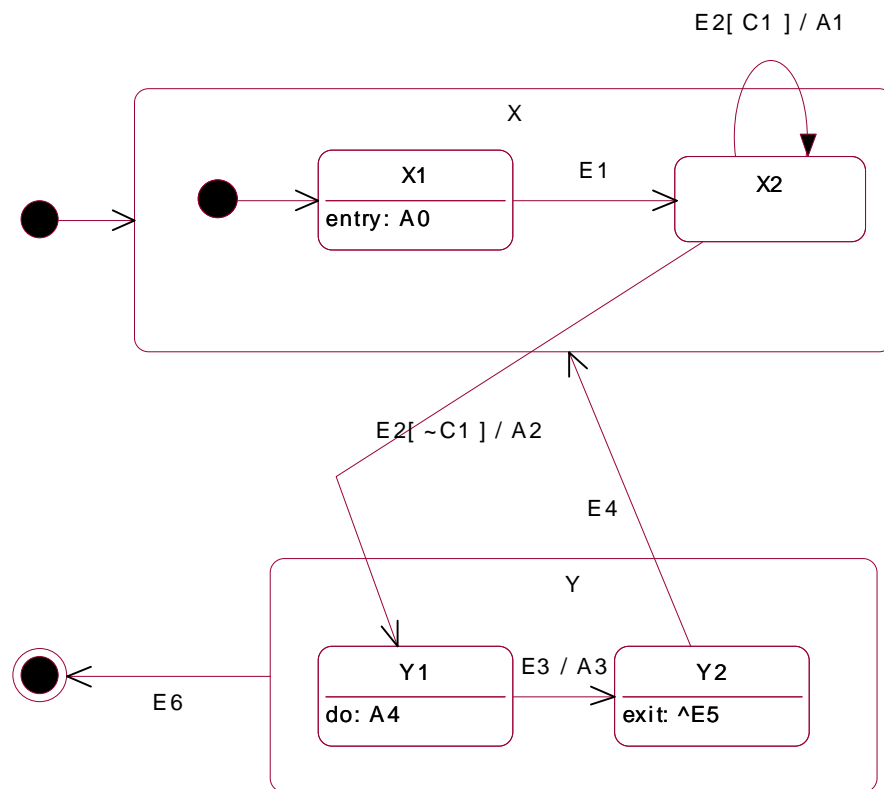


Figura 172 – Diagrama de estados complexo

2.6 Diagramas de atividade

Estes diagramas são uma variante dos fluxogramas, sendo geralmente usados para descrever processos de negócio, ou outros fluxos onde o paralelismo de atividades seja importante. A Figura 15 apresenta um exemplo de diagrama de atividades. A Tabela 16 explica os símbolos usados.

Símbolo	Descrição
Retângulo ovalado	Atividade (passo do fluxo).
Retângulo	Objeto (artefato do fluxo).
Seta cheia	Relação de precedência entre atividades.
Seta pontilhada	Consumo ou produção de objeto por atividade.
Linha horizontal cheia	Ponto de sincronização (onde subfluxos paralelos se juntam).
Pequeno círculo cheio	Estado inicial.
Círculo cheio dentro de círculo vazio	Estado final.

Tabela 238 - Símbolos dos diagramas de atividades

Nesse exemplo, mostra-se que as atividades “Detalhamento dos requisitos de interface” e “Detalhamento dos casos de uso” são realizadas em paralelo, mas a atividade “Detalhamento dos requisitos não funcionais” requer que ambas estejam completas. Os objetos mostrados à direita representam os resultados das atividades.

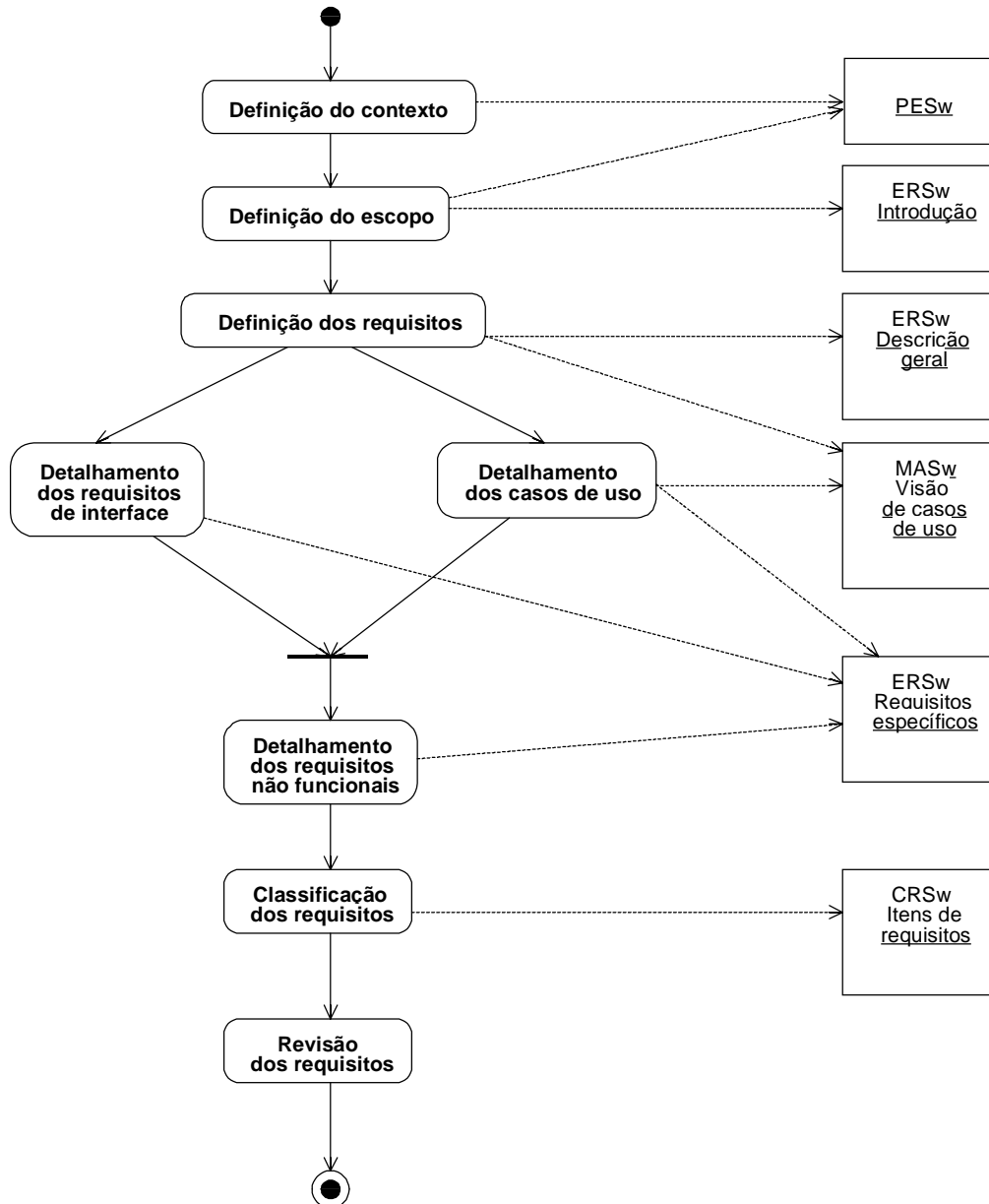


Figura 173 – Exemplo de diagrama de atividades

Um diagrama de atividades mais complexo é mostrado na Figura 18. Ele é dividido em **raias** (*swimlanes*), que representam os diversos papéis de pessoas ou grupos envolvidos no processo (no exemplo, cliente da mercearia, caixeiro e gerente). Os objetos situados sobre as fronteiras das raias representam objetos partilhados entre os papéis. Em um processo de negócio, eles podem representar tanto objetos físicos quanto informacionais. Um losango de decisão tem como saídas subfluxos alternativos (no exemplo, a emissão opcional de nota fiscal).

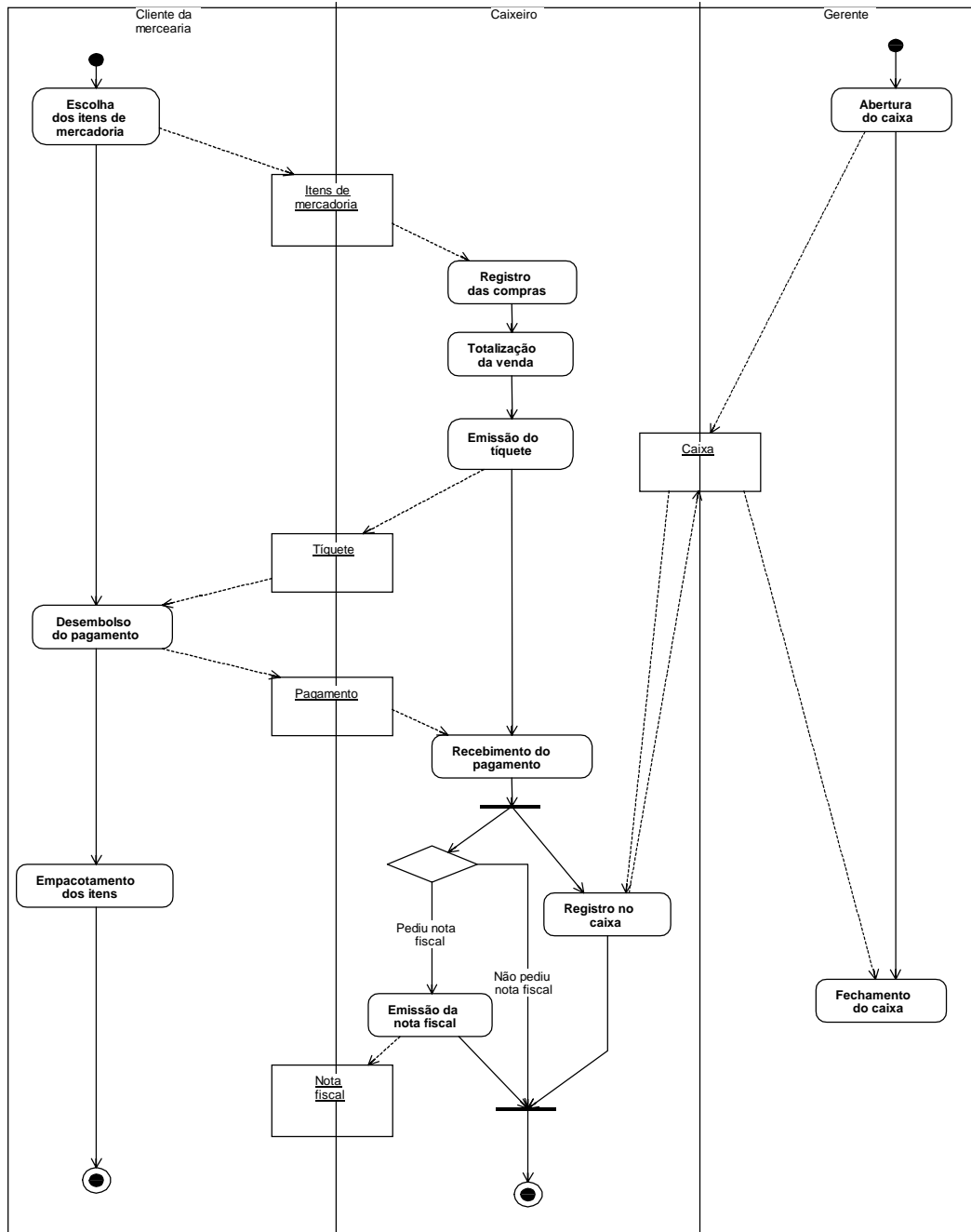


Figura 174 – Exemplo de modelo de processo de negócio

3 Modelagem lógica

3.1 Objetos e classes

Nas metodologias de modelagem orientadas a objetos, os objetos representam entidades discretas, com fronteira bem definida e identidade própria, que encapsulam estado e comportamento. O estado é representado pelos atributos do objeto, e o comportamento pelas respectivas operações. Os objetos interagem entre si trocando mensagens, que são invocações das operações. Objetos similares são agrupados em classes.

Na UML, um objeto é representado por um retângulo, onde o nome do objeto é sublinhado. Quando um objeto aparece em um diagrama UML, podem acontecer as seguintes situações:

- Um objeto pode ter classe indeterminada (Figura 175). Esta é uma situação transitória que pode acontecer durante a análise, mas deve ser resolvida.
- Um objeto pode ter denominação própria e pertencer a uma determinada classe (Figura 176). O nome da classe é separado do nome do objeto por um sinal de dois pontos.
- Um objeto pode ser anônimo, representando uma instância genérica de determinada classe (Figura 177). O campo à esquerda dos dois pontos fica vazio.

Venda
corrente

Figura 175 - Objeto sem classe determinada

Venda corrente
: Venda

Figura 176 - Objeto com indicação da respectiva classe

_
Mercadoria

Figura 177 - Objeto anônimo

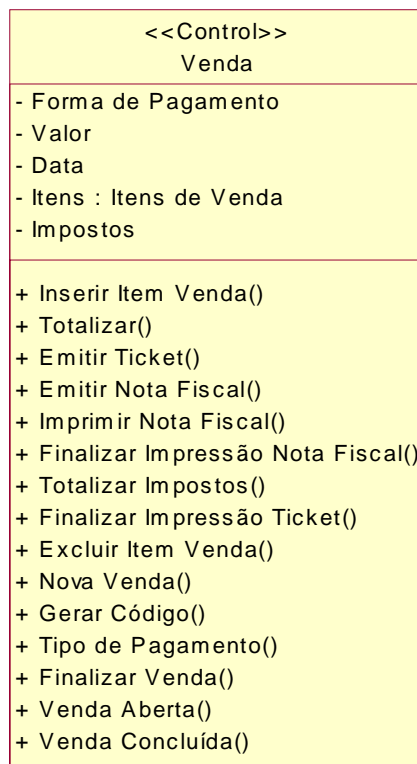


Figura 178 - Representação de classe

Na UML, a classe é representada por um retângulo dividido em três compartimentos, que contêm respectivamente o nome da classe, os atributos e as operações. Para maior clareza nos diagramas, pode-se suprimir cada um dos compartimentos de atributos e operações, ou deixar de mostrar determinados atributos ou operações. São opcionais também: a indicação da visibilidade por caracteres ou ícones; a assinatura (lista de argumentos e tipo de retorno) das operações; e o tipo e valor padrão dos atributos.

