Engenharia de Software:
fundamentos, métodos e padrões

(Excertos de livro a ser publicado pela Editora LTC – proibida a reprodução)

Wilson de Pádua Paula Filho
Março de 2000

Página em branco

Índice analítico

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? Em termos mais práticos: 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 hierarquia[1]: alimentação, moradia, segurança, afeição, auto-estima... Todo produto de engenharia se justifica através da satisfação de uma destas 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 uma 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 às 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.

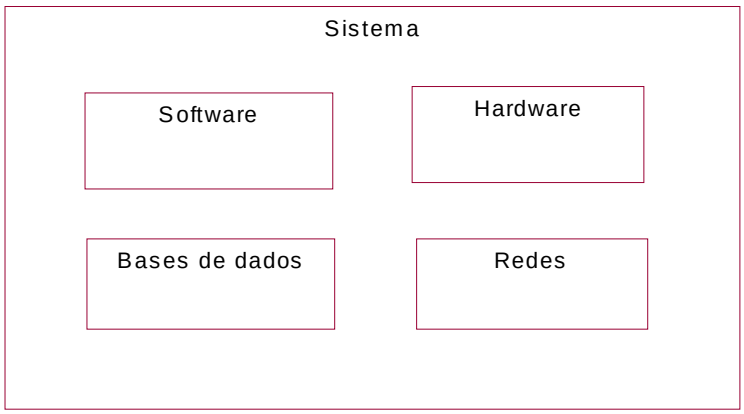

Figura 1 - Sistema de informática e suas partes

Na prática, o engenheiro de software será chamado com freqüência 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:
  - cheios de defeitos;
  - difíceis de usar;

- 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, de dificuldade de uso do próprio sistema, ou de 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, onde, 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: na "boca do 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 desta classe de problemas.

2.2 Produção

2.2.1 Ciclos de vida

A Engenharia de Software se preocupa com o software enquanto 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, um 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.

<table>
<tr><td rowspan="8">Ciclo de vida</td><td colspan="4">Percepção da necessidade</td></tr>
<tr><td rowspan="6">Desenvolvimento</td><td colspan="3">Concepção</td></tr>
<tr><td colspan="3">Elaboração</td></tr>
<tr><td rowspan="4">Construção</td><td colspan="2">Desenho inicial</td></tr>
<tr><td rowspan="3">Liberação</td><td>Desenho detalhado</td></tr>
<tr><td>Codificação</td></tr>
<tr><td>Testes de unidade</td></tr>
<tr><td colspan="2">Testes alfa</td></tr>
<tr><td colspan="3">Transição</td></tr>
<tr><td colspan="4">Operação</td></tr>
<tr><td colspan="4">Retirada</td></tr>
</table>

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, que chamaremos de **gerente do projeto**) e outros recursos. Um projeto representa a execução de um **processo**.

Quando um processo é bem definido, ele definirá subdivisões que permitam avaliar o progresso de um projeto, e corrigir seus rumos quando acontecerem problemas. Estas 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 aos 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 podem conter requisitos incompletos, inconsistentes ou ambíguos. Alguns destes 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. Esta 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 insistir sempre na elaboração de uma boa especificação de requisitos. Faz parte do trabalho deles o convencimento dos 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 necessidades deles 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.

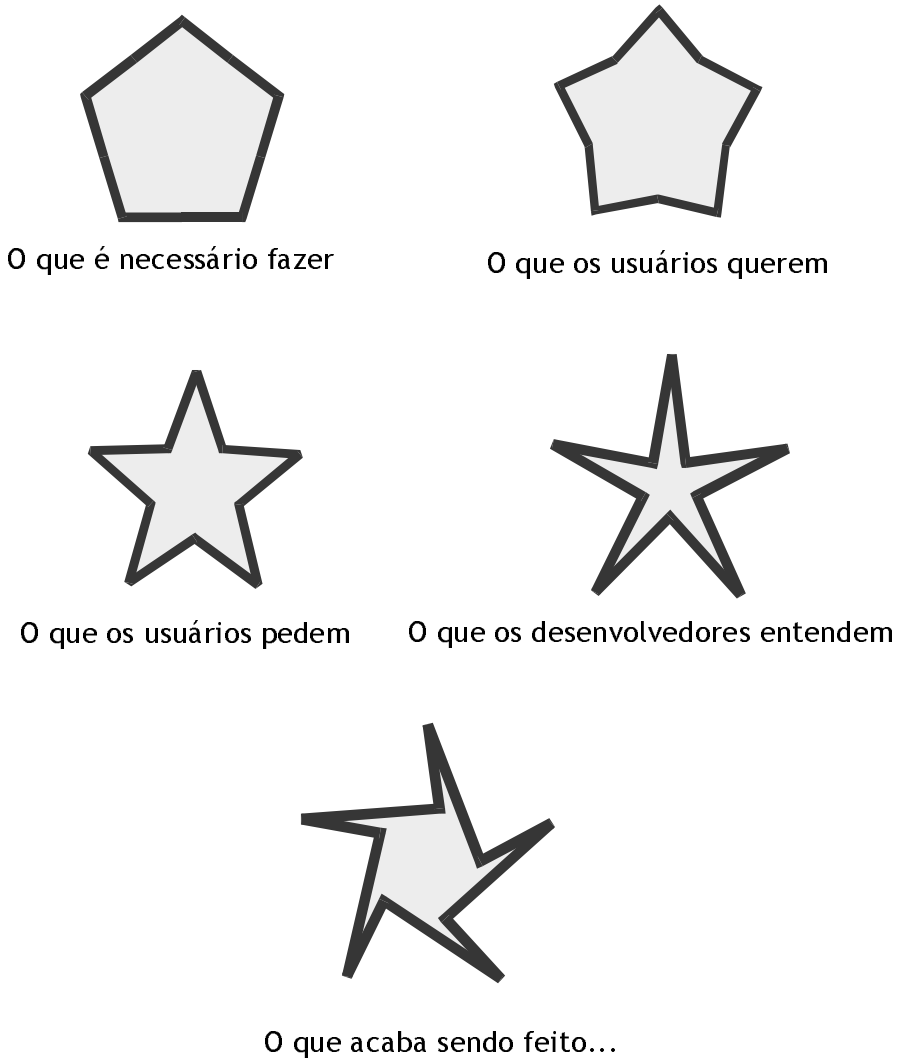

Figura 2 - Como os requisitos evoluem

2.3.4 Gestão dos requisitos

Um problema comum no desenvolvimento de software é a **instabilidade dos requisitos**, que acontece 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 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 inevitáveis alterações.

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 próximo contrato, eles provavelmente escolherão o oferente que prometer menor prazo e/ou menor custo. Se for um projeto interno da organização, farão todo tipo de pressões 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 isto 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 antigüidade, com resultados geralmente desastrosos.

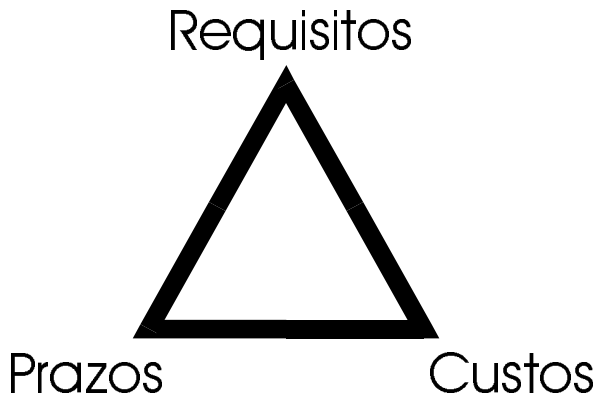

Figura 3 - Um triângulo crítico da Engenharia de Software

Requisitos, prazos e custos formam os vértices de um triângulo crítico. Aumentos de requisitos levam a aumentos de prazos ou custos, ou ambos. Reduções de requisitos podem levar a reduções de prazos ou 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 desta 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 orçamento propostos são realistas; se preciso, recorrendo aos serviços uma terceira parte.

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, estes 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 não previstos 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 através 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**. Ele 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 esta 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 informático, 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 estes defeitos foram causados por erro de uma falível pessoa, 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 disto, como estes requisitos, por definição, não são documentados, é bastante provável que eles não tenham sido considerados 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 do redução dos requisitos de um produto. A qualidade, por outro lado, é conseqüência 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. Estas 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. Isto 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, isto se justifica pelo caráter crítico do sistema: por exemplo, quando defeitos podem colocar pessoas em perigo, ou causar prejuízos materiais vultosos.

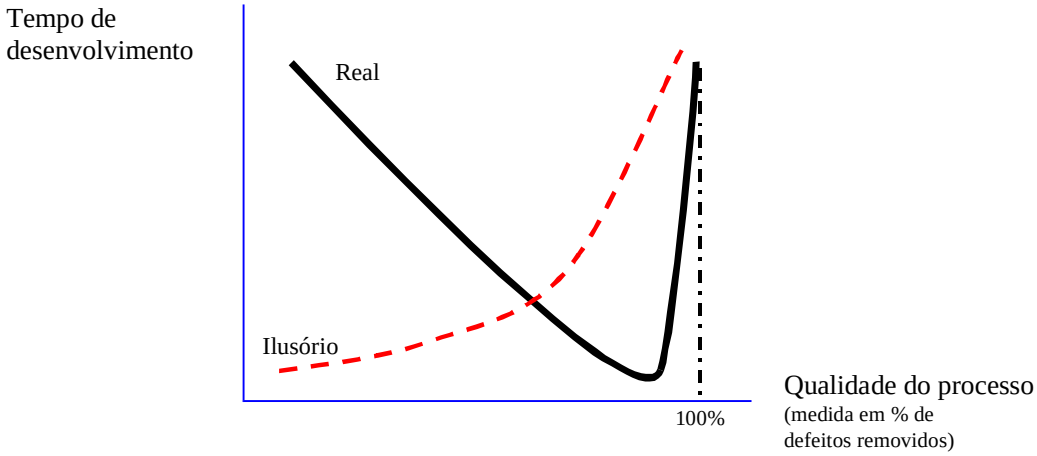

Figura 4 - A curva prazo x qualidade

Vários métodos de garantia da qualidade levam em conta uma limitação humana: somos mais eficazes para achar os defeitos dos outros do que nosso próprios defeitos. Por isto, 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 destes 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 mesmo 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 atualização de documentos e modelos é rotineira 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 destas 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 isto não aconteça, a organização contratante deve estar capacitada em **gestão de contratos**. Para isto, 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 freqüência, 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 isto, 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 freqüentes 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 conseqüências 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 conseqüências 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 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 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 manutenção e evolução dele. Por isto, 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 esta 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 reprodutível. Isto 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 impedimentos 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 eqüivaleria 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. Este 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 estas os conceitos de W. E. Deming, que também teve grande influência na filosofia japonesa de qualidade industrial. Estes conceitos foram adaptados para a área de software por Watts Humphrey [Humphrey90]. A primeira versão oficial do CMM foi divulgada no final dos anos 80, 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 caótico
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.

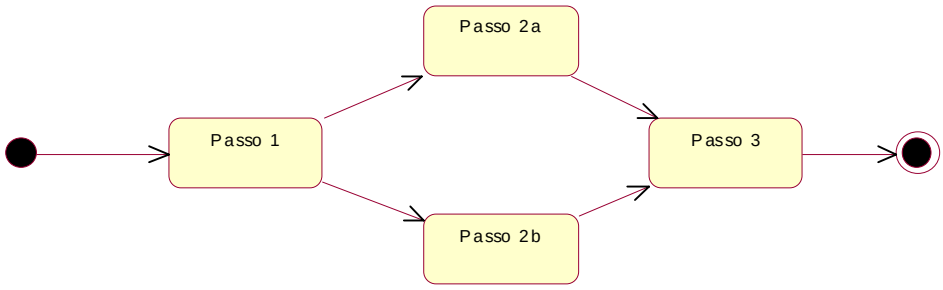

Figura 5 – Passos de um processo

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 uma 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 isto), os desenvolvedores começam imediatamente a codificar, remendando à medida em que os erros vão sendo descobertos. Nenhum processo definido é seguido. Infelizmente, é provavelmente o ciclo de vida mais usado. Para alguns desenvolvedores, este 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.

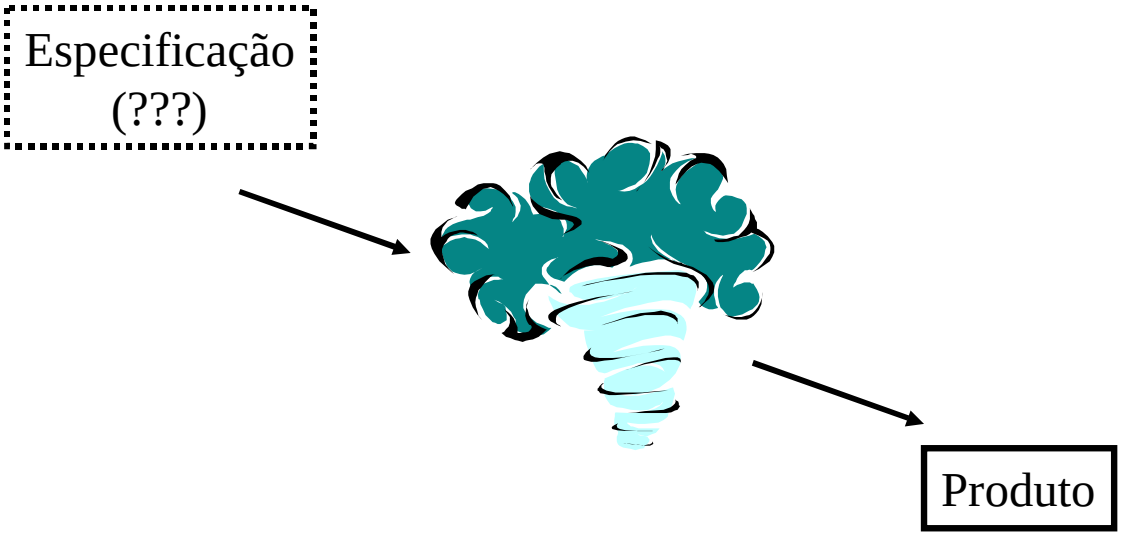

Figura 6 - O modelo de ciclo de vida Codifica e remenda

No modelo de ciclo de vida de **Cascata** (Figura 7), os principais subprocessos são executados em estrita seqüência, 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 este processo seja, em princípio, confiável e utilizável em projetos de qualquer escala. Por outro lado, se interpretado literalmente, é um processo rígido e burocrático, onde 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.

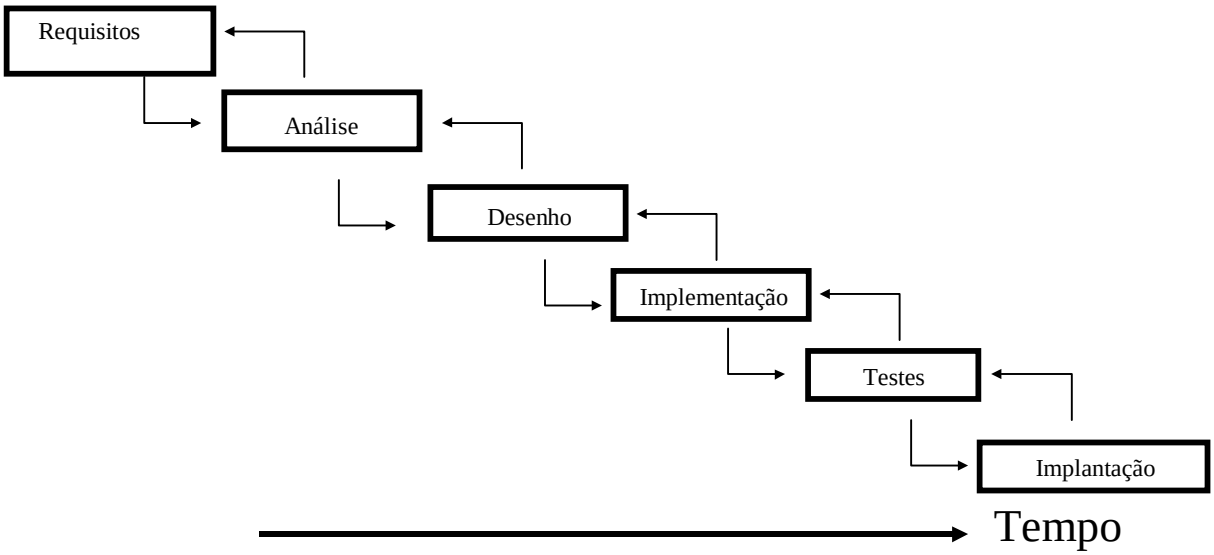

Figura 7 – O modelo de ciclo de vida em Cascata

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 neste modelo de ciclo de vida.

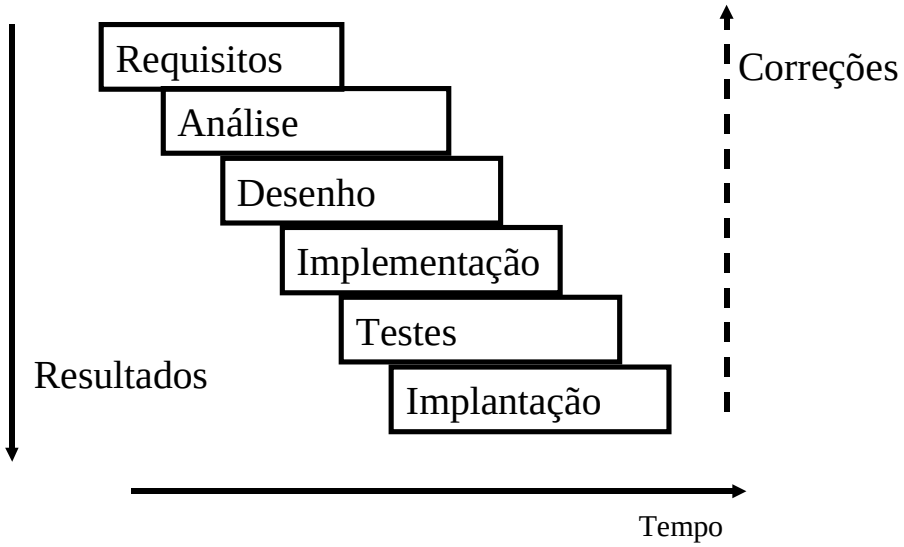

Figura 8 - O modelo de ciclo de vida em Sashimi

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. Isto 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 requer gestão muito sofisticada para ser previsível e confiável.

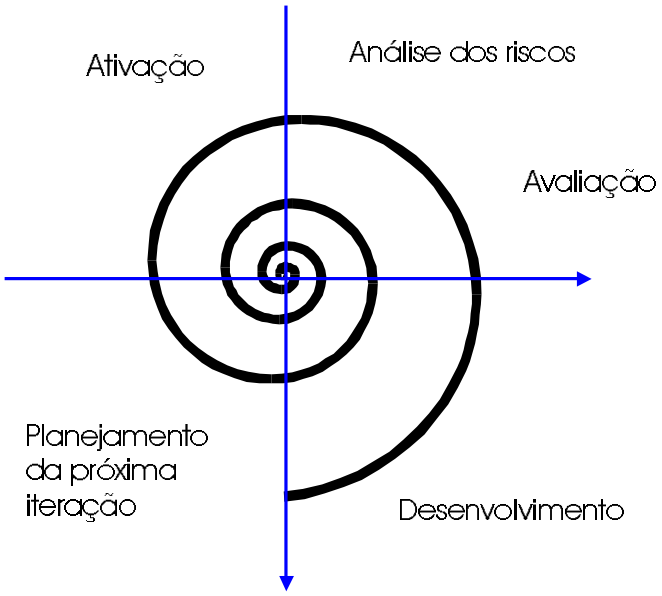

Figura 9 – O modelo de ciclo de vida em Espiral

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.

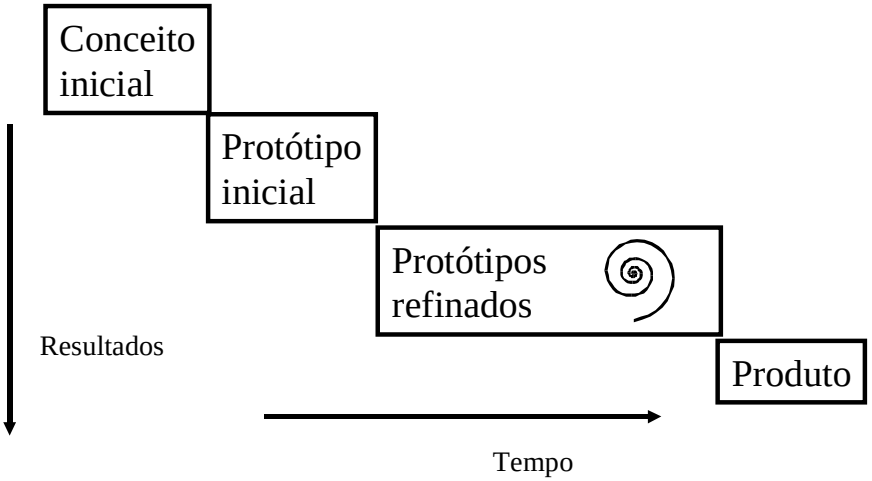

Figura 10 - O modelo de ciclo de vida de Prototipagem evolutiva

O modelo de **Entrega por estágios** (Figura 11) difere do modelo de cascata pela entrega ao cliente de liberações parciais do produto. Isto 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.

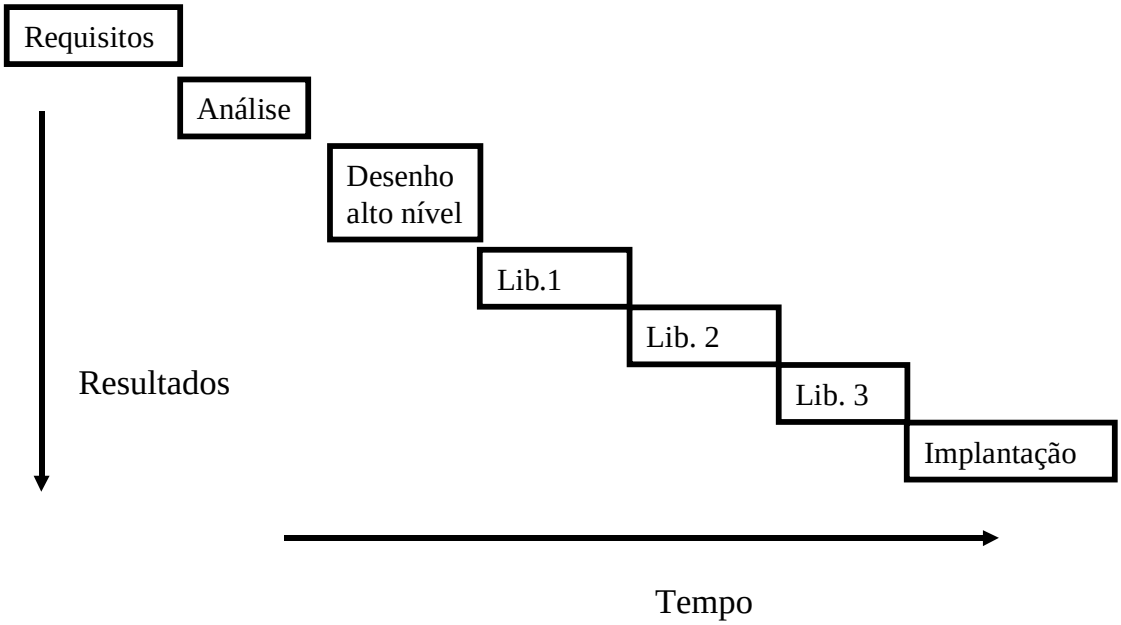

*Figura 11 - O modelo de ciclo de vida de **Entrega por estágios***

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 sistema, 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.

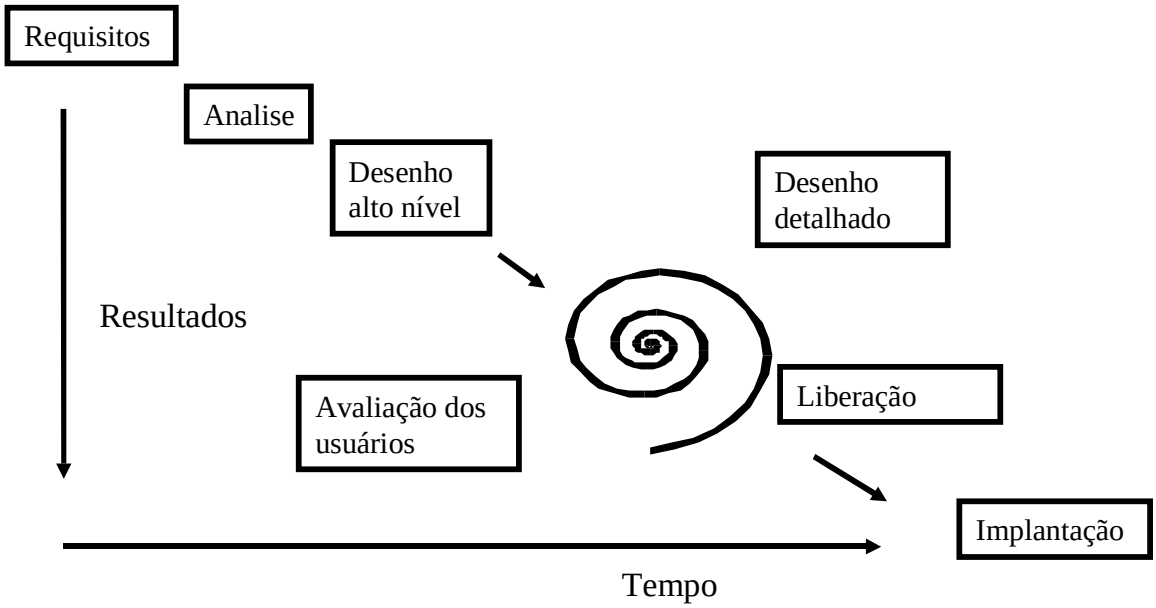

Figura 12 – O modelo de ciclo de vida de Entrega Evolutiva

Em modelos **dirigidos por prazo** ("time-boxed"), o produto é aquilo que se consegue fazer dentro de determinado prazo. Estes 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. Neste 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, este 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. Um 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 a 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. Este conjunto é chamado de Processo Pessoal de Software ("*Personal Software Process*"), ou **PSP**.

Estes processos são aprendidos através de uma seqüência de pequenos projetos, contida nesse 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 como 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 seqüência 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 nestas 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 pelos processo são os de gerente de desenvolvimento, de planejamento, de qualidade, de processo e de suporte, além do líder do time.

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

Tabela 6 - Fases do TSPe – parte 1

O planejamento e controle rigoroso de tamanhos, esforços, prazos e defeitos, característico do PSP, continua a ser feito. 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 Gerias, 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 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. Estes 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 deste 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]. Esta notação é utilizada também para descrição do próprio processo.

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 disto, 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 um 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, onde 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 que 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 PRocesso para Aplicativos eXtensíveis InterativoS, 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 isto, 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 semestre 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 destes 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.
Etapa	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 apresenta 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) quando 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).

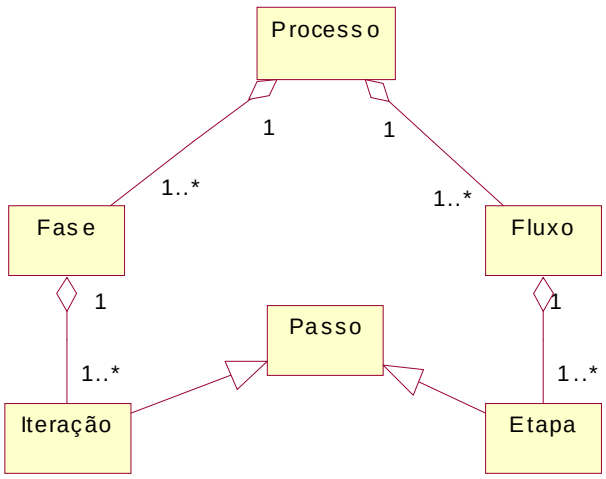

Figura 13 – Divisões de um processo

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.

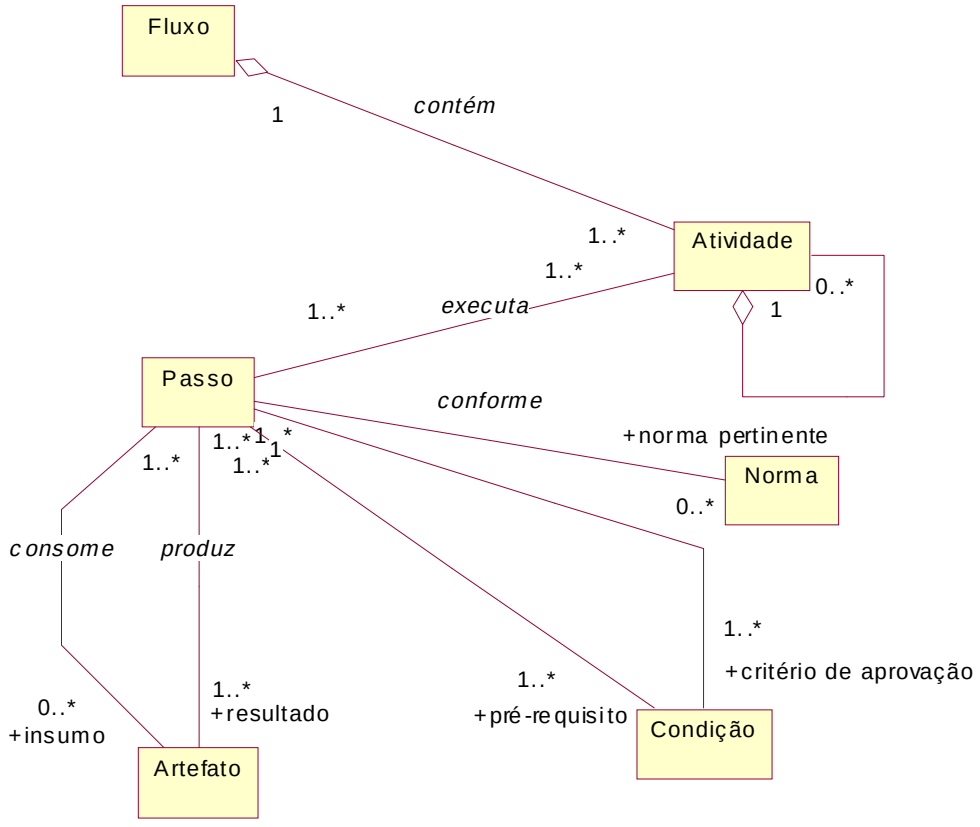

Figura 14 – Relacionamentos entre os elementos do script de um passo

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 obter um conjunto de requisitos de um produto, acordado entre cliente e fornecedor.
Análise	Fluxo que visa 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 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 detalhar e implementar o desenho através de componentes de código e de documentação associada.
Testes	Fluxo que visa 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", onde o foco é a captura dos requisitos junto aos usuários do produto, e de "Análise dos Requisitos", que focaliza o detalhamento e validação dos requisitos através de técnicas de análise.

A construção começa por uma iteração de "Desenho Inicial", onde 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 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 deste 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, a 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 usado para descrever processos de negócio. Os fluxos podem ser encarados como processos de negócio dos desenvolvedores de software.

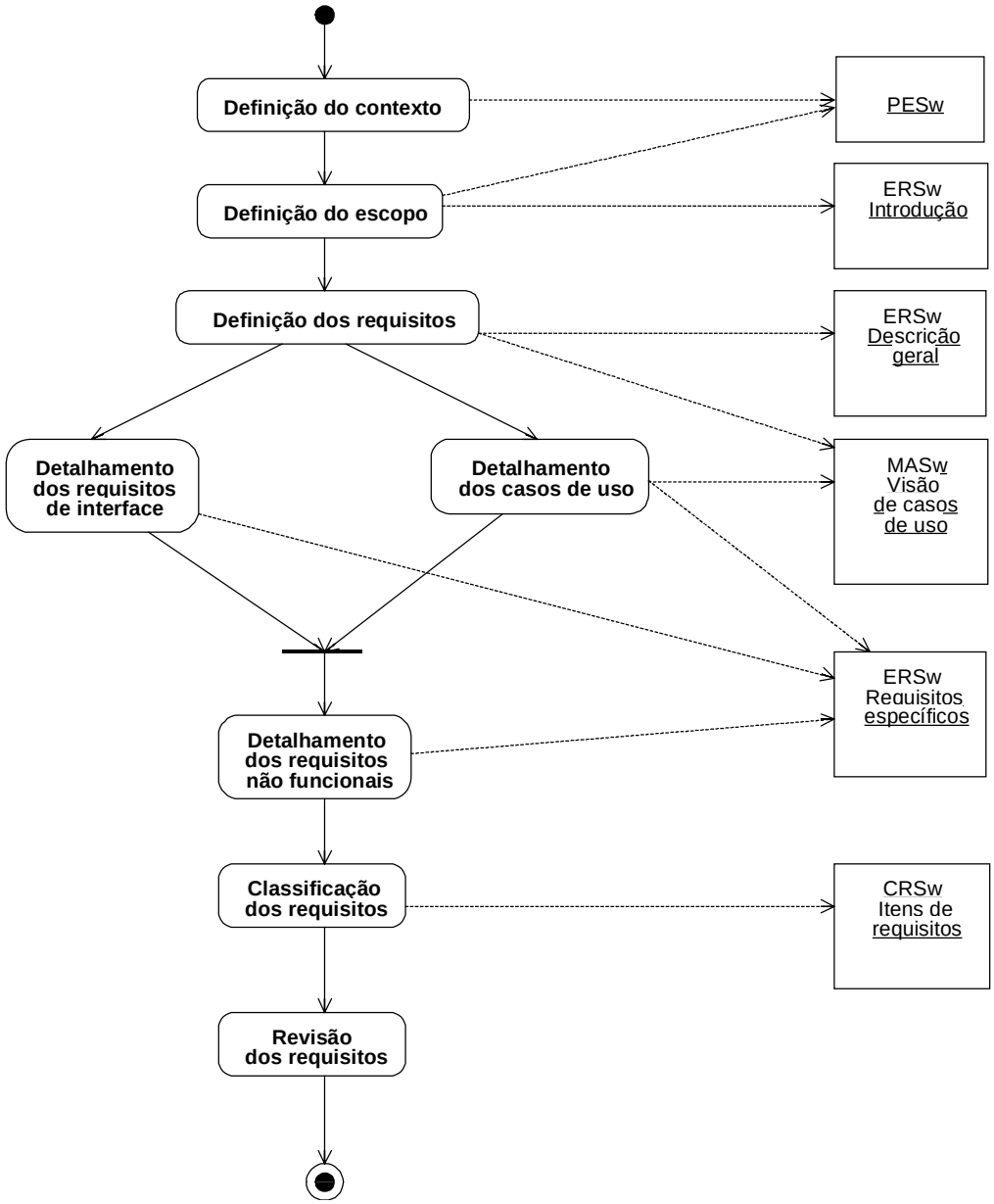

Figura 15 - Exemplo de detalhamento de fluxo

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 estes 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.

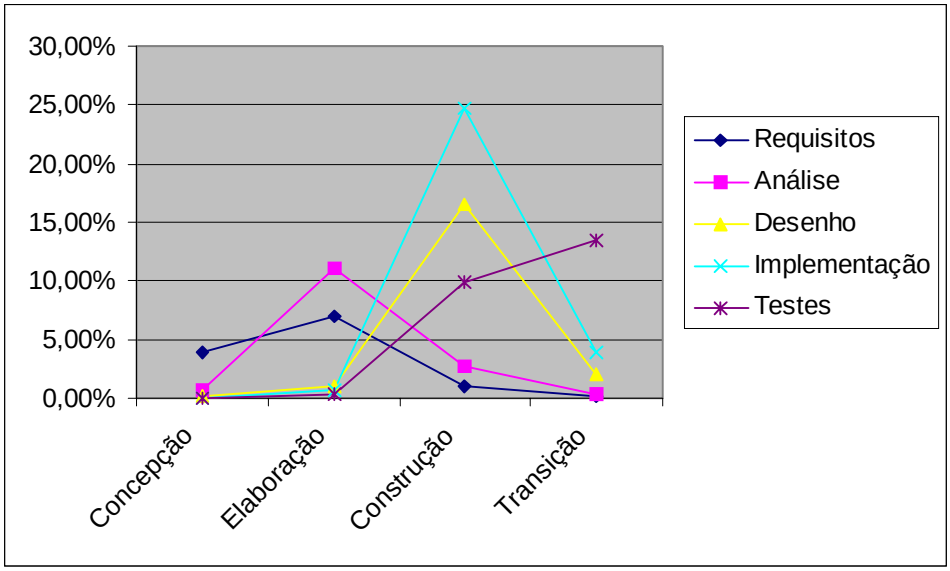

Figura 16 - Gráfico do esforço por fase e fluxo

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 consistente com as iterações que são detalhadas a seguir.