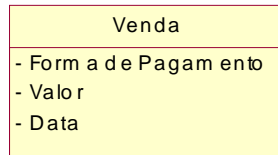


Figura 179 - Representação de classe com supressão das operações

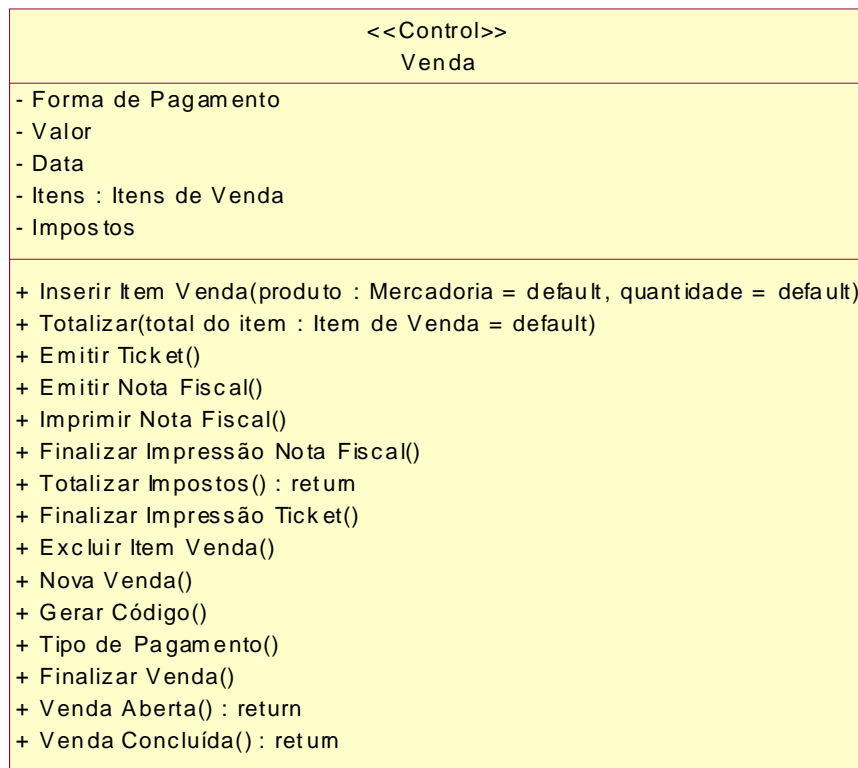


Figura 180 - Representação de classe com detalhamento das assinaturas das operações

3.2 Organização das classes

3.2.1 Pacotes lógicos

Os **pacotes lógicos** são agrupamentos de elementos de um modelo. No modelo de análise, eles podem ser utilizados para formar grupos de classes com um tema comum. Na Figura 45, os pacotes lógicos Compras, Vendas e Administração são usados para agrupar classes especializadas em relação a esses aspectos.

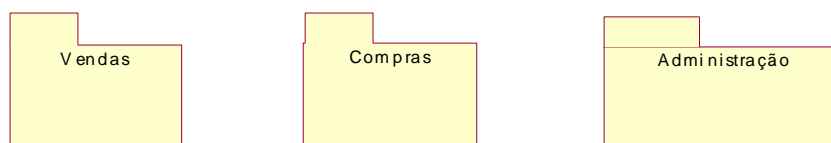


Figura 181 - Pacotes lógicos

Pacotes lógicos podem ter relações de dependência e continência entre si. Na Figura 82, os pacotes lógicos “Classes de fronteira” e “Classes de controle” dependem (importam recursos) de “Coleções de entidades”. “Coleções de entidades” contém os pacotes “Coleções” e “Elementos”. “Classes VB” é um pacote global: seus recursos estão disponíveis para os elementos de todos os outros pacotes.

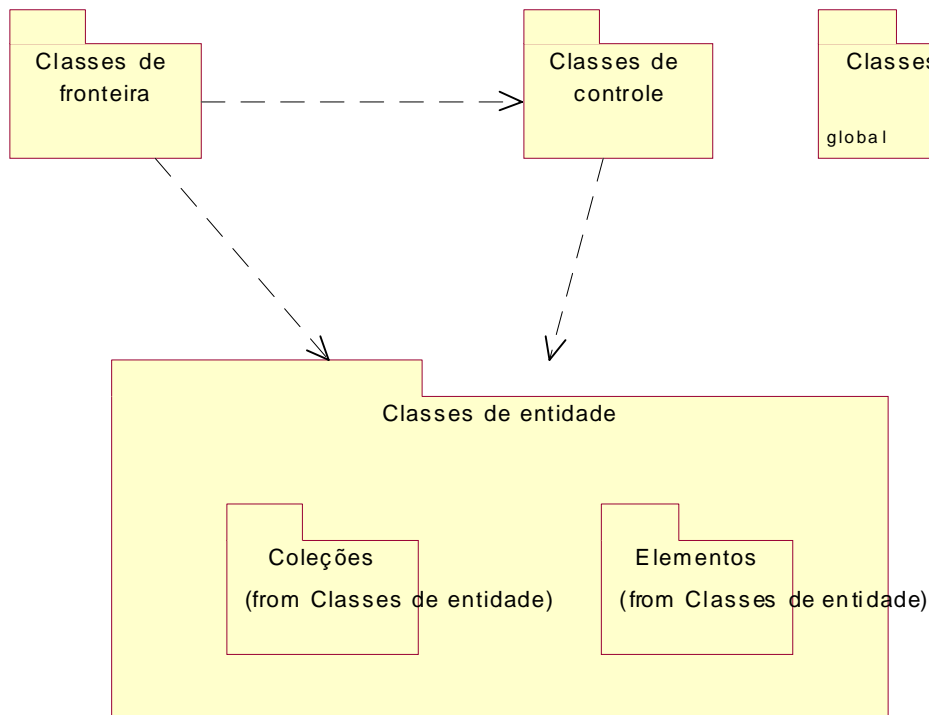


Figura 182 – Relacionamentos entre pacotes lógicos

3.2.2 Estereótipos

Os **estereótipos** são extensões de elementos do modelo. Podem ser usados para denotar especializações significativas de classes. Os atores, por exemplo, podem ser tratados pelas ferramentas de modelagem como classes estereotipadas. Como mostra a Figura 183, estereótipos podem ser indicados através de ícones próprios, ou incluindo-se o nome do estereótipo em aspas francesas (os caracteres « », representados nos desenhos por << >>).

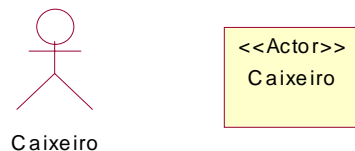


Figura 183 - Representações alternativas do estereótipo "Ator"

Estereótipos são usados para criar especializações da UML em relação a determinadas áreas de modelagem. Jacobson ([Jacobson94], [Jacobson+99]) propõe a divisão das classes do Modelo de Análise de acordo com os estereótipos descritos a seguir, que foram incorporados ao padrão oficial da UML. Na Figura 184 eles são representados de forma textual, e na Figura 185 são representados na forma icônica.

- **Entidades** ("entity") – modelam informação persistente, sendo tipicamente independentes da aplicação. Geralmente são necessárias para cumprir alguma responsabilidade do produto, e freqüentemente correspondem a tabelas de bancos de dados.
- **Fronteiras** ("boundary")– tratam da comunicação com o ambiente do produto. Modelam as interfaces do produto com usuários e outros sistemas, e surgem tipicamente de cada par ator – caso de uso.
- **Controles** ("control") – coordenam o fluxo de um caso de uso complexo, encapsulando lógica que não se enquadra naturalmente nas responsabilidades das entidades. São tipicamente dependentes de aplicação.

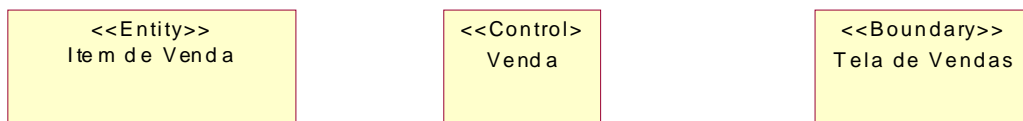


Figura 184 - Exemplo de classes de análise com estereótipos textuais

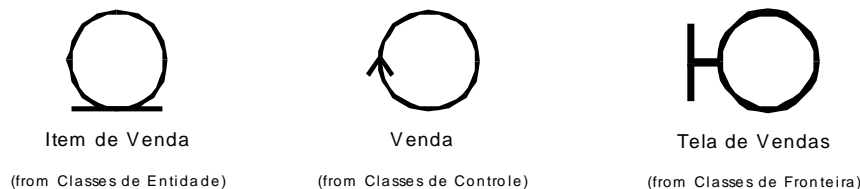


Figura 185 - Exemplo de classes de análise com estereótipos icônicos

3.3 Relacionamentos

3.3.1 Relacionamentos de associação

As associações expressam relações bidirecionais de dependência semântica entre duas classes. Elas indicam a possibilidade de comunicação direta entre os respectivos objetos. Isso significa que faz parte das responsabilidades de um objeto de uma das classes determinar os objetos correspondentes da outra classe. Normalmente, existirão em cada classe operações para cumprir essa responsabilidade.



Figura 186 - Relacionamento de associação

3.3.2 Multiplicidades e papéis

A especificação das associações inclui o seu nome, descrição e possíveis restrições. Em certos casos, é útil batizar também os papéis, assim como especificar vários detalhes destes. Os nomes das associações devem ser simples e significativos. Recomenda-se usar um substantivo que descreva bem a semântica do relacionamento. Pode-se também usar um verbo, desde que esteja claro qual classe é sujeito e qual classe é objeto desse verbo (Figura 187). Uma convenção habitual é batizar o relacionamento de modo que ele seja lido corretamente de cima para baixo ou da esquerda para a

direita. Na última versão da UML, um pequeno triângulo pode ser usado para indicar a direção de leitura, caso necessário.

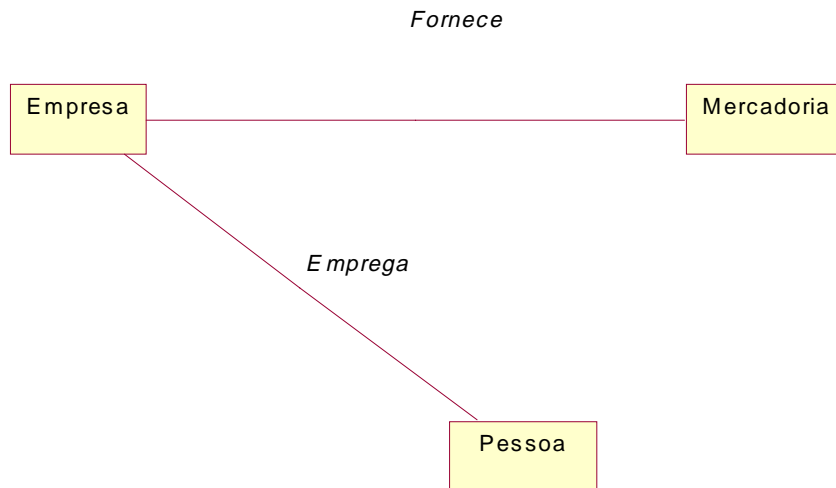


Figura 187 - Relacionamento com nome

A multiplicidade de um participante de um relacionamento indica quantos objetos de uma classe se relacionam com cada objeto da outra classe. Relacionamentos obrigatórios têm multiplicidade mínima 1. A multiplicidade máxima indica o número máximo de instâncias da classe alvo que podem existir simultaneamente. Na Figura 188 modela-se o fato de que uma “Pessoa” pode estar ligada a nenhuma ou uma “Empresa”, enquanto uma “Empresa” pode estar relacionada com uma ou mais instâncias de “Pessoa” (por exemplo, empregar). Uma “Empresa” pode estar relacionada com zero ou mais instâncias de “Mercadoria” (por exemplo, fornecer).

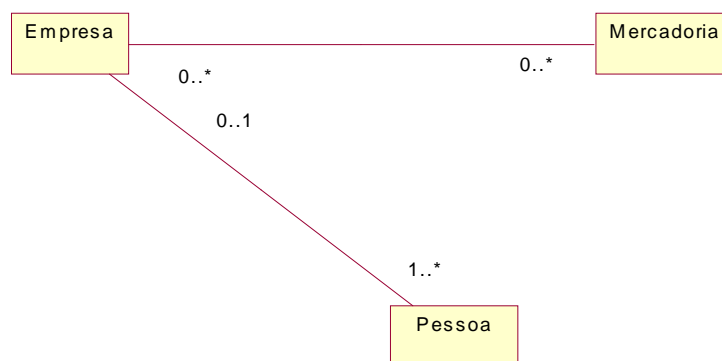


Figura 188 - Relacionamentos com multiplicidades

Os papéis são denominações que exprimem em que qualidade um objeto de uma das classes do relacionamento se relaciona com um objeto da outra classe. Os papéis dos participantes devem ser batizados explicitamente quando participarem de um relacionamento em uma qualidade que não é implícita no respectivo nome da classe. Na Figura 55 indica-se que um objeto da classe "Empresa" participa no papel de "fornecedor" do relacionamento "Fornece", com zero ou mais objetos da classe "Mercadoria", e um objeto dessa classe se relaciona com zero ou mais objetos da classe "Empresa" na qualidade de "produto". Uma "Empresa" pode participar de outros relacionamentos em outras qualidades; por exemplo, no papel de "empregador", em um relacionamento "Emprega" com um objeto da classe "Pessoa".