3.2 Detalhes da fases

3.2.1 Concepção

<table>
<tr><td>Descrição</td><td colspan="4">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.</td></tr>
<tr><td>Pré-requisitos</td><td colspan="4">Solicitação de proposta.</td></tr>
<tr><td>Insumos</td><td colspan="4">(só documentos externos ao projeto).</td></tr>
<tr><td rowspan="7">Atividades</td><td>Fluxo</td><td colspan="3">Tarefas</td></tr>
<tr><td>Requisitos</td><td colspan="3">Definição do escopo do produto.
Definição dos requisitos (preliminar).</td></tr>
<tr><td>Análise</td><td colspan="3">Estudos de viabilidade (opcional).</td></tr>
<tr><td>Desenho</td><td colspan="3">Esboço da arquitetura do produto (opcional).
Estudos de viabilidade (opcional).</td></tr>
<tr><td>Implementação</td><td colspan="3">Prototipagem dos requisitos (opcional).</td></tr>
<tr><td>Testes</td><td colspan="3">Testes dos protótipos dos requisitos (opcional).</td></tr>
<tr><td>Gestão</td><td colspan="3">Levantamento das metas gerenciais.
Estimativas da fase de Elaboração.
Elaboração de proposta de especificação.</td></tr>
<tr><td rowspan="2">Resultados</td><td colspan="2">Artefato</td><td>Sigla</td><td>Partes</td></tr>
<tr><td colspan="2">Proposta de Especificação de Software</td><td>PESw</td><td></td></tr>
<tr><td>Critérios de aprovação</td><td colspan="4">Aprovação em revisão gerencial.
Aprovação da Proposta de Especificação de Software pelo cliente.</td></tr>
<tr><td>Normas pertinentes</td><td colspan="4">Padrão de Proposta de Especificação de Software.</td></tr>
</table>

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 isto, deve durar o mínimo necessário para que se faça uma avaliação grosseira do escopo e 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 estes 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. Neste 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

<table>
<tr><td>Descrição</td><td colspan="3">Levantamento detalhado das funções, interfaces e requisitos não funcionais desejados para o produto.</td></tr>
<tr><td>Pré-requisitos</td><td colspan="3">Ativação terminada.</td></tr>
<tr><td>Insumos</td><td colspan="3">PESw.</td></tr>
<tr><td rowspan="8">Atividades</td><td>Fluxo</td><td colspan="2">Tarefas</td></tr>
<tr><td>Requisitos</td><td colspan="2">Levantamento completo dos requisitos.
Detalhamento das interfaces.
Detalhamento dos casos de uso.
Detalhamento dos requisitos não funcionais.</td></tr>
<tr><td>Análise</td><td colspan="2">Estudos de viabilidade (opcional).</td></tr>
<tr><td>Desenho</td><td colspan="2">Estudos de viabilidade (opcional).</td></tr>
<tr><td>Implementação</td><td colspan="2">Prototipagem dos requisitos (opcional).</td></tr>
<tr><td>Testes</td><td colspan="2">Testes dos protótipos dos requisitos (opcional).</td></tr>
<tr><td>Gestão</td><td colspan="2">Cadastramento dos requisitos.</td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td rowspan="4">Resultados</td><td>Artefato</td><td>Sigla</td><td>Partes adicionadas</td></tr>
<tr><td>Modelo de Análise do Software</td><td>MASw</td><td>Visão de casos de uso.</td></tr>
<tr><td>Especificação dos Requisitos do Software</td><td>ERSw</td><td>Corpo.</td></tr>
<tr><td>Cadastro de Requisitos do Software</td><td>CRSw</td><td>Interfaces, casos de uso e requisitos não funcionais.</td></tr>
<tr><td>Critérios de aprovação</td><td colspan="3">Aprovação em revisão gerencial.</td></tr>
<tr><td>Normas pertinentes</td><td colspan="3">Padrão para Especificação de Requisitos de Software.</td></tr>
</table>

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 a captura das necessidades dos usuários em relação ao produto, expressas na linguagem destes 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 a 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 esta 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. Esta 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. Esta parte da análise fornece o modelo lógico de dados (eqüivalente a um modelo de entidades e relacionamentos), que pode corresponder ao modelo conceitual de um banco de dados usado pelo produto. Estas 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 nesta iteração. A Especificação dos Requisitos agora torna-se 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. Esta proposta é refletida em um Plano de Desenvolvimento. Além disto, existe informação suficiente para planejar as atividades de um grupo de garantia da qualidade, expressas dentro de um Plano da Qualidade.

<table>
<tr><td>Descrição</td><td colspan="4">Modelagem conceitual dos elementos relevantes do domínio do problema e uso deste modelo para validação dos requisitos e planejamento detalhado da fase de Construção.</td></tr>
<tr><td>Pré-requisitos</td><td colspan="4">Levantamento dos requisitos terminado.</td></tr>
<tr><td rowspan="2">Insumos</td><td colspan="2">Antigos</td><td colspan="2">Novos</td></tr>
<tr><td colspan="2">PESw.</td><td colspan="2">ERSw - corpo; MASw - visão de casos de uso.</td></tr>
<tr><td rowspan="8">Atividades</td><td>Fluxo</td><td colspan="3">Tarefas</td></tr>
<tr><td>Requisitos</td><td colspan="3">Revisão e modificação dos requisitos (se necessário).</td></tr>
<tr><td>Análise</td><td colspan="3">Identificação das classe.
Identificação dos atributos e relacionamentos.
Realização dos casos de uso.
Revisão e iteração.</td></tr>
<tr><td>Desenho</td><td colspan="3">Estudos de viabilidade (opcional).
Desenho arquitetônico.</td></tr>
<tr><td>Implementação</td><td colspan="3">Prototipagem dos requisitos (opcional).</td></tr>
<tr><td>Testes</td><td colspan="3">Testes dos protótipos dos requisitos (opcional).</td></tr>
<tr><td>Gestão</td><td colspan="3">Cadastramento dos itens de análise no cadastro de requisitos.
Planejamento do desenvolvimento.
Planejamento da qualidade.</td></tr>
<tr><td></td><td colspan="4"></td></tr>
<tr><td rowspan="8">Resultados</td><td colspan="2">Artefato</td><td>Sigla</td><td>Partes adicionadas</td></tr>
<tr><td colspan="2">Modelo de Análise do Software</td><td>MASw</td><td>Visão lógica.</td></tr>
<tr><td colspan="2">Especificação dos Requisitos do Software</td><td>ERSw</td><td>Anexo – listagem do modelo de análise.</td></tr>
<tr><td colspan="2">Cadastro de Requisitos do Software</td><td>CRSw</td><td>Classes.</td></tr>
<tr><td colspan="2">Memória de Cálculo do Projeto do Software</td><td>MCPSw</td><td>Total.</td></tr>
<tr><td colspan="2">Modelo de Planejamento do Projeto do Software</td><td>MPPSw</td><td>Total.</td></tr>
<tr><td colspan="2">Plano de Desenvolvimento do Software</td><td>PDSw</td><td>Total.</td></tr>
<tr><td colspan="2">Plano da Qualidade do Software</td><td>PQSw</td><td>Total (menos detalhes de implementação).</td></tr>
<tr><td>Critérios de aprovação</td><td colspan="4">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.</td></tr>
<tr><td>Normas pertinentes</td><td colspan="4">Padrão para Especificação de Requisitos de Software.
Padrão para Plano de Desenvolvimento de Software.
Padrão para Plano da Qualidade de Software.</td></tr>
</table>

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 reputação dele 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 esta 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 ser assumidos com o cliente são factíveis, e onde se faz um balanço desta iteração.

Feitas estas 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. Esta 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 nestes 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 a estrutura arquitetônica principal do produto, dividindo-o em camadas, pacotes lógicos e subsistemas. Esta 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 atingir alguns dos mais importantes requisitos funcionais, como os requisitos de desempenho, confiabilidade, portabilidade e manutenibilidade.

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

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 destas 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, onde é feito o balanço da iteração.

<table>
<tr><td>Descrição</td><td colspan="4">Definição interna e externa dos componentes de um produto de software, a nível suficiente para decidir as principais questões de arquitetura e tecnologia, e para permitir o planejamento detalhado das atividades de implementação.</td></tr>
<tr><td>Pré-requisitos</td><td colspan="4">Elaboração terminada.</td></tr>
<tr><td rowspan="2">Insumos</td><td colspan="2">Antigos</td><td colspan="2">Novos</td></tr>
<tr><td colspan="2">MASw; ERSw; CRSw</td><td colspan="2">MCPSw; MPPSw; PDSw; PQSw.</td></tr>
<tr><td rowspan="8">Atividades</td><td>Fluxo</td><td colspan="3">Tarefas</td></tr>
<tr><td>Requisitos</td><td colspan="3">Revisão e modificação dos requisitos (se necessário).</td></tr>
<tr><td>Análise</td><td colspan="3">Revisão e modificação do modelo de análise (se necessário).</td></tr>
<tr><td>Desenho</td><td colspan="3">Desenho dos subsistemas.
Desenho do acesso a dados persistentes.
Desenho das interfaces de usuário.
Elaboração dos casos de uso de desenho.
Desenho das liberações.</td></tr>
<tr><td>Implementação</td><td colspan="3">Prototipagem de problemas de desenho (opcional).</td></tr>
<tr><td>Testes</td><td colspan="3">Planejamento e desenho dos testes de aceitação.</td></tr>
<tr><td>Gestão</td><td colspan="3">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.</td></tr>
<tr><td rowspan="5">Resultados</td><td colspan="2">Artefato</td><td>Sigla</td><td>Partes</td></tr>
<tr><td colspan="2">Insumos</td><td></td><td>Eventuais modificações e detalhes adicionais.</td></tr>
<tr><td colspan="2">Modelo de Desenho do Software</td><td>MDSw</td><td>Todas as visões, em descrição de alto nível.</td></tr>
<tr><td colspan="2">Descrição do Desenho do Software</td><td>DDSw</td><td>Partes 1 a 4, com colocação no anexo das partes já produzidas do MDSw</td></tr>
<tr><td colspan="2">Descrição dos Testes do Software</td><td>DTSw</td><td>Planos e especificações dos testes de aceitação.</td></tr>
<tr><td colspan="2">Bateria de Testes de Regressão do Software</td><td>BTRSw</td><td>Scripts para automação dos testes de aceitação (opcional).</td></tr>
<tr><td>Critérios de aprovação</td><td colspan="4">Aprovação em revisão técnica.
Aprovação do desenho das interfaces de usuário pelos usuários chaves.
Aprovação em auditoria da qualidade.
Aprovação em revisão gerencial.</td></tr>
<tr><td>Normas pertinentes</td><td colspan="4">Padrão para Descrição de Desenho de Software
Padrão para Documentação de Testes de Software</td></tr>
</table>

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 nesta 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.

<table>
<tr><td>Descrição</td><td colspan="4">Implementação de um subconjunto de funções do produto que será avaliado pelos usuários.</td></tr>
<tr><td>Pré-requisitos</td><td colspan="4">Desenho Inicial terminado.
Liberação anterior terminada e avaliada pelos usuários.</td></tr>
<tr><td rowspan="2">Insumos</td><td colspan="3">Antigos</td><td>Novos</td></tr>
<tr><td colspan="3">MASw; ERSw; CRSw; MCPSw; MPPSw; PDSw; PQSw; MDSw; DDSw; DTSw; BTRSw</td><td>CFSw (liberação anterior); CESw (liberação anterior); RTSw (liberação anterior).</td></tr>
<tr><td rowspan="6">Atividades</td><td>Fluxo</td><td colspan="3">Tarefas</td></tr>
<tr><td>Requisitos</td><td colspan="3">Revisão e modificação dos requisitos (se necessário).</td></tr>
<tr><td>Análise</td><td colspan="3">Revisão e modificação do modelo de análise (se necessário).</td></tr>
<tr><td>Desenho</td><td colspan="3">Revisão e modificação do modelo de desenho (se necessário).</td></tr>
<tr><td>Implementação</td><td colspan="3">Desenho detalhado dos componentes desta liberação.
Codificação dos componentes desta liberação.
Compilação dos componentes desta liberação.</td></tr>
<tr><td>Testes</td><td colspan="3">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.</td></tr>
<tr><td>Gestão</td><td colspan="3">Revisão e modificação dos planos de desenvolvimento e da qualidade (se necessário).</td></tr>
<tr><td rowspan="5">Resultados</td><td colspan="2">Artefato</td><td>Sigla</td><td>Partes</td></tr>
<tr><td colspan="2">Insumos</td><td></td><td>Eventuais modificações e detalhes adicionais.</td></tr>
<tr><td colspan="2">Relatórios dos Testes do Software</td><td>RTSw</td><td>Relatórios dos testes de unidade e de integração da liberação (opcional).</td></tr>
<tr><td colspan="2">Códigos Fontes do Software</td><td>CFSw</td><td>Unidades da liberação.
Estruturas provisórias de teste.</td></tr>
<tr><td colspan="2">Códigos Executáveis do Software</td><td>CESw</td><td>Unidades da liberação.
Estruturas provisórias de teste.</td></tr>
<tr><td>Critérios de aprovação</td><td colspan="4">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.</td></tr>
<tr><td>Normas pertinentes</td><td colspan="4">Padrão para Descrição de Desenho de Software
Padrão para Documentação de Testes de Software
Padrão para Desenho Detalhado e Codificação de Software.</td></tr>
</table>

Tabela 23 - Script da Liberação n

Os testes de integração são planejados e desenhados, geralmente aproveitando 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 isto seja feito por uma equipe especializada em testes, independente 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 desta cadeia.

Nesta 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 disto, 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.

<table>
<tr><td>Descrição</td><td colspan="4">Realização dos testes de aceitação, no ambiente dos desenvolvedores, juntamente com elaboração da documentação de usuário e possíveis planos de Transição.</td></tr>
<tr><td>Pré-requisitos</td><td colspan="4">Última liberação terminada e avaliada pelos usuários.</td></tr>
<tr><td rowspan="2">Insumos</td><td colspan="2">Antigos</td><td colspan="2">Novos</td></tr>
<tr><td colspan="2">MASw; ERSw; CRSw; MCPSw; MPPSw; PDSw; PQSw; MDSw; DDSw; DTSw; BTRSw.</td><td colspan="2">CFSw última (liberação); CESw (última liberação); RTSw (última liberação).</td></tr>
<tr><td rowspan="7">Atividades</td><td>Fluxo</td><td colspan="3">Tarefas</td></tr>
<tr><td>Requisitos</td><td colspan="3">Revisão e modificação dos requisitos (se necessário).</td></tr>
<tr><td>Análise</td><td colspan="3">Revisão e modificação do modelo de análise (se necessário).</td></tr>
<tr><td>Desenho</td><td colspan="3">Revisão e modificação do desenho de alto nível (se necessário).</td></tr>
<tr><td>Implementação</td><td colspan="3">Revisão e modificação do desenho detalhado e código (se necessário).
Produção da documentação de usuário.</td></tr>
<tr><td>Testes</td><td colspan="3">Realização dos testes alfa (aceitação no ambiente dos desenvolvedores).</td></tr>
<tr><td>Gestão</td><td colspan="3">Planejamento detalhado da Transição.
Atualização dos planos de desenvolvimento e da qualidade.</td></tr>
<tr><td rowspan="4">Resultados</td><td colspan="2">Artefato</td><td>Sigla</td><td>Partes</td></tr>
<tr><td colspan="2">Insumos</td><td></td><td>Eventuais modificações e detalhes adicionais.</td></tr>
<tr><td colspan="2">Relatórios dos Testes do Software</td><td>RTSw</td><td>Relatórios dos testes alfa.</td></tr>
<tr><td colspan="2">Manual do Usuário do Software</td><td>MUSw</td><td></td></tr>
<tr><td>Critérios de aprovação</td><td colspan="4">Aprovação em auditoria da qualidade.
Aprovação em revisão gerencial.
Aprovação da entrega do produto pelo cliente.</td></tr>
<tr><td>Normas pertinentes</td><td colspan="4">Padrão para Documentação de Testes de Software
Padrão para Documentação de Usuário de Software</td></tr>
</table>

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).

<table>
<tr><td>Descrição</td><td colspan="4">Realização dos testes de aceitação, no ambiente dos usuários.</td></tr>
<tr><td>Pré-requisitos</td><td colspan="4">Construção terminada.
Aceitação da instalação do produto pelo cliente.</td></tr>
<tr><td rowspan="2">Insumos</td><td colspan="2">Antigos</td><td colspan="2">Novos</td></tr>
<tr><td colspan="2">MASw; ERSw; CRSw; MCPSw; MPPSw; PDSw; PQSw; MDSw; DDSw; DTSw; BTRSw.</td><td colspan="2">RTSw (testes alfa); MUSw.</td></tr>
<tr><td rowspan="6">Atividades</td><td>Fluxo</td><td colspan="3">Tarefas</td></tr>
<tr><td>Requisitos</td><td colspan="3">Revisão e modificação dos requisitos (se necessário).</td></tr>
<tr><td>Análise</td><td colspan="3">Revisão e modificação do modelo de análise (se necessário).</td></tr>
<tr><td>Desenho</td><td colspan="3">Revisão e modificação do desenho de alto nível (se necessário).</td></tr>
<tr><td>Implementação</td><td colspan="3">Revisão e modificação do desenho detalhado e código (se necessário).
Revisão e modificação da documentação de usuário (se necessário).</td></tr>
<tr><td>Testes</td><td colspan="3">Realização dos testes beta (aceitação no ambiente dos usuários).</td></tr>
<tr><td>Gestão</td><td colspan="3">Revisão e modificação dos planos de desenvolvimento e da qualidade (se necessário).</td></tr>
<tr><td rowspan="3">Resultados</td><td colspan="2">Artefato</td><td>Sigla</td><td>Partes</td></tr>
<tr><td colspan="2">Insumos</td><td></td><td>Eventuais modificações e detalhes adicionais.</td></tr>
<tr><td colspan="2">Relatórios dos Testes do Software</td><td>RTSw</td><td>Relatórios dos testes beta.</td></tr>
<tr><td>Critérios de aprovação</td><td colspan="4">Aprovação em auditoria da qualidade.
Aprovação em revisão gerencial.
Aprovação dos testes beta pelo cliente.</td></tr>
<tr><td>Normas pertinentes</td><td colspan="4">Padrão para Documentação de Testes de Software</td></tr>
</table>

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, esta 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. Este 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	
Critérios 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 destes instantes. Isto 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 destes documentos é conforme com os padrões do IEEE [IEEE94]. Estes padrões requerem documentos bastante detalhados, mas o processo inclui modelos que facilitam o preenchimento destes. Além disto, em alguns modelos são preenchidas previamente as partes que não variam muito entre projetos conformes com este 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 mostra 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 nesta 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 destes 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 subseqüentes, 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 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 deste procedimentos. Outras verificações típicas destas auditorias são a conformidade com os procedimentos de gestão de configurações, a consistência 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. Nestas últimas avaliações, visa-se principalmente determinar se os requisitos foram corretamente interpretados pelos desenvolvedores.

Melhoria dos processos de software

1 Organizações

1.1 Maturidade das organizações

1.1.1 Sintomas da imaturidade

A produção industrial de software é quase sempre uma atividade coletiva. Alguns produtos são construídos inicialmente por indivíduos ou 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 manutenção e evolução dele. Por isto, 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 esta organização possui para produzir software de boa qualidade, dentro de prazos e custos razoáveis e previsíveis.

Infelizmente para os profissionais, muitas organizações que produzem software são imaturas. Isto ocorre tanto com organizações que produzem software como atividade fim, como com organizações para as quais o software é meio de apoio aos processos de negócio. Alguns sintomas identificam claramente as organizações imaturas.

- Os projetos não são definidos com clareza. Atividades de desenvolvimento de software são disfarçadas de manutenção, ou mesmo realizadas sem nenhum marco formal. Às vezes não se sabe ao certo quem é o responsável por um projeto, ou mesmo se uma atividade faz parte de algum projeto. Os clientes e usuários não sabem exatamente a quem se dirigir. Os gerentes têm dúvidas sobre o quê cobrar de quem.

- As pessoas não recebem o treinamento necessário. Ou não existe disponibilidade de tempo para treinamento, ou as pessoas se inscrevem no treinamento que bem entendem. Os treinamentos são avaliados apenas quanto à satisfação dos treinandos, se tanto. Não se avalia o treinamento quanto ao benefício trazido para os projetos, e não se comparam benefícios com custos. As pessoas trabalham em ambientes inadequados. Não existem planos claros de recrutamento, remuneração e avaliação do desempenho. Não existe boa comunicação entre as pessoas.

- As ferramentas não ajudam realmente a resolver os problemas. As pessoas não têm acesso a ferramentas compatíveis com o estado da arte, ou têm as ferramentas que bem entendem. Os usuários das ferramentas não recebem treinamento e orientação em grau satisfatório. Ferramentas são escolhidas de forma política, sem considerar avaliações técnicas e necessidades de padronização.

- Os procedimentos e padrões, quando existem, são definidos e seguidos de forma burocrática. Muitas vezes, o processo oficial, que existe nos documentos escritos, é rígido demais. Por outro lado, o processo real praticado é, com freqüência, muito diferente do processo oficial, e muito mais relaxado que este. Os gerentes são os primeiros a não levar os processos a sério.

1.1.2 Prejuízos da imaturidade

A organização imatura é ruim para os profissionais técnicos. Os mesmo problemas se repetem de projeto em projeto. Há muitas exigências dos clientes, usuários e gerentes, e poucos recursos. O trabalho é excessivo e estressante: corridas desesperadas contra os prazos são regra. A qualidade de vida no trabalho é ruim, o ambiente é desgastante, e os profissionais são desmotivados.

O leitor dos cartuns do Dilbert facilmente atribuirá aos gerentes a culpa do quadro acima. Isto pode acontecer ou não, mas a organização imatura também é ruim para os gerentes. Eles não têm uma visão realista do progresso dos projetos, e muitos problemas só são levados ao conhecimento deles quando já não têm remédio. Os planos são feitos pro forma: com o projeto em andamento, eles não são consultados, e muito menos atualizados. Os orçamentos e cronogramas estouram rotineiramente. Os gerentes perdem credibilidade diante dos clientes e dos gerentes superiores.

Para completar, a organização imatura é ruim também para os clientes e usuários. Os produtos são de baixa qualidade: apresentam muitos defeitos, e são difíceis de usar. Os clientes e usuários ficam insatisfeitos e reclamam: algumas vezes, por causa de defeitos reais, outras vezes porque não conseguem aprender o uso correto. O resultado final tem sido constatado em muitos inventários realizados por causa do Ano 2000. Tipicamente, a metade do estoque de software das organizações não é usada para nada. Então, para que se gastou tempo e dinheiro no desenvolvimento e manutenção destes produtos?

1.1.3 Compromissos

Prazos políticos raramente são realistas. Em muitas organizações públicas, prazos são determinados pelo Primeiro Método de Fixação de Prazos: "data marcada para cerimônia oficiada por alguma autoridade". Em muitas organizações comerciais, prazos são determinados pelo setor de vendas, através do Segundo Método de Fixação de Prazos: "o que os clientes querem ouvir". Às vezes usa-se o Terceiro Método de Fixação de Prazos: "o que a concorrência pode prometer".

Prazos realistas só podem ser determinados de forma técnica. Mas todas as técnicas de estimativa dependem de dados históricos da própria organização. E só quem trabalha com processos definidos consegue coletar e manter dados históricos. Não é possível aproveitar a experiência de planejamento de projetos anteriores, se os projetos não tiverem uma estrutura comum mínima. Sem prazos realistas, uma organização pode fazer promessas, mas não pode assumir compromissos de verdade.

Os compromissos firmes e realistas têm de ser baseados em especificações bem definidas. Os planos têm de ser feitos com base nas especificações. Os projetos têm de ser geridos de acordo com os planos. As organizações imaturas são tipicamente deficientes em todos estes quesitos.

Em resumo:

- muitas organizações imaturas em software têm o hábito de assumir compromissos não realistas;
- outras querem sinceramente obter maior realismo dos compromissos, mas desconhecem as técnicas de estimativa;
- outras conhecem as técnicas, mas não têm os dados para aplicá-las, porque não usam processos bem definidos e estáveis.

1.1.4 Forças caóticas

Na ausência de processos bem definidos, a gestão dos projetos de desenvolvimento de software torna-se confusa. A pressão dos prazos não deixa tempo para a reflexão, e, por falta de métodos padronizados, os desenvolvedores recorrem a táticas de força bruta. Os usuários e a alta gerência se impacientam com a descumprimento dos compromissos. Como em outras esferas da atividade humana, este estado de coisas leva as pessoas a depositar fé na magia, nos gurus e nos heróis.

O recurso à **magia**, no caso dos desenvolvedores de software, se traduz na crença na **balas de prata** [Brooks95]. As balas de prata míticas eram tidas como único meio eficaz de matar lobisomens. No desenvolvimento de software, as balas de prata são métodos e ferramentas tidos como milagrosos. Divulgadas por vendedores bem treinados, as balas de prata encontram ávida audiência em desenvolvedores e gerentes que preferem acreditar em curas milagrosas. Muitos preferem a promessa de um remédio fulminante do que a realidade da lenta melhoria dos processos, baseada na disciplina

pessoal e de grupo. A crença na mágica bloqueia o pensamento racional. Desenvolvedores e gerentes recorrem à pajelança: "fiz grande mágica, portanto cumprirei cronograma" [Weinberg93].

Os **gurus** são entidades comuns nas organizações imaturas. São pessoas consideradas infalíveis, por dominarem técnicas supostamente ininteligíveis para outros. Determinam ex cathedra os rumos dos projetos. Com o ego incensado, aceitam compromissos cada vez maiores, até não darem mais conta do recado. Quando o desastre é iminente, costumam ser espertos o suficiente para procurar outras plagas, e deixar a culpa para outros. Adoram codificar, e detestam analisar, desenhar, testar, rever e principalmente documentar. Manutenção, para eles, é sempre problema dos outros.

Os **heróis** são personagens mais benévolos. Este realmente têm consciência profissional, mas usam processos informais, que não são transferíveis. Até certo ponto, suprem com o esforço pessoal a falta de técnicas e de organização. Algumas vezes, realmente conseguem salvar alguns projetos. Adquirindo fama messiânica, tendem a ser sobrecarregados até seu limite de competência.

1.2 Escala

1.2.1 Problemas de escala

Problemas de escala são problemas relacionados com tamanho. Problemas de escala existem em todos os ramos da engenharia. Por isto, construir um grande edifício é muito mais difícil que construir um barraco, e realizar uma reação química em escala industrial é muito mais difícil que realizá-la em laboratório.

Para quantificar os problemas de escala de software é preciso medir o **tamanho** dos seus produtos. Definir tamanho é um problema técnico que será tratado posteriormente. Por enquanto, vamos utilizar como medida de tamanho o número de **linhas de código** dos programas fontes. Desde já, é preciso saber que existem várias maneiras diferentes de contar estas linhas; cada maneira leva a contagens diferentes. Como indicador de ordem de grandeza do tamanho, entretanto, o número de linhas de código pode ser considerado satisfatório.

Os problemas de escala são agravados por outros problemas comuns de Engenharia de Software.

- A falta de documentação adequada do código dificulta localizar os defeitos, principalmente quando os codificadores originais não estão mais disponíveis.
- Os programas têm vida muito mais longa que aquela que se imaginou inicialmente.
- A ausência de desenho torna os programas frágeis. Pequenos defeitos localizados podem ter conseqüências graves quanto ao funcionamento de um sistema.
- Mesmo pequenos sistemas têm enorme complexidade. Cada linha de código é como se fosse uma peça móvel de um mecanismo: pode abrigar um defeito fatal para o sistema.
- Além de seus próprios defeitos, um aplicativo pode falhar por causa de defeitos de dados, de ambiente de operação e até de hardware.
- É muito difícil convencer os leigos em informática da gravidade ou mesmo da existência de problemas de Engenharia de Software.

Aliás, é difícil convencer alguns leigos em informática de que eles são leigos em informática.

1.2.2 Problemas de comunicação

O tamanho das equipes que são necessárias para desenvolver um produto de software cresce mais do que proporcionalmente ao aumento de escala deles. O principal problema é o crescimento do esforço de comunicação, como demonstra uma das principais referências da literatura de Engenharia de

Software [Brooks95]. Quando uma equipe é completamente desorganizada, todos têm de se comunicar com todos, e o esforço de comunicação é proporcional ao quadrado do tamanho da equipe. Uma organização hierárquica, com subgrupos e subgerentes, diminui o número de canais de comunicação. Em contrapartida, a equipe se torna menos ágil, e é preciso deslocar mais pessoas de atividades técnicas para atividades gerenciais.

Além disto, as possibilidades de mal-entendidos e de perda de informação crescem muito com o tamanho da equipe. Para combater isto, é preciso fazer documentos mais detalhados, e usar ferramentas e métodos mais padronizados; em suma, tornar os processos mais formais. Embora o trabalho de Watts Humphrey com processos pessoais [Humphrey95] demonstre que processos bem definidos são benéficos mesmo a nível individual, os processos têm de ser tanto mais rigorosos quando maior a necessidade de coordenação de grandes equipes.

Processos rigorosos em projetos de grande porte necessitam, por sua vez, de pessoal próprio. Pessoas têm de ser dedicadas a tarefas como controle de qualidade e gestão de configurações. Outras pessoas têm de ser ocupadas com atividades de padronização: padrões têm de ser redigidos, revistos, distribuídos e atualizados. Os desenvolvedores precisam receber treinamento e assistência do uso de padrões. Possíveis defeitos dos padrões devem ser removidos. A Tabela 33, baseada em [Humphrey95], propõe uma escala logarítmica para o tamanho dos projetos de software.

Outros fatores relacionados com pessoas têm de ser equacionados nos grandes projetos. Por exemplo, o perfil de mão-de-obra varia muito ao longo do desenvolvimento do software; não só em quantidade, mas em tipo de competência requerido. Uma organização que tenha muitos projetos tem maiores possibilidades de balancear os perfis de mão-de-obra, mas isto já traz outro aumento de escala.

Escala	Tamanho em linhas de código	Como pode ser feito
muito pequena	até 100	um bom programador consegue fazer intuitivamente
pequena	até 1.000	um bom programador consegue fazer com processos informais
médio	até 10.000	um bom programador consegue fazer com processos definidos
grande	até 100.000	uma equipe consegue fazer com processos definidos
muito grande	até 1.000.000	uma organização consegue fazer com processos definidos
múltipla	mais de 1.000.000	pode necessitar de um conjunto de organizações cooperantes

Tabela 33 - Escala de projetos de software

1.2.3 Construção em estágios

Uma solução usual para o desenvolvimento de produtos complexos é construí-los em estágios. O produto é entregue aos usuários em sucessivas **versões**. Cada versão é o resultado final de um projeto. Os projetos de novas versões de produtos existentes são chamados de projetos de **evolução**. Tipicamente, novas versões são lançadas em intervalos de um a três anos.

Algumas organizações usam uma hierarquia de versões. Versões maiores indicam acréscimos significativos de funcionalidade ou mudanças significativas de aparência. Versões menores indicam correções e adaptações de pequeno porte. Este estilo de identificação de produtos é muito comum em software de prateleira.

Mesmo dentro de um projeto, pode-se ter vários estágios de construção. Ao resultado de cada estágio chamaremos de **liberação**. Uma liberação é apresentada a usuários escolhidos por sua autoridade em

relação aos requisitos do produto (os **usuários chaves**), e é avaliada por estes. Entretanto, uma liberação não é colocada em operação, mas retorna aos desenvolvedores para que estes aproveitem o feedback dos usuários chaves.

A construção em estágios apresenta uma série de vantagens:

- prazos mais reduzidos para fornecer um produto, ainda que parcialmente completo, ou pelo menos, permitir a avaliação deste;
- redução da incerteza nos requisitos, já que cada versão ataca um subconjunto de necessidades adicionais dos usuários, e a avaliação das liberações permite conferir o entendimento destas necessidades por parte dos desenvolvedores;
- possibilidade de atender às necessidades em ordem de prioridade;
- redução da incerteza no desenho, já que os desenvolvedores de cada versão e liberação podem se apoiar na experiência trazida pelos estágios anteriores.

Por outro lado, a construção em estágios também tem aspectos mais difíceis:

- a divisão em partes tem de ser cuidadosamente planejada, considerando-se aspectos técnicos e gerenciais;
- as partes que compõem um estágio têm de ser integradas com aquelas que são herdadas de estágios anteriores, o que complica o desenho do produto e o planejamento dos testes;
- estruturas provisórias de teste têm de ser construídas, para cada estágio que tem funcionalidade incompleta;
- as interdependências entre as partes que compõem os estágios têm de ser resolvidas;
- o desenvolvimento de um estágio às vezes tem de começar antes que o anterior esteja completamente aceito, não sendo raro encontrarem-se situações em que várias versões do mesmo produto coexistem em diferentes estágios do ciclo de vida.

A construção em estágios, portanto, está longe de ser panacéia. Ela exige mais disciplina dos desenvolvedores, processos mais rigorosos, e maior capacidade gerencial. Entretanto, muitos produtos de software tem longa vida. Por isto, a necessidade de desenhar e planejar para a evolução se faz presente na grande maioria dos projetos de software.

1.3 Riscos

Um grande volume de dados publicados aponta para os riscos que correm os projetos de software executados sem a utilização de processos adequados. Um levantamento publicado em 1994, a partir de uma base de dados de 4.000 projetos [Jones94], constatou a ocorrência freqüente dos seguintes problemas.

- 70% dos projetos de grandes aplicativos sofre de instabilidade dos requisitos. Os requisitos crescem tipicamente cerca de 1% ao mês, atingindo níveis de mais de 25% de inchaço ao final do projeto.
- Pelo menos 50% dos projetos são executados com níveis de produtividade abaixo do normal.
- Pelo menos 25% do software de prateleira e 50% dos produtos feitos por encomenda apresentam níveis de defeitos superiores ao razoável.

- Produtos feitos sob pressão de prazos podem quadruplicar o número de defeitos.
- Pelo menos 50% dos grandes projetos de software estouram seu orçamento e seu prazo.
- Dois terços dos projetos de software muito grandes são cancelados antes do final.
- Os usuários não ficam satisfeitos com 25% dos produtos comerciais para PCs, 30% dos produtos comerciais para "mainframes" e 40% dos produtos feitos por encomenda.
- Tipicamente, 50% do patrimônio de software das empresas não é usado.
- Atritos entre a área de tecnologia da informação e a alta gerência ocorrem em mais de 30% das organizações.
- Atritos com os clientes ocorrem, no desenvolvimento de aplicativos, em 50% dos contratos por administração e 65% dos contratos por empreitada.

1.4 Erros

1.4.1 Os erros clássicos

A organização imatura comete erros que podem ser chamados de **erros clássicos** [McConnell96]. Estes erros são sempre repetidos, apesar de terem soluções conhecidas e publicadas há bastante tempo. Nestas organizações, muitos gerentes são como se dizia de alguns antigos reis da dinastia Bourbon: "nunca esquecem, e nunca aprendem".

Alguns destes erros são relativos ao produto, decorrentes dos requisitos. Outros decorrem de enganos relativos aos fatores da produção: processos, pessoas e tecnologia. Estes fatores formam o segundo triângulo crítico da Engenharia de Software (Figura 17).

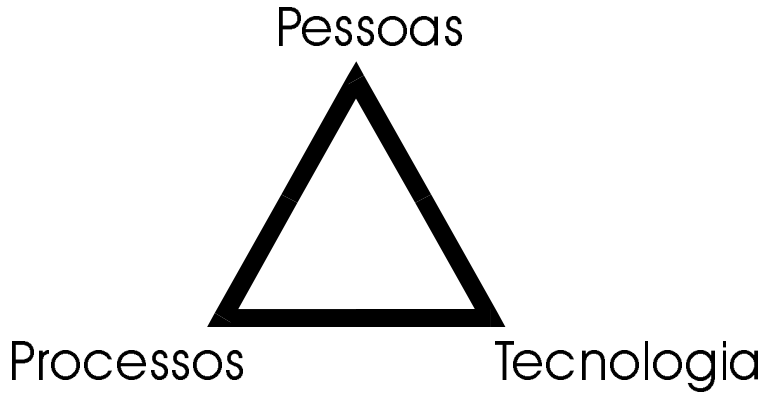

Figura 17 - O segundo triângulo crítico da Engenharia de Software.

1.4.2 Erros relativos ao produto

Os erros relativos ao produto decorrem de defeitos da definição deles. Em alguns casos a definição é deficiente desde a origem. Em outros casos, torna-se deficiente ao longo do projeto. Entre estes erros, destacam-se os seguintes.

- A introdução de características interessantes, mas dispensáveis. Estas características podem ser introduzidas nos requisitos, quando não é analisado o valor de cada um destes. Podem também ser introduzidas pelos desenvolvedores, durante o desenho ou a implementação, por falta de disciplina, agravada por desatenção do gerente de projeto.

- O inchaço causado por adições descontroladas de novos requisitos. Ocorre, comumente, quando todos os desejos dos usuários são imediatamente traduzidos em requisitos, em qualquer ponto do projeto, sem maior análise de impacto. A aceitação descontrolada de novos requisitos pode decorrer de pressões políticas a que o gerente do projeto não é capaz de resistir. Pode também revelar otimismo sem fundamento e vontade de agradar, da parte dos desenvolvedores. O caso pior (e muito freqüente) acontece quando os novos requisitos são impostos a projetos que já estão atrasados.

- O desenvolvimento orientado para a pesquisa, e não para a realização de um produto. Este tipo de erro é comum no ambiente acadêmico. Projetos orientados para a pesquisa têm de existir, mesmo em organizações industriais, para solucionar problemas que não podem ser tratados dentro de projetos normais de produção. Seus métodos, objetivos e filosofia, entretanto, são completamente diferentes destes últimos.