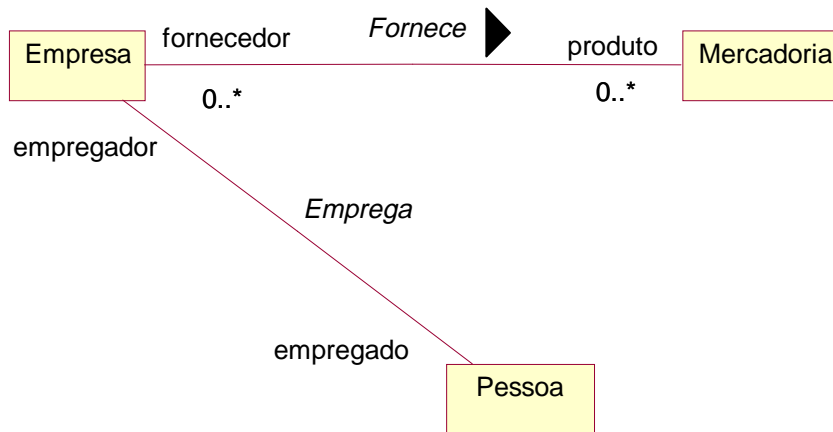


Figura 189 - Relacionamentos com denominação dos participantes

3.3.3 Navegabilidade

Os relacionamentos podem ter direção de navegação. Um relacionamento é navegável da classe A para a classe B se, dado um objeto da classe A, consegue-se obter de forma direta (por exemplo, através de uma operação da classe A) os objetos relacionados da classe B. Por exemplo, a Figura 56 indica que, dada uma "Mercadoria", é possível localizar diretamente o respectivo "Fornecedor", mas a recíproca não é verdadeira.



Figura 190 - Relacionamento com restrição de navegabilidade

3.3.4 Relacionamentos de herança

O relacionamento de herança existe entre classes de natureza mais geral (chamadas de superclasses ou classes bases) e suas especializações (chamadas de subclasses ou classes derivadas). As superclasses contêm atributos ou operações comuns a um grupo de subclasses. Na Figura 191 modela-se o fato de que um "Item de Compra" é um tipo (uma especialização) de "Item de Mercadoria", assim com um "Item de Venda".

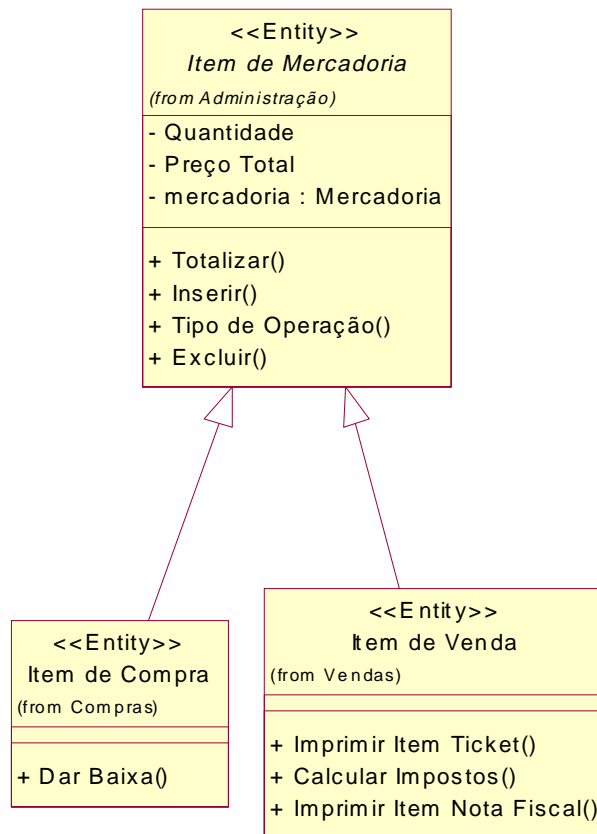


Figura 191 - Exemplo de relacionamentos de herança

3.3.5 Relacionamentos avançados

Um **relacionamento de agregação** é uma associação que reflete a construção física ou a posse lógica. Relacionamentos de agregação são casos particulares dos relacionamentos de associação, e só é necessário distingui-los quando for conveniente enfatizar o caráter "todo-parte" do relacionamento. Geralmente um relacionamento de agregação é caracterizado pela presença da expressão "parte de" na descrição do relacionamento, e pela assimetria da navegação.

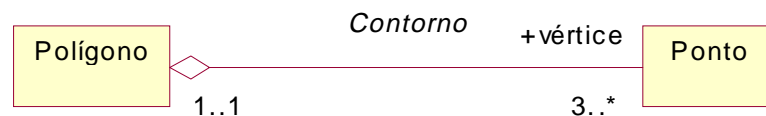


Figura 192 - Relacionamento de agregação

Um tipo mais forte de relacionamento todo-parte é o **relacionamento de composição**. Nesse caso, os objetos da classe parte não têm existência independente da classe todo. A Figura 58 indica que o "centro" de um "Círculo" não tem existência independente deste, enquanto que a Figura 57 indica que cada "ponto" existe independentemente do "Polígono" ao qual serve de "vértice".



Figura 193 - Relacionamento de composição

Uma associação pode ser reflexiva. Uma auto-associação indica um relacionamento entre objetos de mesma classe que desempenham diferentes participações. Na Figura 194 indica-se uma “Pessoa” na qualidade de “chefe” relaciona-se com uma ou mais instâncias de “Pessoa” que exercem o papel de “subordinado”.

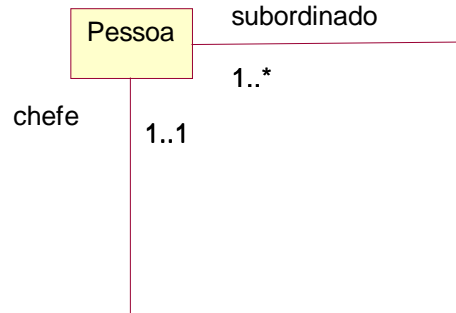


Figura 194 - Associação reflexiva

Os relacionamentos podem ser de natureza complexa. Em certos casos, um relacionamento é mais bem explicado através de uma classe de associação, que exprime atributos e até operações que são propriedades do relacionamento como um todo e não de cada participante isoladamente. Na Figura 195 a classe “Emprego” inclui atributos e operações que não estão ligados a uma “Empresa” ou uma “Pessoa” isolada, mas ao relacionamento entre elas; por exemplo, o atributo “data de início” ou a operação “imprimir atestado de emprego”.

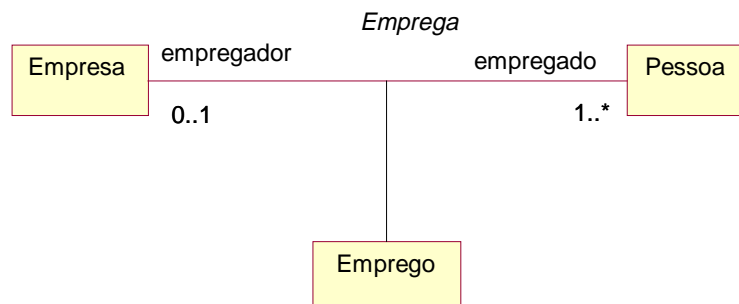


Figura 195 - Classe de associação

Um qualificador é um atributo que restringe um relacionamento através de uma seleção. Na maioria dos casos, cada valor do qualificador está associado a uma única instância da classe alvo. Na Figura 61, o qualificador "número de conta" restringe a participação de objetos da classe "Pessoa" no relacionamento. Um "número de conta" pode estar associado a zero ou um cliente (contas conjuntas não são modeladas).

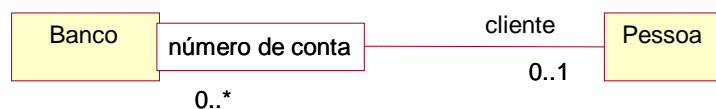


Figura 196 - Relacionamento com qualificador

3.4 Detalhes das classes

3.4.1 Mensagens e operações

Em modelos orientados a objetos, as mensagens representam os mecanismos de interação entre estes. Um objeto só pode receber mensagens que correspondam à invocação de uma operação da respectiva

classe. Durante a execução da modelagem, os diagramas podem conter, em caráter provisório, mensagens não associadas a operações de alguma classe. Entretanto, ao término da análise todas as mensagens têm de ser resolvidas em termos de operações de alguma classe.

Uma mensagem tem as seguintes partes:

- receptor - o objeto que recebe a mensagem;
- operação - a função requisitada do receptor;
- parâmetros - os dados para a operação.

Na Figura 197 modela-se uma interação simples entre dois objetos, através de uma mensagem sem parâmetros.

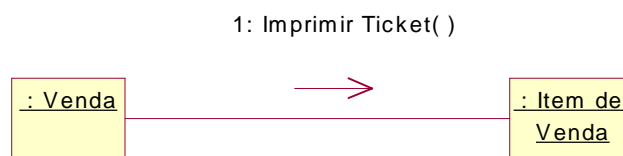


Figura 197 - Exemplo de mensagem sem parâmetro

Na Figura 198 modela-se uma colaboração complexa que envolve um ator (a rigor, uma instância de ator) e vários objetos. Os parâmetros das mensagens são incluídos no modelo.

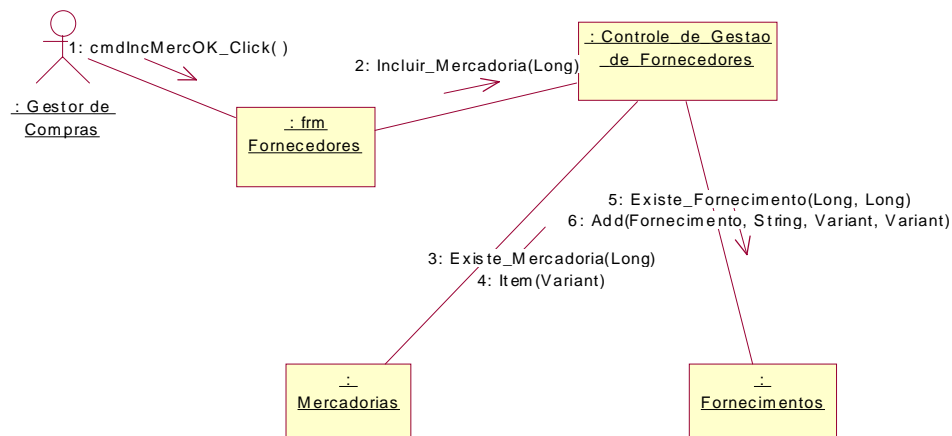


Figura 198 – Exemplo de mensagens com parâmetros

3.4.2 Atributos

Atributos são propriedades que descrevem as classes. Atributos equivalem a relacionamentos de composição em que:

- a classe parte é o tipo do atributo;
- o papel é o nome do atributo.

Atributos e relacionamentos podem ser alternativas para se expressar os mesmos conceitos (Figura 75). A escolha entre atributo e relacionamento deve visar à clareza do modelo. Geralmente atributos são de tipos equivalentes a classes pequenas e reutilizáveis, que representam abstrações de nível superior ao do domínio do problema.

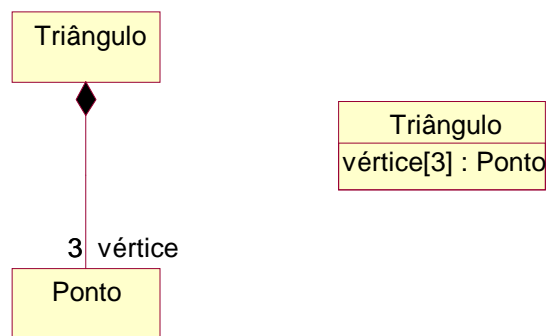


Figura 199 – Equivalência entre atributos e composições

3.4.3 Controle de acesso

Os tipos de acesso a operações ou atributos de cada classe incluem:

- **público**: qualquer outra classe pode usar (símbolo na UML +);
- **protegido**: só subclasses dessa classe podem usar (símbolo na UML #);
- **privado**: nenhuma outra classe pode usar diretamente (símbolo na UML -);
- **implementação**: declarado no corpo de uma operação.

Exceções ao controle normal de acesso, para determinadas classes clientes, podem ser feitas declarando-se essas classes como **amigas** da classe fornecedora.

Na Figura 89 são mostrados os controles de acesso dos atributos e operações de uma classe implementada em Visual Basic. Note-se que todos os atributos são privados. São públicas as operações provenientes do modelo de análise e as operações de acesso aos atributos; são privadas as operações de início e término da vida dos objetos.

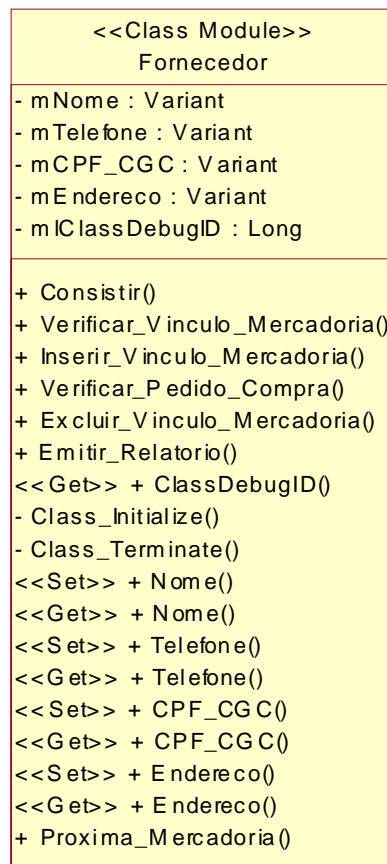


Figura 200 - Exemplo de controles de acesso

Note-se também, na Figura 89, o uso de estereótipos para indicar que construções da linguagem alvo serão utilizadas para implementar a classe (<< Class Module >>) e as operações de acesso aos atributos (<< Get >>, << Set >>).

3.5 Diagramas de interação

3.5.1 Diagramas de seqüência

Os diagramas de seqüência enfatizam o ordenamento temporal das ações. Eles são construídos de acordo com as seguintes convenções:

- linhas verticais representam os objetos;
- setas horizontais representam as mensagens passadas entre os objetos;
- rótulos das setas são os nomes das operações;
- a posição na vertical mostra o ordenamento relativo das mensagens;
- o diagrama pode ser complementado e esclarecido por anotações.