1.4.3 Erros relativos a processos

Os erros relativos a processos são comuns em organizações que utilizam processos informais. Ocorrem também naquelas que adotam processos oficiais rígidos e burocráticos, que não são realmente seguidos pelos desenvolvedores. Estes erros incluem os seguintes.

- Tempo desperdiçado antes do início do projeto. Cada dia de atraso no início de um projeto é um dia que muito dificilmente será recuperado depois. Este desperdício de tempo decorre de muitos problemas de gestão das organizações, como inércia para a tomada de decisões e ausência de planejamento estratégico. Como estes atrasos são geralmente de responsabilidade da alta gerência, dificilmente são questionados.

- Pressões causadas por prazos excessivamente otimistas. Como vimos acima, estas promessas geralmente resultam da combinação de pressões de clientes e alta gerência com otimismo não justificado por parte de desenvolvedores e gerentes de projeto. Mesmo quando os primeiros marcos de um projeto começam a atrasar, é comum a crença de que os atrasos vão ser compensados de alguma maneira. De que maneira, ninguém sabe explicar muito bem.

- Planejamento insuficiente dos projetos. Uma causa comum é a omissão, nas estimativas, de determinadas tarefas tidas como overhead, como garantia da qualidade, reuniões, treinamento etc. É comum também a ausência de análise dos riscos: o plano é todo baseado nas hipóteses mais otimistas.

- Falta de controles gerenciais adequados para acompanhar os projetos, e fazer cumprir o planejado. A lista dos riscos do projeto, se por acaso foi levantada durante o planejamento, não é mantida atualizada.

- Abandono dos planos, sob pressão para o cumprimento de prazos impossíveis. Um caso comum é o corte de atividades consideradas como "não essenciais". As vítimas prediletas são as atividades de análise, desenho, planejamento dos testes e garantia da qualidade. A conseqüência normal destes cortes é fazer mais coisas erradas, gastar mais tempo corrigindo os erros, e aumentar mais ainda os atrasos.

- Codificação desenfreada, baseada em desenho insuficiente ou inexistente. Pode ser conseqüência de pressão de prazos, ou simplesmente indisciplina de desenvolvedores sem maturidade técnica e sem controle.

- Falha de subcontratados, comum em casos de terceirização sem controle competente de compromissos.

- Entrega prematura do produto, também conhecida como Estratégia do Vietnã: "quando é impossível ganhar a guerra, declare vitória e bata em retirada".

1.4.4 Erros relativos a pessoas

Muitos gerentes de projeto nunca tiveram nenhuma formação relativa a desenvolvimento humano, recursos humanos ou gestão de pessoas. Por isto, são comuns erros que poderiam ser tratados com técnicas consagradas desta área. O assunto de gestão de pessoas está fora do escopo deste texto, mas resumimos a seguir alguns dos problemas mais comuns.

- Falta de patrocínio eficaz para o projeto. Segundo estudo recente [Keil+98], baseado em entrevistas com gerentes de projeto de vários países, a falta de comprometimento da alta gerência é considerada como o principal risco para os projetos de software. Um problema comumente associado é o predomínio da política sobre a substância. Os gerentes de projeto investem mais tempo em manobras políticas do que realmente cumprir sua missão de supervisão e controle.

- Falta de participação das partes interessadas em um produto, principalmente na engenharia de requisitos. Uma conseqüência freqüente é que informação vital deixa de ser repassada pelos usuários aos desenvolvedores. [Keil+98] lista entre os riscos mais sérios a ausência de comprometimento dos usuários, a falta de envolvimento adequado dos usuários e mal-entendidos a respeito dos requisitos.

- Atritos entre desenvolvedores e usuários ou clientes, causados por expectativas irreais destes e otimismo sem fundamento daqueles. [Keil+98] aponta também como um dos maiores riscos a falta de gestão das expectativas dos usuários e clientes. Ocorre às vezes que gerentes de projetos bem conduzidos não consigam repassar aos clientes e usuários a sensação de progresso rumos aos objetivos, por deficiência de comunicação.

- Defeitos de formação de staff do projeto. Estes incluem métodos inadequados de seleção de pessoal, a presença de funcionários problemáticos e a dependência em relação aos heróis. É famosa a Lei dos Projetos Atrasados de Brooks: "colocar mais gente em um projeto atrasado faz com que ele atrase mais ainda" [Brooks95]. Geralmente é preciso desviar os desenvolvedores atuais de seu trabalho para treinar e orientar os novatos, e o overhead de comunicação aumenta. Raramente o aumento nominal da força de trabalho compensa estas perdas.

- Escritórios apinhados e barulhentos. Vários estudos mostram que o hábito de muitas organizações atuais de colocar seu pessoal em baias apertadas ou grandes salões abertos contribui significativamente para diminuir a produtividade. O problema é particularmente sério, dado o grau de concentração que o trabalho de engenheiro de software exige. [Jones94] apresenta alguns dados a respeito; [Adams97] e demais livros de cartuns do Dilbert retratam o quotidiano dos habitantes das baias.

- Falta de motivação dos desenvolvedores, que geralmente surge em conseqüência dos outros problemas. A falta de motivação não é resolvida simplesmente pela promessa de recompensas monetárias, mas exige remédios mais profundos, como os discutidos em [Humphrey96], [McConnell96] e [Yourdon97].

1.4.5 Erros relativos à tecnologia

A engenharia de software trabalha com tecnologia de ponta. Muitas vezes, é exatamente o fascínio da alta tecnologia que atrai as pessoas que trabalham na área. Mas a fé cega nas soluções tecnológicas também é causa de alguns erros clássicos.

- Como foi discutido acima, a crença nas balas de prata fecha os olhos de muitos desenvolvedores e gerentes de projeto às limitações da tecnologia. É comum a estimativa exagerada dos ganhos de produtividade trazidos por determinados métodos e ferramentas. Este otimismo é fomentado por alguns vendedores, e facilmente transformado em crença fanática.

- Mudança de ferramentas no meio do projeto. Mesmo que as novas ferramentas sejam realmente boas, o tempo perdido com treinamento, adaptações, trabalho refeito e problemas a serem contornados em geral não compensa os possíveis ganhos. Mudanças de tecnologia devem ser cuidadosamente avaliadas, planejadas e introduzidas, uma de cada vez, em projetos piloto que não estejam sob pressão de prazos.

- Em poucos casos, a falta de automação de algumas atividades. O desenvolvimento de software é bastante intensivo em mão de obra, e poucas tarefas são suscetíveis de um grau mais avançado de automação. A gestão de configurações, entretanto, pelo volume de itens que deve tratar, não é viável sem uso de ferramentas automatizadas. A análise e o desenho baseados em modelos mais avançados, como os modelos orientados a objetos, necessitam do apoio de uma ferramenta de modelagem. A execução de determinados tipos de teste precisa ser automatizada, por causa do volume e da freqüência com que devem ser feitos.

2 A melhoria dos processos

2.1 Motivação para a melhoria

2.1.1 Prioridades de melhoria

A capacitação de uma organização também requer a aplicação de processos. Estes processos têm etapas, resultados intermediários e pontos de controle. A aplicação dos processos de capacitação é realizada por meio de projetos, que precisa de ter custos, prazos e responsáveis bem definidos.

Os projetos de capacitação levam algum tempo para mostrar os primeiros resultados significativos. Entretanto, é preciso mostrar resultados em prazo razoável, para justificar o investimento feito, e manter a motivação das pessoas envolvidas. É necessário, portanto, escolher áreas prioritárias para investimento. Onde o retorno do investimento em capacitação é mais rápido?

Dos três fatores da produção, a tecnologia é o mais sexy, para muitos profissionais. Existe um otimismo natural quanto aos resultados da aplicação de novas tecnologias. Levado ao exagero, este otimismo está na raiz da crença nas balas de prata. Entretanto, a tecnologia tem seu próprio ritmo de evolução. Tecnologias promissoras evoluem para um beco sem saída. Tecnologias consideradas inferiores pelos especialistas lançam raízes permanentes, graças a forças de mercado.

Além disto, a tecnologia só oferece retorno do investimento quando colocada nas mãos de pessoas capacitadas, trabalhando dentro de processos adequados. Toda introdução de nova tecnologia tem uma **curva de aprendizado**: as pessoas precisam ser treinadas, cometem inicialmente muitos erros, e por isto tornam-se menos produtivas durante algum tempo. Algumas tecnologias mais complexas só começam a pagar-se depois de vários projetos. Além disto, costuma ser contraproducente empregar tecnologias novas com os processos antigos.

Investir na capacitação das pessoas é absolutamente necessário. Entretanto, não é fácil introduzir e manter um programa consistente de capacitação de pessoas. Formar pessoas é difícil, caro e demorado. Recrutar pessoas capacitadas também: não há sinais de que a oferta de pessoas com alta qualificação em informática venha a se igualar à demanda, em futuro próximo. Além disto, muitas pessoas produzem menos que a capacidade delas permitiria. Isto pode acontece por falta de liderança, por deficiência de apoio, e por inadequação de processos.

Dos investimentos nos fatores de produção, as mudanças de processo podem trazer melhorias a prazo mais curto. Processos também não fazem milagres, mas a melhoria dos processos traz retorno em prazos relativamente curtos. Isto é ilustrado por alguns dados que mostraremos em seguida. Tal como a melhoria da tecnologia e das pessoas, a melhoria dos processos também requer seu próprio processo: deve ser feita em etapas bem definidas e controladas.

2.1.2 Processos de software

Chamaremos de **processo de software** ao conjunto de atividades, métodos, práticas e transformações, usado para desenvolver e manter produtos de software. Lembramos que o conceito de produto de software inclui os artefatos associados, como documentos e modelos. Os processos de software são processos de negócio das organizações desenvolvedoras e mantenedoras de software.

Em organizações com baixa maturidade de capacitação em software, os processos geralmente são informais. Processos informais existem apenas na cabeça de seus praticantes. Geralmente são processos individuais. Podem ser parcialmente transferidos para outras pessoas, por transmissão oral e por imitação.

Por outro lado, um **processo definido** tem documentação que detalha todos os seus aspectos importantes: o que é feito, quando, por quem, as coisas que usa e as coisas que produz. Existem muitas formas de representação de processos definidos. Exemplos de possíveis formas são:

- seqüências de passos descritos em linguagem natural;
- tabelas onde cada linha corresponde a um passo do processo;
- representações gráficas, como os fluxogramas e notações eqüivalentes.

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 reprodutível. Isto 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 impedimentos 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. 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;
- em prazo mais curto.

Ou seja, bons processos devem ajudar a produzir: **melhor; mais barato; mais rápido**.

2.1.3 Mal-entendidos comuns

Para acertar os rumos da melhoria dos processos, é preciso desfazer alguns mal-entendidos comuns.

- A qualidade do software não pode ser medida. Portanto, a apreciação da qualidade de um produto de software é subjetiva.
- Os principais problemas da produção de software são técnicos. Portanto, podem ser resolvidos através da aplicação de tecnologia.

- Os principais problemas da produção de software são causados por deficiências das pessoas. Portanto, podem ser resolvidos por uma combinação de terceirização, ameaças e punições.

- A engenharia de software é diferente das outras engenharias. Portanto, os métodos normais da engenharia não são aplicáveis.

- A gestão de projetos de software é diferente das outras formas de gestão. Portanto, os métodos normais da gestão de projetos não são aplicáveis.

A qualidade do software pode ser medida. Produtos complexos como o software, porém, não têm uma única medida de qualidade. Existem muitas medidas úteis: defeitos achados, produtividade etc. Escolher medidas adequadas, e ser capaz de coletar, normalizar e analisar estas medidas já requer um grau mais avançado de capacitação. Sem medidas, os processos não podem ser avaliados e muito menos melhorados. Mesmo medidas rudimentares podem ser úteis, entretanto. Por isto os programas de melhoria dos processos requerem, desde o início, algumas medidas mínimas para justificá-los.

Os principais problemas da produção de software não são técnicos. Com um mau processo, não adianta usar boa tecnologia. Os problemas gerenciais são mais básicos que os técnicos. Os projetos têm de ser conduzidos dentro de parâmetros mínimos de boa gestão, ou os investimentos em tecnologia não terão retorno. Muitas organizações que têm pessoas com alta capacitação técnica cometem erros gerenciais grosseiros.

Os principais problemas da produção de software não são causados por deficiências das pessoas. As pessoas geralmente erram por uma das seguintes razões:

- têm informação imprecisa, confusa ou incompleta;

- não têm os recursos necessários;

- têm métodos e procedimentos mal definidos;

- não foram treinadas adequadamente;

- não sabem seguir os procedimentos que têm.

Ao avaliar uma organização, muitos consultores descobrem que as pessoas que ali trabalham têm boa consciência dos problemas existentes de produção. Muitas vezes, conhecem inclusive alguma solução, e não conseguem ser ouvidas pela alta gerência. Freqüentemente, o papel do consultor consiste em levar estes pontos de vista aos níveis superiores de decisão.

A engenharia de software tem algumas diferenças em relação a outras engenharias. A indústria de software não é uma indústria de fabricação em série. Mesmo quando se trata de software de prateleira, o custo de reprodução, embalagem e distribuição de cada cópia é geralmente muito pequeno, comparado com o rateio do custo de desenvolvimento. O software não é baseado em princípios físicos estáveis, como a maioria das engenharias, e sim em abstrações lógicas. Talvez isto explique por que o software seja visto como uma espécie de magia negra por boa parte do público leigo.

A gestão dos projetos de software não é essencialmente diferente de outras formas de gestão. A engenharia de software requer mais disciplina gerencial, e não menos. Na maioria das engenharias, as leis físicas impõem limites claramente visíveis ao que pode ser feito. Na engenharia de software, a criatividade não é limitada por leis físicas, e sim pela capacidade humana de entender e dominar a complexidade. Para manter a complexidade sob controle, os gerentes devem exigir planos detalhados, e sistemas de acompanhamento destes planos, com pontos de controle bem definidos. Em outras palavras, os gerentes podem e devem exigir respostas claras para perguntas simples [Humphrey90].

2.1.4 Benefícios da melhoria dos processos

Os programas de melhoria de processos devem ser justificáveis através de análises de **retorno do investimento**. Estas análises procuram medir, para cada unidade monetária investida, quantas unidades monetárias retornam em determinado prazo, através da redução de custos ou do aumento da renda. Existem muitas práticas de Engenharia de Software; os programas de melhoria devem dar prioridade às práticas com melhor retorno de investimento.

Por exemplo, os dados seguintes. sustentam algumas das práticas mais prioritárias para melhoria dos processos ([Jones94], [McConnell96]).

- Captar um requisito correto é 50 a 200 vezes mais barato que corrigi-lo durante a implementação ou em operação. Portanto, a engenharia e a gestão dos requisitos estão entre as práticas de maior retorno de investimento.

- Fazer um desenho correto é 10 vezes mais barato que corrigi-lo durante os testes de aceitação. Portanto, o desenho tem forte impacto nos custos dos projetos, embora menos que a engenharia de requisitos.

- Refazer defeitos de requisitos, desenho e código consome 40% a 50% do custo total dos projetos. Portanto, a garantia da qualidade se paga rapidamente, na medida em que diminui a necessidade de refazer.

- Cada hora gasta em prevenção de defeitos representa de 3 a 10 horas menos de correção de defeitos. Dentre as atividades de qualidade, as atividades ligadas à prevenção de defeitos são mais eficazes que aquelas que focalizam a correção.

Em [Jones98] apresenta-se uma tabela comparativa do retorno de investimento para os principais métodos e técnicas da Engenharia de Software. Alguns dos dados relativos a práticas tratadas neste texto são resumidos na Tabela 34. O retorno de investimento é apresentado como percentagem do investimento inicial, em um prazo médio de quatro anos.

Prática	Retorno do investimento
Reutilização de artefatos de alta qualidade	4.375
Inspeções de código	1.600
Inspeções de desenho	1.550
Ferramentas de gestão de projetos	1.300
Gestão de configurações (total)	1.200
JAD	1.200
Programação orientada a objetos	1.100
Estruturação do código	1.000
Ferramentas de gestão dos requisitos	950
Métricas de pontos de função	900
Reutilização de código de alta qualidade	700
Testes de regressão	600
Reutilização do desenho	600
Revisões informais de código	550
UML	550
Revisões informais de desenho	500
Testes de unidade	500
Testes de aceitação	500
Reutilização de requisitos	500
Reutilização de planos de projeto	500
Reutilização de casos de teste	500
Desenho orientado a objetos	450
Reutilização de documentação de usuário	375
Gestão de configurações (código)	350
Reutilização de interfaces	350
Prototipagem informal	300
Padrões ISO e IEEE	300
Métricas de contagem de linhas lógicas	275
Métricas orientadas a objetos	170

Tabela 34 - Retorno de investimento em práticas de Engenharia de Software

2.2 Realização da melhoria

2.2.1 Condições para a melhoria de processos

Um programa de melhoria de processos requer muitos cuidados. Em organizações onde este tipo de programa nunca foi realizado existem muitas oportunidades de fracasso. Um programa fracassado de melhoria de processos pode ser pior do que nenhum programa, porque desmoraliza o próprio conceito. Para evitar o fracasso, algumas precondições importantes devem ser observadas.

- O programa deve ter forte apoio da alta direção da organização.
- Os gerentes dos projetos devem ter participação ativa no programa.
- Todos os que usarão os processos devem ser envolvidos.

- Os processos atuais devem ser conhecidos.
- Investimentos significativos devem ser feitos.
- O programa deve ter estágios intermediários bem definidos, onde são localizados os pontos de controle.

Grandes mudanças de processo têm de começar do topo. Toda mudança requer liderança baseada em uma visão estratégica. Esta visão deve ser encampada por patrocinadores com suficiente força dentro da organização. Eles podem sustentar custos que são certos e de curto prazo, a troco de benefícios incertos de médio e longo prazo. Eles podem ter uma visão de sobrevivência da organização, mais abrangente que os resultados deste semestre ou deste ano. E eles têm um escopo de interesse mais amplo que o de cada projeto.

Problemas de processo são de responsabilidade dos gerentes dos projetos. Só através da melhoria de processos é possível evitar a repetição de erros e incorporar as lições dos projetos passados. O gerente de projeto que não tem como prioridade a melhoria dos processos está condenado a apagar incêndios eternamente. É papel do gerente de projeto propor a sua equipe metas desafiadoras, em termos de custos, prazos e qualidade dos resultados. Por outro lado, estas metas devem ser claramente viáveis, sob pena de não serem realmente levadas a sério. Uma vez propostas as metas de processo, elas devem ser cobradas, insistindo-se em criar uma cultura de excelência.

Naturalmente, erros e problemas acontecerão. O gerente de projeto deve então procurar as deficiências de processo que causaram os problemas. Gerentes que dão prioridade à caça de culpados estão, na realidade, convidando seus subordinados a esconder problemas. Gerentes que são os primeiros a quebrar as regras estão deixando bem claro que estas regras não são realmente para valer.

Todos os que serão afetados pelas mudanças de processos têm de ser envolvidos. O custo deste envolvimento é sempre baixo, comparado com os prejuízos que podem ser causados por quem teme a mudança, não a leva a sério, ou não a entende. Processos imaturos são baseados na improvisação individual. Eles fomentam nas pessoas mal informadas sensações de liberdade, criatividade e até poder, que na realidade são ilusórias, porque pagas com riscos, incerteza e desperdício. Processos maduros permitem a ação mais estruturada, eficiente e coletiva. Reduzindo a incerteza e o estresse, eles contribuem para melhor qualidade de vida no trabalho.

A mudança eficaz requer conhecimento dos processos atuais. Não adianta ter um mapa quando não sabemos onde estamos. Mesmo que os processos atuais sejam informais, é preciso saber quais os pontos fortes e fracos deles. Sem o entendimento destes, é difícil estabelecer prioridades de melhoria. É preciso diagnosticar, para poder receitar. Além disto, como já foi citado, muitas vezes existem, dentro da própria organização, pessoas com boas idéias para resolver os problemas atuais.

Melhorias de processos requerem investimentos significativos. É preciso ter pelo menos um pequeno grupo de pessoas cujo foco seja a melhoria dos processos, e não problemas específicos de cada projeto. Treinamento é fundamental: a melhoria dos processos envolve a absorção de conceitos que não são triviais, e que vão até contra a intuição de algumas pessoas. O programa de treinamento tem de ter planejamento, recursos, cronograma e obrigatoriedade. Algumas ferramentas têm de ser adquiridas e implantadas.

O amadurecimento dos processos se faz passo a passo. Para ser viável, a melhoria deve ser feita em etapas planejadas. O processo de melhoria dos processos de software deve também ter pontos de controle onde a alta direção da organização possa decidir sobre seus rumos. Estes pontos podem corresponder a estágios intermediários de capacitação. A experiência da indústria de software deu origem a vários modelos de capacitação, que propõem áreas prioritárias para investimento em melhoria de processos.

2.2.2 Estabilização dos processos

Durante cada projeto, a estabilidade do processo é necessária. É normal, entretanto, descobrir defeitos e inadequações dos processos, à medida em que eles vão sendo implantados. Como qualquer produto, os processos apresentam mais defeitos quando começam a ser usados. Cada problema de processo é uma oportunidade de melhoria. De projeto para projeto, é natural que a acumulação de experiência leve a modificar os processos para melhor.

Por outro lado, mudanças de processos não devem ser feitas sob pressão, de prazos ou de outra natureza. No curso normal dos projetos, surgem emergências e situações imprevistas. Faz parte do trabalho dos gerentes o tratamento correto destas situações que exigem respostas de curtíssimo prazo. Este tratamento não deve prejudicar os interesses de médio longo prazo da organização, que são afetados pelos programas de melhoria dos processos. Para servir estes interesses, um programa de melhoria de processos deve incluir práticas que estabilizem os ganhos conseguidos.

Mudanças de processos não são sustentáveis por si só. A adoção de novos métodos passa sempre por estágios: implantação, prática, proficiência e finalmente naturalidade. Mesmo as ações dos melhores profissionais têm de ser acompanhadas, revisadas e checadas. Estes procedimentos de controle são normais e necessários, e por causa disto a introdução de práticas de garantia da qualidade é geralmente uma das primeiras ações de um programa de melhoria de processos. Quando conseguem ser sustentados por maior prazo, os processos se tornam parte da cultura da organização. Atinge-se, então, o estágio da naturalidade, em que as práticas deles se tornam parte da rotina profissional.

Para garantir a estabilidade dos novos processos, os programas de melhoria introduzem práticas de institucionalização. Estas práticas meios visam criar infra-estrutura e cultura que suportam as práticas fins dos novos processos. Ancorando-as na cultura da organização, elas tornam a execução dos processos independente de pessoas específicas. As seguintes práticas contribuem para a estabilidade da mudança de processos.

- Formação de grupos cuja atividade fim é a melhoria dos processos.
- Formação de estruturas organizacionais adequadas e estáveis.
- Desenvolvimento de recursos de apoio a processos, como padrões, modelos, exemplos, documentação de referência e bases de dados.
- Realização de programas de treinamento, orientação, aconselhamento e comunicação.
- Estabelecimento de medições do grau de progresso em relação à melhoria de processos.
- Estabelecimento de mecanismos de controle e verificação.

2.2.3 Os personagens da mudança

A melhoria de processos é iniciada, promovida, realizada e controlada por pessoas. Em programas bem sucedidos, as pessoas envolvidas desempenham diversos papéis, de acordo com sua função, experiência e personalidade. Geralmente, é necessário ter participantes do programa que assumam os seguintes papéis.

- **Campeões** - os que provocam o início da mudança, defendendo a causa à maneira dos paladinos medievais.
- **Patrocinadores** - autoridades que reconhecem o valor da mudança e dão apoio oficial, protegendo os programas de melhoria contra as pressões do dia a dia.
- **Agentes** - os que planejam e implementam a mudança, inclusive os grupos de melhoria dos processos e os gerentes dos projetos.

2.2.4 Um modelo de programas de melhoria

Alguns dos conceitos mais estabelecidos sobre a melhoria de processos de software originaram-se no SEI (Software Engineering Institute), ligado à Carnegie-Mellon University e patrocinado pelo Departamento de Defesa americano. A contribuição mais conhecida deste instituto é o paradigma CMM (Capability Maturity Model), que será discutido em detalhe no próximo capítulo.

O SEI desenvolveu um modelo de programas de melhoria de processos, chamado de IDEAL [McFeeley96], que pode ser usado como mecanismo de implementação do CMM e de outros paradigmas de capacitação em processos. Pode ser usado também para implementação de programas de melhoria de tecnologia e de capacitação de pessoas.

A Tabela 35 apresenta detalhes do modelo IDEAL. O modelo propõe cinco fases maiores para a realização de um ciclo de melhoria. Ao fim de cada ciclo, atinge-se um estágio de capacitação. Cada ciclo repete os passos do modelo. Possivelmente, a motivação e o patrocínio estarão bem estabelecidos a partir do segundo ciclo. As cinco fases maiores têm os objetivos seguintes.

- **Início** - lançar as bases para um programa bem sucedido.
- **Diagnóstico** - determinar onde se está e onde se quer chegar.
- **Estabelecimento** - planejar os detalhes de como chegar ao destino.
- **Ação** - realizar os planos.
- **Lições** - aprender com a experiência.

Fase		Atividades	
Nome adotado neste texto	**Nome em inglês**	**Nome**	**Descrição**
Início	Initiating	Motivação	Motivar a organização para mudar os processos.
		Patrocínio	Obter patrocinadores para a mudança.
		Infra-estrutura	Criar a infra-estrutura mínima necessária, principalmente um Grupo de Melhoria de Processos.
Diagnóstico	Diagnosing	Aferição	Caracterizar os estados atual e desejado dos processos.
		Recomendações	Desenvolver recomendações para resolver os problemas pela aferição.
Estabelecimento	Establishing	Priorização	Estabelecer prioridades para os esforços de mudança, considerando a realidade da organização.
		Abordagem	Definir a estratégia de mudança.
		Planejamento	Elaborar plano detalhado de realização, considerando prazos, custos, riscos, pontos de controle, responsabilidades e outros elementos.
Ação	Acting	Criação da solução	Reunir os elementos da solução, como padrões, modelos, ferramentos e treinamento.
		Testes pilotos	Testar a solução em um ou mais projetos pilotos.
		Refinamento	Aperfeiçoar a solução, para refletir o que for aprendido com os testes pilotos.
		Implantação completa	Transferir a solução para o restante dos projetos da organização.
Lições	Learning	Análise e validação	Coletar, analisar, checar e documentar as lições aprendidas.
		Proposição de ações futuras	Desenvolver recomendações para o próximo ciclo de melhoria de processos.

Tabela 35 - O modelo IDEAL de programas de melhoria

Página em branco

Capacitação em processos de software

1 O modelo CMM

1.1 Bases

1.1.1 Modelos de capacitação

Um programa de melhoria de processos não pode ser desenhado de forma intuitiva. Ele deve refletir o acervo de experiência dos profissionais e organizações da área. 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. Uma organização pode ser aferida ou avaliada, comparando-se suas práticas reais com aquelas que o modelo de capacitação prescreve ou recomenda. Esta aferição produz um diagnóstico da organização quanto aos seus processos. O diagnóstico serve de base para recomendações de melhoria de processos, e estas recomendações podem ser consolidadas em um plano de melhoria, como sugere o modelo IDEAL.

Esta primeira subseção trata do paradigma mais difundido atualmente de capacitação de software, que é o modelo CMM. As subseções seguintes tratam do sistema de comprometimento que serve de base para o CMM, e de um exemplo detalhado de uma área chave (disciplina específica) do CMM

1.1.2 O que é o CMM

Um modelo de capacitação particularmente importante para a área de software é o **CMM** (Capability Maturity Model), do Software Engineering Institute, citado anteriormente a respeito do modelo IDEAL. O CMM é patrocinado pelo Departamento de Defesa americano, que o utiliza para avaliação da capacidade de seus fornecedores de software. O CMM 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 estas os conceitos de W. E. Deming, que também teve grande influência na filosofia japonesa de qualidade industrial. Estes conceitos foram adaptados para a área de software por Watts Humphrey [Humphrey90]. A primeira versão oficial do CMM foi divulgada no final dos anos 80, e é descrita em relatórios técnicos do SEI ([Paulk+93], [Paulk+93a]) e um livro ([Paulk+95]).

O CMM é um modelo de capacitação específico para a área de software. Estão fora de seu escopo outras áreas importantes para a sobrevivência de uma organização produtora de software, como marketing, finanças, administração. Mesmo áreas importantes da informática, como hardware e bancos de dados, estão fora do escopo do CMM. Portanto, a aplicação do CMM, ou paradigma eqüivalente, não garante por si só a viabilidade de um organização, embora possa ser um fator importante de melhoria da eficácia e competitividade.

Além disto o CMM focaliza os processos, fator 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.

O CMM em vigor na época em que este texto está sendo escrito tem o número de versão 1.1. Já existe uma proposta de nova versão, chamada de CMMI, que integra os aspectos de processos de software, de engenharia de sistemas e de definição de produtos. O CMMI proposto é bastante mais complexo que o CMM versão 1.1, e vários anos serão necessários para que ele atinja o grau de aceitação que este último tem na indústria mundial de software. Por isto, utilizaremos o SW-CMM 1.1 como referência principal deste texto.

1.1.3 Níveis de maturidade

O CMM é um exemplo de modelo de capacitação de **arquitetura em estágios**. Isto significa que as práticas que ele descreve ou recomenda são agrupadas em níveis de maturidade. Estes níveis são escolhidos de acordo com os seguintes critérios:

- representar fases históricas razoáveis, na vida de organizações típicas;
- prover degraus intermediários de maturidade, em seqüência razoável;
- sugerir medidas de progresso e objetivos intermediários;
- definir, para cada estágio, prioridades de melhoria.

A arquitetura em estágios reconhece a dificuldade de que uma organização melhore todos os seus processo simultaneamente. Oferece, por isto, uma seqüência de estados intermediários que possam ser atingidos em tempo relativamente curto. Além disto, é possível resumir o resultado de uma avaliação ou aferição em um único número, representativo do estágio de maturidade alcançado. Embora seja uma medida extremamente simplista, este número é útil para comparar organizações produtoras de software.

A Tabela 3 resume os níveis do CMM, destacando as características mais marcantes de cada nível. As subseções seguintes apresentarão uma descrição mais detalhada.

Número do nível	Nome do nível	Característica da organização	Característica dos processos
Nível 1	Inicial	Não segue rotinas	Processos caótico
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 36 - Níveis do CMM

A Figura 18 mostra as percentagens encontradas em 1989, em um levantamento realizado pelo SEI entre organizações americanas de software. Naquela época, não foram encontradas organizações situados nos níveis 4 e 5. Um levantamento publicado em março de 1999 é mostrado na Figura 19. Este levantamento abrange cerca de 800 organizações produtores de software, compreendendo aquelas que, nos cinco anos anteriores, foram avaliadas pelo SEI ou por aferidores credenciados. Vê-se, neste levantamento, que cerca de metade das organizações avaliadas já apresenta nível 2 ou superior. Existe já um número significativo de organizações nível 4; as organizações nível 5 formam ainda uma elite reduzida. Note-se que uma organização pode ser um setor de uma empresa; grandes empresas geralmente não são avaliadas como um todo, mas sim a nível de laboratório ou divisão.

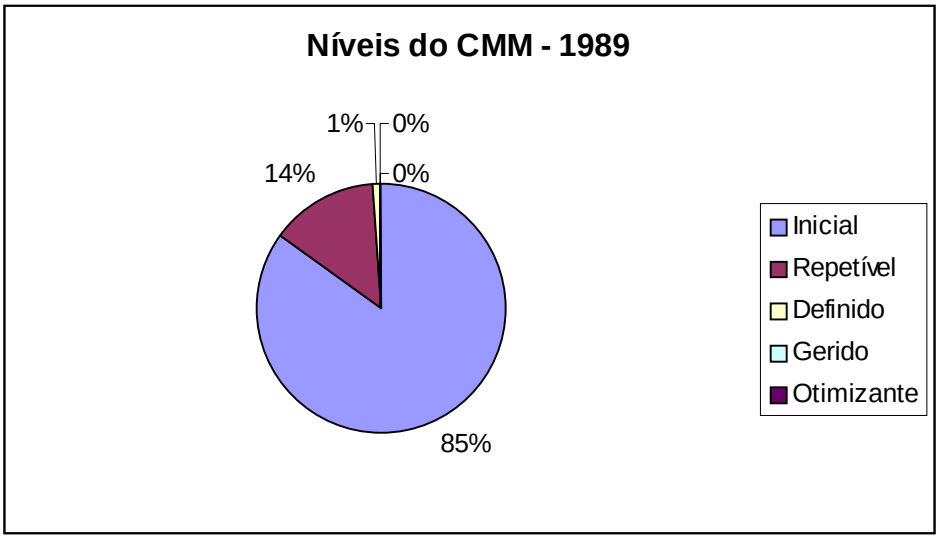

Figura 18 - Níveis encontrados em 1989

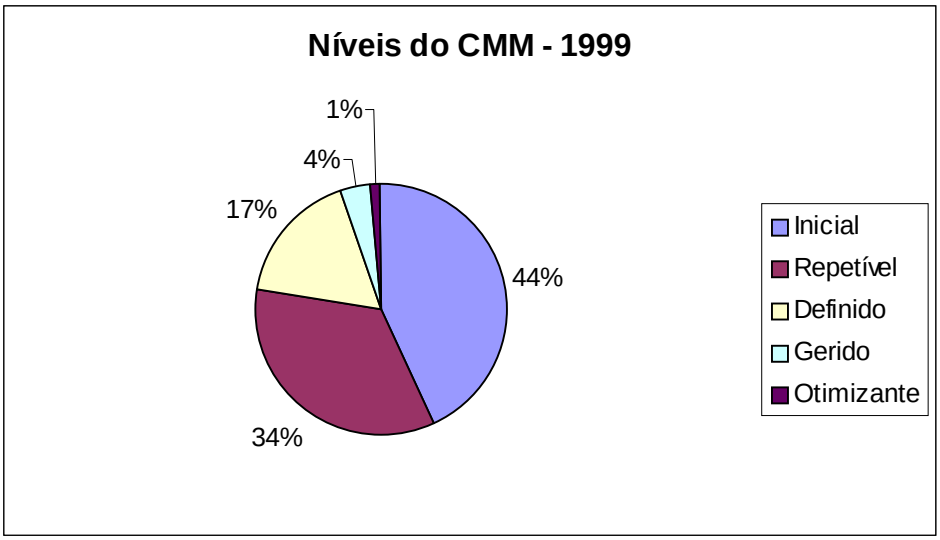

Figura 19 - Níveis encontrados em 1999

1.2 Áreas chaves

1.2.1 Introdução

Cada nível do CMM (exceto o nível 1) é composto de várias **áreas chaves de processo** ("key process areas", ou KPAs). Cada área chave identifica um grupo de atividades correlatas que realizam um conjunto de metas consideradas importantes, quando executadas em conjunto. As áreas chaves de um nível identificam as questões que devem ser resolvidas para atingir este nível.

No CMM, cada área chave reside em um único nível de maturidade. Para ser classificada em um determinado nível, uma organização tem de ter implementado completamente as áreas chaves deste nível e todos os níveis inferiores[3].

[3] Uma exceção é a área chave de Gestão da Subcontratação, do nível 2, que, obviamente, não é aplicável a organizações que não subcontratam desenvolvimento de software.

1.2.2 Práticas chaves

Cada área chave define um conjunto de **metas**, que representam o estado atingido por uma organização que domine a área chave. Para atingir as metas da área chave, a organização deve implementar um conjunto de práticas chaves. Estas práticas descrevem procedimentos gerenciais e técnicos, descritos no CMM, com grau de detalhe variável.