A UML prevê que os retângulos situados nas linhas verticais devem indicar o tempo de vida dos objetos. Esta convenção não foi seguida neste texto, por limitação da ferramenta de modelagem utilizada.

<p>O <u>Gestor de Compras</u> seleciona a mercadoria.</p> <p>O <u>Merci</u> verifica se existe algum pedido pendente que contenha esta mercadoria.</p> <p>Se não houver pedido pendente contendo a mercadoria a ser excluída,</p> <ul style="list-style-type: none">o <u>Merci</u> desvincula a mercadoria dos fornecedores (os fornecedores não mais fornecerão a mercadoria que esta sendo excluída);o <u>Merci</u> faz a remoção da mercadoria. <p>Se houver pedido pendente contendo a mercadoria a ser excluída,</p> <ul style="list-style-type: none">o <u>Merci</u> emite uma mensagem de erro.

Tabela 239 - Exemplo de fluxo de caso de uso

Os diagramas de seqüência são orientados para exprimir, de preferência, o desenrolar temporal de seqüências de ações. É mais difícil representar lógicas de seleção e repetição sem prejudicar a inteligibilidade do diagrama. Os roteiros representam desdobramentos da lógica do caso de uso. É preferível usar diagramas separados para representar roteiros resultantes de diferentes caminhos lógicos. Por exemplo, a lógica contida no fluxo da Figura 64 pode ser descrita através dos diferentes roteiros (Figura 65 e Figura 66). Para simplificar o exemplo, os diagramas não usam classes de fronteira.

<p>O <u>Caixeiro</u> registra itens de mercadoria.</p> <p>O <u>Merci</u> totaliza venda.</p> <p>O <u>Caixeiro</u> registra modo de venda.</p> <p>Se venda a prazo,</p> <ul style="list-style-type: none">o <u>Caixeiro</u> insere venda em contas a receber. <p>Senão,</p> <ul style="list-style-type: none">o <u>Caixeiro</u> registra pagamento.
--

Figura 201 – Exemplo de fluxo com lógica de seleção: Operação de Venda

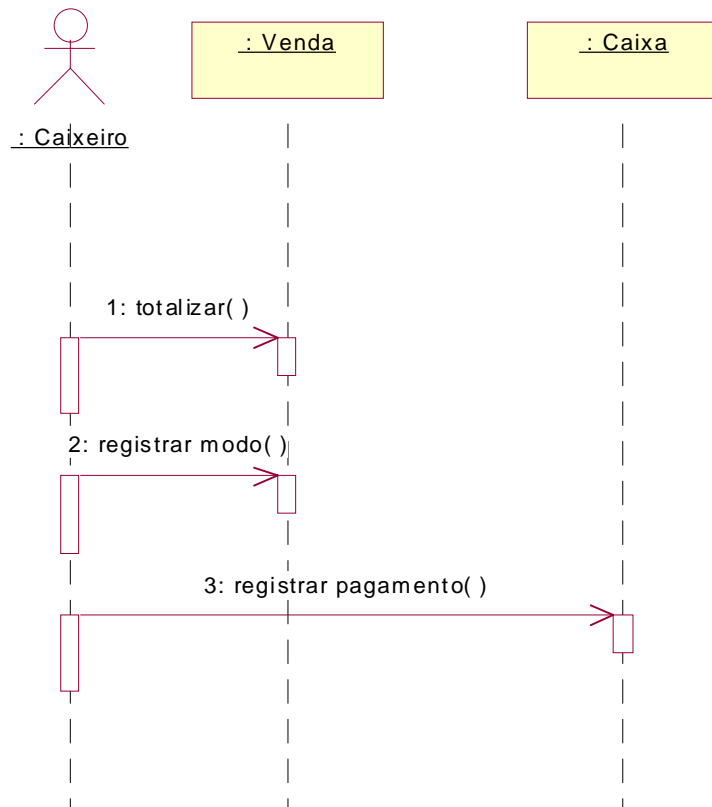


Figura 202 – Exemplo de roteiro alternativo: Operação de Venda à Vista

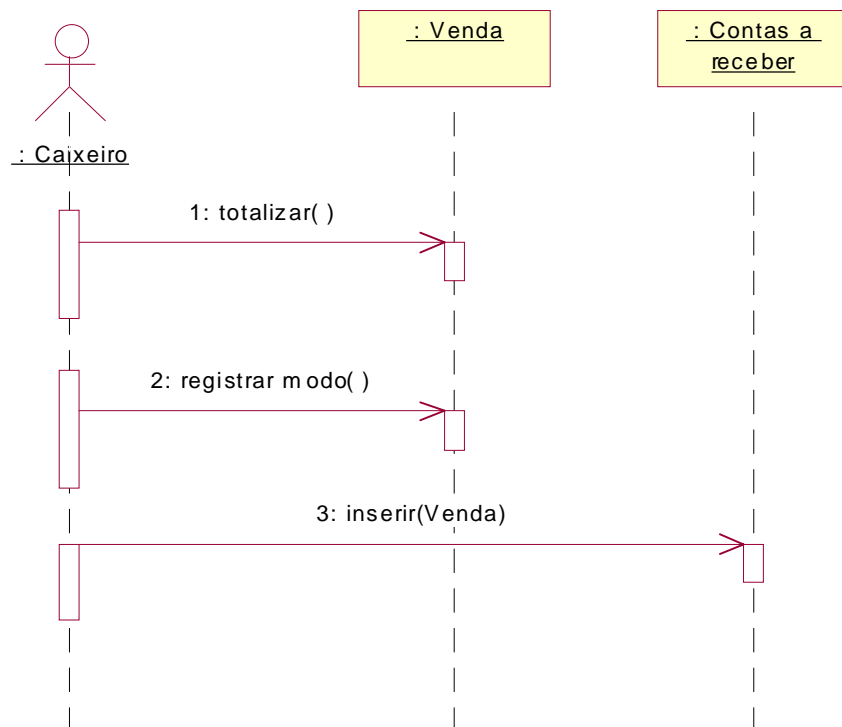


Figura 203 – Exemplo de roteiro alternativo: Operação de Venda a Prazo

Subfluxos curtos podem ser inseridos em um roteiro, indicando-se a lógica de ativação deles por meio de **expressões de recorrência**:

- [condição] - lógica de seleção;
- *[condição] - lógica de iteração.

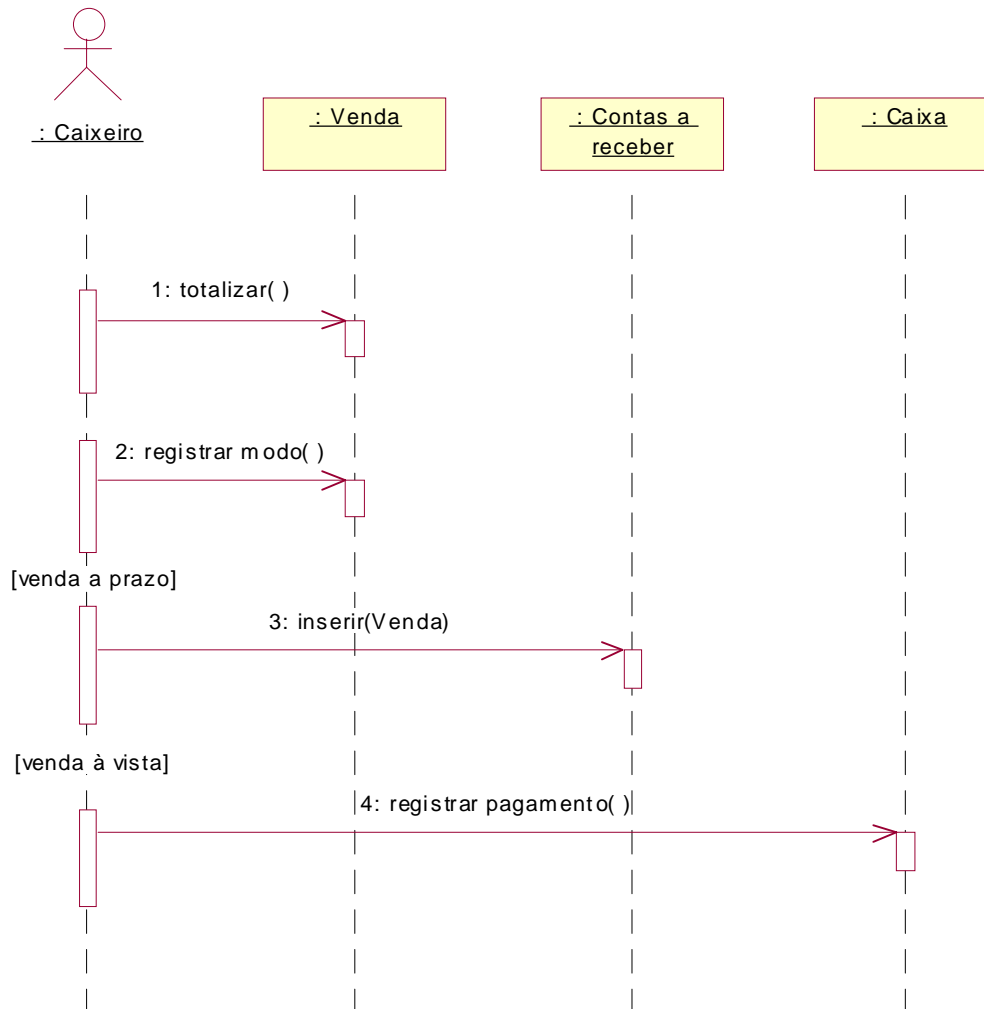


Figura 204 – Representação de lógica por meio de restrições

Diagramas de sequência podem ser complementados por meio de anotações. Os detalhes de processamento que não correspondem a interações entre classes foram lançados como anotações no diagrama da Figura 93. Estas anotações podem servir como especificações das operações das classes envolvidas.

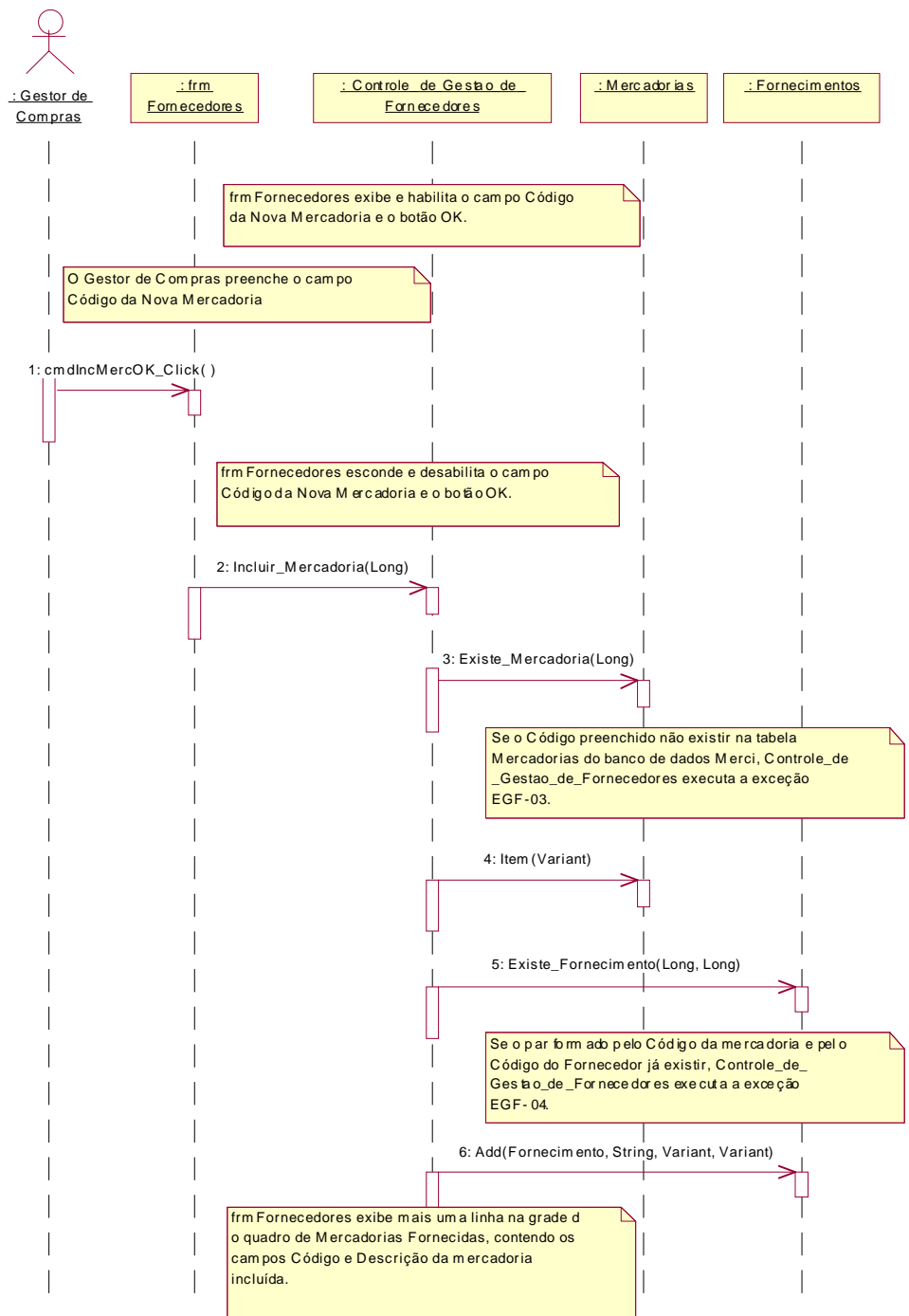


Figura 205 – Exemplo de diagrama de sequência para realização de um caso de uso

3.5.2 Diagramas de colaboração

Os diagramas de colaboração enfatizam os relacionamentos entre os objetos participantes. Eles são construídos de acordo com as seguintes convenções:

- nodos representam os objetos;

- os arcos representam as mensagens passadas entre os objetos;
- os rótulos dos arcos são os nomes das operações;
- os números de seqüência mostram o ordenamento relativo das mensagens;
- as anotações podem complementar o diagrama.