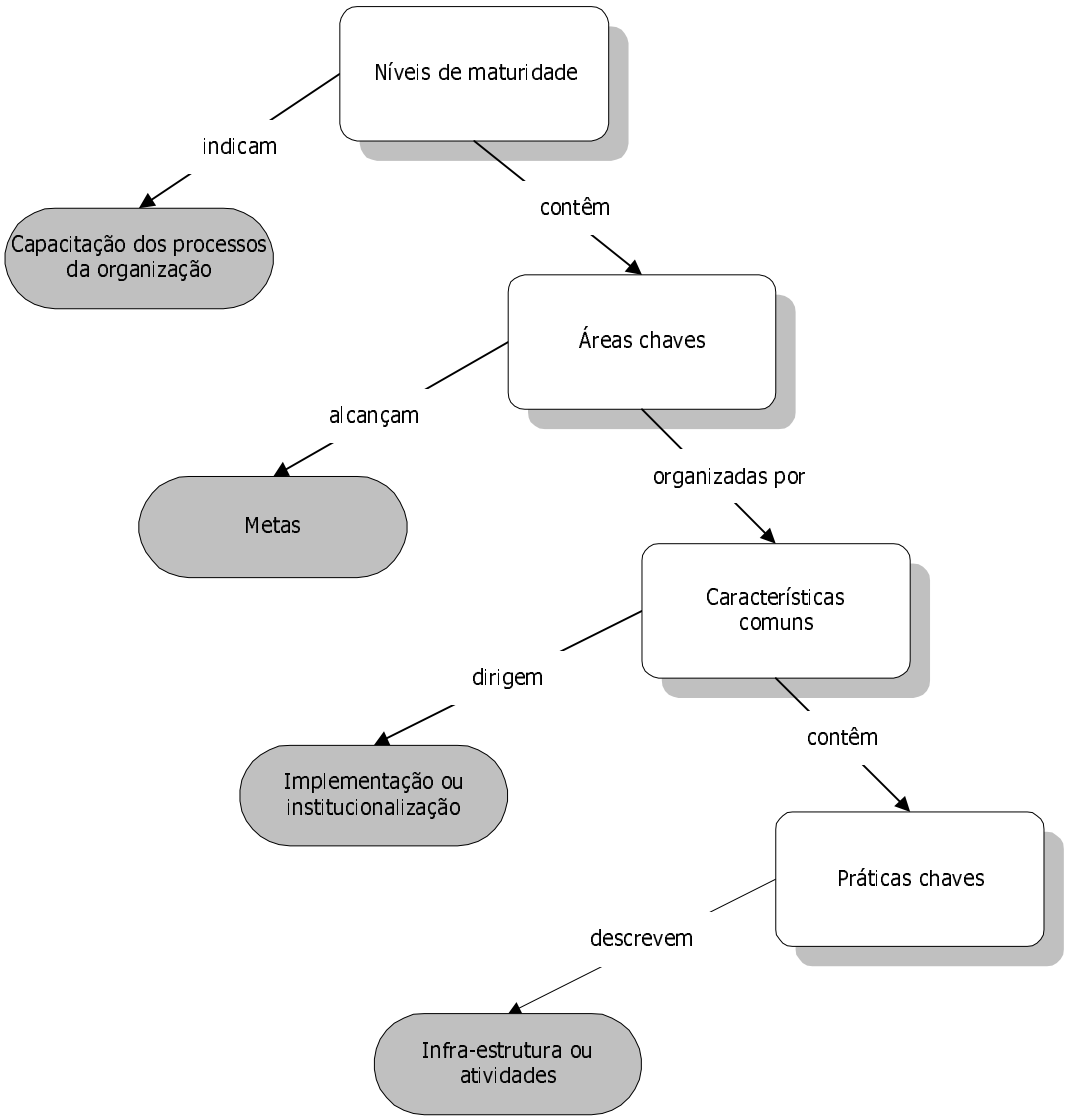

Figura 20 - Inter-relação dos elementos do SW-CMM

O grupo principal de práticas chaves é chamado de "Atividades a executar" ("activities to perform"). Estas atividades estão relacionadas diretamente com as metas que a área chave pretende atingir. Neste texto, serão também chamadas de **atividades de implementação**.

Por outro lado, um tema central do CMM é a estabilidade da melhoria conseguida. Folclore ou realidade, são muito difundidas histórias de organizações que recaem nos velhos processos viciados assim que o comitê avaliador dá sua nota e se retira. Por isto, o CMM definiu, em cada área chave, vários grupos de **atividades de institucionalização**. Estas atividades formam uma espécie de sistema de travas organizacionais, que dificulta o retrocesso nas atividades de implementação. As atividades de institucionalização são divididas nos seguinte grupos:

- **Comprometimento em executar** – condições que garantem a permanência da melhoria dos processos, geralmente associadas à existência de patrocinadores para a melhoria da área, e à existência de políticas documentadas.

- **Capacitação para executar** – fatores que contribuem para que a área seja efetivamente implantada, geralmente compreendendo estruturas organizacionais, recursos e treinamento.

- **Medição e análise** – medições básicas necessárias para avaliar o status da área chave.

- **Verificação da implementação** – ações que garantem a conformidade das demais atividades com os processos estabelecidos. No SW-CMM, geralmente abrangem ações de verificação por parte da gerência superior da organização, dos gerentes dos projetos e de um grupo independente de Garantia da Qualidade.

Apresenta-se como exemplo, a seguir, a estrutura da área chave de Gestão de Requisitos. Esta área é a primeira do nível do CMM, e uma das mais simples em quantidade de práticas chaves. A documentação do CMM apresenta maiores detalhes a respeito de cada prática. A descrição apresentada na Tabela 37 é simplificada em relação à definição oficial das práticas do CMM.

As metas a atingir requerem que todas as atividades técnicas e gerenciais dos projetos sejam baseadas nos requisitos do produto que será desenvolvido, e que os artefatos produzidos sejam todos consistentes com estes requisitos. Para isto, prevêem-se três atividades a executar:

- os requisitos devem sofrer revisão prévia dos grupos afetados, dos quais o grupo dos desenvolvedores é o mais óbvio;

- estes requisitos devem ser usados como base para o planejamento e para as diversas atividades do projeto;

- se os requisitos forem alterados ao longo de um projeto, estas alterações devem ser aprovadas pelos grupos afetados e incorporadas de forma disciplinada.

Para garantir que estas atividades sejam executadas de forma permanente e estável, o CMM requer que a organização tenha uma política documentada para a área; não basta existir um costume não escrito. Todo projeto tem de ter um responsável pelos requisitos, oficialmente designado. Todos os requisitos alocados ao software (dentre os requisitos do sistema), têm der ser documentados. As atividades de gestão de requisitos devem receber recursos suficientes, e o pessoal envolvido deve ser treinado no assunto. Devem ser feitas medições de status da gestão de requisitos; por exemplo, o grau de estabilidade dos requisitos de cada projeto. E, finalmente, a gestão de requisitos deve ser acompanhada, em nível mais alto, pela alta direção da organização, chamada no CMM de Gerência Executiva (“senior management”); a nível detalhado, pelos gerentes dos respectivos projetos; e por um grupo independente de Garantia da Qualidade, que fornece à Gerência Executiva informações independentes dos gerentes de projeto.

Metas	Atividades de engenharia e gestão de software baseadas em requisitos documentados. Consistência permanente de planos, produtos e atividades com os requisitos.	
Práticas chaves	Comprometimento em executar	Existência de política escrita para Gestão de Requisitos.
	Capacitação para executar	Designação de responsáveis pelos requisitos, em todos os projetos. Documentação dos requisitos alocados ao software. Existência de recursos e orçamento adequados para Gestão de Requisitos. Treinamento da equipe de software e equipes correlatas em Gestão de Requisitos.
	Atividades a executar	Revisão prévia dos requisitos pelos grupos afetados. Uso dos requisitos como base para planos, produtos e atividades. Revisão e incorporação ao projeto das mudanças de requisitos.
	Medição e análise	Status das atividades de Gestão de Requisitos.
	Verificação da implementação	Revisão periódica das atividades de Gestão de Requisitos pela Gerência Executiva. Revisão periódica e por eventos das atividades de Gestão de Requisitos pelos gerentes dos projetos. Revisão e auditoria das atividades de Gestão de Requisitos pelo Grupo de Garantia da Qualidade de Software.

Tabela 37 – Estrutura da área chave de Gestão de Requisitos

Com pequenas variações, as atividades de institucionalização das outras áreas chaves do CMM funcionam de forma semelhante. Por outro lado, as atividades de implementação são bastante específicas de cada área chave.

1.3 Níveis

1.3.1 A organização nível 1

A organização nível 1 representa o estágio inicial dos produtores de software. Ela utiliza processos informais e métodos ad hoc, às vezes descritos como caóticos. Muitas destas organizações são bem sucedidas, já que o mercado de software é ainda extremamente tolerante em relação à má qualidade dos produtos. Muitas vezes a qualidade do marketing pode ocultar deficiências técnicas, e existe pouca competição, em muitos setores deste mercado. A cultura no Nível Inicial é muito baseada no valor dos indivíduos. É comum a dependência em relação a heróis técnicos e gerenciais.

A organização nível 1 geralmente não é capaz de fazer estimativas de custo ou planos de projeto; se faz, não é capaz de cumpri-los. As ferramentas não são integradas com os processos, e não são aplicadas com uniformidade pelos projetos. Geralmente, a codificação é a única fase dos processos de desenvolvimento que merece atenção. Engenharia de requisitos e desenho são fracos ou inexistentes;

mudanças de requisitos e de outros artefatos ocorrem sem controle. A instalação e manutenção costumam ser deficientes, sendo encaradas como atividades de pouca importância.

Os gerentes destas organizações geralmente não entendem os verdadeiros problemas, por falta de processos que lhes dêem visibilidade real em relação ao progresso dos projetos. São comuns os casos de gerentes com formação exclusivamente administrativa, que não entendem os problemas técnicos dos projetos. Existem também aqueles que chegaram a gerentes como promoção da carreira técnica, e não têm a mínima formação em práticas gerenciais.

Podem existir processos definidos no papel, que não são aplicados na realidade, ou são sempre contornados, com a cumplicidade e até a pressão dos gerentes. Na melhor das hipóteses, os processos são seguidos quando os projetos estão em fase tranqüila; em crise, abandonam-se os métodos, e reverte-se à codificação desenfreada.

1.3.2 A organização nível 2

A tônica da organização nível 2 é ser capaz de cumprir compromissos. No nível repetível, uma organização é capaz de assumir compromissos referentes a requisitos, prazos e custos com alta probabilidade de ser capaz de cumpri-los. Isto requer o domínio das seguintes áreas chaves:

- a **gestão de requisitos** permite definição e controle dos requisitos em que se baseiam os compromissos;
- o **planejamento de projetos** prevê prazos e custos para cumprimento dos compromissos, como bases técnicas e não apenas intuitivas;
- a **supervisão e acompanhamento de projetos** confere o atendimento dos compromissos, comparando o conseguido com o planejamento, e acionando providências corretivas sempre que haja desvios significativos em relação aos compromissos;
- a **gestão da subcontratação** cobra de organizações subcontratadas para desenvolver partes do software os mesmos padrões de qualidade que a organização principal oferece a seus clientes;
- existe um grupo de **garantia da qualidade**, que confere o cumprimento dos compromissos, de forma independente em relação aos projetos;
- a **gestão de configurações** garante a consistência permanente dos resultados dos projetos, entre si e com os requisitos, ao longo do projeto, mesmo quando ocorram alterações nos compromissos.

Sigla em inglês	Nome em inglês	Tradução adotada neste texto
RM	Requirements Management	Gestão de Requisitos
SPP	Software Project Planning	Planejamento de Projetos
PTO	Software Project Tracking and Oversight	Supervisão e Acompanhamento de Projetos
SSM	Software Subcontract Management	Gestão da Subcontratação
SQA	Software Quality Assurance	Garantia da Qualidade
SCM	Software Configuration Management	Gestão de Configurações

Tabela 38 – Áreas chaves do CMM nível 2

A organização nível 2 é disciplinada a nível dos projetos. Por isto, ela sabe estimar e controlar projetos semelhantes a projetos anteriores bem sucedidos. Entretanto, ela corre riscos diante de vários tipos de mudanças a que as organizações estão sujeitas, tais como:

- mudanças de ferramentas e métodos, trazida pela evolução de tecnologia;
- mudanças de tipos de produto, causadas por variações dos mercados;
- mudanças de estrutura organizacional, causadas por diversos fatores da dinâmica das organizações.

1.3.3 A organização nível 3

O nível 3 conduz da gestão de projetos à engenharia de produtos. Este nível de organização não repete simplesmente os sucessos de projetos anteriores, mas estabelece uma infra-estrutura de processos que permite a adaptação a vários tipos de mudanças. Este nível de organização requer o domínio das seguintes áreas chaves:

- estabelecimento formal de um **grupo de processos de engenharia de software**, responsável pelas atividades de desenvolvimento, melhoria e manutenção de processos de software;
- estabelecimento de um **processo padrão de software** a nível da organização, a partir do qual devem ser derivados os processos definidos para os projetos;
- estabelecimento de um **programa de treinamento** em processos de software, a nível da organização;
- **gestão integrada dos projetos**, baseada nos processos definidos para os projetos, com o uso de procedimentos documentados para gestão de tamanho, esforços, prazos e riscos;
- padronização a nível da organização dos métodos de **engenharia de produtos de software**, abrangendo engenharia de requisitos, testes, desenho, codificação e documentação de uso;
- **coordenação entre os grupos** que participam de projetos de sistemas, a nível da organização;
- coordenação de **revisões técnicas** a nível da organização.

Sigla em inglês	Nome em inglês	Tradução adotada neste texto
OPF	Organization Process Focus	Focalização dos Processos da Organização
OPD	Organization Process Definition	Definição dos Processos da Organização
TP	Training Program	Programa de Treinamento
ISM	Integrated Software Management	Gestão Integrada de Software
SPE	Software Product Engineering	Engenharia de Produtos de Software
IC	Intergroup Coordination	Coordenação entre Grupos
PR	Peer Reviews	Revisões Técnicas

Tabela 39 – Áreas chaves do CMM nível 3

A organização nível 3 sabe manter-se dentro do processo, mesmo durante as crises. As ferramentas passam a ser aplicadas de forma sistemática, padronizada e coerente com os processos. Com isto,

passam a contribuir significativamente para melhoria da produtividade e qualidade. Por outro lado, o conhecimento dos processos, por parte da organização nível 3, ainda é basicamente qualitativo. Existe uma base de dados de processos, povoada com os dados recolhidos dos projetos; esta base é usada para gestão dos projetos, mas não é ainda aplicada, de forma sistemática e a nível da organização, para atingir metas quantitativas de desempenho de processo e de qualidade de produto.

1.3.4 A organização nível 4

Na organização nível 4, o domínio dos processos de software evolui para uma forma quantitativa. Isto não quer dizer que apenas organizações deste nível devam coletar métricas de processo. Todas as áreas chaves do CMM contêm pelo menos uma prática de medição e análise, que sugere métricas adequadas para medir o sucesso da implantação da respectiva área. A organização nível 3 constrói e mantém uma base de dados de processos.

Coleta de dados é uma atividade cara: é necessário definir com precisão e antecipação os dados que vão ser coletados. Estes dados têm de ser criticados, consistidos e normalizados para terem alta qualidade. A organização nível 4 é proficiente em coletar métricas e gerir a base de dados de processo, que é povoada e analisada por profissionais treinados. Além disto, sabe intervir nos processos para atingir metas de qualidade dos produtos. Este nível tem apenas duas áreas chaves:

- a **gestão quantitativa dos processos** controla o desempenho dos processos usados pelos projetos;
- a **gestão da qualidade de software** promove o entendimento quantitativo da qualidade dos produtos de software, permitindo atingir metas quantitativas desejadas.

Sigla em inglês	Nome em inglês	Tradução adotada neste texto
QPM	Quantitative Process Management	Gestão Quantitativa dos Processos
SQM	Software Quality Management	Gestão da Qualidade de Software

Tabela 40 – Áreas chaves do CMM nível 4

A organização nível 4 passa da engenharia de produtos à qualidade de processos e produtos. Ela tem elementos para decidir, por exemplo, qual deve ser a fração de recursos dos projetos destinada à garantia da qualidade, considerando o nível máximo de defeitos que se quer admitir nos produtos. Este domínio quantitativo dos processos é necessário para atingir um estado de melhoria contínua.

1.3.5 A organização nível 5

A organização nível 5 atinge um estado em que os processos estão em melhoria contínua, sendo otimizados para as necessidades de cada momento. As seguintes áreas chaves são executadas:

- prevenção dos defeitos, através da identificação e remoção das causas deles;
- gestão da evolução tecnológica, com procedimentos sistemáticos de identificação, análise e introdução de tecnologia apropriada;
- uso dos dados de processo para gestão das mudanças de processos, colocando-os em melhoria contínua.

Sigla em inglês	Nome em inglês	Tradução adotada neste texto
DP	Defect Prevention	Prevenção de Defeitos
TCM	Technology Change Management	Gestão das Mudanças de Tecnologia
PCM	Process Change Management	Gestão das Mudanças de Processos

Tabela 41 – Áreas chaves do CMM nível 5

1.4 Alternativas

1.4.1 Outros modelos de capacitação

Existem vários outros modelos, além do SW-CMM, que podem ser usados como referência para avaliação de capacitação das organizações em processos de software. Alguns destes modelos foram desenvolvidos por empresas privadas, como os modelos da SPR (Software Productivity Research) e HP (Hewlett-Packard). Outros representam combinações do SW-CMM com outros padrões, como o Trillium, desenvolvido por um grupo de grandes empresas da área de telecomunicações.

Alguns modelos alternativos importantes foram criados por organizações que desenvolvem padrões, como a ISO (International Organization for Standardization) e o IEEE (Institute of Electrical and Electronics Engineers). São muitos difundidos no Brasil os modelos da série ISO-9000; o modelo aplicável à área de processos de software será discutido na subseção seguinte. Alguns dos padrões de Engenharia de Software do IEEE serviram de modelo para diversos padrões incluídos neste texto; eles serão citados no respectivo capítulo.

Outros modelos de maturidade foram desenvolvidos pelo SEI:

- recursos humanos (P-CMM);
- engenharia de sistemas (SE-CMM);
- aquisição de software (SA-CMM);
- definição de produtos (IPD-CMM).

O modelo SE-CMM é de arquitetura contínua, como o Spice. O SEI chegou a publicar versões preliminares do SW-CMM v.2, mas recentemente decidiu unificar o SW-CMM, SE-CMM e IPD-CMM em um único modelo, chamado de CMMI (Capability Maturity Model Integration). Este modelo é apresentado em versões de arquitetura em estágios e arquitetura contínua.

1.4.2 CMM e ISO-9000

A denominação ISO-9000 abrange uma família de padrões com âmbitos diferentes. Por exemplo, o padrão ISO-9001 descreve sistemas de qualidade para processos industriais, abrangendo as atividades de desenho, desenvolvimento, produção, instalação e assistência técnica. O padrão ISO-9000-3 (que não deve ser confundido com o ISO-9003) representa uma adaptação do ISO-9001 para o desenvolvimento, fornecimento e manutenção de software, levando em conta certas características que são específicas deste ramo industrial.

É difícil uma comparação direta entre CMM e ISO-9000-3. Há grandes diferenças de detalhamento: as práticas do CMM são descritas a nível muito mais detalhado, com grande número de exemplos e sugestões de implementação. A estrutura também é completamente diferente: o modelo ISO-9000-3 não tem arquitetura em estágios. Uma organização passa ou não passa na avaliação ISO-9000, feita por uma entidade certificadora. Finalmente, o CMM não tem caráter de certificação, mas apenas de

informação de interesse da organização aferida ou de um seu cliente que tenha encomendado a avaliação.

Um estudo comparativo do CMM e dos modelos ISO-9001 e ISO-9000-3 é apresentado em [Paulk94] e resumido em [Paulk+95]. Segundo este estudo, as práticas do modelo ISO-9000-3 cobrem a maior parte do CMM nível 2, boa parte do CMM nível 3 e algumas práticas do CMM níveis 4 e 5. A grosso modo, uma organização que implemente o modelo ISO-9000-3 provavelmente estará situada no nível 2 do CMM, com parte do nível 3 coberta. Por outro lado o CMM não cobre algumas práticas previstas no ISO-9000-3, como inclusão de produtos, implantação e manutenção de software. Estas práticas deveriam ser implementadas adicionalmente por organizações que queiram obter a certificação ISO-9000.

Mais recentemente, a ISO tem desenvolvido o modelo ISO-15504 (conhecido como Spice), que representa um guia para avaliação de maturidade de processos. O modelo ISO-15504 tem **arquitetura contínua**; isto é, a avaliação da organização é separada por área de processos, não existindo o conceito de níveis de maturidade globais.

1.5 Benefícios

O SW-CMM foi adotado como referência principal deste texto pelas seguintes razões:

- existência de informação detalhada publicada, tanto em relação ao modelo em si ([Paulk+93], [Paulk+93a], [Paulk+95]), quanto à sua base conceitual ([Humphrey90]) e à interpretação por avaliadores ([Dymond95]);

- arquitetura em estágios, que orienta quanto à escolha de áreas prioritárias para a implementação da melhoria de processos;

- existência de referências publicadas com dados sobre benefícios observados em organizações que implementaram o CMM (por exemplo, [Diaz+97], [Dion93], [Goldenson+95], [Herbsleb+94], [Humphrey+91] e [Sims94]).

O estudo de [Herbsleb+94] resumiu as avaliações de um conjunto de organizações produtoras de software, chegando aos seguintes resultados agregados:

	Faixa	**Mediana**
Duração do programa de melhoria de processos	1 a 9 anos	3,5 anos
Custo anual total	$ 49K a $ 1.202K	$ 245K
Custo anual por engenheiro de software	$ 490 a $ 2.004	$ 1375
Ganho anual de produtividade	9% a 67%	35 %
Redução anual de tempo até o mercado	15% a 23%	19 %
Ganho anual de detecção precoce de defeitos	6% a 25%	22 %
Redução anual em defeitos achados em operação	10% a 94%	39 %
Retorno do investimento	400% a 880%	500 %

Tabela 42 – Ganhos obtidos em implementações do CMM

Além disto, os seguintes benefícios intangíveis foram relatados nesse estudo:

- melhoria do moral;

- melhoria da qualidade de vida;

- diminuição de horas extras;

- maior estabilidade do ambiente de trabalho;
- menor rotatividade da equipe;
- melhoria da comunicação;
- melhoria da qualidade percebida pelos clientes.

Capers Jones, que em [Jones94] havia criticado o CMM pela escassez de resultados publicados, relatou os seguintes valores de retorno de investimento, em [Jones98]:

Nível do CMM	RI – 1 ano %	RI – 2 anos %	RI – 4 anos %
SEI CMM 2	100	115	250
SEI CMM 3	175	250	500
SEI CMM 4	300	400	950
SEI CMM 5	350	475	1275

Tabela 43 – Retorno do investimento na implantação dos níveis do CMM

1.6 Estrutura organizacional

1.6.1 Visão geral

A entidade visada pelo CMM é a **Organização**. A organização é o volume de controle do CMM; as aferições focalizam as práticas de uma organização, que pode ser uma companhia, um divisão de uma companhia, agência de governo, ou uma ramificação de qualquer uma destas entidades. Considera-se que, em uma organização, o desenvolvimento de produtos é feito através de **projetos**.

Para o CMM, um projeto é um empreendimento que visa desenvolver ou melhorar um produto de informática. O produto pode envolver outros componentes, além de software. Tipicamente, cada projeto tem seu próprio orçamento e cronograma. O CMM não distingue subprojetos, embora estes possam existir em organizações reais. Atividades permanentes, executadas rotineiramente, sem datas de início e fim, não são projetos. Note-se que em organizações reais algumas destas atividades podem ser denominadas como projetos.

1.6.2 Papéis organizacionais

O CMM descreve um conjunto de práticas recomendáveis de engenharia e gestão, e não de leis exatas. Ele tenta ser independente da estrutura da organização alvo, de modo que possa ser utilizado com a grande variedade de estruturas que existe nas organizações da vida real. Por outro lado, a descrição de algumas funções e estruturas é necessária para o correto entendimento do CMM.

Em organizações de grande porte, com centenas de programadores, é mais fácil definir uma correspondência entre as estruturas do CMM e as estruturas reais. Para organizações de menor porte, o CMM deve ser interpretado adequadamente. Para isto, devem ser bem entendidos os **papéis** organizacionais mencionados no CMM. Os papéis são unidades de responsabilidades que podem ser assumidas por um ou mais indivíduos.

Estes papéis são usados para descrever funções lógicas. Eles devem ser mapeados nas estruturas reais de cada organização. Por exemplo, é comum atribuir à mesma função organizacional real vários papéis lógicos do CMM. Isto é possível porque, em organizações menores, às vezes não há necessidade de que os papéis sejam desempenhados por pessoas em tempo integral.

1.6.3 Gerentes

O CMM descreve diversos tipos de **gerente** (manager). Um gerente é o responsável técnico e administrativo de uma área da organização. Esta responsabilidade inclui as tarefas de planejamento, organização, direção e controle desta área. Em todas as áreas chaves são atribuídas responsabilidades aos seguintes papéis.

- **Gerência Executiva** (Senior Management) - representa o nível hierárquico e estratégico mais alto da organização. É responsável pelas decisões estratégicas da organização. Em uma organização saudável, a Gerência Executiva focaliza principalmente as questões de longo prazo. Envolve-se em todos os projetos, em nível mais alto de abstração. À Gerência Executiva cabem as decisões sobre metas e recursos para melhoria dos processos. Esta melhoria dificilmente ocorre se a Gerência Executiva não for a parte mais interessada nisto.

- **Gerente de projeto** (Project Manager) - tem responsabilidade completa por um projeto, inclusive a direção, controle e administração deste. O gerente de um projeto deve ser o único responsável por este, aos olhos do cliente. Cabe ao gerente liderar a equipe do respectivo projeto.

O CMM prevê que certas decisões devem ser reservadas à Gerência Executiva. Por exemplo, apenas a Gerência Executiva pode assumir compromissos com clientes externos (ou alterações nestes compromissos). A Gerência Executiva deve realizar reuniões periódicas para verificar a execução das práticas de todas as áreas chaves do CMM. Deve também supervisionar todos os projetos, e resolver pendências sobre as quais não for possível acordo entre os gerentes de projeto e os responsáveis pela garantia da qualidade.

Outros papéis gerenciais menores são mencionados em algumas práticas do CMM. Em projetos maiores, o gerente de projeto pode ser auxiliado por um ou mais **gerentes imediatos** (First Line Managers), responsáveis por uma divisão do projeto. Dentro de uma divisão, podem ser formadas equipes para resolver tarefas específicas; estas equipes são chefiadas por um **líder de tarefa** (task leader).

1.6.4 Grupos

Um **grupo** é um conjunto de profissionais e unidades que tem uma responsabilidade definida. Um grupo não é necessariamente uma equipe de várias pessoas. Ele pode ser formado por uma pessoa em tempo parcial ou integral, por pessoas de um ou vários departamentos, ou até por departamentos inteiros.

O CMM chama de Grupo de Engenharia de Software os profissionais responsáveis pelas tarefas técnicas de desenvolvimento e manutenção de software. Como o CMM focaliza o desenvolvimento e não a manutenção, o Grupo de Engenharia de Software geralmente deve ser entendido como a equipe técnica de um projeto de desenvolvimento. Neste texto, será usado o termo **desenvolvedores**.

O CMM atribui muitas funções aos "grupos relacionados com software". Estes grupos representam uma disciplina de Engenharia de Software praticada a nível da organização e não de projetos individuais. Estes grupos dão suporte às atividades das equipes dos projetos, e serão chamados, neste texto, de grupos de suporte. Os principais grupos requeridos pelo CMM são os seguintes.

- **Grupo de Garantia da Qualidade de Software**, o **GGQSw** ("SQA - Software Quality Assurance") - planeja e implementa atividades que asseguram a qualidade do produto. Requerido em todas as organizações de nível 2 ou superior. Deve ter, obrigatoriamente, autonomia em relação aos projetos, embora deva colaborar com estes na resolução de problemas relativos à qualidade. É responsável por fornecer à Gerência Executiva um canal de informação sobre problemas, independente dos gerentes de projeto.

- **Grupo de Gestão de Configurações de Software**, o **GGCSw** ("SCM - Software Configuration Management") - planeja, coordena, e implementa ações para gerir um sistema centralizado de guarda de configurações de software. Requerido em todas as organizações de nível 2 ou superior. Não é este grupo quem decide que alterações solicitadas aos projetos devem ser realizadas, mas é o responsável operacional pela biblioteca de configurações de software.

- **Grupo de Engenharia de Processos de Software**, o **GEPSw** ("SEPG - Software Engineering Process Group[4]") - responsável pela definição, manutenção e melhoria dos processos de software da organização. Requerido em todas as organizações de nível 3 ou superior. Note-se que, mesmo em organizações nível 1, deve existir um grupo responsável pelas tarefas de melhoria dos processos. Entretanto, só na descrição do nível 3 este grupo tem suas tarefas formalmente enunciadas.

Outros grupos mencionados no CMM incluem os seguintes.

- Em projeto de sistemas, o Grupo de Engenharia de Sistemas especifica requisitos de nível de sistema, alocando requisitos de sistema a hardware, software e outros componentes. É responsável por especificar interfaces entre componentes, e assegurar a padronização destes.

- O Grupo de Testes de Software planeja, desenha e realiza testes de aceitação do produto de um projeto. Para garantir a eficácia destes testes, deve atuar independentemente dos desenvolvedores.

- O Grupo de Treinamento coordena e executa as atividades de treinamento em software. Para isto, administra diversos veículos de instrução e difusão. Em organizações reais, pode fazer parte do setor de Recursos Humanos, ou pode ser uma extensão do Grupo de Engenharia de Processos de Software.

1.6.5 Exemplo de estrutura organizacional

A Figura 21 mostra um exemplo de estrutura organizacional inspirada no CMM. A estrutura aí mostrada é uma estrutura lógica. Assim, é possível que cada gerência ou cada grupo corresponda a uma comissão, uma pessoa em tempo integral ou uma pessoa em tempo parcial. A mesma pessoa pode acumular funções lógicas da estrutura. A Tabela 44 descreve as atribuições de cada papel. Em relação ao CMM, essa estrutura apresenta algumas diferenças.

A Gerência Executiva foi desdobrada em uma Gerência Geral e em gerências especializadas para as áreas de Desenvolvimento, Qualidade, Manutenção e Suporte Técnico. As tarefas reservadas à Gerência Executiva podem ser exercidas pelo gerente geral, delegadas aos gerentes especializados ou realizadas coletivamente pelo grupo superior de gerentes. A Tabela 45 descreve alguma possíveis delegações à Gerência de Desenvolvimento. As delegações devem fazer parte de uma política documentada da organização.

Foram incluídas as áreas de Manutenção e Suporte Técnico. Embora obviamente necessárias em qualquer organização produtora de software, estes assuntos não foram incluídos no escopo do SW-CMM 1.1, a não ser por algumas poucas referências em práticas relacionadas.

Os três principais grupos de suporte mencionados no CMM foram agrupados em uma Gerência da Qualidade. Esta vinculação facilita a ligação destes grupos com a Gerência Executiva e entre si. Em uma organização pequena ou principiante, é razoável atribuir ao mesmo grupo de pessoas a

[4] Uma tradução mais literal seria "Grupo de Processos de Engenharia de Software". A denominação aqui adotada (Grupo de Engenharia de Processos de Software) procura enfatizar que, para esta disciplina, os processos são eles próprios um produto, e o desenvolvimento e manutenção deles é uma atividade de engenharia.

responsabilidade por estas três áreas. Membros das equipes dos projetos só devem participar das tarefas destes grupos em caráter ad-hoc, nunca atuando em relação aos próprios projetos.

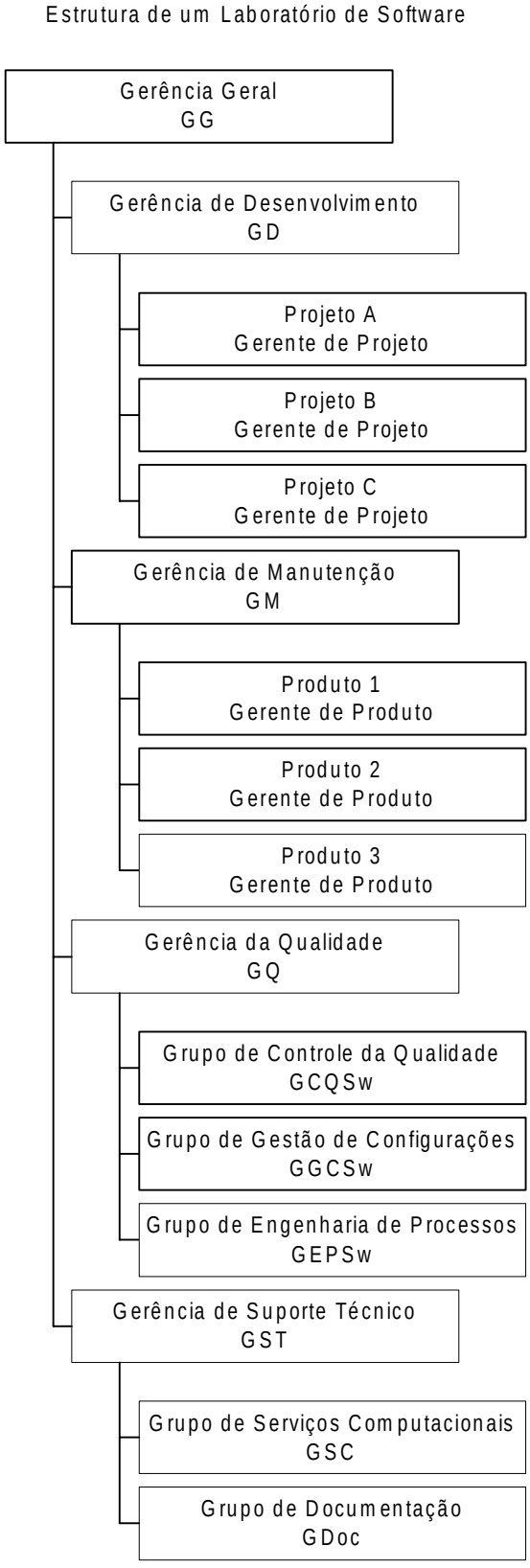

Figura 21 – Estrutura organizacional básica

Função	Sigla	Atribuições
Gerência Geral	**GG**	Gerência responsável pelas decisões de mais alto nível referentes à organização. Inclui o planejamento e controle administrativo e financeiro da organização.
Gerência de Desenvolvimento	**GD**	Gerência responsável por um conjunto de projetos de software ativos na organização.
Gerência de Manutenção	**GM**	Gerência responsável por um conjunto dos produtos de software a cuja operação a organização dá suporte.
Gerência da Qualidade	**GQ**	Gerência responsável pelo conjunto de atividades referentes à qualidade, à gestão de configurações e à melhoria dos processos da organização.
Gerência de Suporte Técnico	**GST**	Gerência responsável pelo conjunto de atividades de suporte técnico aos projetos da organização.
Grupo de Recursos Computacionais	**GRC**	Grupo responsável pelo fornecimento de serviços computacionais aos desenvolvedores de software.
Grupo de Documentação	**GDoc**	Grupo responsável pela guarda, atualização e distribuição de documentação on-line.
Grupo de Garantia da Qualidade de Software	**GGQSw**	Grupo responsável por planejar e implementar atividades que asseguram a qualidade dos produtos.
Grupo de Gestão de Configurações de Software	**GGCSw**	Grupo responsável por planejar, coordenar, e implementar ações para gerir um sistema centralizado de guarda de configurações de software.
Grupo de Engenharia de Processos de Software	**GEPSw**	Grupo responsável pela definição, manutenção e melhoria dos processos de software da organização. Considera-se que este grupo engloba o Grupo de Treinamento.

Tabela 44 - Descrição de papéis organizacionais

Número	Responsabilidade	Delegação
1	A Gerência Executiva deve revisar as atividades de Gestão de Requisitos de Software.	Delegar à Gerência de Desenvolvimento a tomada de providências, quando as mudanças de requisitos de um projeto atingirem nível considerado anormal.
2	A Gerência Executiva deve revisar e homologar todos os compromissos assumidos com clientes e outras organizações.	Delegar à Gerência de Desenvolvimento a aprovação de compromissos que não envolvam aspectos financeiros ou políticos significativos.
3	A Gerência Executiva deve realizar reuniões periódicas para revisar as atividades de gestão de projetos de software sob sua alçada.	Delegar à Gerência de Desenvolvimento a realização destas reuniões, acompanhando os projetos, em situações normais, apenas através dos Relatórios Mensais de Projeto de Software.

Tabela 45 - Delegações para a Gerência de Desenvolvimento

1.7 Institucionalização

1.7.1 Visão geral

Uma vez que uma organização consiga melhorar seus processos, aparece a questão de como manter o que foi conseguido geralmente a duras penas. Nos primeiros tempos, o retorno do investimento nem sempre é evidente de imediato. Quando as primeiras crises acontecem, muitos têm reações de pânico e

acham que a bagagem de processo deve ser a primeira a jogar no mar. Os que resistiram à mudança dos processos aproveitarão todas as oportunidades possíveis para reclamar da suposta burocracia de processo, ou, no mínimo, para argumentar que o projeto específico deles é diferente, e portanto deve ser dispensado dos processos.

Estes problemas ocorrem com todos os tipos de alteração da cultura organizacional. Modas organizacionais aparecem e são adotadas sofregamente, com um mínimo de espírito crítico e adaptação à realidade local. Quando surgem as primeiras dificuldades, as pessoas apelam para a improvisação, e a moda vai sendo desacreditada. Finalmente, quando acontecem problemas sérios, muitos pessoas culpam o modelo, para não ter de admitir a própria culpa. Até aparecer uma nova moda, que fecha o ciclo.

A proposta do CMM para fugir deste círculo vicioso está nas atividades de institucionalização. Elas estabelecem diversas amarrações organizacionais para garantir a prática e estabilidade das atividades principais. Elas são discutidas nas subseções seguintes. Espera-se que, depois de certo tempo, as vantagens da melhoria dos processos se tornem evidentes. Quando o nível de aceitação dos processos se torna satisfatório, eles entram para a cultura da organização, e passam a ser praticados como hábito.

1.7.2 Comprometimento em Executar

Os Comprometimentos em Executar descrevem ações que a organização deverá realizar para garantir que as melhorias de processo sejam estáveis e duradoura. Tipicamente, envolvem definições de políticas organizacionais ou de responsabilidades.

Normalmente, cada área chave deve ter uma política documentada, isto é, escrita e oficializada. O CMM sugere itens que devam constar de cada política, mas o detalhamento destas deve ser baseado em necessidades específicas da organização. Como pode ser visto na Tabela 46, todas as áreas chaves do nível 2 prevêem a existência de uma política documentada.