Nos diagramas de colaboração a ordenação das mensagens é mostrada apenas pela numeração delas. Na Figura 207 mostra-se o diagrama de colaboração que corresponde ao diagrama de seqüência da Figura 206.

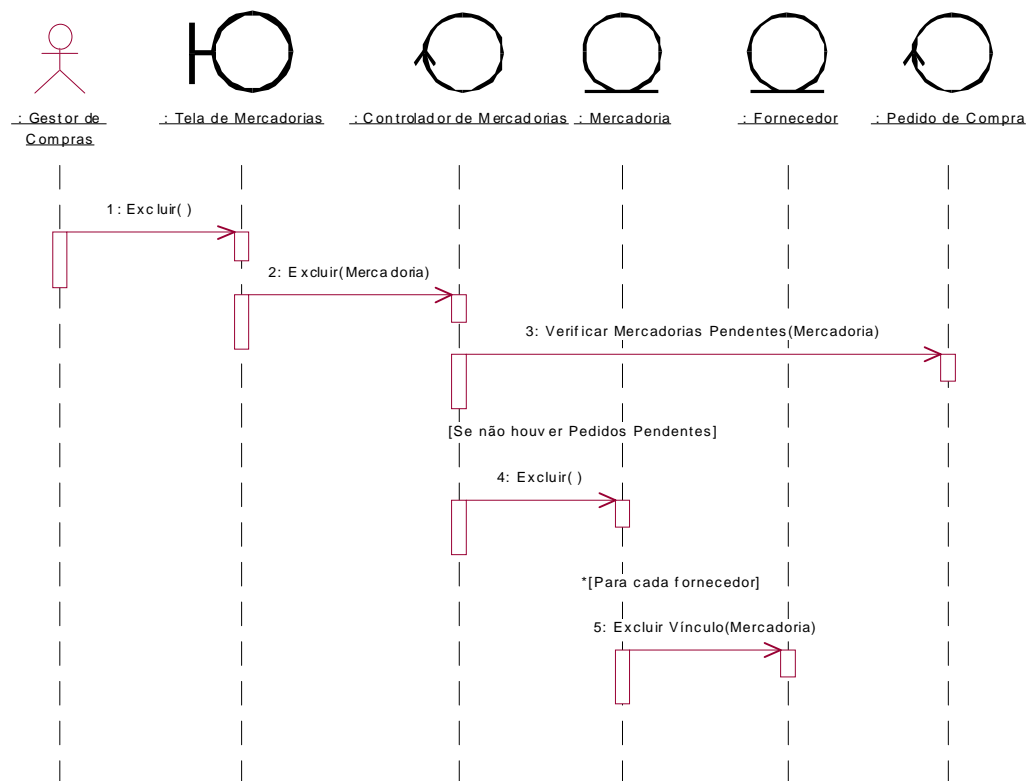


Figura 206 - Realização de fluxo através de diagrama de seqüência

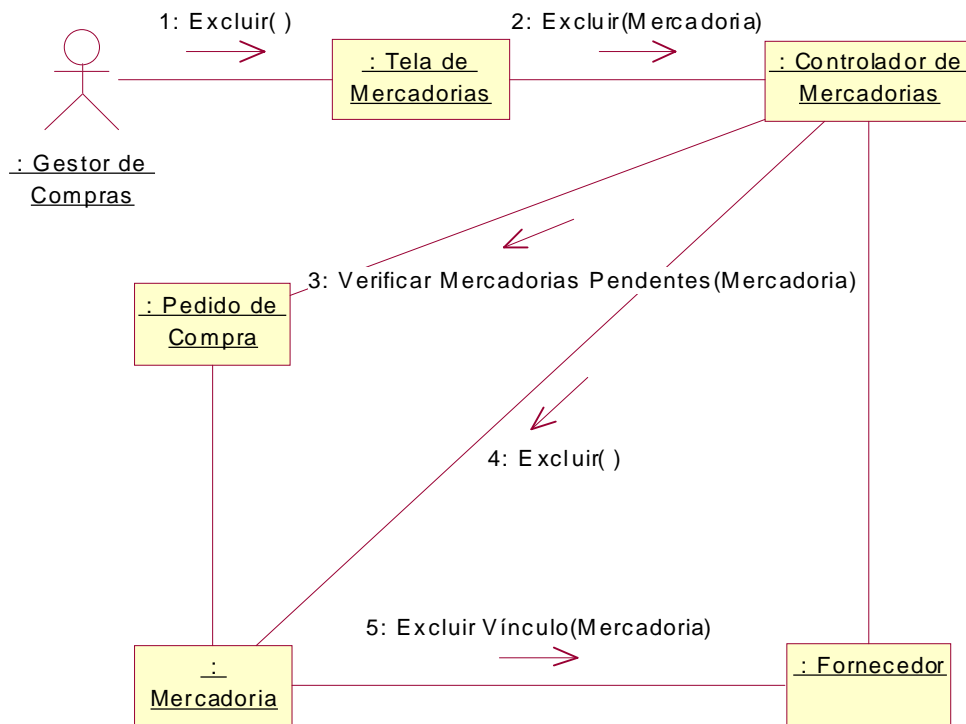


Figura 207 - Realização de fluxo através de diagrama de colaboração

3.5.3 Diagramas de objetos

Diagramas de colaboração mostram interações entre objetos, que geralmente têm o objetivo de cooperar para realizar roteiro de um caso de uso. Diagramas de objetos podem ser utilizados também para outras finalidades. Na Figura 109, um diagrama de objetos ilustra a integração *bottom-up*. No exemplo, as unidades U21 e U22 são testadas utilizando-se os controladores C1 e C2. No passo seguinte da integração, os controladores são substituídos pela unidade U1. “Substituído por” é uma ligação entre dois objetos.

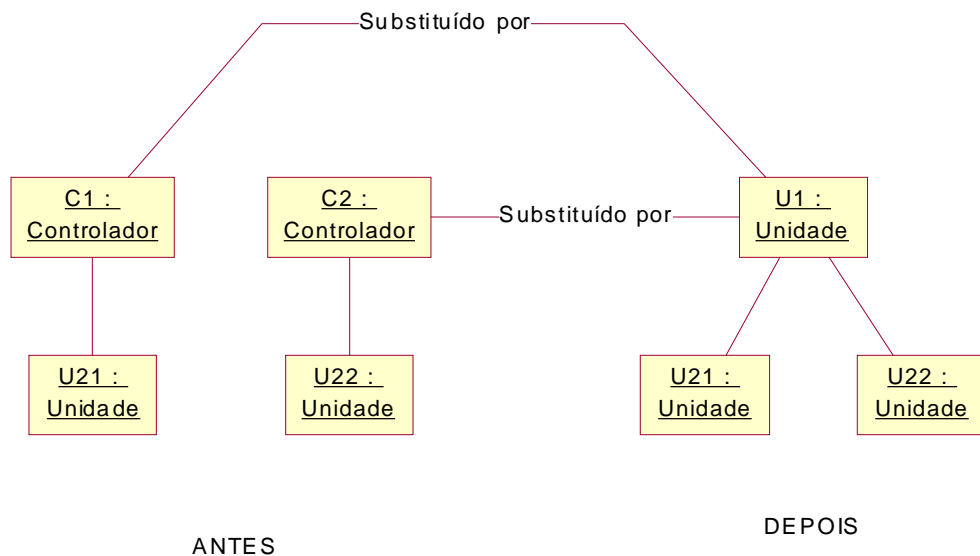


Figura 208 – Exemplo diagrama de objetos utilizado para ilustrar testes

Os diagramas de objetos podem ser usados para mostrar a passagem de dados e de controle. Na Figura 123, por exemplo, mostram-se como objetos uma unidade sob teste e os componentes utilizados para testá-la. As setas simples representam a direção de fluxo de controle: do objeto que chama (cliente do serviço ou emissor da mensagem) para o objeto chamado (fornecedor do serviço ou receptor da mensagem). As setas com bolinhas indicam o fluxo de dados (direção de passagem de parâmetros).

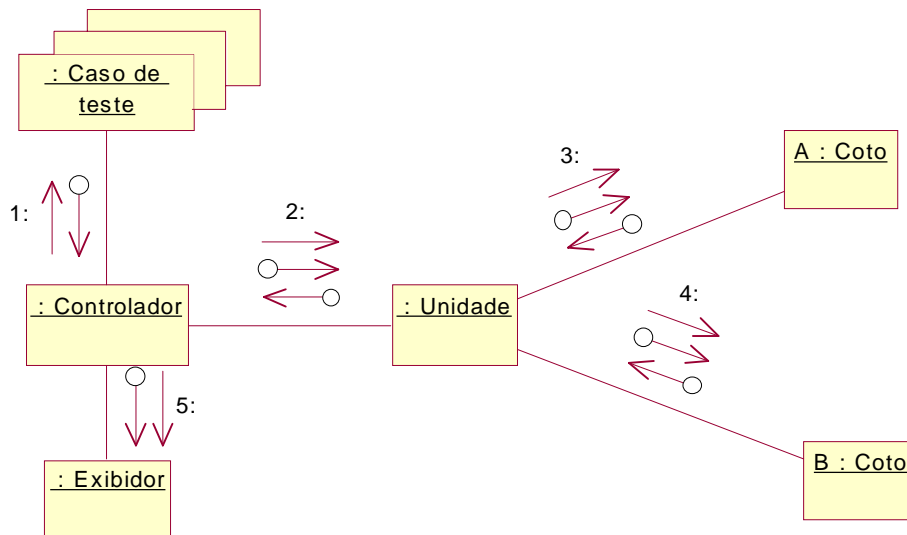


Figura 209 - Componentes de teste

4 Modelagem física

4.1 Visão de componentes

A visão de componentes descreve os aspectos estáticos do modelo físico. Ela mostra os arquivos de código fonte e objeto utilizados na implementação, assim como componentes reutilizados do ambiente e de outros projetos. Mostra também as dependências existentes entre eles, que indicam o impacto das alterações realizadas em cada componente. Na Figura 210, mostra-se que o componente executável “Merci_10” depende de vários componentes de código fonte correspondentes a formulários do Visual Basic e de um componente de código binário da tecnologia COM (um controle ActiveX de grade de entrada).

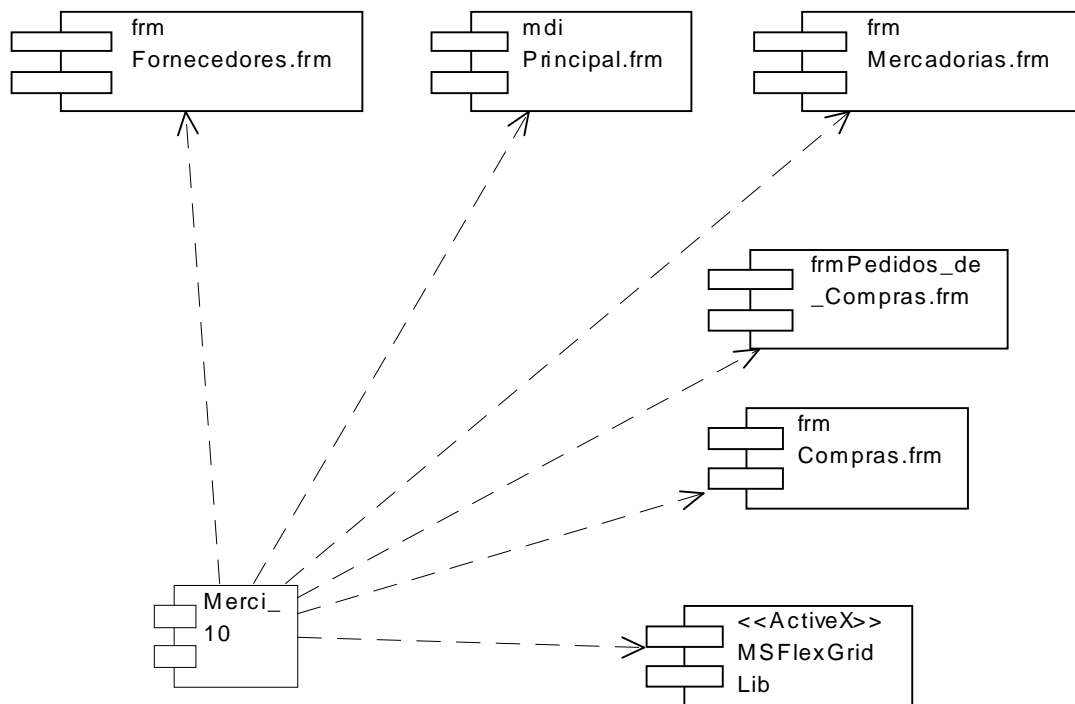


Figura 210 – Exemplo de diagrama de componentes (visão estática)

Os componentes podem ser implementados por meio de muitos tipos diferentes de arquivos. Nos diagramas de componentes, esses tipos de arquivos podem ser destacados por meio de estereótipos, representados por meio de identificadores ou de ícones próprios. A Figura 112 mostra alguns tipos de arquivos fontes, com os respectivos estereótipos icônicos. A Figura 113 mostra alguns exemplos de componentes de código objeto comuns em ambientes Windows, mostrados com estereótipos textuais.