Nas áreas de Planejamento de Projetos de Software, Acompanhamento e Controle de Projetos de Software e Gestão da Subcontratação o CMM prevê a designação de um gerente responsável, e descreve as atribuições deste.

1.7.3 Capacitação para Executar

As Capacitações para Executar são condições prévias para a implantação correta dos processos de desenvolvimento de software. Em alguns casos referem-se aos projetos, e em outros à organização como um todo. Tipicamente envolvem recursos, estrutura da organização e treinamento.

Todas as áreas chaves requerem que recursos materiais e financeiros estejam disponíveis para a respectiva execução. Os recursos materiais podem se constituir de pessoal com determinadas competências, ou de ferramentas específicas da área. Os recursos financeiros devem ser realmente disponíveis e não apenas orçados.

Quase todas as áreas chaves têm requisitos de treinamento. Este treinamento pode ser fornecido de diversos meios: aulas, treinamento por vídeo, reuniões, treinamento baseado em computador ou mentoreação. Os membros da organização devem ter disponibilidade real de tempo e recursos para participar dos treinamentos necessários à respectiva função. A partir do nível 3, é exigido que este treinamento faça parte de um programa sistemático, planejado e avaliado.

Área chave do CMM nível 2	Comprometimentos em Executar
Gestão dos Requisitos	Todo projeto segue uma política organizacional documentada para gestão dos requisitos de sistema alocados ao software.
Planejamento de Projetos de Software	Todo projeto de software tem um gerente, que é responsável por negociar compromissos e elaborar o plano de desenvolvimento de software.
	Todo projeto segue uma política organizacional documentada para planejamento de projetos de software.
Acompanhamento e Controle de Projetos de Software	Todo projeto de software tem um gerente, que é responsável pela atividades e resultados do projeto.
	Todo projeto segue uma política organizacional documentada para gestão de projetos de software.
Garantia da Qualidade de Software	Os projetos seguem uma política organizacional documentada para implementar a garantia da qualidade de software.
Gestão de Configurações de Software	Os projetos seguem uma política organizacional documentada para implementar a gestão de configurações de software.
Gestão da Subcontratação	Os projetos seguem uma política organizacional documentada para gestão de subcontratos de software.
	Todo projeto com subcontratação tem um gerente de subcontrato, como responsável pelo estabelecimento e gestão do subcontrato.

Tabela 46 - Comprometimentos em Executar das áreas chaves do nível 2 do CMM

Para alguns papéis e áreas chaves, o CMM admite uma orientação no lugar do treinamento. A orientação envolve menos profundidade e formalismo. Pode ser dada, por exemplo, através de consultoria dos grupos de suporte às equipes dos projetos.

1.7.4 Medição e Análise

As Medições e Análises servem para determinação do estado atual dos processos da organização. Elas fornecem os elementos para que a Gerência Executiva avalie com mais precisão o grau de sucesso da implantação de cada área chave. Idealmente, devem servir de base para o cálculo do retorno do investimento em cada área.

1.7.5 Verificação da Implementação

O CMM considera que, na maioria das organizações, a aderência aos processos estabelecido não acontece espontaneamente. Por isto, as práticas de todas as áreas chaves são sujeitas a no mínimo três instâncias de averiguação: pela Gerência Executiva; pelos gerentes de projeto; e pelo Grupo de Garantia da Qualidade de Software.

A averiguação periódica pela Gerência Executiva tem por objetivo fornecer à alta administração a percepção das atividades de software, no nível adequado. Esta verificação deve ser feita de forma periódica. O período exato depende dos costumes da organização e do nível de envolvimento que a Gerência Executiva está disposta a assumir. Dependendo do tamanho da organização, um intervalo de um a três meses costuma ser um período razoável.

A averiguação pelos gerentes de projetos deve ser tanto periódica quanto dirigida por eventos. Os eventos dependem da características de cada projeto, estando comumente associados aos marcos destes. Por outro lado, são também necessárias reuniões mais freqüentes de acompanhamento rotineiro dos projetos; por exemplo, a cada semana, ou o quanto seja necessário para uma boa percepção do andamento. Estas reuniões são mais detalhadas que as da Gerência Executiva, devido ao envolvimento operacional que o gerente de projeto deve ter.

A maioria das atividades do Grupo de Garantia da Qualidade de Software (GGQSw) é descrita detalhadamente nas práticas da área chave de Garantia da Qualidade de Software. Além disto, em quase todas as áreas chaves cabe ao GGQSw verificar se as atividades dos projetos estão sendo conduzidas de acordo com as práticas adotadas. Em caso de problemas, o CMM prescreve que o GGQSw deve procurar resolvê-los, inicialmente, em conjunto com o respectivo gerente de projeto. Caso isto não seja possível, a pendência deve ser decidida pela Gerência Executiva.

2 O sistema de comprometimento

2.1 A filosofia de comprometimento

Na rota do CMM, o caminho para a melhoria dos processos começa pela capacitação em cumprir compromissos. A organização deve aprender a cumprir o que promete a seus clientes, em termos de requisitos, de custos, de prazos e de qualidade. Isto não significa eliminar as incertezas inerentes a qualquer planejamento. Significa que os erros de previsão deverão se tornar pequenos e aleatórios. No nível inicial, estes erros são geralmente grandes e sistemáticos; raramente, por exemplo, alguém entrega mais cedo do que prometeu.

Um tema central do nível 2 do CMM é o sistema de comprometimento, tal como definido por Watts Humphrey [Humphrey90]. Este sistema é semelhante à filosofia de comprometimento que pode ser observado na norma ISO-9000. Ele considera que os seguintes elementos são a base de um comprometimento eficaz:

- avaliação cuidadosa das tarefas que devem ser executadas, dos recursos efetivamente disponíveis e dos prazos necessários;
- concordância entre todas as partes sobre o que deve ser feito, por quem deve ser feito e quando deve ser feito;
- divisão do desenvolvimento em atividades de menor complexidade e de menor duração, de maneira a facilitar a previsão, medição e avaliação de cada atividade;
- fases terminadas de forma bem definida, fechadas por pontos de controle onde o resultado da fase é avaliado de maneira previamente combinada entre as partes;
- tomada de medidas corretivas completas, o mais cedo possível, quando são detectados problemas;
- caráter aberto, público e preciso dos compromissos, sem jogos políticos e sem promessas vagas;
- consciência de todos os envolvidos quanto à responsabilidade em cumprir os compromissos buscando ajuda sempre que for preciso;
- se um compromisso não puder ser cumprido, renegociação antecipada e não após o fato consumado;
- boa vontade de todas as partes, sem a qual nenhum dos outros elementos funciona.

O CMM traduz estes elementos nos seguintes requisitos, que estão embutidos em muitas práticas das áreas do nível 2.

- Todos os compromissos assumidos com o cliente requerem aprovação de Gerência Executiva. Isto é particularmente importante quando estes compromissos envolvem entrega de resultados.

Dizia o filósofo do futebol Neném Prancha que "pênalti é tão importante que deveria ser batido pelo presidente do clube". O compromisso com o cliente deve ser encarado como o pênalti.

- Os compromissos só podem ser assumidos depois de um processo formal de revisão e concordância. Isto significa que compromissos que dependem de desenvolvedores e de outros profissionais devem ser analisados por estes, em caráter oficial. Os gerentes não podem prometer e só depois comunicar aos desenvolvedores o que foi prometido.

- Deve haver mecanismos para conduzir da forma correta os processos de revisão e concordância. Isto significa que a concordância não pode ser dada através de uma conversa no cafezinho. Ela deve resultar de uma análise conduzida de acordo com padrões predefinidos, incluídos no processo de desenvolvimento.

O próprio sistema de comprometimento deve ser documentado. Isto quer dizer que as regras acima não devem ser simplesmente tácitas, mas devem estar escritas em documentos de política da organização. Estes documentos devem estar disponíveis a todos os interessados. Alguns exemplos de regras que devem constar da política documentada são os seguintes.

- Que nível de aprovação requer cada tipo de comprometimento? Quais compromissos podem ser assumidos pelos desenvolvedores, quais requerem aprovação dos gerentes de projeto, e quais são da alçada da Gerência Executiva?

- Quando estas aprovações são necessárias? Que tipo de aprovação requer cada ponto de controle do projeto?

- Qual a preparação para os pedidos de aprovação? Que documentos são necessários para a decisão? Que tipos de revisão devem ser feitos?

- Como são tratados os compromissos problemáticos? Quais os mecanismos para parar projetos que têm resultados não aprovados?

Os gerentes têm obrigações particularmente importantes em relação aos compromissos. Eles devem ter muito cuidado ao assumir compromissos; mas uma vez assumidos, devem fazer muito esforço para cumpri-los. Devem apoiar os profissionais técnicos nas negociações com os clientes e usuários, e lutar por todos os recursos necessários ao projeto. O cronograma de reuniões deve ser sempre mantido em dia. Os gerentes jamais devem impor compromissos aos profissionais de sua equipe; a posição destes quanto à viabilidade dos compromissos deve ser sempre respeitada. Por outro lado, devem cobrar dos profissionais o máximo de seriedade em relação aos compromissos assumidos.

2.2 Bases materiais do comprometimento

O sistema de comprometimento abrange uma filosofia de trabalho, mas tem de ser traduzido em artefatos reais. Os principais artefatos do sistema de comprometimento são a Especificação de Requisitos, o Plano de Desenvolvimento e o Plano da Qualidade.

Primeiro, o trabalho a ser executado deve ter uma definição documentada, completa, consistente e precisa. Esta definição consiste na Especificação de Requisitos. Ela deve ser no mínimo aprovada pelo cliente; é muito importante que os usuários efetivos, ou seus representantes autorizados, participem da redação dela.

O Plano de Desenvolvimento definirá os recursos necessários, os custos previstos e os prazos possíveis. Estes custos, prazos e recursos são calculados de forma técnica documentada, por pessoal proficiente em planejamento de projetos. Incluirá também a análise dos riscos gerenciais e técnicos dos compromissos, indicando alternativas e contramedidas para o caso em que alguns destes riscos se concretizem. Este plano identificará todos os recursos necessários, inclusive de pessoal. Se for o caso, definirá as necessidades de treinamento ou recrutamento de pessoas.

Finalmente, o Plano da Qualidade estabelecerá os mecanismos de controle necessários para garantir o cumprimento dos compromissos. Definirá responsáveis e datas para as diversas ações de garantia da qualidade, como revisões técnicas e auditorias. As ações relativas à gestão de configurações serão previstas neste plano, ou em um plano próprio.

Pode-se dizer que todas as áreas chaves do nível 2 do CMM têm como foco o cumprimento dos compromissos. A Tabela 47 resume os pontos em que cada área chave toca o sistema de comprometimento.

Área Chave	Relação com o sistema de comprometimento
Gestão de Requisitos	Definição do que vai ser feito; regras para alteração dos compromissos.
Planejamento dos Projetos	Definição de custos, prazos e recursos; previsão dos riscos.
Supervisão e Acompanhamento dos Projetos	Comparação entre previsto e realizado; monitorização dos riscos; replanejamento, quando necessário.
Garantia da Qualidade	Definição de mecanismos de controle externos ao projeto; acompanhamento do Plano da Qualidade do projeto.
Gestão de Configurações	Manutenção da história dos compromissos; guarda das versões oficiais dos resultados.
Gestão da Subcontratação	Definição e cobrança dos compromissos dos subcontratados.

Tabela 47 - ***Relação das áreas chaves do CMM nível 2 com o sistema de comprometimento***

Página em branco

Requisitos

1 Princípios

1.1 Visão geral

O fluxo de Requisitos reúne as atividades que visam 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 por ERSw).

Projetos de produtos mais complexos geralmente precisam de maior investimento em Engenharia de Requisitos que projetos de produtos mais simples. A Engenharia de Requisitos é também mais complexa no caso de produtos novos. Quando um projeto visa 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 das fases do 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 48 – Artefatos de Requisitos

No Praxis, os requisitos estão contidos nos artefatos enumerados na Tabela 48. 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, 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 sumária 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: Qual 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 destes 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 este fato, todo o 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 Manual, 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 isto, 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 respectivas 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 como parte do desenho, e não devem fazer parte da 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 abaixo 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. As características mais importantes são as seguintes.

- **Correta** - Todo requisito presente na 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 em sua 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 porque existe cada requisito, e quem ou o que o originou. Isto é importante para que se possa avaliar o impacto da mudança daquele requisito, e dirimir dúvidas de interpretação.

- **Rastreabilidade para frente** - deve ser possível localizar quais os resultados do desenvolvimento que serão afetados por cada requisito. Isto é importante para garantir que

os itens de análise, desenho, código e testes cubram todos os requisitos, e para localizar os itens que serão afetados por uma mudança nos requisitos.

2 Atividades

2.1 Visão geral

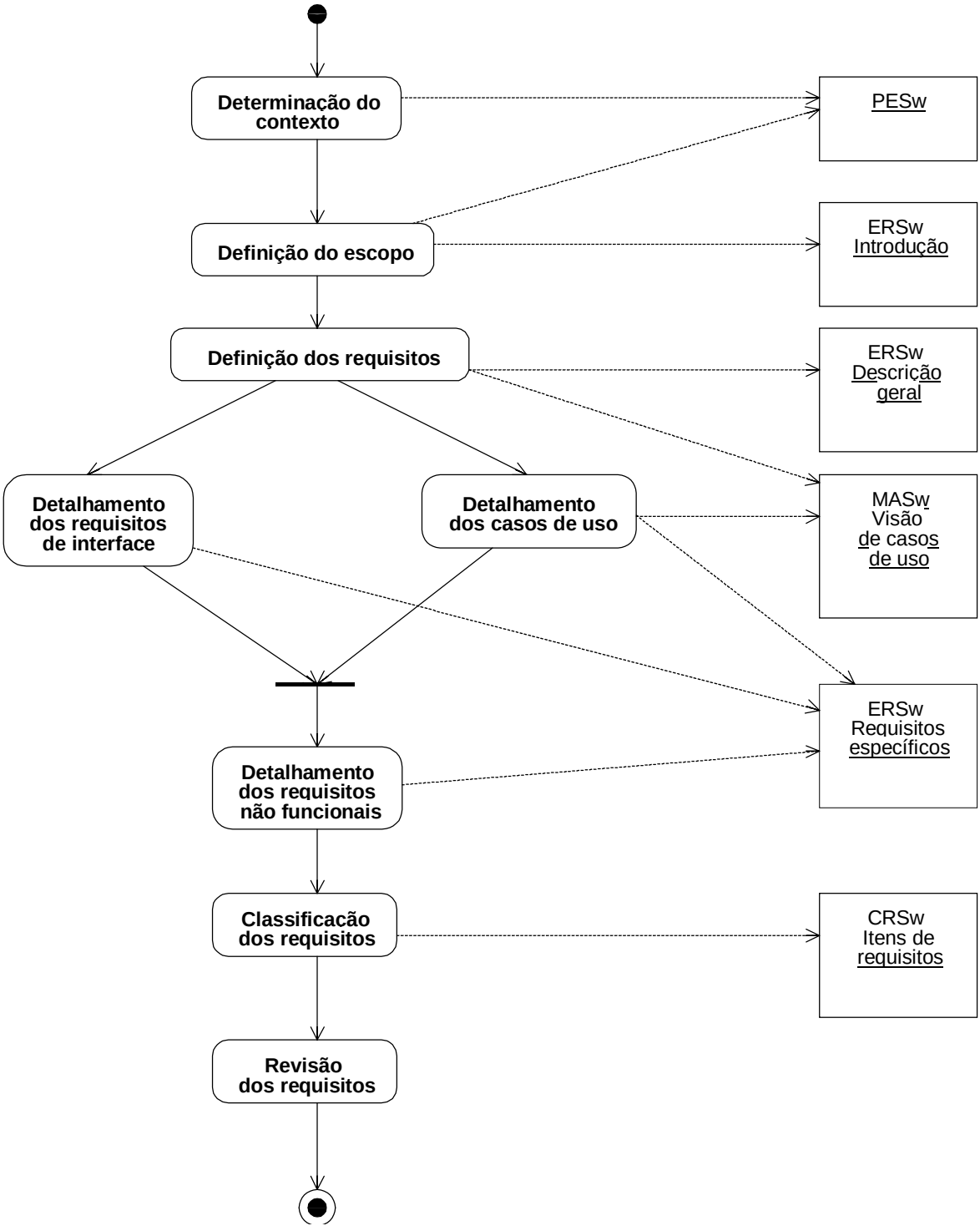

Figura 22 – Atividades e artefatos do fluxo de Requisitos

A Figura 22 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 Software, 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. Estas 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 destas 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 como requisitos. Normalmente, é feito um esboço das interfaces de usuário, levantado através de um protótipo executável ou de estudos em papel. Estes 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 como 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 as estabilidade e 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 aos critérios de qualidade de requisitos, e se a Especificação dos Requisitos do Software está clara e bem entendida por todas as partes interessadas.

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 operará um produto de software. Esta atividade é a mais variada de todas. Dependendo da complexidade e 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]).

Todas 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 22, para descrever o próprio fluxo de Requisitos. A Figura 23 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.

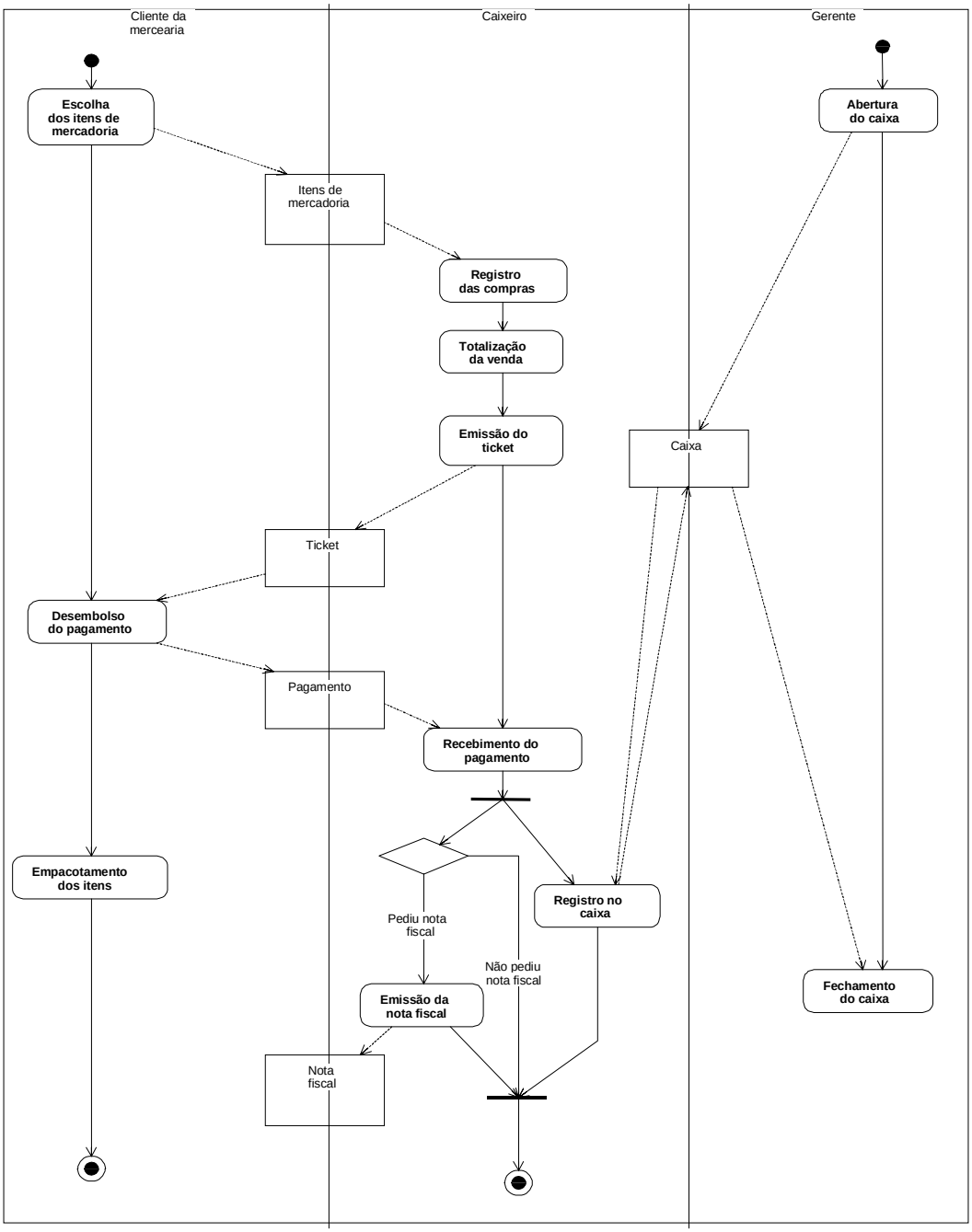

Figura 23 – Exemplo de modelo de processo de negócio

Em relação à Figura 22, a Figura 23 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 desta 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 49). 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 49 - Exemplo de Missão do produto

2.2.2.2 Limites

Deve-se determinar os limites do produto (Tabela 50), ou seja, o que o produto não fará. Isto 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 deste 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 50 - 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 50) 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 isto, 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 51 - 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 consistentes com aquelas usadas em todos os documentos do projeto. Ele incluirá siglas, abreviações e termos relevantes para todas as partes interessadas. Por isto, 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 deste projeto. Naturalmente, é muito cedo para fazer-se qualquer estimativa decentes destes parâmetros (embora muitos clientes esperem isto). Durante a fase de Concepção, entretanto, é importante determinar pelo menos o prazo e custo da fase de Elaboração. Para isto, 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 desta 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.

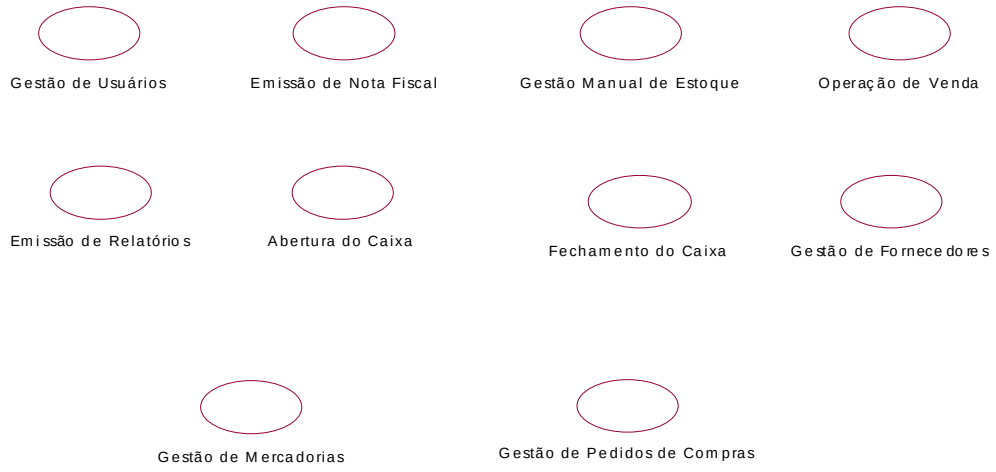

Figura 24 - Exemplos de casos de uso

A Figura 24 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 52). 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 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 52 - Exemplo de lista de casos de uso

2.2.3.3 Atores

Os papéis dos usuários do produto são modelados através dos atores (Figura 25). 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 25 - 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 responsabilidade do respectivo papel (Tabela 53). Deve-se também identificar as características mais importantes do respectivo grupo de usuários. Exemplos de características importantes são cargo ou função, permissão de acesso, freqüência de uso, nível educacional, proficiência no processo de negócio e proficiência em informática (Tabela 54).

Número de ordem	Ator	Definição
1	Caixeiro	Funcionário operador comercial de caixa.
2	Gerente	Funcionário responsável pela abertura e fechamento do caixa, alem do cadastro de usuários.
3	Gestor de Compras	Funcionário responsável por: • 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 inventario do estoque da mercearia e por manter estes níveis coerentes com as bases de dados do Merci.

Tabela 53 - Exemplo de descrição de atores

Número de ordem	Atores	Permissão de acesso	Freqüência 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 54 - 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 26). 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; convenciona-se, neste 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 27).

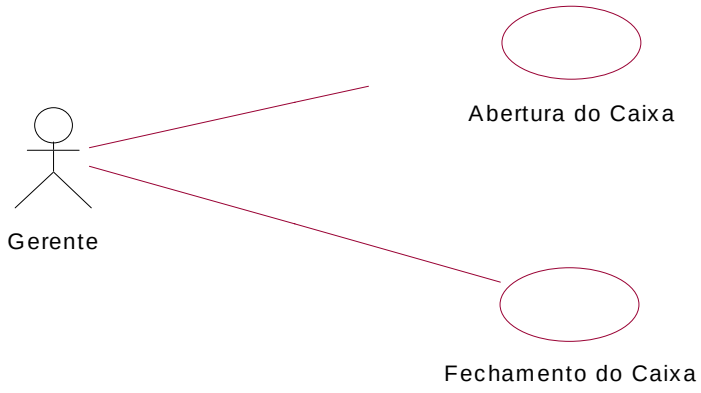

Figura 26 - Exemplo de casos de uso de um ator

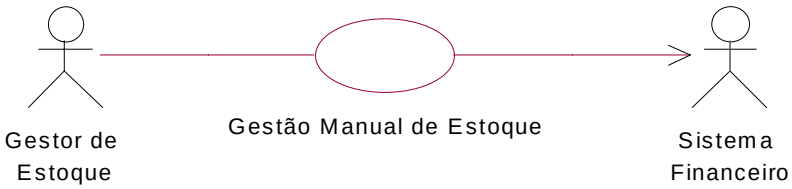

Figura 27 - Relacionamento entre caso de uso com mais de um ator

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 28, 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 29).

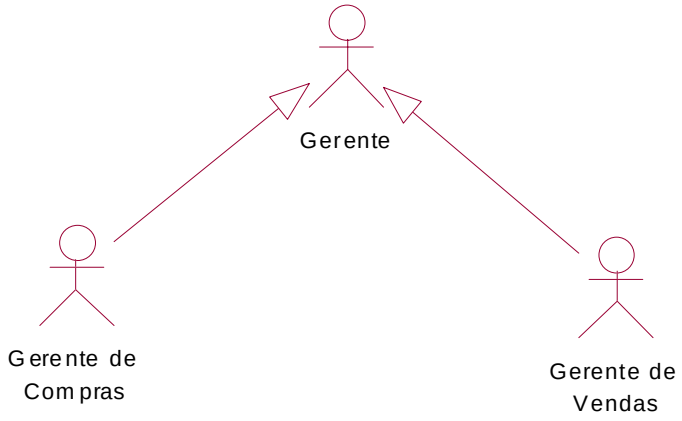

Figura 28 – Herança entre atores

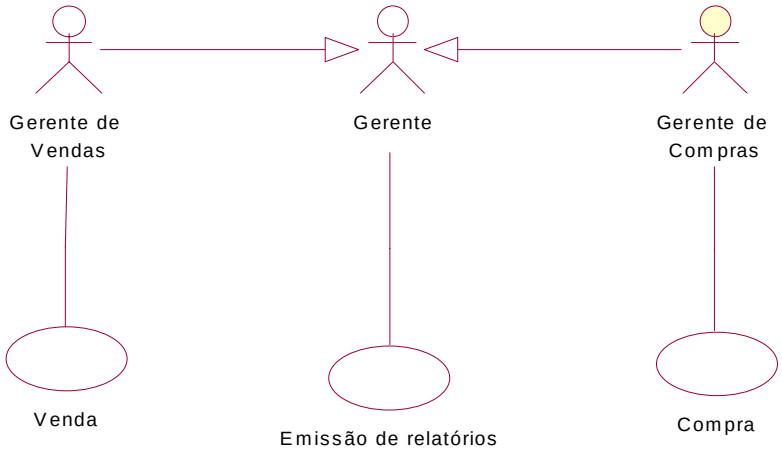

Figura 29 – Uso de atores genéricos

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.

Neste diagrama, os usuários, sistemas externos e outros componentes de um sistema maior são representados por atores, enquanto que 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 30).

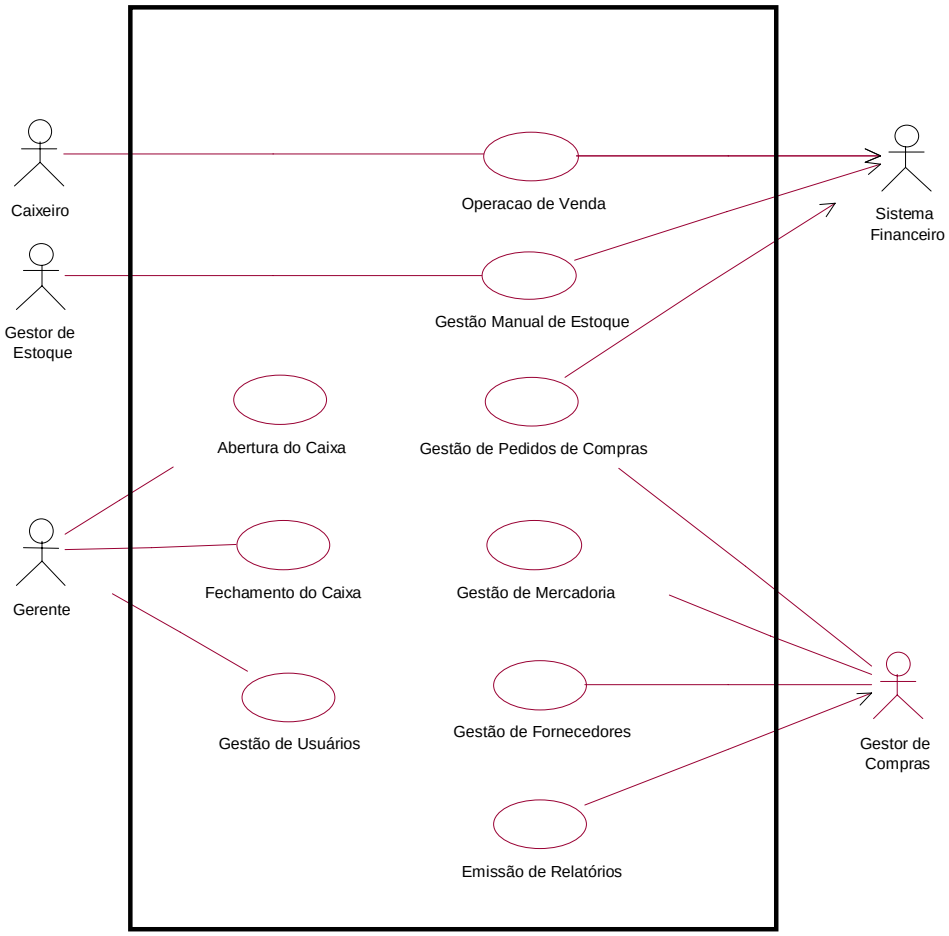

Figura 30 - Diagrama de contexto

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 nas quais os demais requisitos se baseiam. Casos estas 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 55 – 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 56 – 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 57 - 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 estes 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 eqüivalentes. 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 desta no sentido de reduzir erros do usuário.

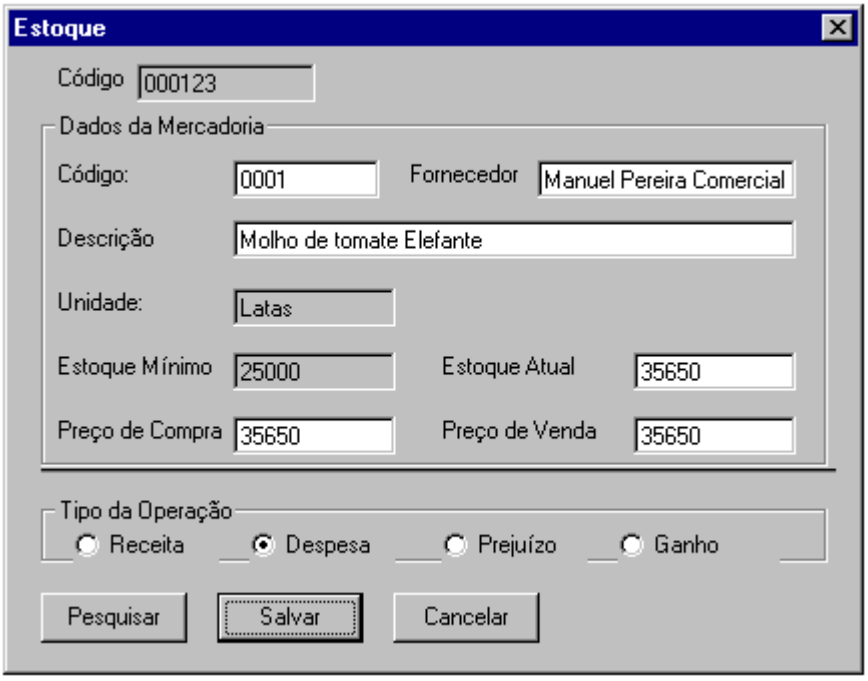

Figura 31 – Exemplo de leiaute de interface

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 destas 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 entendimento deles. 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. Esta notação pode ser usada para representar fluxos alternativos ou anormais complexos. O caso de uso "estendido" é referenciado nas pré-condições do caso de uso "extensor".

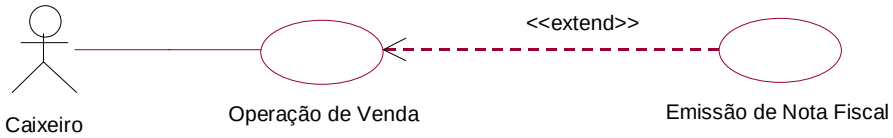

Figura 32 - Exemplo de caso de uso de extensão

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.

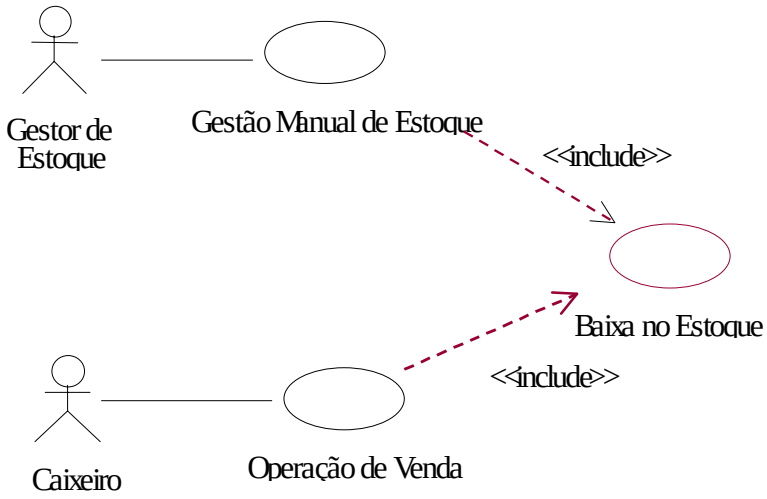

Figura 33 - Exemplo de caso de uso de inclusão

O caso de uso A inclui o caso de uso B quando B representa uma atividade complexa, comum a vários casos de uso. Esta 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 “incluidor”.

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 como uma espécie de macro, cuja única finalidade é abreviar as descrições dos fluxos dos casos de uso incluidores.

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 58). 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 seqüência de passos (Tabela 59). 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. Isto 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 freqüência (Tabela 60). Os fluxos alternativos descrevem condições pouco habituais ou exceções (Tabela 61). Os subfluxos devem ser detalhados dentro do fluxo de Requisitos; por outro lado, apenas os fluxos alternativos mais importantes são detalhados como parte deste 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 58 - 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 59 - 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 60 - Exemplo de subfluxo: registro de item em Operação de Venda

Se o Gestor de Compras solicitar:
o Merci imprime o pedido de compra.

Tabela 61 - 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.

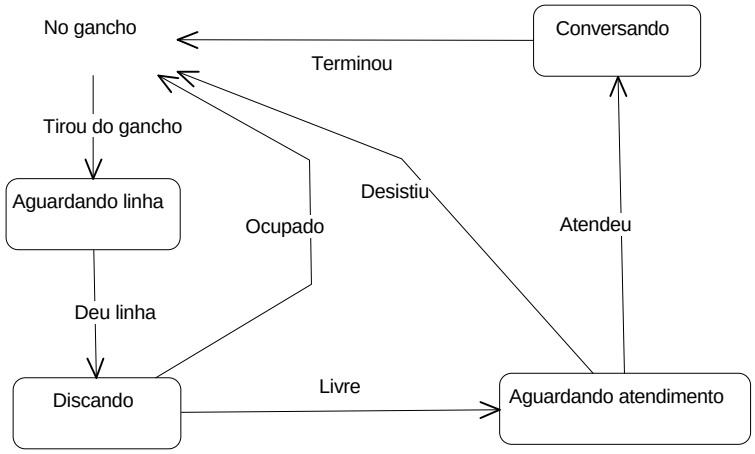

Figura 34 - Exemplo de diagrama de estado: fluxo de uma ligação telefônica

Para casos de uso complexos, pode-se descrever seu comportamento de maneira mais formal, através de um diagrama de estados ou diagrama de atividade da UML. Os detalhamentos dos estados e transições destes diagramas deve 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 são raramente 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. Nesta 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 devam 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 62 - 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. Neste 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 arquitetura do produto.

2.2.6.3 Requisitos de dados persistentes

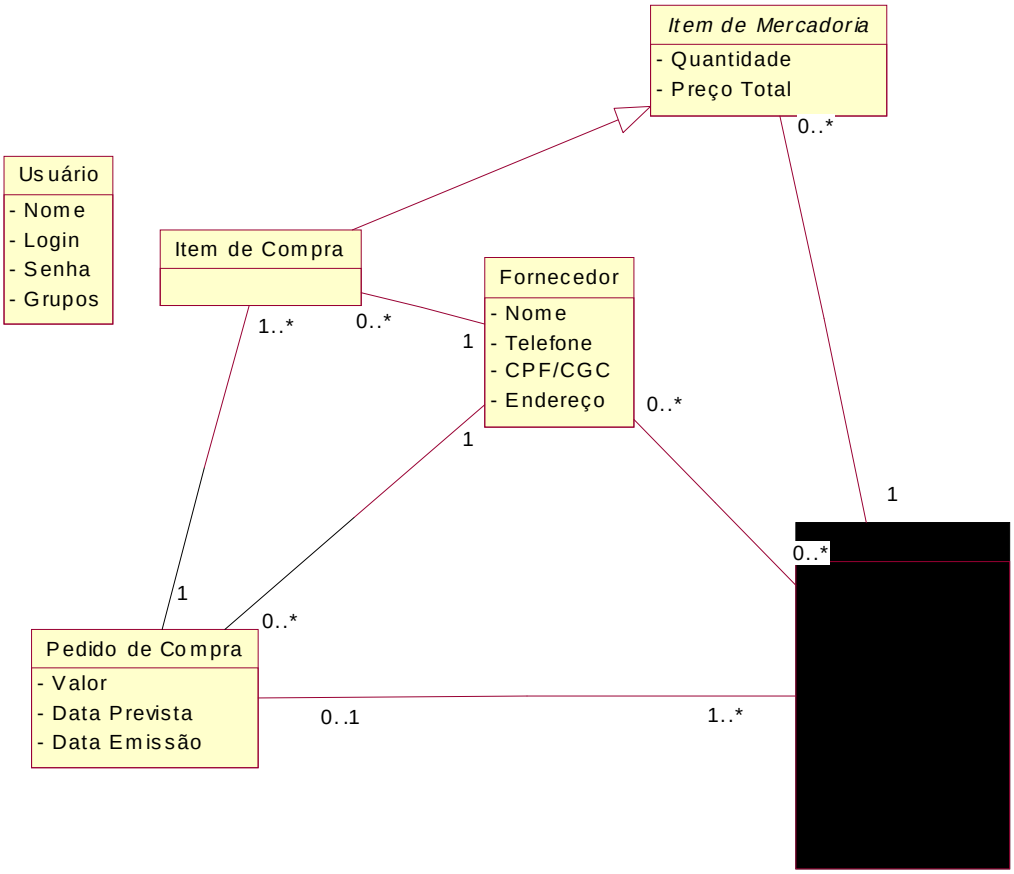

Figura 35 - Exemplo de diagrama de dados persistentes

Deve-se levantar as estruturas lógicas dos dados persistentes que sejam usadas pelo produto (Figura 35). 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 este 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 63 - Exemplos de restrição ao desenho (expansibilidade)

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

Tabela 64 - 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. Esta norma inclui atributos de funcionalidade (Tabela 67), confiabilidade (Tabela 68), usabilidade (Tabela 69), manutenibilidade (Tabela 70) e portabilidade (Tabela 71), 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 65 - 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 66 - 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 67 - Propostas da ISO-9126 para subcaracterísticas de funcionalidade

Subcaracterística	Definição	Possíveis métricas
Maturidade	Freqüência 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 68 - 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 69 - 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.	Percentagem 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.

Tabela 70 - 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 71 - 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, erão 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 36 mostra um exemplo de organização do cadastro inicial. Os campos desta 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 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 36 – 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. Nesta 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 Manual, dentro do padrão para Especificação de Requisitos.

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 que aumentam a eficácia das atividades de levantamento de requisitos, e a qualidade dos requisitos resultantes. A aplicação destas 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, estes 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 esta 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 incremental[5] e iterativa[6] de um aplicativo complexo. Ele não faz parte da Engenharia

[5] Por partes.

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 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.

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 como 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** – linguagem de quarta geração 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 decida 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 concluiu-se 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 tal desde o começo; pode ser o caso, por exemplo, em relação às interfaces de usuário.

---

[6] Que pode ser revista várias vezes, em função da avaliação dos Usuários.

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 deste 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 codificadores[7].

Segundo [Jones94], o uso de protótipos descartáveis oferece um retorno de investimento de um fator de 200 %, em um ano, e de 1000 %, em quatro anos.

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 onde 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;

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

- produzir um primeiro esboço da interface 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.

Em seguida 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 proceso.

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 tipicamente de 1 a 10 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 disto. 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 a focalização das discussões, que deve dominar técnicas de condução de reunião, e ser treinado em JAD, em particular;
- os **patrocinadores executivos**, 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 deve ser a de fornecer informação sobre a viabilidade das idéias levantadas;
- o **relator**, participante da equipe do projeto, encarregado de redigir as atas de reunião.

Normalmente, os participantes do lado do cliente deverão ser usuários efetivos do produto, e não necessariamente seus gerentes. 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, deve-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 destes resultados, para os responsáveis pelo projeto, juntamente com um balanço desta etapa. Isto 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 acima, 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 leiaute 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 acima 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. Isto 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. Este 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. Estas 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 disto, 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 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 impossíveis de prazo;
  - 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 estes 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. Freqüentemente os requisitos levantados:

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

A Figura 37 ilustra a diferença entre requisitos reais e levantados.

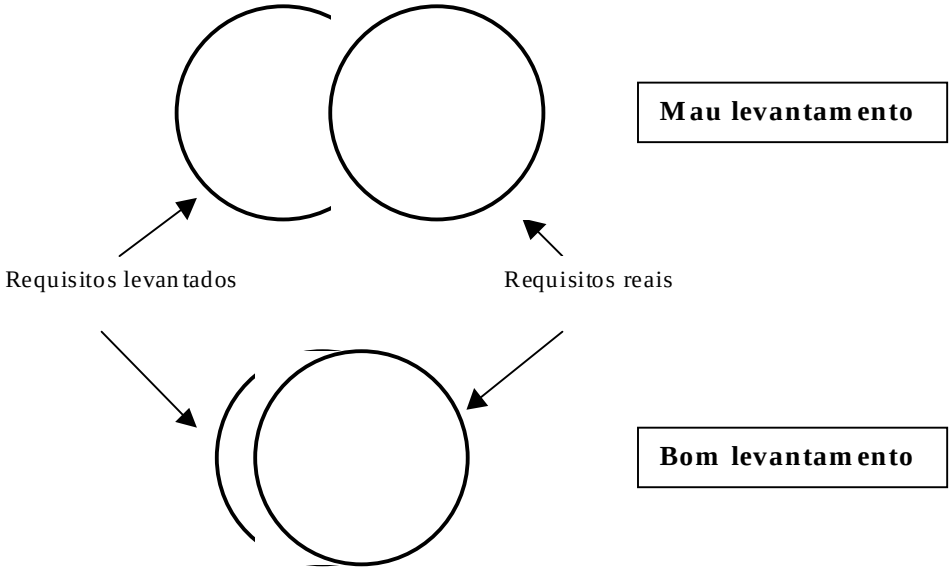

Figura 37 – Requisitos reais versus levantados

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 freqüentes e visíveis para o cliente.

O desenvolvimento incremental, baseado em um desenho arquitetônico robusto, seguido de liberações bem planejadas, atende a estes requisitos. As próprias liberações se constituem em resultados intermediários concretos, muito mais eficazes que relatórios de progresso, quanto ao convencimento dos clientes.

Análise

1 Princípios

1.1 Objetivos

O fluxo da Análise visa os seguintes objetivos:

- verificar a qualidade dos requisitos obtidos através do fluxo de Requisitos;
- detalhar estes 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 Analise 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 da 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 que 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. Isto 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. Isto 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.

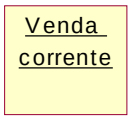

Figura 38 - Objeto sem classe determinada

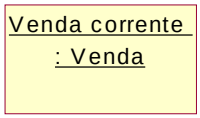

Figura 39 - Objeto com indicação da respectiva classe

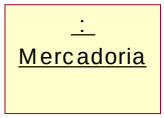

Figura 40 - Objeto anônimo

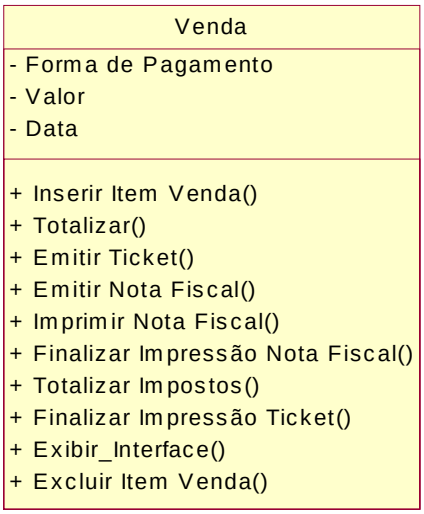

Figura 41 - Representação de classe

Na UML, a classe é representa 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. É opcional 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 este nível de detalhe nos Modelos de Análise.

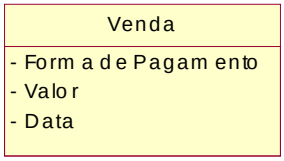

Figura 42 - Representação de classe com supressão das operações

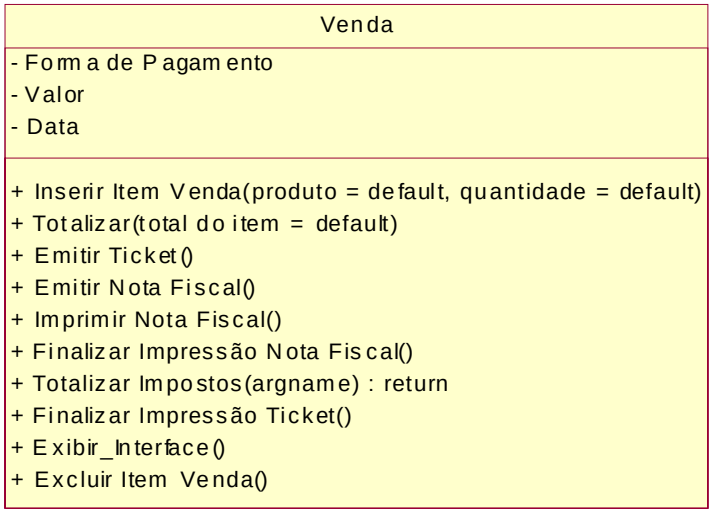

Figura 43 - Representação de classe com detalhamento das assinaturas das operações

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 abaixo 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].

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 estados
Diagramas de classes	Modelos E-R
Diagramas de interação	DFD, DSSD, JSD, SADT
Especificações de classes e relacionamentos	Dicionários de dados

Tabela 72 – Notações estruturadas correspondentes à UML

2 Atividades

2.1 Visão geral

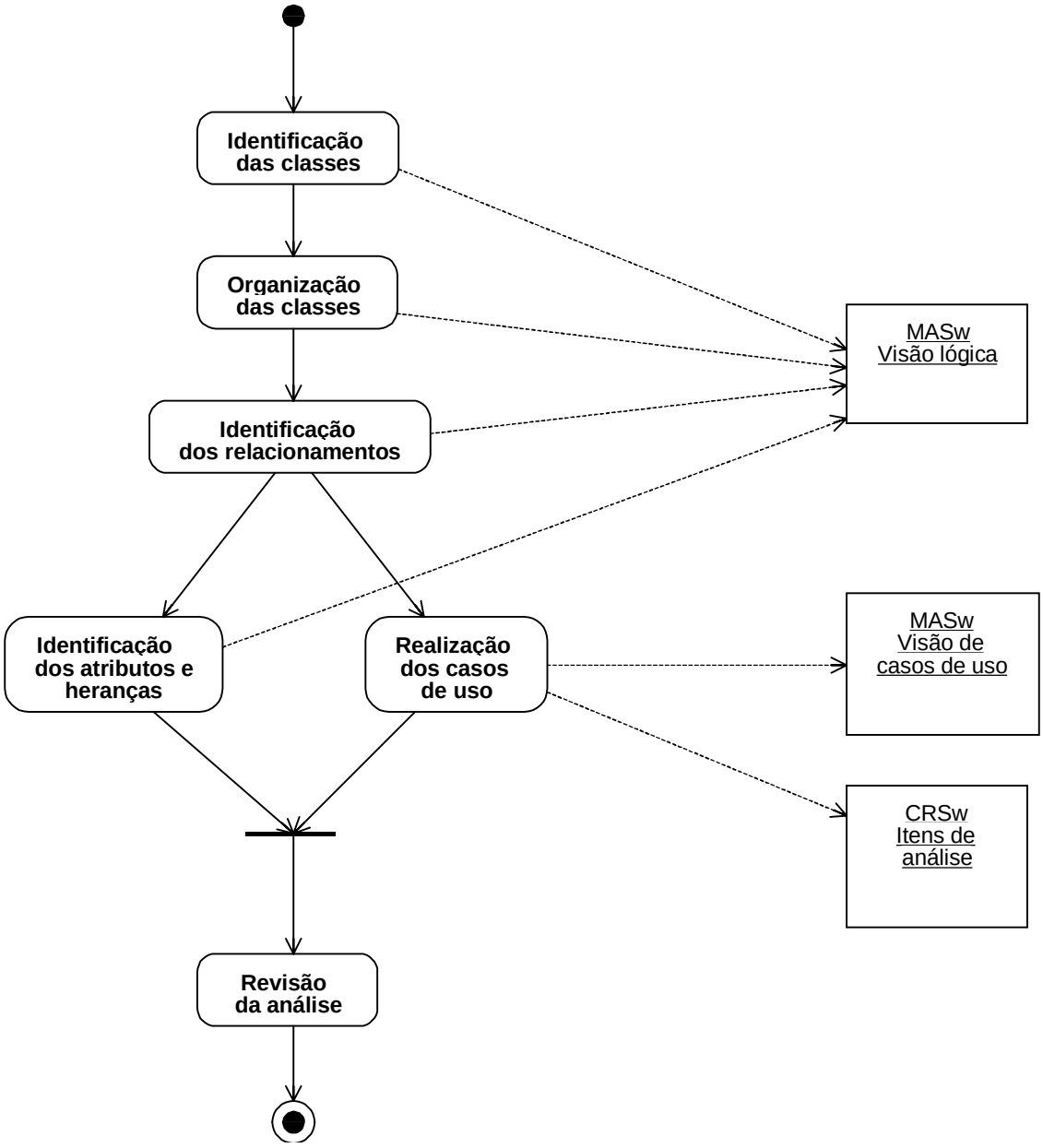

Figura 44 – Atividades e artefatos do fluxo de Análise

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 atribui-lhes 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 estas 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. Esta 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. Isto ajuda a localizar imprecisões e outros tipos de problemas nos fluxos, e permite definir as operações das classes de análise. Esta 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 esta 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

Nesta 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. Devem ser observados os seguintes detalhes, na pesquisa dos substantivos:

- 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**. Estes cartões são usados pelos participantes da sessão de modelagem (por exemplo, em um JAD) para lançar propostas de classes à medida em que os 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 45 - 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 46 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.

O *caixeiro* faz a abertura da venda.

O *caixeiro* registra os itens vendidos, informando a identificação e a quantidade do item.

O **Merci** totaliza a venda para o cliente da mercearia.

O *caixeiro* encerra a venda.

O **Merci** emite o ticket de caixa para o cliente da mercearia.

O *caixeiro* registra a forma de pagamento.

O **Merci** faz a baixa no estoque das mercadorias vendidas.

Figura 46 - 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 73. As classes candidatas são representadas na Figura 47.

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 73 - Exemplo de análise de classes candidatas

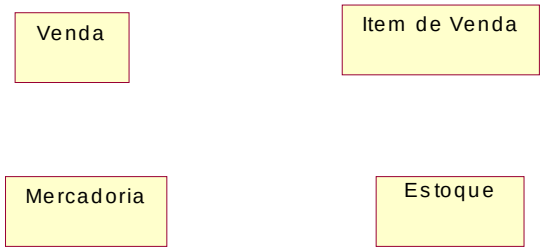

Figura 47 - Classes candidatas encontradas

Quando se tem um modelo de Entidades - Relacionamentos disponível, as entidades deste 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. Deve-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.

Descrição:
armazena a informação relativa a um item de uma venda.
Responsabilidades:
comandar baixa no estoque;
calcular impostos;
imprimir linha de ticket e da nota fiscal.
Colaborações:
Venda, Mercadoria
Regras e restrições:
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.
Exemplos:
seis cervejas Rottenbeer em lata;
duas caixas de pregos tamanho 2.

Figura 48 - 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 74 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 74 - Sintomas de problemas na documentação das classes

2.2.1.3 Término da atividade

Ao fim desta 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 desta 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.

Seguem-se um exemplo das classes iniciais que poderiam ser encontradas na análise do sistema de informatização de mercearia.

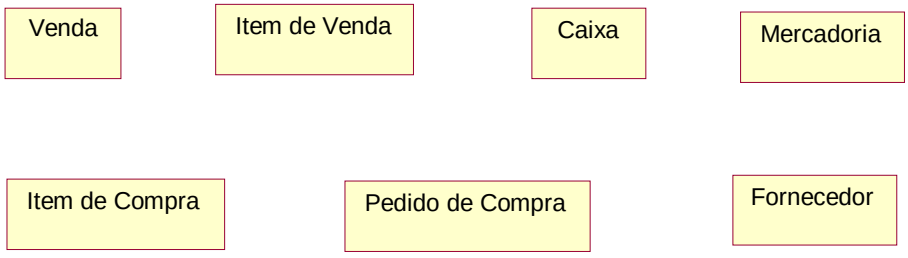

Figura 49 – Classes chaves de um sistema de informatização de mercearia

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. Estes 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 50, os pacotes lógicos Compras, Vendas e Administração são usados para agrupar classes especializadas em relação a estes aspectos do funcionamento da mercearia.

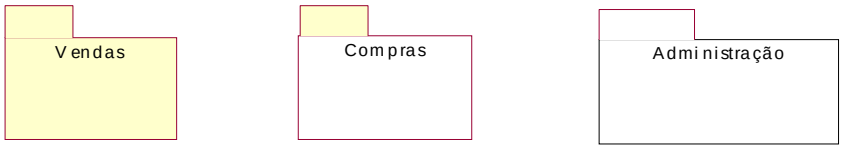

Figura 50 - Exemplos de pacotes lógicos

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 << >>).

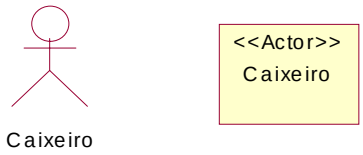

Figura 51 - Representações alternativas do estereótipo "Ator"

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.

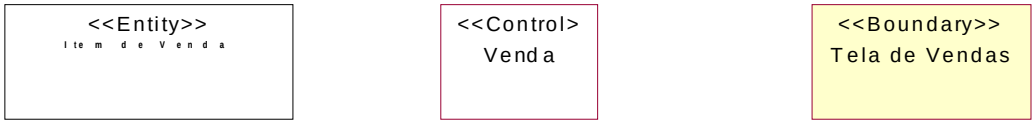

Figura 52 - Exemplo de classes de análise com estereótipos textuais

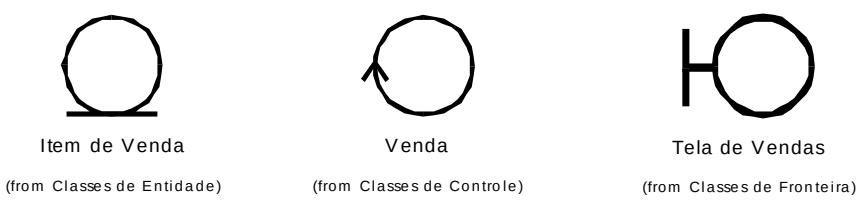

Figura 53 - Exemplo de classes de análise com estereótipos icônicos

Para simplificar, usaremos geralmente a notação estereótipos textuais para as classes de análise[8]. Geralmente, a maioria das classes localizadas através dos métodos descritos para a atividade "Identificação das classes" corresponde a classes <<Entidade>>. Por isto, as classes de <<Fronteira>> e <<Controle>> serão aqui tratadas com maior detalhe.

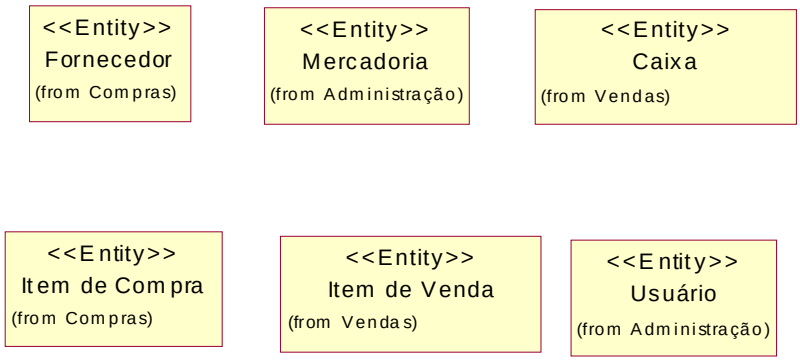

Figura 54 - Exemplos de classes de entidade

2.2.2.2 Classes de fronteira

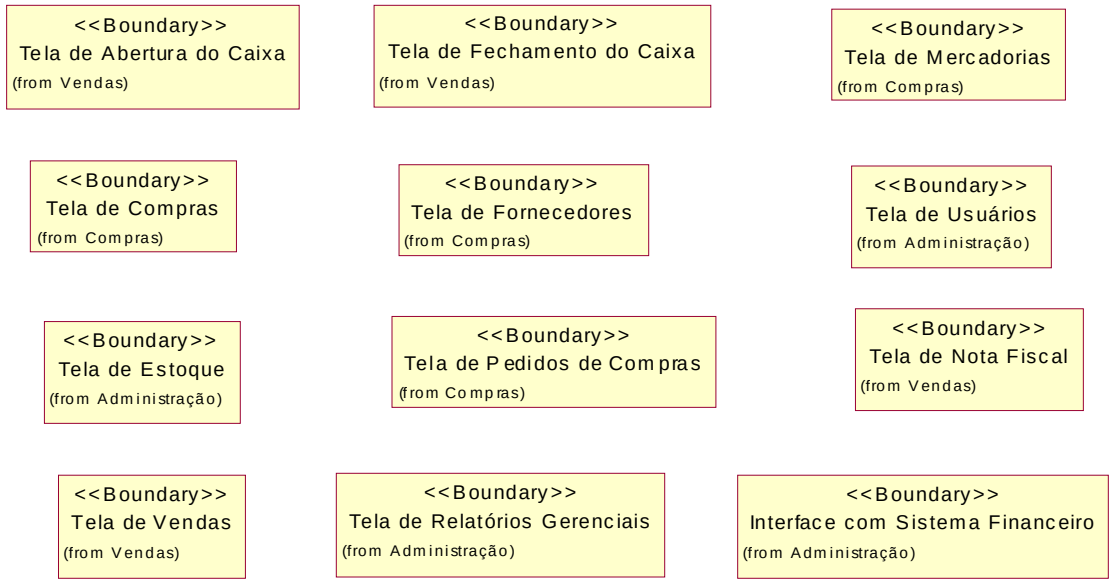

Figura 55 – Exemplo de classes de fronteira

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

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 este 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.

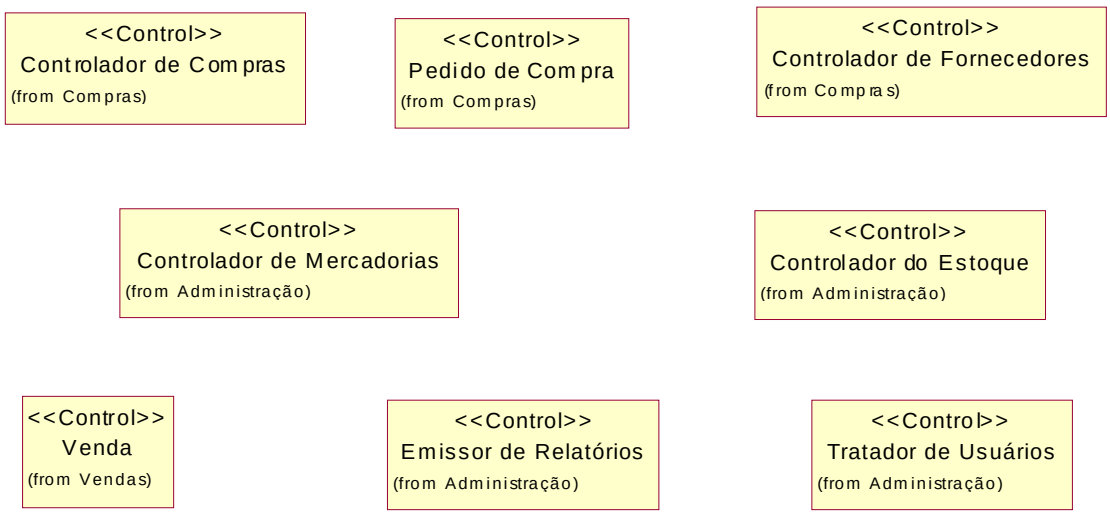

Figura 56 – Exemplo de classes de controle

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 do domínio apenas aspectos que são inerentes a cada conceito modelado. Com isto, as classes do domínio 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, este 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 desta 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 desta 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

Nesta 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. Isto 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 esta responsabilidade.

Figura 57 - 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 destes.

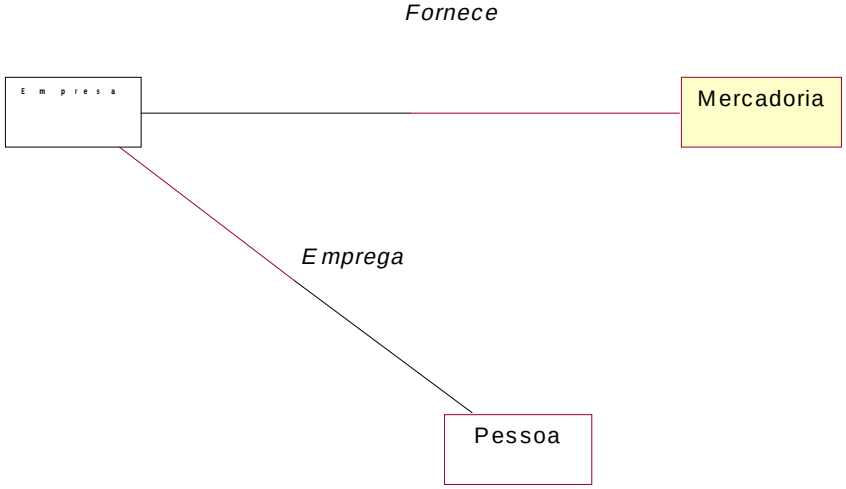

Figura 58 - Relacionamento com nome

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 bem descreva a semântica do relacionamento. Pode-se também usar um verbo, deste que estejam claros qual classe é sujeito e qual classe é objeto deste 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. 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.

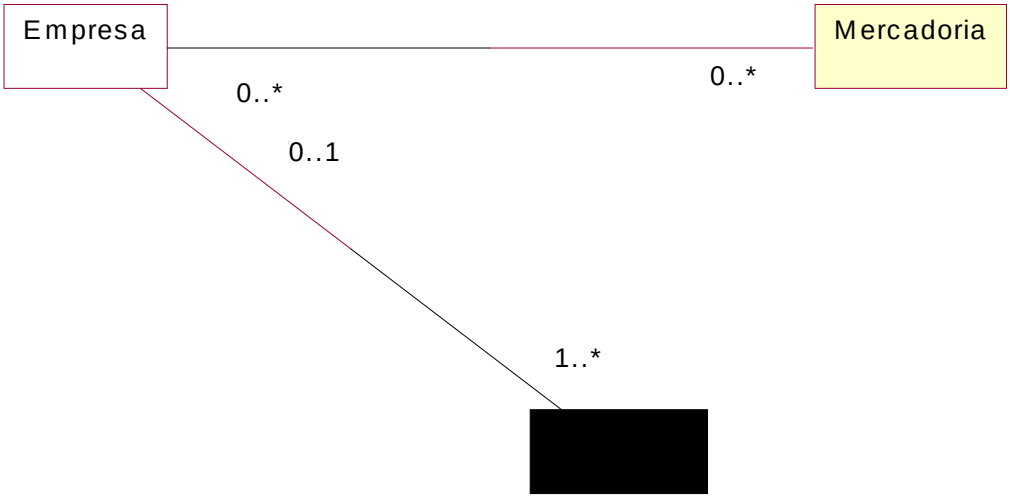

Figura 59 - Relacionamentos com multiplicidades

A multiplicidade de um participantes 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.

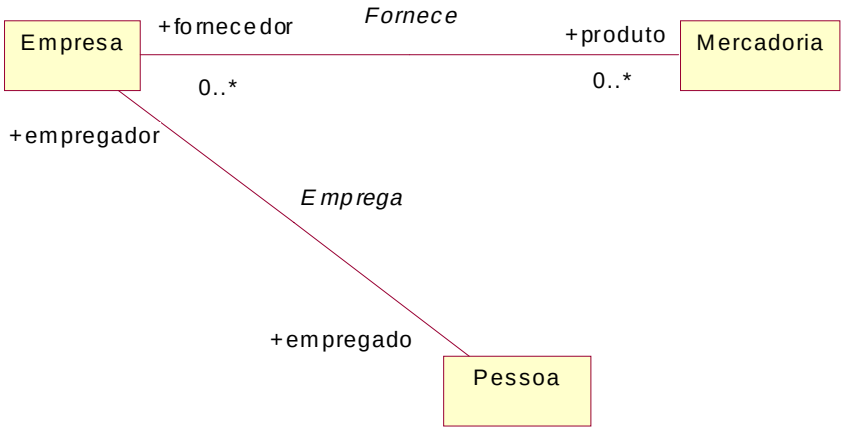

Figura 60 - Relacionamentos com denominação dos participantes

Na Figura 60 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 desta 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.

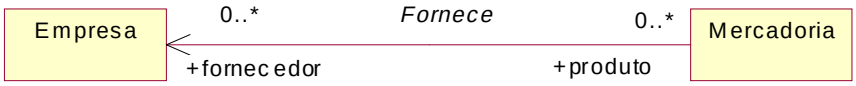

Figura 61 - Relacionamento com restrição de navegabilidade

Por exemplo, a Figura 61 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.

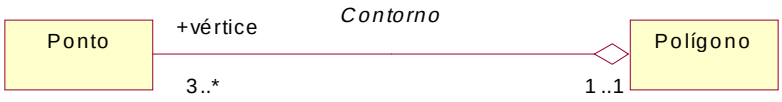

Figura 62 - Relacionamento de agregação

Um tipo mais forte de relacionamento todo-parte é o **relacionamento de composição**. Neste caso, os objetos da classe parte não têm existência independente da classe todo. A Figura 63 indica que o "centro" de um "Círculo" não tem existência independente deste, enquanto que a Figura 62 indica que cada "ponto" existe independentemente do "Polígono" ao qual serve de "vértice".

Figura 63 - Relacionamento de composição

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

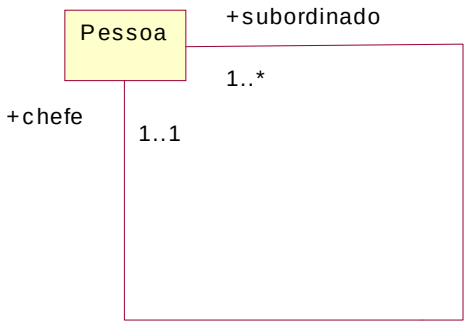

Figura 64 - Associação reflexiva

Os relacionamentos podem ser de natureza complexa. Em certos casos, um relacionamento é melhor 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.

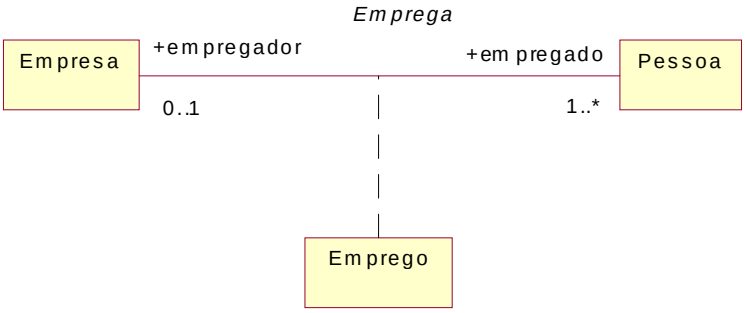

Figura 65 - Classe de associação

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 66, 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).

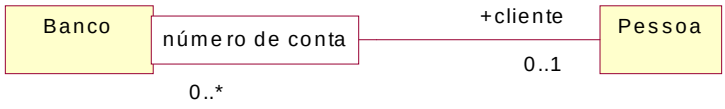

Figura 66 - Relacionamento com qualificador

2.2.3.5 Término da atividade

Ao fim desta 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.

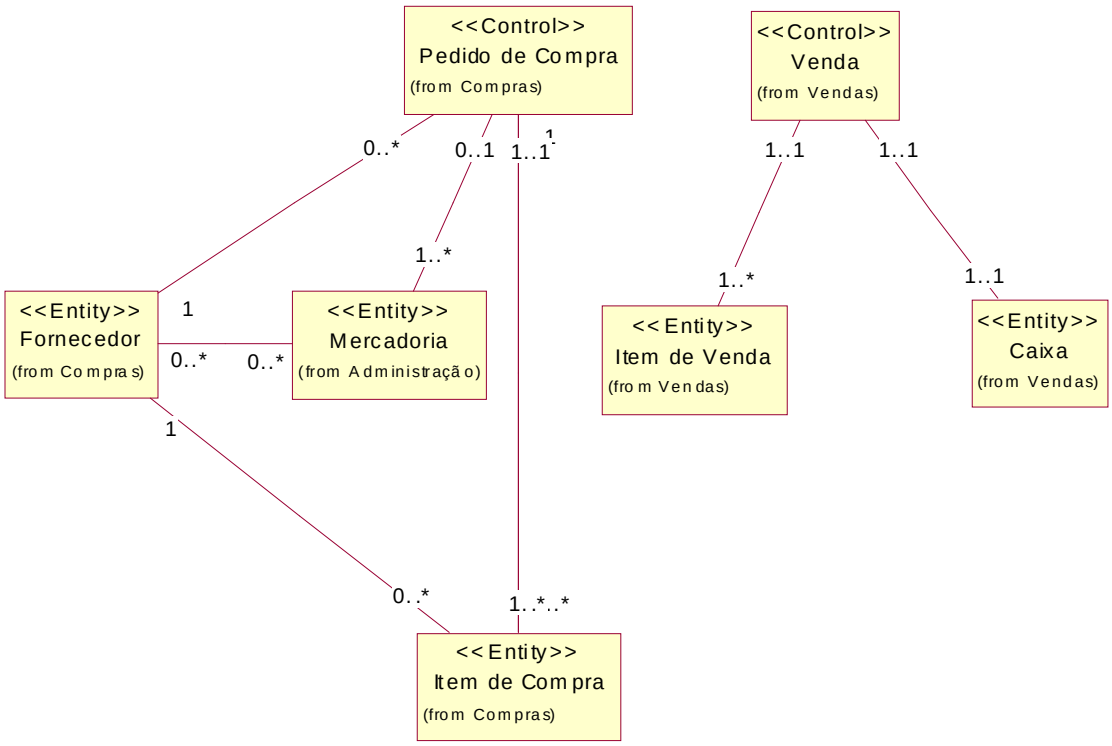

Figura 67 - Exemplo de diagrama de classes com relacionamentos

São os seguintes os resultados desta 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

Nesta atividade devem ser definidas as operações de cada classe. As operações devem ser suficiente para fazer a **realização** dos casos de uso, isto é, a tradução dos fluxos destes em termos de interações entre objetos das classes achadas. Estas 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.

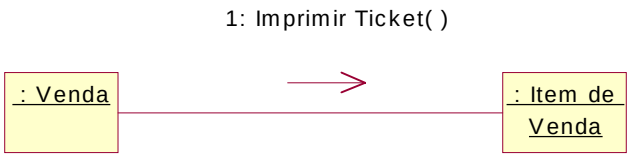

Figura 68 - Exemplo de mensagem

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 estes 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 melhor 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.

O *Gestor de Compras* seleciona a mercadoria.

O **Merci** verifica se existe algum pedido pendente que contenha esta mercadoria.

Se não houver pedido pendente contendo a mercadoria a ser excluída:

O **Merci** desvincula a mercadoria dos fornecedores (os fornecedores não mais fornecerão a mercadoria que esta sendo excluída).

O **Merci** faz a remoção da mercadoria.

Se houver pedido pendente contendo a mercadoria a ser excluída

O **Merci** emite uma mensagem de erro.

Tabela 75 - 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 69 pode ser descrita através dos diferentes roteiros (Figura 70 e Figura 71). Para simplificar o exemplo, os diagramas não usam classes de fronteira.

O *Caixeiro* registra itens de mercadoria.

O **Merci** totaliza venda.

O *Caixeiro* registra modo de venda.