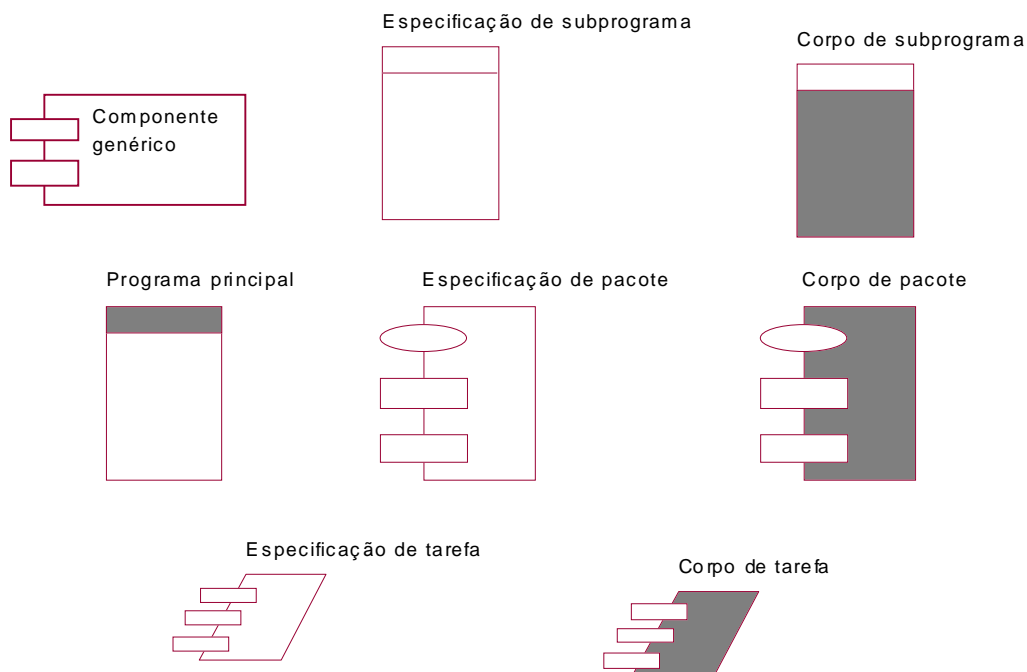


Figura 211 – Exemplo de componentes de código fonte

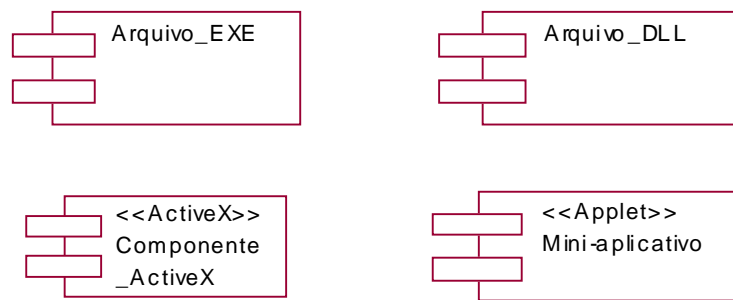


Figura 212 – Exemplo de componentes de código objeto

As interfaces representam um subconjunto dos recursos implementados por um componente, que têm um tema comum. Por exemplo, na Figura 114, um componente ActiveX oferece duas interfaces distintas. Aplicativos que não tratam de vendas a prazo, por exemplo, possivelmente não utilizarão a interface *Gestao_de_Clientes*, utilizando apenas os recursos oferecidos através da interface *Operacao_de_Venda*.

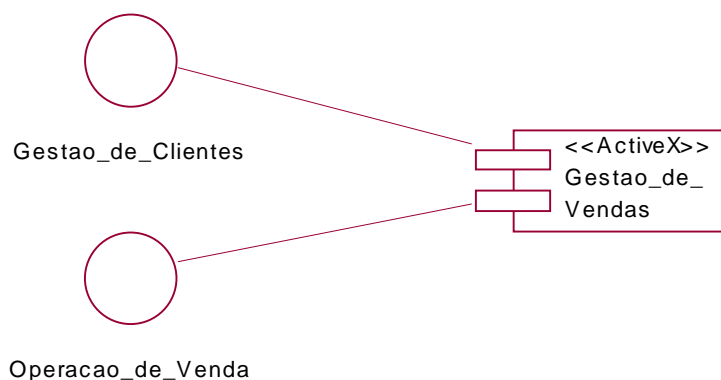


Figura 213 – Exemplo de interfaces de componente

4.2 Visão de desdobramento

O modelo físico dinâmico é descrito pelo diagrama de desdobramento, que mostra os processadores, os dispositivos e as conexões utilizados na implementação de um sistema. Em cada processador, pode-se mostrar os processos que nele executarão. Esse modelo é particularmente importante no caso de sistemas distribuídos. É comum a utilização de estereótipos para indicar diferentes tipos de processadores e dispositivos.

Na Figura 214 mostra-se que os processadores “Cliente de WWW 1”, “Cliente de WWW 2” e “Servidor de bancos de dados” são ligados ao processador “Servidor de WWW” através de conexões “Internet”. O “Servidor de WWW” executa os processos “Personal Web Server” e “Aplicativo CGI”. O “Cliente de WWW 1” está ligado a um dispositivo “Impressora”.

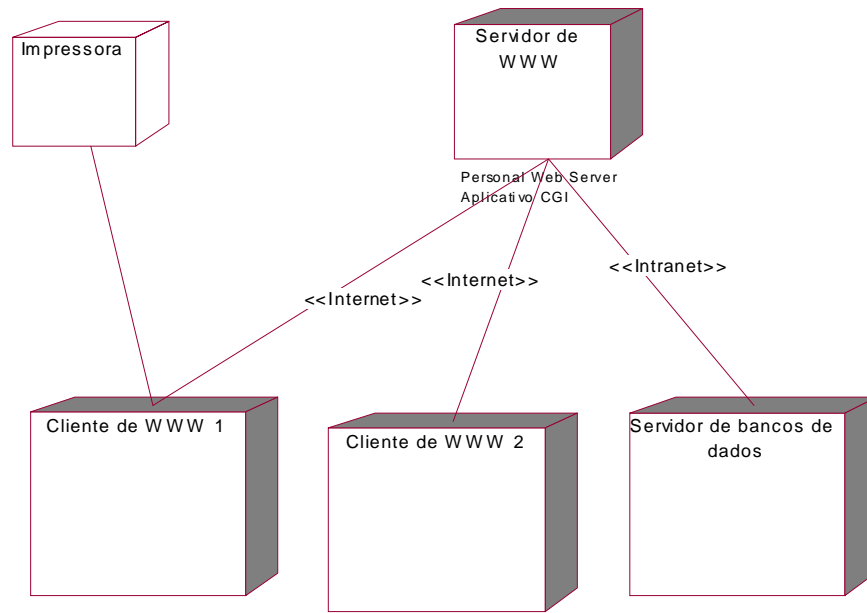


Figura 214 - Exemplo de diagrama de desdobramento

Página em Branco

Glossário

Este glossário procura, sempre que possível, ser consistente com as definições contidas em [IEEE94] e [Rumbaugh+99]. Diferenças, quando introduzidas, destinam-se a compatibilização entre esses padrões, adaptações em relação ao processo Praxis ou simplificações de natureza didática. A menos de observação em contrário, entende-se que todos os termos abaixo definidos referem-se a software.

Termo	Versão em inglês	Definição
Ação	<i>Action</i>	(UML) Unidade de processamento que resulta em mudança de estado de um item.
Acoplamento	<i>Coupling</i>	Maneira e grau de interdependência entre rotinas ou módulos.
Agregação	<i>Aggregation</i>	(UML) Associação que exprime um relacionamento entre todo e parte. Cf. composição, associação .
Análise	<i>Analysis</i>	<ol style="list-style-type: none"> 1. (Praxis) Fluxo que visa a detalhar, estruturar e validar os requisitos, em termos de um modelo conceitual do problema, de forma que estes possam ser usados como base para o planejamento e acompanhamento detalhados da construção do produto. 2. (UML) Estágio de um sistema que captura os requisitos e o domínio do problema, focalizando o que deve ser feito, e não como fazê-lo.
Aplicativo	<i>Application (software)</i>	Software que se destina a satisfazer necessidades específicas de um usuário.
Arquitetura	<i>Architecture</i>	Estrutura organizacional de um produto, que inclui a sua decomposição em partes e as conexões e interações entre estas. Cf. configuração, desenho .
Arquitetura de processo	<i>Process architecture</i>	Arcabouço conceitual para a organização dos elementos de um processo
Artefato	<i>Artifact</i>	Coisa concreta produzida em um projeto (por exemplo, código executável, código fonte, modelo ou documento).
Asserção	<i>Assertion</i>	Expressão lógica que deve ser sempre satisfeita durante a execução de um programa, se este estiver correto.
Associação	<i>Association</i>	Relacionamento semântico simples entre instâncias de uma classe.
Atividade	<i>Activity</i>	<ol style="list-style-type: none"> 1. (Praxis) Termo genérico para unidades de trabalho executadas em um passo de um processo. Cf. pacote de trabalho, passo. 2. (UML) Unidade não-atômica de execução de uma máquina de estados.
Ator	<i>Actor</i>	(UML) Abstração representativa de entidades externas que interagem com um produto ou sistema.
Atributo	<i>Attribute</i>	(UML) Descrição de um espaço com nome e tipo, dentro de uma classe, onde cada objeto desta classe mantém um valor deste tipo.
Auditoria	<i>Audit</i>	Exame de um resultado ou produto por avaliadores independentes, com o objetivo de verificar a conformidade com especificações, padrões, exigências contratuais e outros critérios. Cf. revisão técnica, inspeção .
Bala de prata	<i>Silver bullet</i>	Tecnologia ou método ao qual se atribuem poderes milagrosos.
Bateria de testes	<i>Test bucket</i>	Conjunto de testes que são executados durante

		determinado passo de um processo.
Característica	<i>Feature</i>	Aspecto distintivo de um item de software. Cf. requisito .
Caso de teste	<i>Test case</i>	Especificação das entradas, resultados previstos e condições de execução para um item a testar. Cf. procedimento de teste .
Caso de uso	<i>Use case</i>	Especificação de seqüências de ações, inclusive variantes, que um sistema ou produto pode executar quando interage com seus usuários ou com sistemas externos.
Ciclo de vida	<i>Life cycle</i>	Conjunto da história de um software, desde a percepção de sua necessidade até sua retirada de operação. Cf. projeto, desenvolvimento .
Classe	<i>Class</i>	(UML) Descritor para um conjunto de objetos que partilham os mesmos atributos, operações, relacionamentos e comportamento.
Cliente	<i>Client, customer</i>	Entidade que contrata a execução de um projeto, ou seu representante autorizado, com poder de aceitação de propostas e produtos. Cf. usuário .
Codificação	<i>Coding</i>	Expressão de um programa de computador em uma linguagem de programação. Cf. implementação .
Coesão	<i>Cohesion</i>	Maneira e grau com que as tarefas realizadas por uma rotina ou um módulo são relacionadas com as realizadas por outra rotina ou módulo. Cf. acoplamento .
Comissão de controle de configurações	<i>Configuration control board, CCB</i>	Grupo responsável por avaliar e autorizar ou não alterações em itens de configuração. Cf. grupo de gestão de configurações de software .
Componente	<i>Component</i>	<ol style="list-style-type: none"> 1. Uma das partes que constituem um produto ou sistema. Cf. módulo, rotina, unidade. 2. (UML) Parte física substituível de um sistema, que encapsula a implementação e realiza um conjunto de interfaces.
Composição	<i>Composition</i>	(UML) Agregação na qual coincidem os tempos de vida do todo e das partes. Cf. agregação .
Concepção	<i>Inception</i>	(Praxis) Fase na qual as necessidades dos usuários e os conceitos da aplicação são analisados o suficiente para justificar a especificação de um produto de software, resultando em uma proposta de especificação.
Condição de guarda	<i>Guard condition</i>	(UML) Condição que deve ser satisfeita para permitir o disparo de uma transição associada. Cf. transição .
Configuração	<i>Configuration</i>	Disposição de um sistema ou componente, definida pelo número, natureza e interconexões de suas partes constituintes. Cf. arquitetura, desenho .
Construção	<i>Build</i>	Estágio parcialmente operacional de um produto. Cf. liberação, versão, construção (construction) .
Construção	<i>Construction</i>	(Praxis) Fase na qual é desenvolvida (desenhada, implementada e testada) uma liberação completamente operacional de um produto, que atende aos requisitos especificados. Cf. Construção (build) .

Controlador	<i>Driver</i>	Componente de software que invoca outras unidades de software, possivelmente controlando e monitorando a execução destas. Cf. coto .
Coto	<i>Stub</i>	Componente que substitui provisoriamente uma unidade de software, como o objetivo de testar outras unidades que dependem dessa unidade. Cf. controlador .
Desenho	<i>Design</i>	<ol style="list-style-type: none"> 1. Processo de definição da arquitetura, interfaces, componentes e outras características de construção de um produto. 2. (UML) Estágio de um sistema que descreve como ele será implementado, em nível lógico superior ao do código. 3. (Praxis) Fluxo que visa a formular um modelo estrutural do produto, que sirva de base para a implementação, definindo os componentes a desenvolver e a reutilizar, assim como as interfaces entre si e com o contexto do produto. 4. O resultado do processo ou do fluxo de desenho. Cf. arquitetura, configuração.
Desenvolvedor	<i>Developer</i>	Profissional que exerce tarefas técnicas em um projeto de desenvolvimento de software.
Desenvolvimento	<i>Development</i>	Processo pelo qual as necessidades dos usuários são traduzidas em um produto de software. Cf. implementação .
Documento	<i>Document</i>	Artefato que pode ser lido por uma pessoa.
Elaboração	<i>Elaboration</i>	(Praxis) Fase na qual a especificação de um produto é detalhada o suficiente para modelar conceitualmente o domínio do problema, validar os requisitos em termos deste modelo conceitual e permitir um planejamento acurado da fase de construção.
Engenharia de requisitos	<i>Requirements engineering</i>	Conjunto das técnicas de levantamento, documentação e análise dos requisitos.
Especificação de requisitos	<i>Requirements specification</i>	Documento que estabelece os requisitos de um produto ou sistema.
Estado	<i>State</i>	<ol style="list-style-type: none"> 1. Condição ou modo de existência de um sistema ou componente. 2. Valores assumidos em um dado instante pelas variáveis que definem um sistema ou componente. 3. (UML) Condição ou situação da vida de um objeto, durante a qual ele executa uma atividade ou espera por algum evento.
Estereótipo	<i>Stereotype</i>	(UML) Nova espécie de elemento de modelagem, definido dentro de um modelo, baseado em um elemento existente.
Etapa	<i>Workflow step</i>	(Praxis) Passo constituinte de um fluxo.
Evento	<i>Event</i>	(UML) Especificação de uma ocorrência relevante, localizada no espaço e no tempo. Cf. transição .
Evolução	<i>Evolution, enhancement</i>	Melhoria e aumento de funcionalidade de um produto, ao longo das versões produzidas durante seu ciclo de vida.