Se venda a prazo

O *Caixeiro* insere venda em contas a receber.

Senão

O *Caixeiro* registra pagamento.

Figura 69 – Exemplo de fluxo com lógica de seleção: Operação de Venda

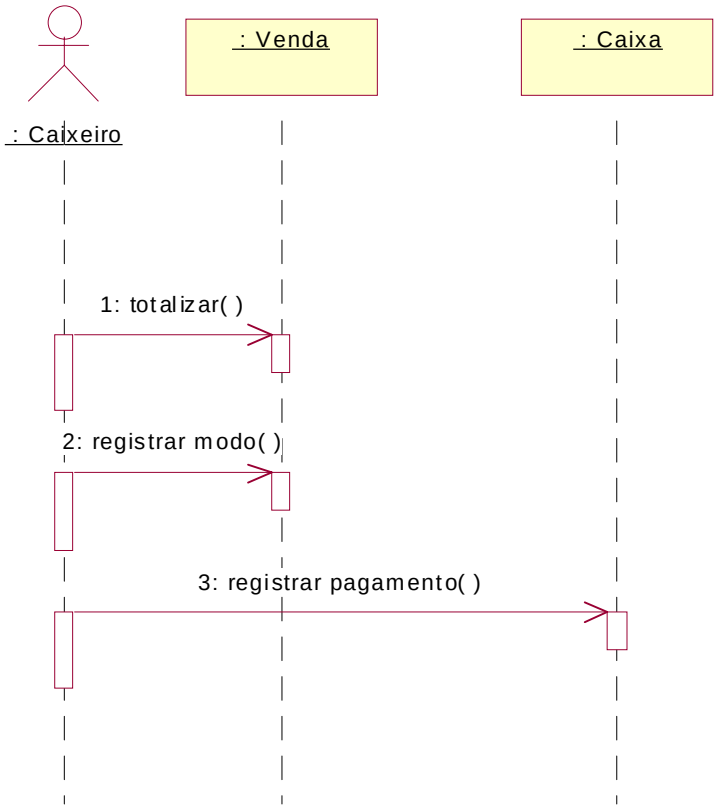

Figura 70 – Exemplo de roteiro alternativo: Operação de Venda à Vista

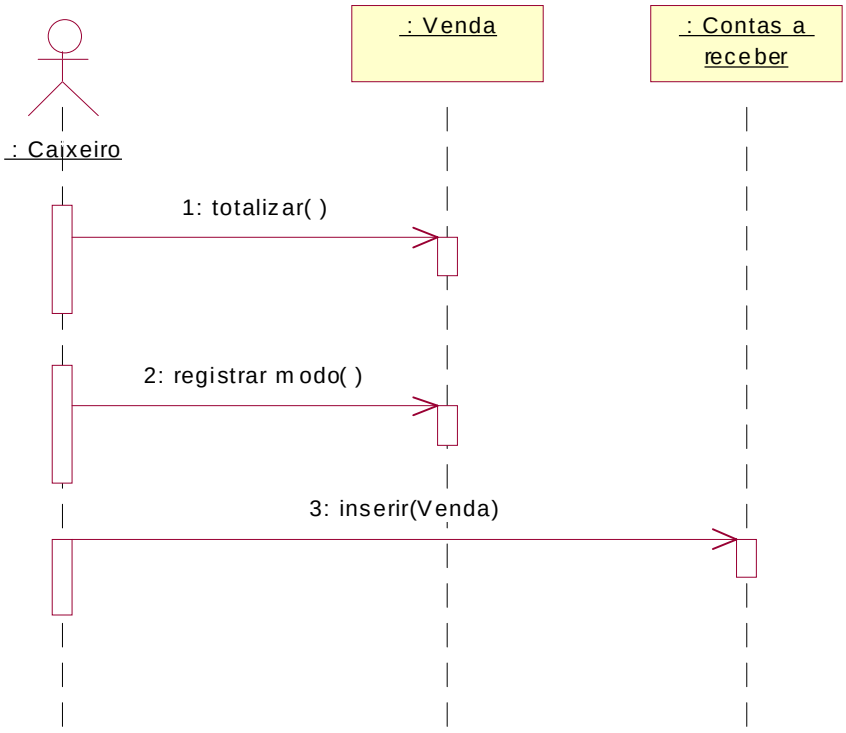

Figura 71 – Exemplo de roteiro alternativo: Operação de Venda a Prazo

Por outro lado, só há necessidade de desenhar diagramas de seqüência 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.

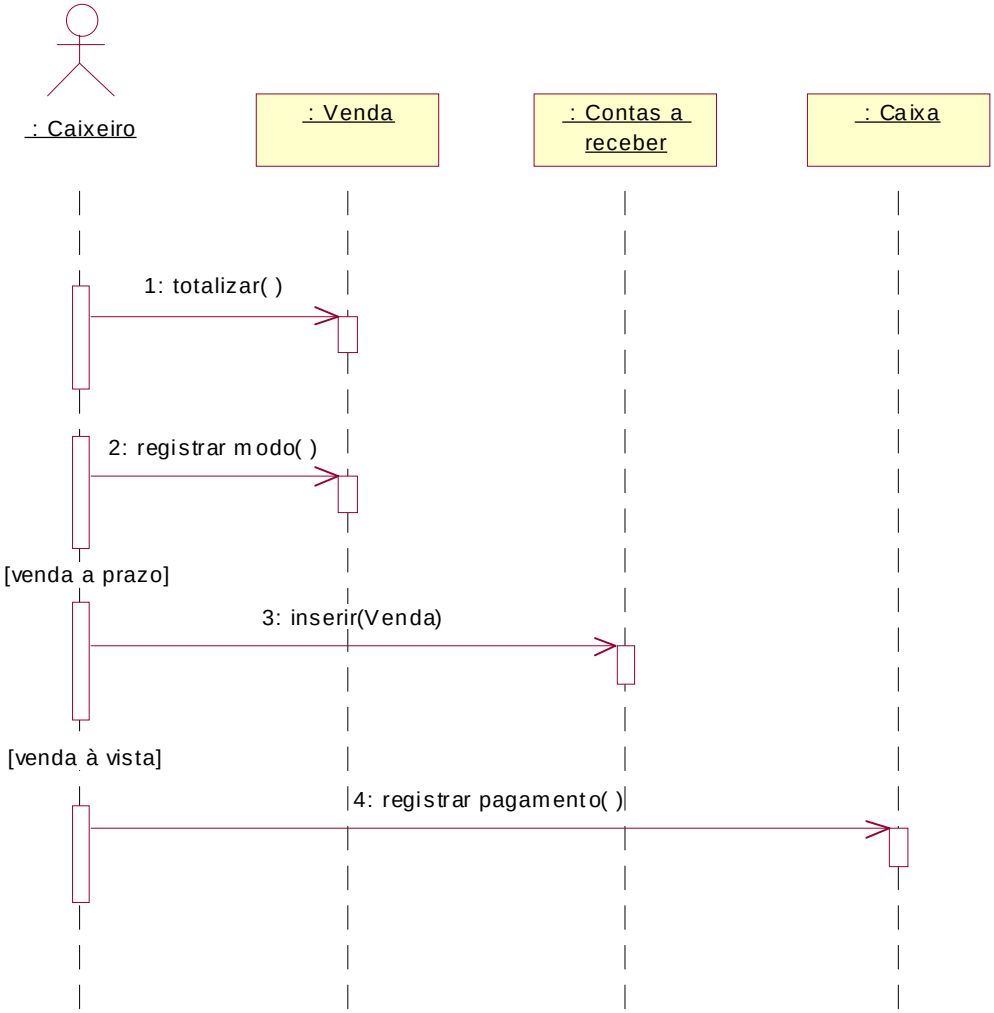

Figura 72 – Representação de lógica por meio de restrições

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 seqüência 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.

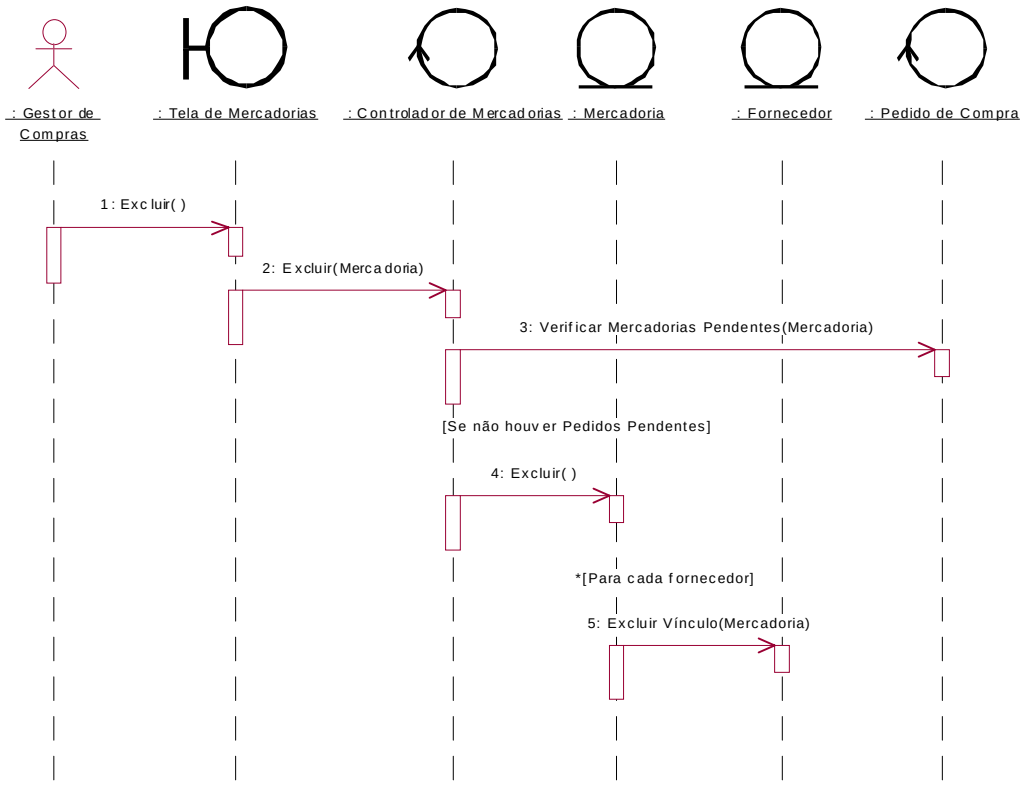

Figura 73 - Realização de fluxo através de diagrama de seqüência

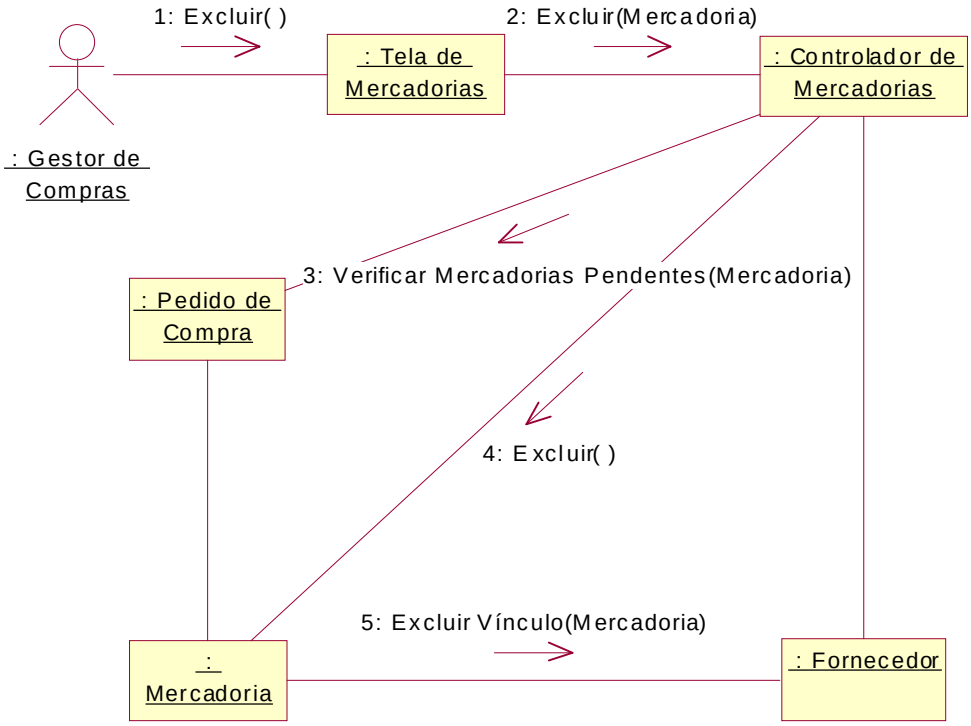

Figura 74 - Realização de fluxo através de diagrama de colaboração

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 estas 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 75). Outro sintoma de relacionamento é a presença de objetos nas assinaturas das operações. A Figura 76 indica um relacionamento entre “Venda” e “Contas a receber”.

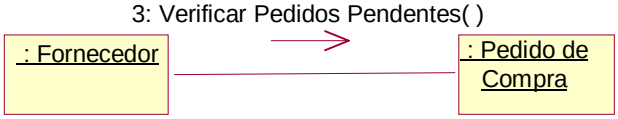

Figura 75 – Existência de mensagens entre objetos

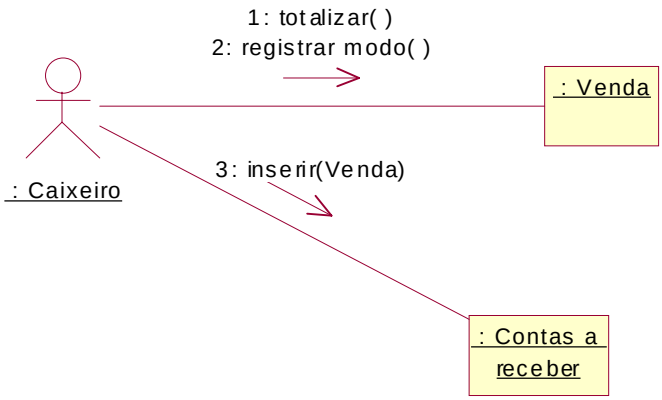

Figura 76 – Presença de objeto em assinatura de operação

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. Isto geralmente indica a necessidade de partir as operações em outras mais simples. Deve-se evitar argumentos que funcionem apenas como chaves de entrada, isto é, de condição para uma seleção interna à operação. Isto 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																								x								
3	Tela de Estoque	Cl. fronteira			x																									x						
4	Tela de Fechamento do Caixa	Cl. fronteira				x																														
5	Tela de Fornecedores	Cl. fronteira					x																			x										
6	Tela de Mercadorias	Cl. fronteira						x																			x									
7	Tela de Nota Fiscal	Cl. fronteira							x								x																			
8	Tela de Pedidos de Compras	Cl. fronteira								x																		x								
9	Tela de Relatórios	Cl. fronteira									x																									
10	Tela de Usuários	Cl. fronteira										x																	x							
11	Tela de Vendas	Cl. fronteira											x																		x					
12	Sistema Financeiro	Cl. fronteira																			x									x	x					
13	Caixa	Cl. entidade																				x			x											
14	Fornecedor	Cl. entidade													x	x		x	x							x	x	x								
15	Item de Compra	Cl. entidade																x	x									x		x						
16	Item de Mercadoria	Cl. entidade												x				x	x									x		x	x	x				
17	Item de Venda	Cl. entidade															x			x											x	x				
18	Mercadoria	Cl. entidade												x		x		x	x								x	x		x						
19	Usuário	Cl. entidade																											x							
20	Nota Fiscal	Cl. entidade																					x											x		
21	Pedido de Compra	Cl. controle													x	x		x	x							x	x	x								
22	Venda	Cl. controle															x	x		x	x	x										x	x	x		
23	Controlador de Mercadorias	Cl. controle														x											x									
24	Controlador de Estoque	Cl. controle												x																x						
25	Controlador de Compras	Cl. controle																																		
26	Controlador de Fornecedores	Cl. controle													x											x										
27	Emissor de Relatórios	Cl. controle																																		
28	Tratador de Usuários	Cl. controle																											x							

Figura 77 – Exemplo de rastreamento entre itens de análise e de requisitos

A realização dos casos de uso permite determinar quais classes são dependentes de quais requisitos. Esta informação é lançada na Cadastro dos Requisitos do Software, como mostra a Figura 77. Os números dos itens de requisitos correspondem à Figura 36. 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 desta atividade, devem estar:

- determinadas as operações requeridas para realizar os casos de uso através de diagramas de interação;
- determinados os argumentos de algumas destas operações, quando necessário para o entendimento do modelo;
- atribuídas as operações às respectivas classes.

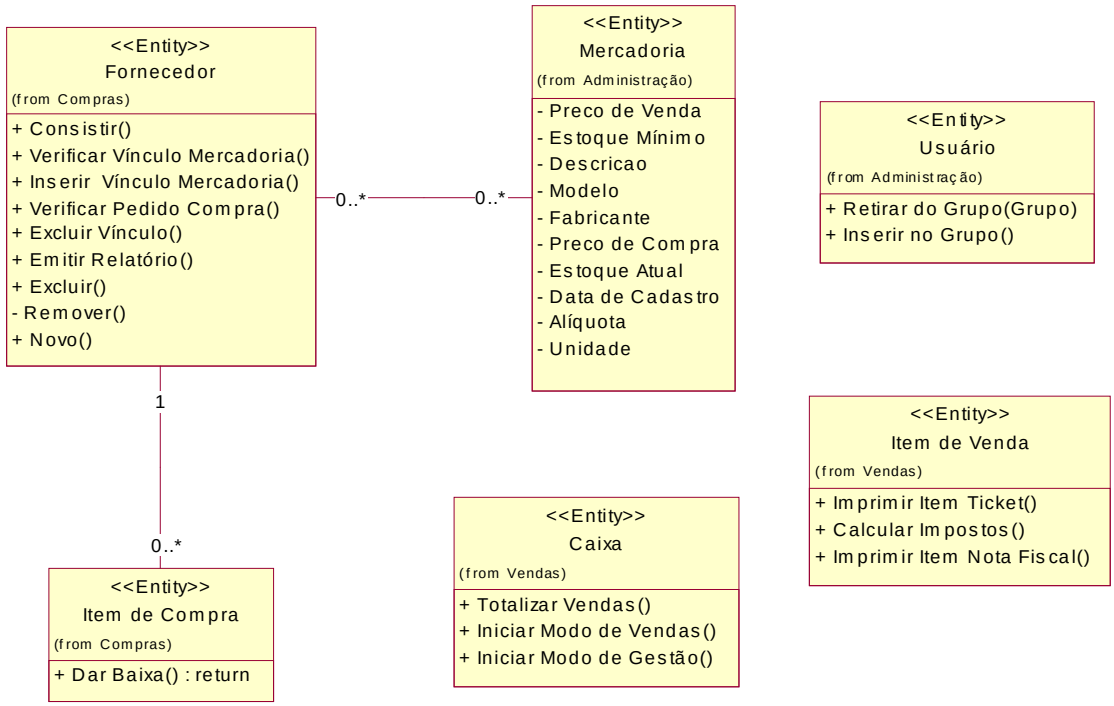

Figura 78 - Diagrama de classes com operações

São os seguintes os resultados desta 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 eqüivalem a relacionamentos de composição onde:

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

Atributos e relacionamentos podem ser alternativas para expressar os mesmos conceitos (Figura 79). A escolha entre atributo e relacionamento deve visar a clareza do modelo. Geralmente atributos são de tipos eqüivalentes a classes pequenas e reutilizáveis, que representam abstrações de nível superior ao do domínio do problema.

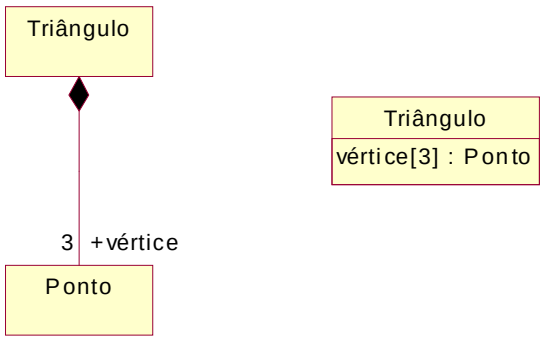

Figura 79 – Eqüivalência entre atributos e composições

2.2.5.2 Definição 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 freqüentemente são adjetivos ou possessivos que descrevem um nome de classe.
- Evitar, nesta 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 estes 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 base) 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.

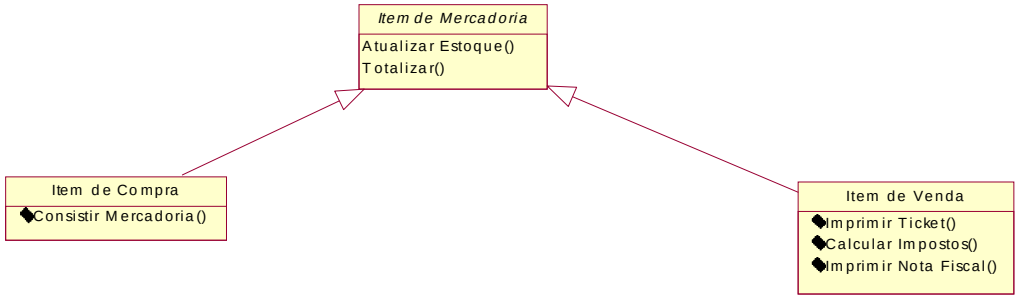

Figura 80 - Exemplo de relacionamentos de herança

2.2.5.4 Término da atividade

Ao fim desta atividade, devem estar:

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

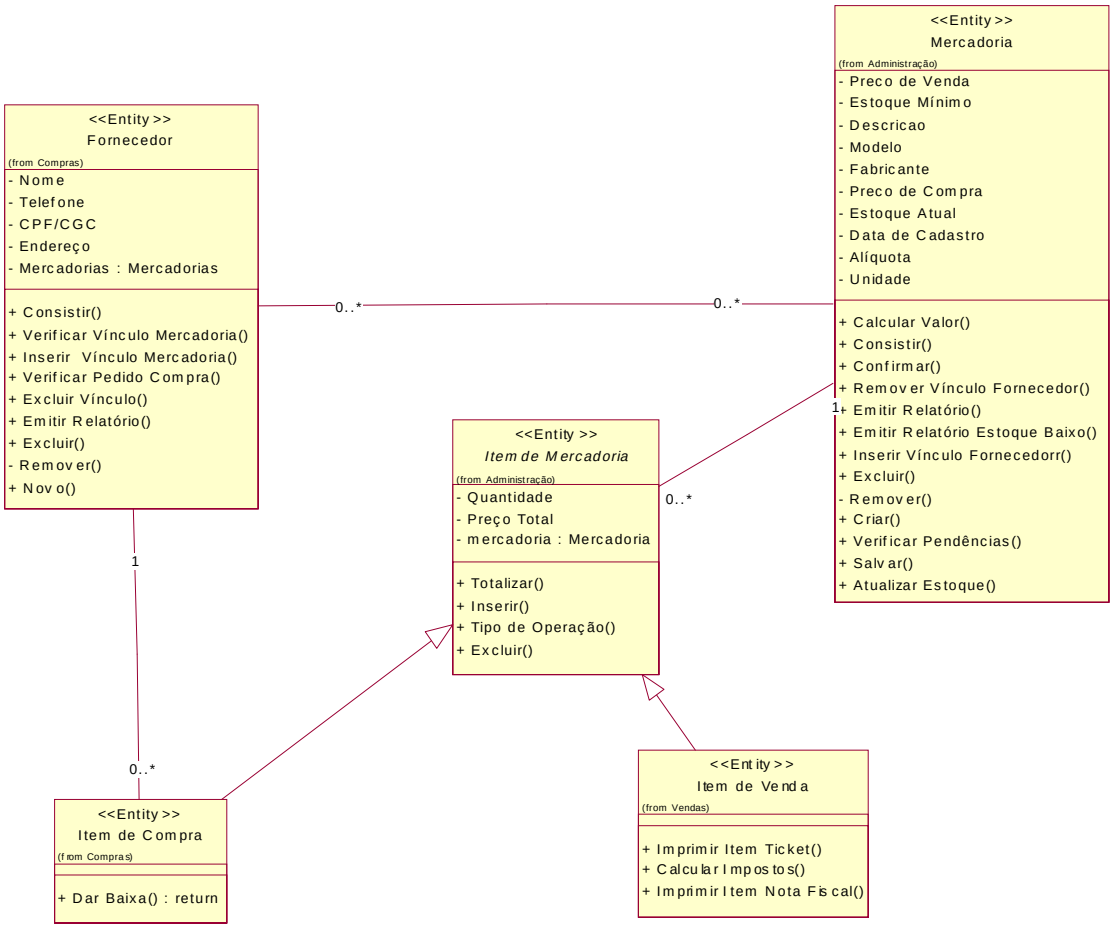

Figura 81 – Diagrama de classes com atributos e relacionamentos de herança

São os seguintes os resultados desta 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, onde são analisados em conjunto o Modelo de Análise e a Especificação dos Requisitos do Software. Para facilitar esta 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 desta 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. Este 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 este 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 acima descritas. Alterações na Especificação dos Requisitos do Software podem também revelar-se necessárias. Pode-se considerar a análise encerrada, pelo 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 destas 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;
- identificadas 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, junto 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 deste 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 aqui tratados 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.

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, com duração típica de uma semana. Os recursos necessários e forma de condução das reuniões são semelhantes aos dos JADs de requisitos.

Esta 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, deve-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 continuar o 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. Isto permite que o cliente avalie os requisitos propostos em conjunto com as estimativas de prazos e custos. Com isto, 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. Isto permite que o grupo de revisão técnica desta

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, deve-se indicar as multiplicidades das classes participantes. A denominação dos relacionamentos e dos papéis das classes participantes só deve ser feita quanto isto contribuir para melhor entendimento do modelo. Restrições que não possam ser expressas através da UML devem ser indicadas através de anotações nos diagramas.

3.3.3 Especificações das classes

Deve-se descrever no Modelo de Análise o mínimo de informação que é 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.

Deve-se, por outro lado, incluir no Modelo de Análise todas as operações que correspondem a mensagens dos diagramas de interação. Deve-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.

Página em branco

Proposta de Especificação de Software

1 Introdução

Este padrão descreve uma estrutura de Proposta de Especificação de Software (PESw). Este 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 desta proposta é delimitar e dimensionar o esforço da fase de Elaboração. Isto é particularmente importante para justificar, perante o cliente, o preço desta fase. Além disto, 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.

Este 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. Neste 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 oferecer apoio informatizado ao controle de vendas, de estoque, de compra e de fornecedores da mercearia Pereira & Pereira.

Tabela 76 - Exemplo de Missão do produto

A declaração da 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 77 - 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 um tempo a ser definido na especificação de requisitos; deverá ser possível a expansão dos pontos de venda.

Tabela 78 - 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 as seguintes metas gerenciais do cliente: prazo máximo de desenvolvimento: 12 meses; custo máximo de desenvolvimento: R$ 60.000,00.

Tabela 79 - 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 80 - 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 adicionais do Merci 2.0: 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 prazo de entrega do produto serão determinados na atividade 6.

Tabela 81 - 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 todo 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, estes aspectos devem ser especificados em documentos separados. Isto é particularmente importante nos projetos que envolvem bancos de dados. O povoamento e controle da qualidade dos bancos de dados é considerado como fora do escopo do software, embora possa 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 destas 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 particular Especificação de Requisitos (documento). Entretanto, chama-se aqui atenção sobre 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 partilha de dados

    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 Merci, bem como definir para os desenvolvedores o produto a ser feito. Público alvo: cliente, usuários e desenvolvedores do projeto Merci.

Tabela 82 - 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. Neste caso, os componentes maiores devem ser enumerados. Resultados normais de um projeto, tais como documentos previstos no processo padrão, não são considerados como 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 fase de Engenharia de Requisitos.

3. Se necessário, esclarecer também os limites do produto, ou seja, o que o produto não fará. Por exemplo, isto pode ser conveniente para evitar falsas expectativas quanto a algum tópico, ou para ressaltar funções e atributos que serão implementadas por outros componentes de um sistema maior, ou em versões futuras deste 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]), isto 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 a partir da Proposta de Especificação de Software, não é necessário que seja completamente consistente com esta; modificações dos itens da proposta são aceitáveis, desde que feitas de acordo com o cliente. Neste 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 83 - 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 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 84 - 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 isto, 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 85 - 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 isto 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 deste projeto.

Número de ordem	Termo	Definição
1	Abertura do Caixa	Inicialização do caixa, autorizando o caixeiro a trabalhar. Ë 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	Modo de Gestão	Modo de operação do Merci no qual o sistema está disponível para a Gestão de Mercadorias, Gestão Manual de Estoque, Gestão de Compras, Gestão de Fornecedores, Gestão de Usuários e Abertura do Caixa.
10	Modo de Venda	Modo de operação do Merci no qual o sistema está liberado para a Operação de Venda e Fechamento do Caixa.
11	Nota Fiscal	Documento exigido pela legislação fiscal para fins de fiscalização.
12	Ticket de Venda	Um relatório impresso pelo Merci que exibe e totaliza os itens referentes a uma venda efetuada.

Tabela 86 - Exemplo de Definições e siglas

2.1.5 Visão geral deste documento (ERSw-1.5)

Descreve-se aqui o restante da ERSw contém, indicando sua estrutura básica. Caso nesta particular ERSw 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 87 - Exemplo de Visão geral deste 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**, isto 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 82). 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. Neste diagrama, os usuários, sistemas externos e outros componentes de um sistema maior são representados por atores, enquanto que 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.

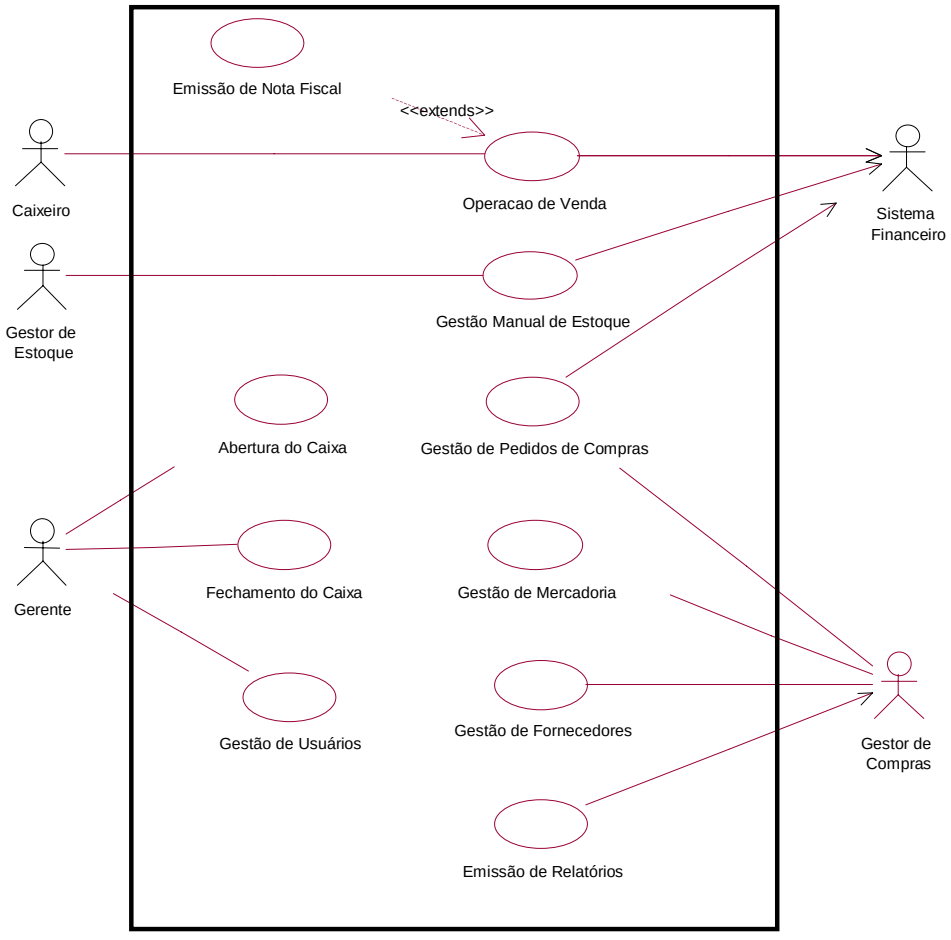

Figura 82 - Diagrama de contexto

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 adeqüe 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
12	Relatório de Estoque Baixo	Gestor de Compras	Emissão de Relatório	Lista mercadorias cujo estoque esteja 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.
19	Tela de Acesso	Todos	Controle de acesso	Solicita o usuário e senha para identificação das funções disponíveis ao usuário solicitante e respectiva consistência com a função pedida.
20	Tela Principal			Interface inicial do MERCI.
21	Tela de Exceções			Tratamento das exceções com apresentação das mensagens de erro.

Tabela 88 - 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, relatório[9] etc.

Número de ordem	Nome	Ator	Caso de uso	Descrição
12	Relatório de Estoque Baixo	Gestor de Compras	Emissão de Relatório	Lista mercadorias cujo estoque esteja 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	Emissão de Pedido de Compra	Pedido de Compras solicitado.
17	Relação de Pedidos de Compra	Gestor de Compras	Emissão de Relatório	Lista dos pedidos de compra da mercearia.
18	Ticket de Venda	Caixeiro	Operação de Venda	Ticket de caixa correspondente a uma Venda.

Tabela 89 - 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, 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	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 desta interface é informar a um Gerenciador Financeiro as transações diárias, desonerando o Merci de consultas e relatórios referentes à administração financeira.

Tabela 90 - Exemplo de descrição de interface de software

[9] Relatórios podem ser on-line (telas e janelas informativas não interativas) ou impressos.

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 deste 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, 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 este 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 91 - 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 Merci 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 Merci no qual o sistema está liberado apenas para a Operação de Venda, Emissão de Nota Fiscal e Fechamento do Caixa.

Tabela 92 - 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 acima pode ser desdobrado em modos diferentes, desde que esta diferença seja significativa do ponto de vista do desenvolvimento (por exemplo, tenha conseqüências 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 93 - 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 isto 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 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 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 94 - 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, freqüência de uso, nível educacional, 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: • 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 estes níveis coerentes com as bases de dados do Merci.

Tabela 95 - Exemplo de descrição de atores

Número de ordem	Atores	Permissão de acesso	Freqüência 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, 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 96 - 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 97 - 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 98 - Exemplo de hipótese de trabalho

2.2.6 Requisitos adiados (ERSw-2.6)

Descrevem-se aqui os requisitos que foram identificados durante a elaboração desta especificação, mas cujo atendimento se decidiu deixar para versões futuras. O preenchimento desta subseção serve para registrar idéias no momento de seu aparecimento, e 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 99 - 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 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 100 - 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 Merci.
O formato do registro financeiro consiste destes 5 campos, separados por uma vírgula. Cada linha do arquivo correspondendo 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 101 - Exemplo de formato de interface de software

O nome e 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 indica-se aqui o conteúdo e formato dos seguintes elementos de cada interface, quando aplicáveis (exemplos em Tabela 100 e Tabela 101):

- 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, nos caso de interfaces gráficas de usuário, a inclusão dos seguintes elementos:

- um esboço do leiaute gráfico sugerido para a interface (Figura 83);
- uma descrição dos relacionamentos com outras interfaces (Tabela 102);
- um diagrama de estados, caso necessário para melhor entender-se o comportamento requerido da interface (Figura 84);
- uma lista dos campos de dados da interface (Tabela 103);
- uma lista dos comandos (botões de ação ou controles eqüivalentes) da interface (Tabela 104);
- observações.

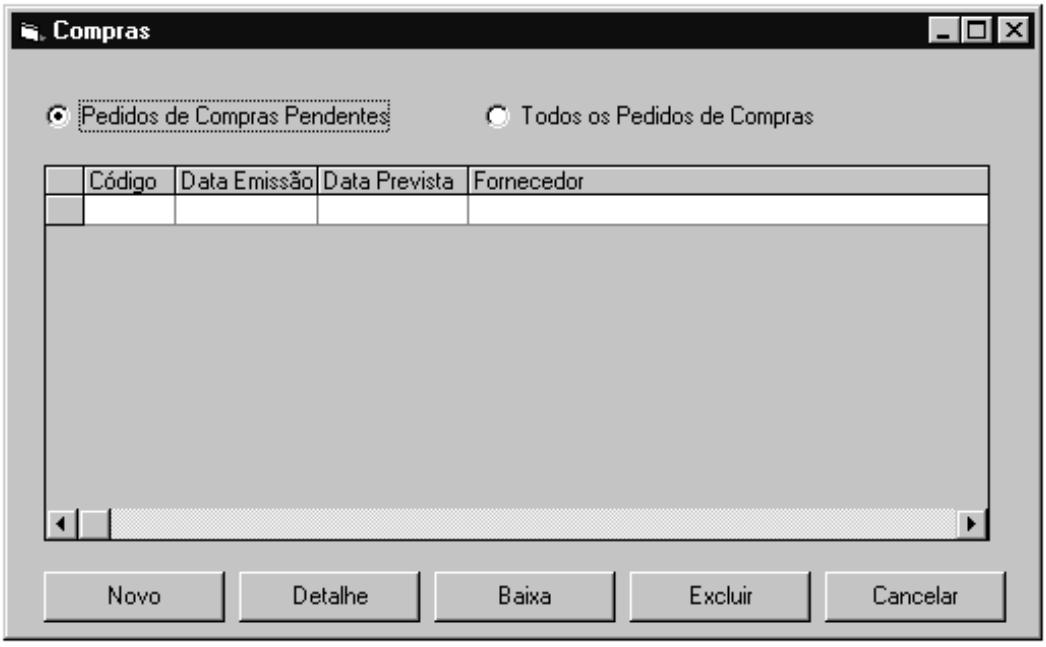

Figura 83 - Exemplo de esboço de leiaute de interface de usuário

O botão Cancelar retorna à interface principal. O botão Emitir NF chama a interface Tela de Nota Fiscal.