	<i>enhancement</i>	Cf. manutenção .
Fase	<i>Phase</i>	(Praxis) Divisão maior de um processo, para fins gerenciais, que corresponde aos pontos principais de aceitação por parte do cliente.
Fluxo	<i>Workflow</i>	(Praxis) Subprocesso caracterizado por um tema técnico.
Garantia da qualidade	<i>Quality assurance</i>	Conjunto planejado e sistemático de ações necessárias para estabelecer um nível adequado de confiança na qualidade de um produto.
Gerência executiva	<i>Senior management</i>	Função organizacional responsável pela alta administração e pelas decisões estratégicas de uma organização.
Gerente de projeto	<i>Project manager</i>	Pessoa que tem responsabilidade completa por um projeto, inclusive a sua direção, o seu controle e a sua administração.
Gestão de configurações	<i>Configuration management</i>	Conjunto de procedimentos técnicos e gerenciais para identificação de artefatos e gestão de alterações destes.
Grupo de engenharia de processos de software	<i>Software engineering process group, SEPG</i>	Grupo responsável pela definição, manutenção e melhoria dos processos de software da organização.
Grupo de garantia da qualidade de software	<i>Software quality assurance group, SQA</i>	Grupo que planeja e implementa atividades que asseguram a qualidade de um produto de software, responsável por fornecer à gerência executiva um canal de informação sobre problemas, independentemente dos gerentes de projeto.
Grupo de gestão de configurações de software	<i>Software configuration management group, SCM</i>	Grupo que planeja, coordena, e implementa ações para gerir um sistema centralizado de guarda de configurações de software. Cf. comissão de controle de configurações .
Implementação	<i>Implementation</i>	(Praxis) Fluxo que visa a detalhar e implementar o desenho através de componentes de código e de documentação associada.
Inspeção	<i>Inspection</i>	Análise crítica de um material, efetuada por um grupo independente, com a utilização de um processo formal, para identificar defeitos e problemas de conformidade com padrões, na qual a geração de uma lista padronizada de defeitos é obrigatória, requerendo-se a ação dos produtores para remoção desses defeitos. Cf. revisão técnica .
Instância	<i>Instance</i>	Entidade individual, com identidade e valor próprios.
Integração	<i>Integration</i>	Processo de combinar componentes para formar um sistema maior.
Interativo	<i>Interactive</i>	Relativo a um sistema ou modo de operação no qual cada ação de um usuário provoca uma resposta imediata do sistema. Cf. on-line, iterativo .
Interface	<i>Interface</i>	<ol style="list-style-type: none"> 1. Fronteira de passagem de informação entre componentes de um produto ou sistema. 2. (UML) Conjunto nomeado de operações que caracteriza o comportamento de um elemento de

		modelagem.
Item a testar	<i>Test item</i>	Item de software que é objeto de um teste.
Item de configuração	<i>Configuration item</i>	Conjunto de unidades de software que é tratado como entidade única para fins de gestão de configurações. Cf. linha de base .
Iteração	<i>Iteration</i>	1. Processo de executar repetidamente uma sequência de passos. 2. (Praxis) Passo constituinte de uma fase, no qual se atinge um conjunto bem definido de metas parciais de um projeto.
Iterativo	<i>Iterative</i>	Que é realizado em iterações. Cf. iterativo .
Liberação	<i>Release</i>	Estágio parcialmente operacional de um produto, que é submetido à avaliação de usuários em determinado marco de um projeto. Cf. construção, versão .
Linha de base	<i>Baseline</i>	Conjunto revisto e aprovado formalmente de resultados de um projeto, associado a um marco deste projeto, que serve de base para o desenvolvimento futuro, e que só pode ser alterado através de procedimentos formalizados. Cf. item de configuração .
Manutenção	<i>Maintenance</i>	Processo de modificação de um produto após a entrega deste, geralmente para remover defeitos ou corrigir problemas. Cf. evolução .
Marco	<i>Milestone</i>	Ponto de tempo que representa um estado significativo de um projeto, geralmente associado à conclusão e aprovação de um ou mais resultados do projeto. Cf. resultado .
Mensagem	<i>Message</i>	(UML) Transporte de informação de um objeto a outro, com a expectativa de que isto provocará uma atividade.
Método	<i>Method</i>	(UML) Implementação de uma operação. Cf. operação .
Módulo	<i>Module</i>	Unidade de programa discreta e identificável em relação à compilação, combinação com outras unidades e carga. Cf. componente, rotina, unidade .
Multiplicidade	<i>Multiplicity</i>	(UML) Restrição aplicável à cardinalidade (tamanho) de um conjunto.
Navegação	<i>Navigation</i>	(UML) Travessia de uma conexão em um grafo.
Objeto	<i>Object</i>	(UML) Entidade discreta, com fronteira bem definida e identidade, que encapsula estado e comportamento. Cf. instância .
On-line	<i>On-line</i>	Relativo a um sistema ou modo de operação no qual os dados de entrada são captados diretamente no ponto de origem, ou os dados de saída são transmitidos diretamente para o ponto em que são usados. Cf. iterativo .
Operação	<i>Operation</i>	(UML) Especificação de uma consulta ou transformação que um objeto pode ser chamado a executar. Cf. mensagem, método .
Pacote	<i>Package</i>	Mecanismo de agrupamento de elementos de um modelo.
Pacote de trabalho	<i>Work package</i>	Etapas de um projeto que é tratada como indivisível, para

		fins de planejamento e controle.
Pacote lógico	<i>Logical package</i>	(UML) Pacote de classes e elementos correlatos de modelagem.
Papel (em um relacionamento)	<i>(Relationship) role</i>	(UML) Condição na qual um objeto participa de um relacionamento.
Papel (organizacional)	<i>(Organizational) role</i>	Unidade de responsabilidade dentro de uma organização, que pode ser assumida por um ou mais indivíduos.
Passo (de caso de uso)	<i>(Use case) step</i>	Sentença que descreve cada uma das divisões sequenciais do comportamento de um caso de uso.
Passo (de processo)	<i>(Process) step</i>	Divisão formal de um processo, com pré-requisitos, entradas, critérios de aprovação e resultados definidos.
Persistência	<i>Persistence</i>	Propriedade pela qual um objeto continua a existir mesmo depois da execução do programa que o criou.
Procedimento de teste	<i>Test procedure</i>	Conjunto detalhado de instruções para execução de testes. Cf. caso de teste .
Processo	<i>Process</i>	Conjunto de passos bem definidos para a execução de um projeto ou atividade. Cf. projeto.
Processo de negócio	<i>Business process</i>	Processo que faz parte da área de aplicação, onde, tipicamente, alguns procedimentos são executados por pessoas e outros são automatizados através do computador.
Processo de software	<i>Software process</i>	Processo usado para atividades relacionadas com software, como desenvolvimento, manutenção, aquisição e contratação. Neste livro, o termo refere-se ao processo de desenvolvimento de software, quando não qualificado.
Processo definido	<i>Defined process</i>	Processo que tem documentação que detalha o que é feito (produto), quando (passos), por quem (agentes), as coisas que usa (insumos) e as coisas que produz (resultados).
Produto	<i>Product</i>	Conjunto completo de itens de software, com os respectivos procedimentos e documentos, que é entregue a um cliente.
Projeto	<i>Project</i>	Unidade gerencial que cobre uma execução de um processo de desenvolvimento de software. Cf. processo .
Qualidade de processo	<i>Process quality</i>	Grau em que um processo garante a qualidade dos respectivos produtos.
Qualidade de produto	<i>Product quality</i>	Grau de conformidade de um produto com os respectivos requisitos.
Qualificador	<i>Qualifier</i>	(UML) Índice para travessia de uma associação.
Realização	<i>Realization</i>	(UML) Relacionamento entre uma especificação e a respectiva implementação.
Relacionamento	<i>Relationship</i>	(UML) Conexão semântica reificada entre elementos de um modelo. Cf. associação .
Requisito	<i>Requirement</i>	1. Característica que um produto deve possuir para que seja aceito. 2. Expressão documentada desta característica.
Requisito explícito	<i>Explicit</i>	Requisito descrito em um documento que arrola os

	<i>requirement</i>	requisitos de um produto, ou seja, um documento de especificação de requisitos.
Requisito implícito	<i>Implicit requirement</i>	Requisito decorrente de expectativas dos clientes e usuários, que são cobradas por estes, embora não documentadas.
Requisito normativo	<i>Normative requirement</i>	Requisito que decorre de leis, regulamentos, padrões e outros tipos de normas a que o tipo de produto deve obedecer.
Requisitos	<i>Requirements</i>	(Praxis) Fluxo que visa a obter um conjunto de requisitos de um produto, acordado entre cliente e fornecedor.
Resultado	<i>Work product, Result</i>	Artefato produzido em uma atividade de um projeto, cuja aprovação é condição para a conclusão definitiva desta atividade. Cf. produto, documento .
Revisão de apresentação	<i>Walthrough</i>	Revisão na qual o autor apresenta o material em ordem lógica, sem limite de tempo, a um grupo que checa o material à medida que ele vai sendo apresentado. Cf. inspeção, revisão técnica, revisão informal .
Revisão informal	<i>Desk check</i>	Análise de um material, efetuada pelo autor, com a possível assistência de partes, sem a utilização de um processo formal, para identificar defeitos e problemas de conformidade com padrões. Cf. inspeção, revisão técnica, revisão de apresentação .
Revisão técnica	<i>Peer review</i>	Análise de um material, efetuada por um grupo de pares dos autores, com a utilização de um processo formal, para identificar defeitos e problemas de conformidade com padrões. Cf. inspeção, revisão informal, revisão de apresentação .
Roteiro (de caso de uso)	<i>Scenario</i>	(UML) Seqüência de ações que ilustra um possível comportamento de um caso de uso.
Roteiro (de revisão)	<i>(Review) script</i>	Seqüência de passos que devem ser seguidos na revisão de um tipo específico de material.
Rotina	<i>Routine</i>	Parte de um programa que é chamada por outros programas ou partes de programa. Cf. módulo, unidade, componente .
Sistema	<i>System</i>	Coleção de componentes organizados para realizar uma função ou conjunto de funções.
Sistema embutido	<i>Embedded system</i>	Sistema físico do qual o software é parte indistinguível.
Software	<i>Software</i>	Parte programável de um sistema de informática, juntamente com a documentação associada.
Software de prateleira	<i>Off-the-shelf software</i>	Software comercial, vendido no mercado aberto.
Teste	<i>Test</i>	Atividade na qual um produto, sistema ou componente é executado sob condições especificadas, os resultados da execução são observados e registrados, e algum aspecto deste produto, sistema ou componente é avaliado. Cf. validação, verificação .
Teste de aceitação	<i>Acceptance test</i>	Teste que tem por objetivo verificar se um produto atende aos requisitos especificados para ele.

Teste de integração	<i>Integration test</i>	Teste que tem por objetivo verificar as interfaces entre as partes de uma arquitetura de produto
Teste de regressão	<i>Regression test</i>	Teste que tem por objetivo assegurar que alterações em partes do produto não afetem as partes já testadas.
Teste de unidade	<i>Unit test</i>	Teste individual de uma unidade ou grupo de unidades.
Testes	<i>Tests</i>	(Praxis) Fluxo que visa a verificar os resultados da implementação, através do planejamento, desenho e realização de baterias de testes.
Transação	<i>Transaction</i>	Seqüência de passos de processamento que não pode ser interrompida sem risco de provocar o aparecimento de um estado defeituoso de um sistema.
Transição	<i>Transition</i>	<ol style="list-style-type: none"> 1. (Praxis) Fase na qual um produto é colocado à disposição de uma comunidade de usuários para testes finais, treinamento e uso inicial. 2. (UML) Relacionamento entre dois estados de uma máquina de estados, que indica que um objeto no primeiro estado executará ações específicas e passará para o segundo estado quando ocorrer um evento especificado, desde que condições de guarda especificadas sejam satisfeitas. Cf. evento, condição de guarda.
Unidade	<i>Unit</i>	<ol style="list-style-type: none"> 1. Parte logicamente separável de um programa. Cf. componente, módulo, rotina. 2. Elemento separadamente testável de um programa.
Usuário	<i>User</i>	Pessoa que efetivamente usará um produto. Cf. cliente .
Usuário chave	<i>Key user</i>	Usuário designado pelo cliente como representativo, normalmente experiente no processo de negócio em que o produto será usado, e com poder de decisão sobre os requisitos deste.
Validação	<i>Validation</i>	Processo de avaliar um sistema, produto ou componente para determinar se ele satisfaz os respectivos requisitos. Cf. verificação, teste .
Verificação	<i>Verification</i>	Processo de avaliar um sistema, produto ou componente para determinar se os resultados de um passo do respectivo processo de desenvolvimento satisfazem as condições impostas no início do passo. Cf. validação, teste .
Versão	<i>Version</i>	Estágio da evolução de produto, que é entregue a clientes e usuários como resultado final de um projeto. Cf. construção, versão .

Página em branco

Bibliografia

- [ABNT93] ABNT. *NBR ISO 9000-3. Normas de gestão da qualidade e garantia da qualidade. Parte 3: Diretrizes para a aplicação da NBR 19001 ao desenvolvimento, fornecimento e manutenção de software*. ABNT, Rio de Janeiro - RJ, Nov. 1993.
- [ABNT94] ABNT. *NBR ISO/IEC 9126: Tecnologia da informação - Avaliação de produto de software - Características da qualidade e diretrizes para o seu uso*. ABNT, Rio de Janeiro - RJ, Ago. 1994.
- [Adams97] Scott Adams. *The Dilbert Principle: A Cubicle's-Eye View of Bosses, Meetings, Management Fads & Other Workplace Afflictions*. Harperbusiness, 1997.
- [Albuquerque+99] Jones Albuquerque, Claudionor Coelho Jr., Wilson de Pádua Paula Filho, Antônio Otávio Fernandes. *Using PSP on Undergraduate Computer Science Program*. Proceedings of the Symposium on Software Technology - 1999 (SoST'99). 28th International Conference of the Argentine Informatics and Operations Research Society (SADIO). Buenos Aires - Argentina, pp. 1-6, 1999.
- [Ambler99] Scott Ambler. Tracing Your Design. *Software Development Magazine*. Abr. 1999.
- [Blaha+98] Michael Blaha e William Premerlani. *Object-Oriented Modeling and Design for Database Applications*. Prentice Hall, Englewood Cliffs, NJ, 1998
- [Blaha+99] Michael Blaha e William Premerlani. Using UML to Design Database Applications. *Rose Architect* 1(3), Spring 1999.
- [Booch+97] Grady Booch, Ivar Jacobson e James Rumbaugh. *UML Document Set, version 1.0*. <http://www.rational.com/uml/references/>.
- [Booch+99] Grady Booch, Ivar Jacobson e James Rumbaugh. *The Unified Modeling Language User Guide*. Addison-Wesley, Reading - MA, 1999.
- [Booch94] Grady Booch. *Object-Oriented Analysis and Design with Applications* 2nd. ed. Benjamin/Cummings, Redwood City - CA, 1994.
- [Booch96] Grady Booch. *Object Solutions: Managing the Object-Oriented Project*. Addison-Wesley, Reading - MA, 1996.
- [Brooks95] Frederick P. Brooks, Jr. *The Mythical Man-Month*. Addison-Wesley, Reading - MA, 1995.
- [Carpenter+95] Maribeth B. Carpenter e Harvey K. Hallman. *Training Guidelines: Purchasing Training for a Software Organization*. CMU/SEI-95-TR-010. Software Engineering Institute, Pittsburgh - PA, Dec. 1995.
- [Carpenter+95a] Maribeth B. Carpenter e Harvey K. Hallman. *Training Guidelines: Creating a Training Plan for a Software Organization*. CMU/SEI-95-TR-007. Software Engineering Institute, Pittsburgh - PA, Sep. 1995.
- [Cheng+95] Lin Chih Cheng, Carlos Alberto Scapin, Carlos Augusto de Oliveira, Eduardo Krafetuski, Fátima B. Drumond, Flávio S. Boan, Luiz R. Prates e Renato M. Vilela. *QFD: Planejamento da Qualidade*. Escola de Engenharia da UFMG. Fundação Christiano Ottoni. Belo Horizonte, MG,

- 1995.
- [Constantine+99] Larry L. Constantine e Lucy A.D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*. Addison-Wesley, Reading - MA, 1999.
- [Curtis+95] Bill Curtis, William E. Hefley e Sally Miller. *Overview of the People Capability Maturity Model*. CMU/SEI-95-MM-01. Software Engineering Institute, Pittsburgh - PA, Sep. 1995.
- [Curtis+95a] Bill Curtis, William E. Hefley e Sally Miller. *People Capability Maturity Model*. CMU/SEI-95-MM-02. Software Engineering Institute, Pittsburgh - PA, Sep. 1995.
- [Davis93] Alan M. Davis. *Software Requirements: Objects, Functions and States*. Prentice-Hall, Upper Saddle River - NJ, 1993.
- [Diaz+97] Diaz, M. e Sligo, J., How Software Process Improvement Helped Motorola. *IEEE Software*, 114(5), Set./Out. 1997.
- [Dion93] Raymond Dion Process Improvement and the Corporate Balance Sheet. *IEEE Software* 10(4), Jul. 1993, pp. 28-35.
- [Dreger89] J. Brian Dreger. *Function Point Analysis*. Prentice-Hall, Englewood Cliffs - NJ, 1989.
- [Dymond95] Kenneth M. Dymond. *A Guide to the CMM*. Process Inc. US, 1995.
- [Fiorini+98] Fiorini, S.T., Staa, A.v. e Baptista, R.M., *Engenharia de Software com CMM*, Brasport, 1998.
- [Floyd+93] Thomas D. Floyd, Stu Levy e Arnold B. Wolfman. *Winning the New Product Development Battle*. IEEE, New York - NY, 1994.
- [Fowler+97] Martin Fowler e Kendall Scott. *UML Distilled – Applying the Standard Object Modeling Language*. Addison-Wesley, Reading - MA, 1997.
- [Fowler97] Martin Fowler. *Analysis Patterns: Reusable Object Models*. Addison-Wesley, Reading - MA, 1997.
- [Freedman+93] Daniel P. Freedman e Gerald M. Weinberg. *Manual de Walkthroughs*. Makron Books do Brasil, São Paulo - SP, 1993.
- [Gamma+94] Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading - MA, 1994.
- [Gause+89] Donald C. Gause e Gerald M. Weinberg. *Explorando Requerimentos de Sistemas*. Makron Books do Brasil, São Paulo - SP, 1991.
- [Goldenson+95] Dennis R. Goldenson e James D. Herbsleb. *After the Appraisal: A Systematic Survey of Process Improvement, its Benefits, and Factors that Influence Success*. Software Engineering Institute, CMU/SEI-95- TR-009, Ago. 1995.
- [Haley96] Haley, Thomas J.. Software Process Improvement at Raytheon. *IEEE Software*, Nov. 1996.
- [Herbsleb+94] James Herbsleb, Anita Carleton et al. *Benefits of CMM-Based Software Process Improvement: Initial Results*. Software Engineering Institute, CMU/SEI-94-TR-13, Ago. 1994.
- [Hitt85] William D. Hitt. *Management in Action: Guidelines for New Managers*.

- Batelle Press. Columbus-OH, 1985.
- [Hitt88] William D. Hitt. *The Leader-Manager: Guidelines Action*. Batelle Press. Columbus-OH, 1988.
- [Hix+93] Deborah Hix e H. Rex Hartson. *Developing User Interfaces: Ensuring Usability through Product and Process*. Wiley, New York - NY, 1993.
- [Humphrey+91] Watts S. Humphrey, Terry R. Snyder e Ronald R. Willis, "Software Process Improvement at Hughes Aircraft. *IEEE Software* 8(4), Jul. 1991.
- [Humphrey90] Watts S. Humphrey. *Managing the Software Process*. Addison-Wesley, Reading - MA, 1990.
- [Humphrey95] Watts S. Humphrey. *A Discipline for Software Engineering*. Addison-Wesley, Reading - MA, 1995.
- [Humphrey96] Watts S. Humphrey. *Managing Technical People: Innovation, Teamwork, and the Software Process*. Addison-Wesley, Reading - MA, 1996.
- [Humphrey97] Watts S. Humphrey. *Introduction to the Personal Software Process*. Addison-Wesley, Reading - MA, 1997.
- [Humphrey99] Watts S. Humphrey. *Introduction to the Team Software Process*. Addison-Wesley, Reading - MA, 1999.
- [Hyde97] Randall Hyde. *IMA Software Development Guidelines*. http://webster.ucr.edu/Page_softeng/softDevGuide.html
- [IEEE94] IEEE. *IEEE Standards Collection - Software Engineering*. IEEE, New York - NY, 1994.
- [Jacobson+94a] Ivar Jacobson, Maria Ericsson e Agneta Jacobson. *The Object Advantage: Business Process Reengineering with Object Technology*. Addison-Wesley, Reading - MA, 1994.
- [Jacobson+97] Ivar Jacobson, Martin Griss e Patrik Jonsson. *Software Reuse: Architecture, Process and Organization for Business Success*. Addison-Wesley, Reading - MA, 1997.
- [Jacobson+99] Ivar Jacobson, James Rumbaugh e Grady Booch. *Unified Software Development Process*. Addison-Wesley, Reading - MA, 1999.
- [Jacobson94] Ivar Jacobson. *Object-Oriented Software Engineering*. Addison-Wesley, Reading - MA, 1994.
- [Jones94] Capers Jones. *Assessment and Control of Software Risks*. Yourdon Press - Prentice-Hall, Upper Saddle River - NJ, 1994.
- [Jones98] Capers Jones. *The Year 2000 Software Problem*. Addison-Wesley, Reading - MA, 1998.
- [Keil+98] Mark Keil, Paul E. Cule, Kalle Lyytinen e Roy C. Schmidt. A Framework for Identifying Software Process Risks. *CACM* 41(11), Nov. 1998.
- [Laplante93] Phillip A. Laplante. *Real-Time Systems Design and Analysis*. IEEE, New York - NY, 1993.
- [Loomis94] Mary E.S. Loomis, Hitting the relational wall. *JOOP*, Jan. 1994.
- [Love93] Tom Love. *Object Lessons*. SIGS Books, New York - NY, 1993.
- [McConnell93] Steve McConnell. *Code Complete: A Practical Handbook of Software*

- Construction*. Microsoft Press, 1993.
- [McConnell96] Steve McConnell. *Rapid Development*. Microsoft Press, 1996.
- [McFeeley96] Bob McFeeley. *IDEAL: A User's Guide for Software Process Improvement*. CMU/SEI-96-HB-001. Software Engineering Institute, Pittsburgh - PA, Feb. 1996.
- [Mead+96] Nancy Mead, Lawrence Tobin e Suzanne Couturiaux. *Best Training Practices Within the Software Engineering Industry*. CMU/SEI-96-TR-036. Software Engineering Institute, Pittsburgh - PA, Nov. 1996.
- [Meyers92] Scott Meyers. *Effective C++*. Addison-Wesley, Reading - MA, 1992.
- [Meyers96] Scott Meyers. *More Effective C++*. Addison-Wesley, Reading - MA, 1996.
- [Paula+98a] Wilson de Pádua Paula Filho e Cláudio Ricardo Guimarães Sant'Ana. *Manual de Engenharia de Produtos de Software - Parte I: Recomendações*. RT – DCC - 008/1998.
- [Paula+98b] Wilson de Pádua Paula Filho e Cláudio Ricardo Guimarães Sant'Ana. *Manual de Engenharia de Produtos de Software - Parte II: Padrões e Modelos*. RT – DCC - 009/1998.
- [Paula+98c] Wilson de Pádua Paula Filho e Cláudio Ricardo Guimarães Sant'Ana. *Manual de Engenharia de Processos de Software - Parte I: Políticas*. RT – DCC - 015/1998.
- [Paula+98d] Wilson de Pádua Paula Filho e Cláudio Ricardo Guimarães Sant'Ana. *Manual de Engenharia de Processos de Software - Parte II: Padrões e Modelos*. RT – DCC - 016/1998.
- [Paula95] Wilson de Pádua Paula Filho. Implantação de um programa da qualidade em um convênio Universidade-Empresa. *Workshop sobre Qualidade e Produtividade em Software*. IX Simpósio Brasileiro de Engenharia de “Software”, Recife - PE, 1995.
- [Paula96] Wilson de Pádua Paula Filho (ed.). *Manual de Processos versão 1.0 - CASE : Programa de Capacitação em Sistemas de Engenharia*. RT DSE 18/96. DCC-ICEX-UFMG, Belo Horizonte - MG, 1996.
- [Paulk+93] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis e Charles V. Weber. *Capability Maturity Model for Software, Version 1.1*. CMU/SEI-93-TR-24. Software Engineering Institute, Pittsburgh - PA, Feb. 1993.
- [Paulk+93a] Mark C. Paulk, Charles V. Weber, Suzanne M. Garcia, Mary Beth Chrissis e Marilyn Bush. *Key Practices of the Capability Maturity Model, Version 1.1*. CMU/SEI-93-TR-25. Software Engineering Institute, Pittsburgh - PA, Feb. 1993.
- [Paulk+95] Mark C. Paulk, Charles V. Weber, Bill Curtiss e Mary Beth Chrissis. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley, Reading - MA, 1995.
- [Paulk94] Mark C. Paulk. *A Comparison of ISO 9001 and the Capability Maturity Model for Software*. CMU/SEI-94-TR-012. Software Engineering Institute, Pittsburgh - PA, Ago. 1994.
- [Perry95] William Perry. *Effective Methods for Software Testing*. JohnWiley, New York, 1995.

- [Pressman95] Roger Pressman. *Engenharia de Software*. 3a ed. Makron Books do Brasil, S. Paulo, 1995.
- [Quatrani98] Terry Quatrani. *Visual Modeling with Rational Rose and UML*. Addison-Wesley, Reading - MA, 1998.
- [Robertson+99] Suzanne Robertson e James Robertson. *Mastering the Requirements Process*. Addison-Wesley, Harlow-England, 1999.
- [Rumbaugh+99] James Rumbaugh, Ivar Jacobson e Grady Booch. *Unified Modeling Language Reference Manual*. Addison-Wesley, Reading - MA, 1999.
- [Rumbaugh91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy e William Lorensen. *Object-Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, NJ, 1991.
- [Saiedian+95] Hossein Saiedian e Richard Kuzara. SEI Capability Maturity Model's Impact on Contractors. *IEEE Computer* 28(1), Jan. 1995.
- [Schneider98] Geri Schneider e Jason P. Winters. *Applying Use Cases: A Practical Guide*. Addison-Wesley, Reading - MA, 1998.
- [Shneiderman92] Ben Shneiderman. *Designing the user interface: strategies for effective human-computer interaction*. Addison-Wesley, 2^a. edição, 1992.
- [Sims94] David Sims. Motorola India Self-Assesses at Level 5. *IEEE Software*, Vol. 11(2). Mar. 1994.
- [Taylor90] David A. Taylor. *Object-Oriented Technology: a Manager's Guide*. Addison-Wesley, Reading - MA, 1990.
- [Weinberg93] Gerald M. Weinberg. *Software com Qualidade: Pensando e Idealizando Sistemas*. Makron Books do Brasil, São Paulo - SP, 1993.
- [Weinberg94] Gerald M. Weinberg. *Software com qualidade vol. 2: Medidas de Primeira Ordem*. Makron Books do Brasil, São Paulo - SP, 1994.
- [White94] Iseult White. *Using the Booch Method: a Rational Approach*. Benjamin/Cummings, Redwood City - CA, 1994.
- [Yourdon97] Edward Yourdon. *Death March: The Complete Software Developer's Guide to Surviving 'Mission Impossible' Projects*. Prentice-Hall, Upper Saddle River - NJ, 1997.

Página em branco