Tabela 102 - Exemplo de descrição de relacionamento com outras interfaces

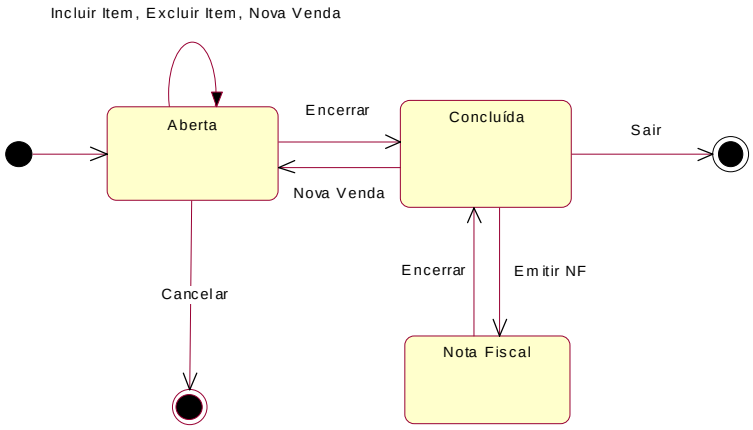

Figura 84 - Exemplo de diagrama de estados para interface de usuário

Diagramas de estado devem ser usados quanto 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 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/AAAA	Data	Preenchido pelo Merci / não editável.
3	Data Prevista	Maior que Data Emissão.	DD/MM/AAAA	Data	Preenchido pelo Merci / não editável.
4	Fornecedor	Não vazio.	Até 30 caracteres.	Texto	Preenchido pelo Merci / não editável.

Tabela 103 - 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, estes campos podem ser substituídos por soluções funcionalmente eqüivalentes, desde que isto 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 104 - 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 destes requisitos, a nível suficiente para o desenho do produto, de seus testes de aceitação e de seu manual de usuário.

A forma de descrição funcional adotada neste 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. Nesta 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 que 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 módulo 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.

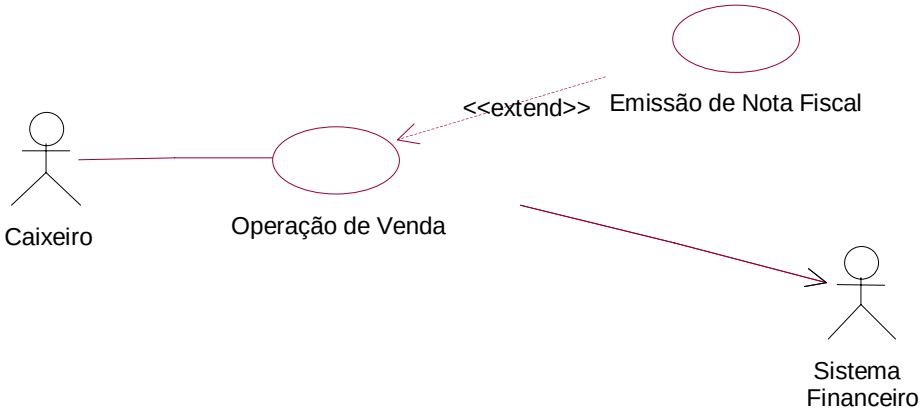

Figura 85 - Exemplo de diagrama local a um caso de uso

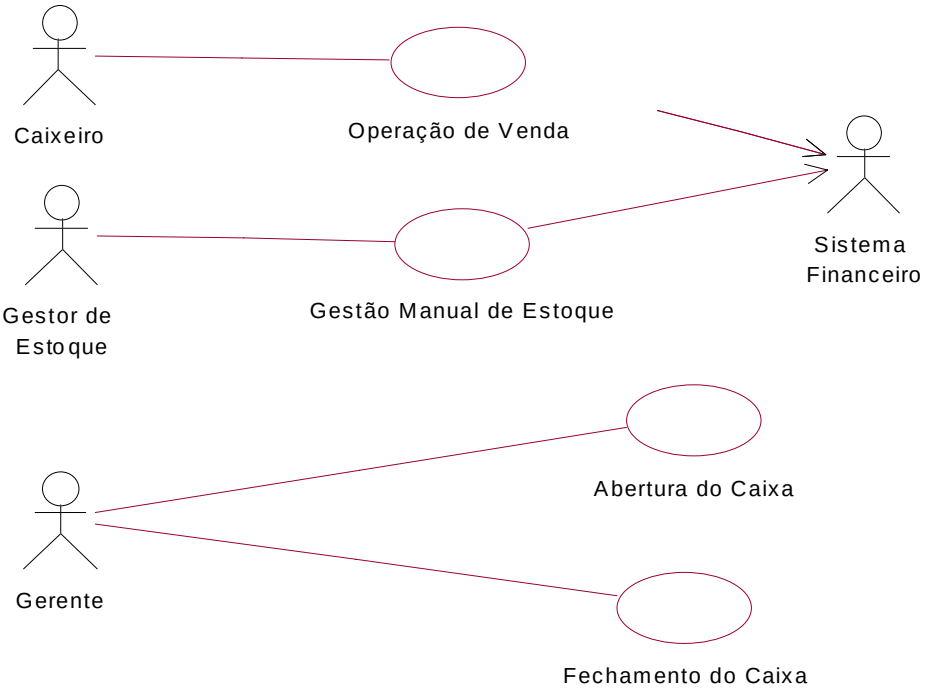

Figura 86 - Exemplo de proposta de casos de uso para uma liberação executável

É fundamental que este diagrama seja legível; caso necessário, 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:

- precondições para a realização do caso de uso;

- fluxo principal do caso de uso, descrito na forma de uma seqüência 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 freqüentes);
- 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.
Merci deve estar no Modo de Vendas.

Tabela 105 - 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, 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 106 - 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 107 - Exemplo de subfluxo: registro de item em Operação de Venda

Se o Gestor de Compras solicitar:
o Merci imprime o pedido de compra.

Tabela 108 - 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 este 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. Esta 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 informados 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 109 - Exemplo de requisito de desempenho

2.3.3.3 Requisitos de dados persistentes

Descrevem-se aqui estruturas lógicas de dados persistentes[10] 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 permanente[11].

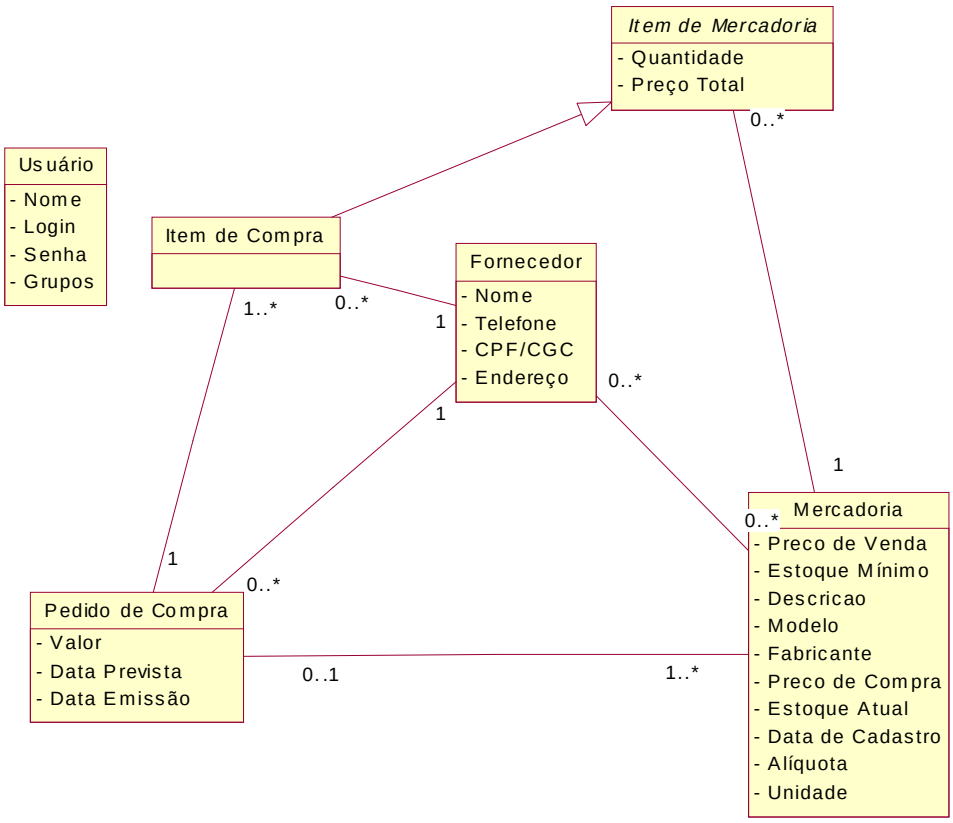

Tabela 110 - Exemplo de diagrama de dados persistentes (classes)

Deve-se apresentar um diagrama de entidades e relacionamentos ou um diagrama das classes permanentes, mostrando seus relacionamentos e atributos.

Além disto, devem ser apresentados os requisitos específicos aplicáveis a estas estruturas lógicas de dados persistentes. Eles podem incluir:

- tipos de informação que devam ser suportados;
- freqüência de uso;
- restrições de acesso;

[10] Isto é, que mantêm seu valor após a execução do programa.

[11] Grupo de objetos permanentes da mesma classe, armazenados em um banco de dados orientado a objetos ou de forma simulada, através de outro tipo de banco de dados.

- restrições de integridade;
- requisitos de guarda e retenção de dados.

2.3.3.4 Restrições ao desenho

Esta subseção descreve restrições ao desenho que sejam impostas por padrões externos. Estas 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 111 - Exemplos de restrição ao desenho (legal)

2.3.3.5 Atributos de qualidade

Esta 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 Merci com um dia de treinamento.

Tabela 112 - 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 x benefício;
- informação auxiliar para os leitores da Especificação dos Requisitos doSoftware, sobre tópicos de informática ou da aplicação;
- requisitos de embalagem, distribuição e instalação etc.

3 Revisão da Especificação dos Requisitos doSoftware

3.1 Introdução

Esta subseção apresenta um roteiro para revisão da Especificação dos Requisitos do Software (ERSw). Ao fim desta revisão, deve-se assegurar que a ERSw:

- esteja conforme com este 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 deste modelo;

- forneça informação suficiente para o desenho do produto, de seus testes de aceitação e do seu manual de usuário.

Neste 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 página 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 destes 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. Esta 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. Esta 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;

- é claramente descrita a aplicação do produto especificado, inclusive suas metas e seus benefícios, quantificados se possível;

- esta subseção é consistente com outros possíveis documentos de nível mais alto.

3.5.1.3 Revisão da subseção 1.3 Materiais de referência

Esta subseção da ERSw identifica todas as fontes de informações utilizadas, inclusive entrevistas e reuniões. Este item deve estar suficientemente detalhado para que os revisores possam avaliar se o contato com o cliente e usuários é completo.

Ainda neste item, é importante verificar se foram incluídas referências à documentação das versões correntes (se houver), que permitam sua rápida localização. Os meios de aquisição dos documentos enumerados devem ser apresentados (e.g.: relatórios arquivados pelo fornecedor, relatórios arquivados pelo cliente, material de sítios Web etc.).

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. Todos as definições apresentadas na ERSw devem ser agrupadas nesta 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:

- as interfaces importantes, para todos os elementos relevantes do ambiente do produto, foram definidas;

- o diagrama apresentado é claro e pode ser compreendido sem textos complementares.

Bancos de dados partilhados, arquivos partilhados, interfaces com outros sistemas, interfaces com usuários, e outras interfaces externas relevantes devem ser incluídas no diagrama. Todos devem ser representados por atores, na notação de casos de uso.

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 estes dispositivos;
- não são incluídos dispositivos cujo suporte é 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 estas 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, sigla, versão, 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 estes recursos;
- não são incluídos recursos cujo suporte é 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 nesta 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 esta seção descreve aspectos relevantes que possam limitar as opções dos desenvolvedores, e apenas estes aspectos.

3.5.2.5 Revisão da subseção 2.5 Hipóteses de trabalho

Verificar se esta seção descreve fatores que se supõem sejam atendidos para que o produto possa ser desenvolvido e implantado, e apenas estes 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 neste padrão. Deve ter sido seguida a organização por casos de uso e classes, recomendada por este padrão, ou uma forma alternativa de organização, recomendada pelo processo personalizado adotado para o projeto.

Os seguintes princípios devem ser verificados:

- cada requisito deve atender as características de qualidade dos requisitos (precisão, consistência etc.);
- cada requisito relacionado com outros documentos deve conter referência a estes documentos;
- a organização dos requisitos deve ser orientada para facilitar a legibilidade e 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 113).

Casos de uso	Foram todos descritos sucintamente na subseção 2.2 Funções do produto.	
	Todos os casos de uso presentes nos diagramas e na subseção 2.2 Funções do produto 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 113 – 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 partilhados que sejam usados pelo produto;
- estes 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 estes 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. Esta 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 seguinte (Tabela 114).

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 foram anotadas, se existentes.
		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 destas.
		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 a 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 114 – 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 sua eficácia de 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 este 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 técnica[12], concentra-se na análise de aspectos selecionados, um de cada vez. É obrigatória a geração de uma lista de defeitos com classificação padronizada, requerendo-se a ação dos produtores para remoção destes 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 de pares, que checa o material à medida que ele vai sendo apresentado. Podem ser simulados os passos de um procedimento. Este tipo de revisão não exige muita preparação prévia, e pode ser feita com maior número de participantes, por terem estes papel mais passivo. As revisões de apresentação têm eficácia média para detecção de defeitos. Elas podem ser usadas nos marcos de projeto em que se requer sejam feitas apresentações ao cliente.

- A **revisão gerencial** é conduzida pelo gerente de um projeto, com o objetivo principal de avaliar os problemas técnicos e gerenciais deste, assim como o seu progresso em relação aos

[12] Alguns autores ignoram a distinção entre revisões técnicas e inspeções.

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. Este tipo de revisão pode ser realizada 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 resultado em revisão.

Em casos excepcionais, é permitida a presença de assistentes, para fins de treinamento, avaliação ou transmissão de conhecimento. Estes 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. Este 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 isto é determinar que cada membro deva fazer pelo menos um comentário positivo e outro negativo, a respeito do resultado do projeto 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 serão feitas interrupções externas à reunião, e que os membros da revisão não serão 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 houver a ausência de algum dos participantes, a revisão deverá ser cancelada, e nova data para a revisão deverá ser marcada pelo líder. Este cancelamento é obrigatório nos seguintes casos:

- só houver um autor do documento e este não comparecer;
- o líder não comparecer e não houver outra pessoa qualificada para a posição;
- o relator não comparecer e não houver outra pessoa qualificada para a posição;
- após o remanejamento das funções dos membros, devido a ausências inesperadas, não sobrarem 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, apresentamos 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 desta 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 possuir dificuldade pessoal com qualquer um dos revisores, que possa interferir em sua habilidade de liderar a revisão.

A Tabela 115 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 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 do 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 115 - 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 desta revisão em particular;
- compreender o jargão e 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 116 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.
	Dispor 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 e registrar o total de horas gasto na preparação da revisão.
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 116 - 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 (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. Neste caso, os usuário 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. Este relatório apresenta um cabeçalho com um identificador único da revisão, tipo (revisão técnica ou uma das alternativas), data, hora de início e hora 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. Este dado é importante para 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);
- **revisão não foi completada** (foi necessário cancelar ou interromper a reunião de revisão, e nova revisão será marcada).

Identificação da revisão	**Identificador**	MERCI-RRT-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 117 - 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 receberá as assinaturas destes.

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. Isto 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 destes aspectos, a determinação da prioridade de correção destes 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, 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 118 mostra o padrão de classificação de defeitos do PSP ([Humphrey95]). Esta classificação detalhada permite identificar os tipos de erros mais freqüentes, e definir possíveis contramedidas. Por exemplo, pode-se introduzir nos padrões de desenho detalhado e codificação regras mais estritas que combatam os tipos de erros mais freqüentes. 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 118 – Classificação dos defeitos de desenho detalhado e código

4.3 Listas de tópicos

O Relatório de Revisão Técnica poderá apresentar os anexos opcionais seguintes.

- Uma **Lista de Tópicos do Projeto**, que contém sugestões para melhorias no projeto. Estas sugestões não devem ser discutidas, mas simplesmente anotadas na ordem que aparecerem, e podem ser aceitas ou não pelo gerente do projeto.

- Uma **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 desta lista são os grupos de suporte, como os grupos de Garantia da Qualidade e Engenharia de Processos.

Página em branco

Desenho

1 Princípios

1.1 Objetivos

O fluxo de Desenho (design ou projeto[13]) 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 possa eventualmente 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 possa 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 119 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 necessário para 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.

[13] 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 119 – 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 do módulos. Como isto pode aumentar o acoplamento entre módulos, estas 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

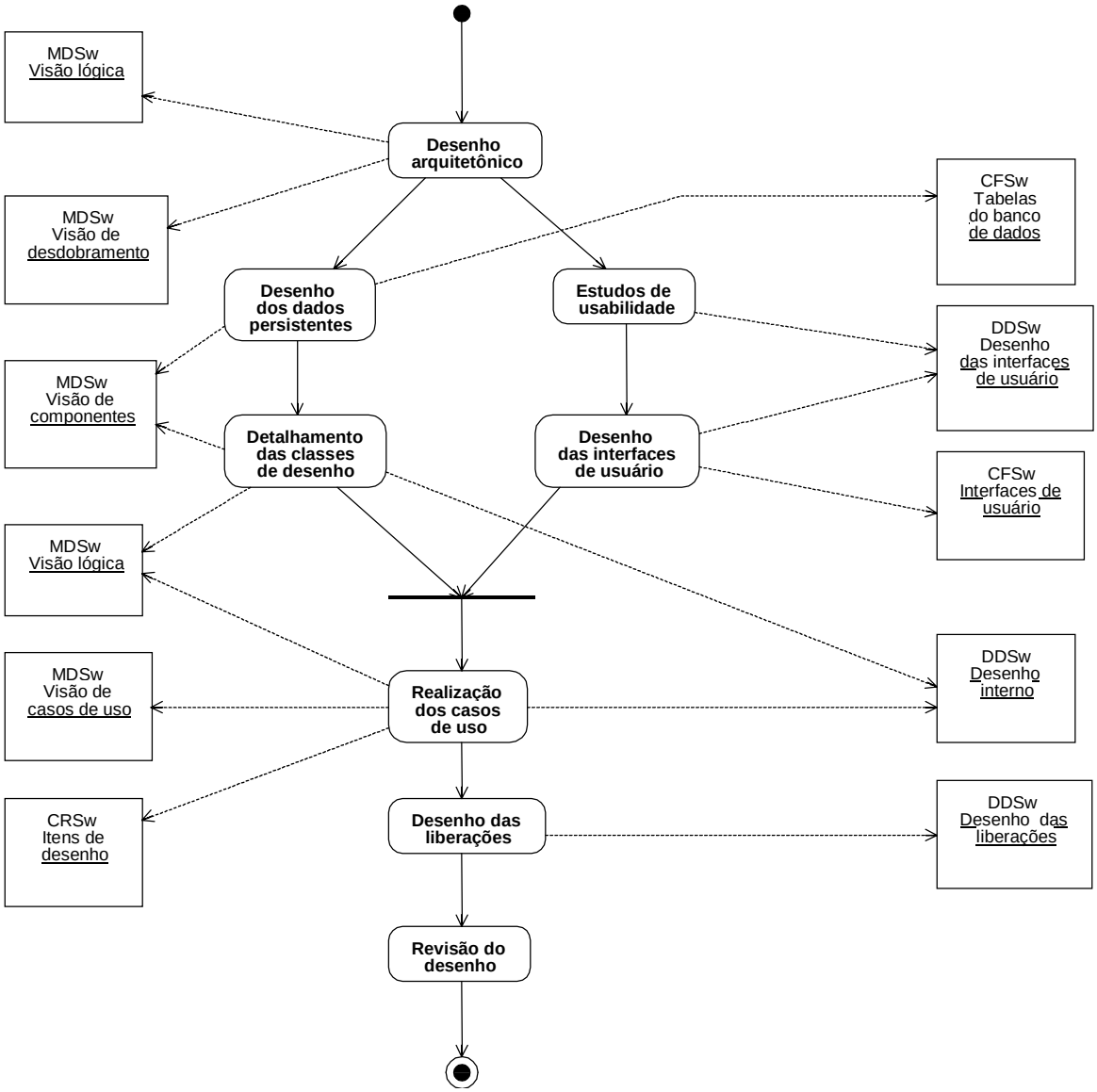

Figura 87 – Atividades e artefatos do fluxo de Desenho

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 isto 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 de trabalhar destes. 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 estas duas primeiras atividades sejam realizadas principalmente na fase de Elaboração, principalmente, 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 estas 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. Esta 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. Esta 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 divididas 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 deste 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 que 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**).

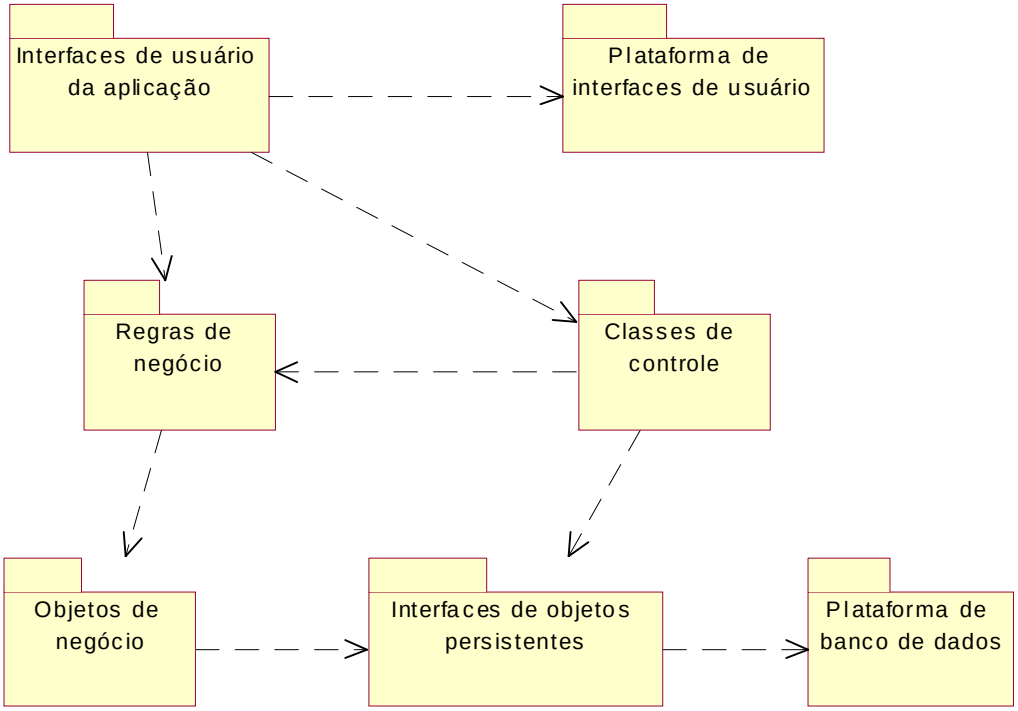

Figura 88 – Exemplo de pacotes lógicos

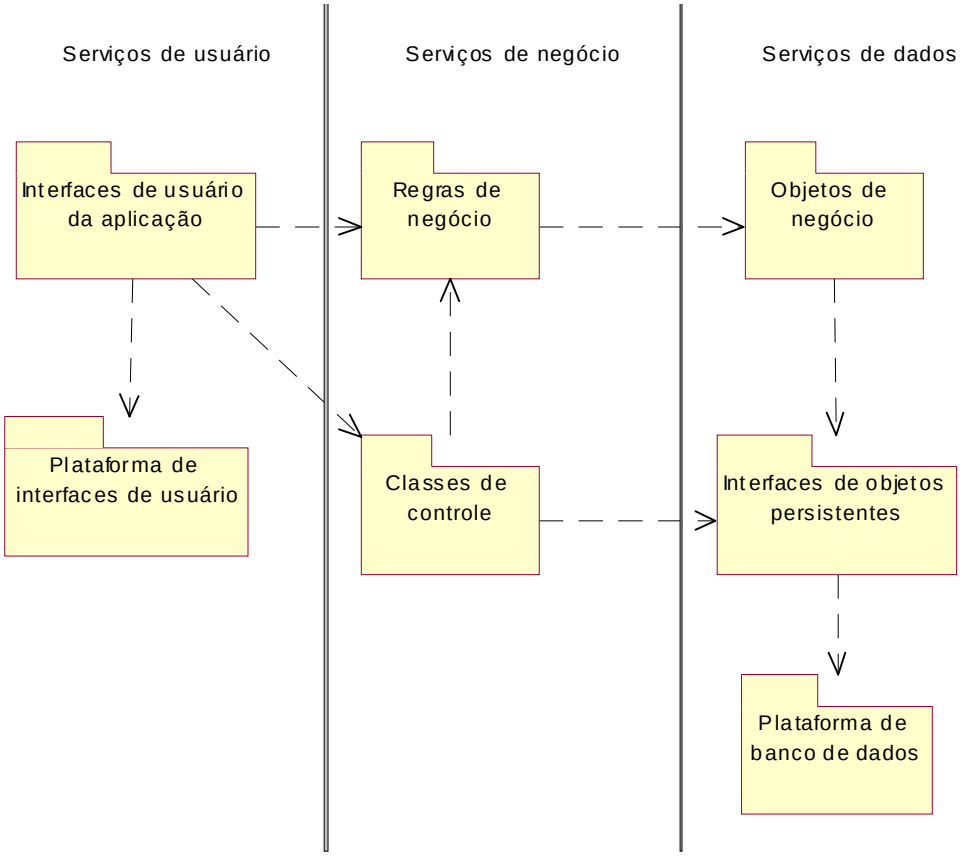

Figura 89 – Exemplo de arquitetura em camadas

2.2.1.3 Organização da arquiteturas

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.

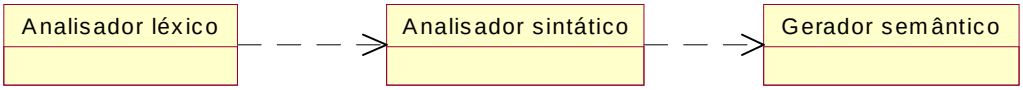

Figura 90 – Exemplo de arquitetura em filtros

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 120 – 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 isto, 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 88, por exemplo, os pacotes obedecem à divisão mostrada na Tabela 120.

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 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.

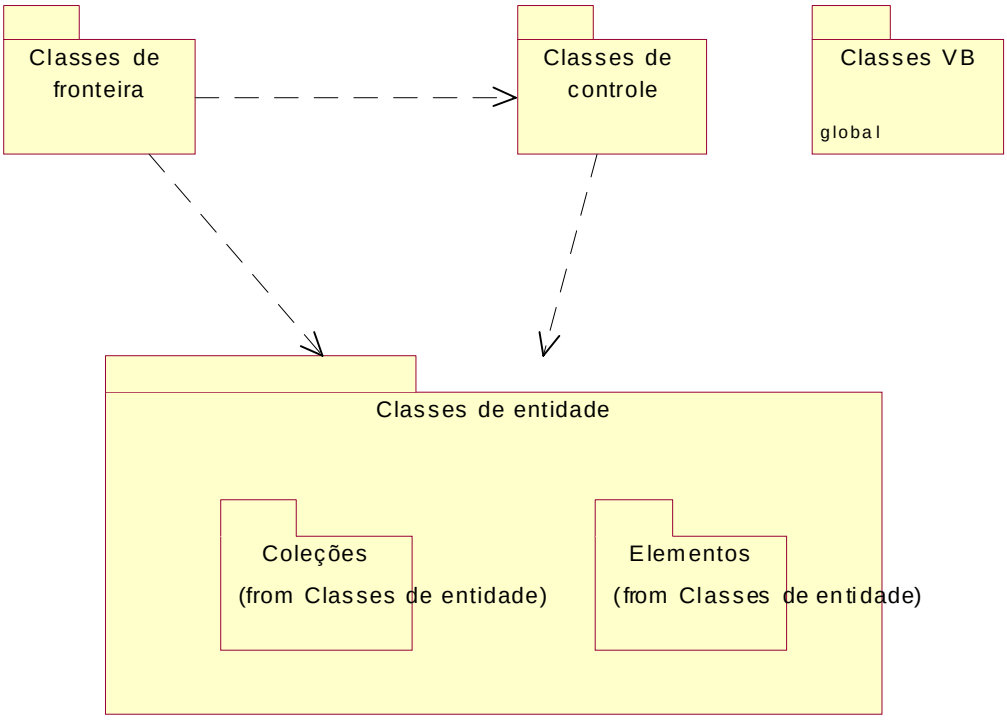

Figura 91 – Exemplo de organização de pacotes lógicos

A Figura 91 mostra parte dos pacotes lógicos desenhados para um sistema de informatização de Mercearia. Aparecem os pacotes descritos na Tabela 121. Esta 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.	`modErrorHandling`
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 121 – 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 como 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, considerando-se a freqüência de uso?
- **Satisfação do usuário**: quão satisfeitos os usuários se sentem com o produto, de maneira geral?

Estes fatores podem ser balanceados. Se for viável um longo tempo de aprendizado, a produtividade dos usuários pode ser melhorada através 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. Estas 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 um 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 isto, 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. Neste ambiente, existem ferramentas que permitem extrair as classes de fronteira correspondentes a partir do código dos formulários (técnica de engenharia reversa). Este 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.

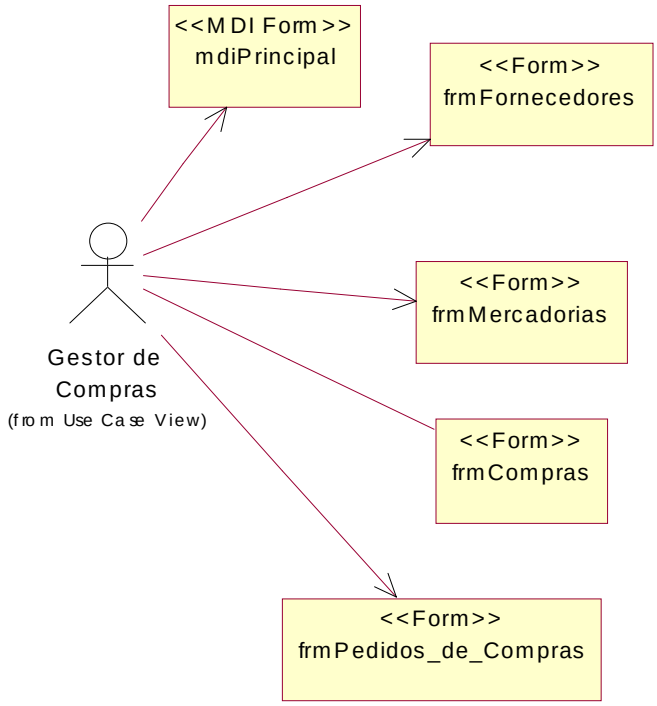

Figura 92 – Exemplo de classes de interfaces de usuário

Quando é necessário mostrar como as classes de fronteira do produto se relacionam com as classes constituintes da plataforma de desenvolvimento, as classes necessárias destas plataforma devem ser importadas para o modelo. Na Figura 55, 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 que 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 93).

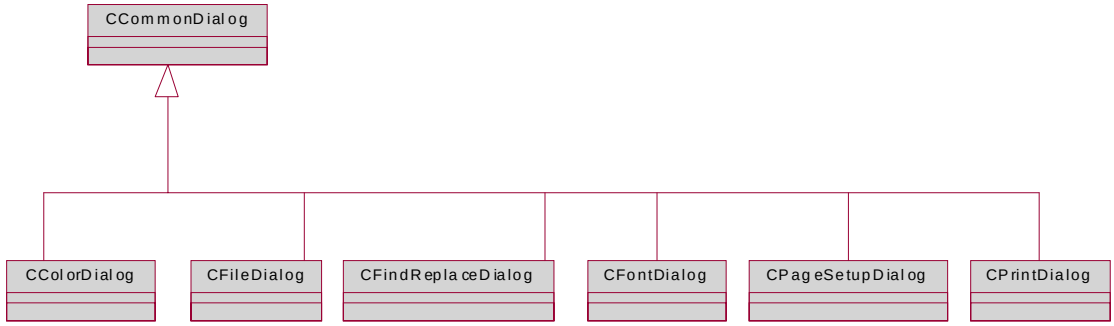

Figura 93 – Classes de diálogo MFC

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. Estas 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 isto, 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 implementação deles. 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, independente da natureza dos objetos. As classes parametrizadas são **instanciadas** para formar as classes do domínio. Na Figura 94, 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`.

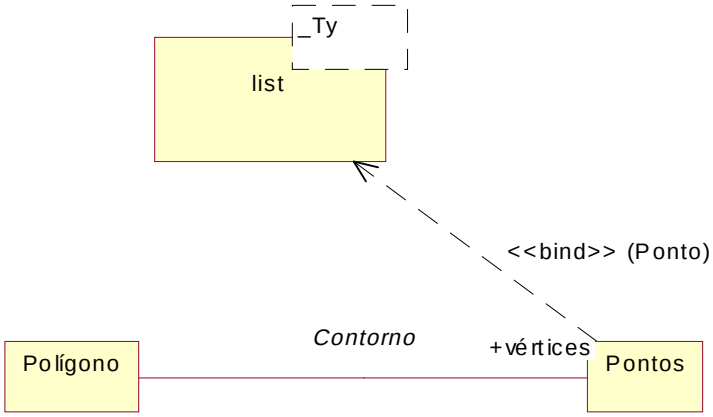

Figura 94 - Exemplo de uso de classe parametrizada

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 através de um tipo de dados específico da linguagem (Figura 95).

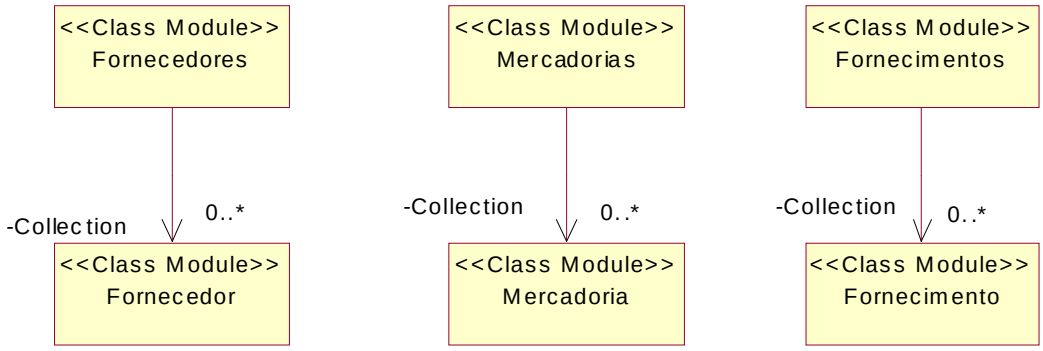

Figura 95 – Coleções em Visual Basic

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 96 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 estas classes serão implementadas como módulos de classe do Visual Basic. Os papéis indicam os nomes das variáveis que serão usadas para implementar os relacionamentos.

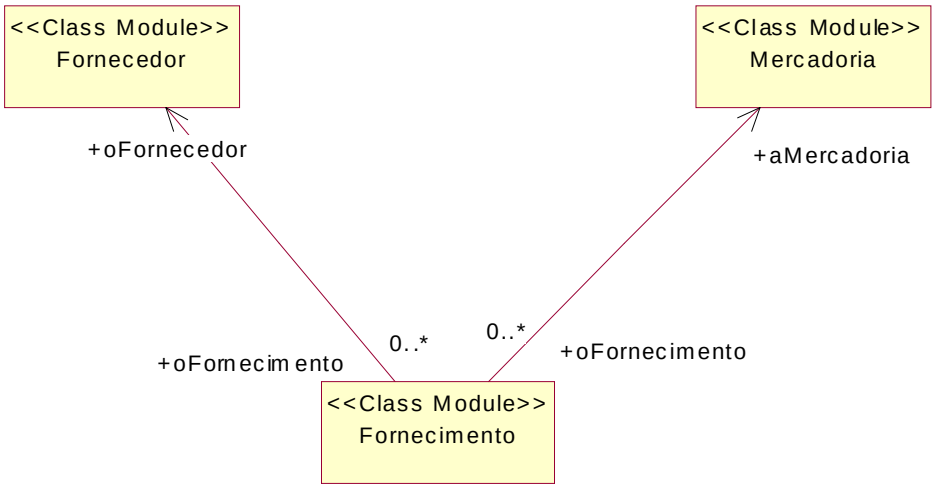

Figura 96 – Relacionamentos com navegação restrita

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 estes relacionamentos serão implementados, observando-se os tempos de vida dos objetos.

- **Agregação por valor**: o objeto todo contém o objeto parte em seu espaço[14]. 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.

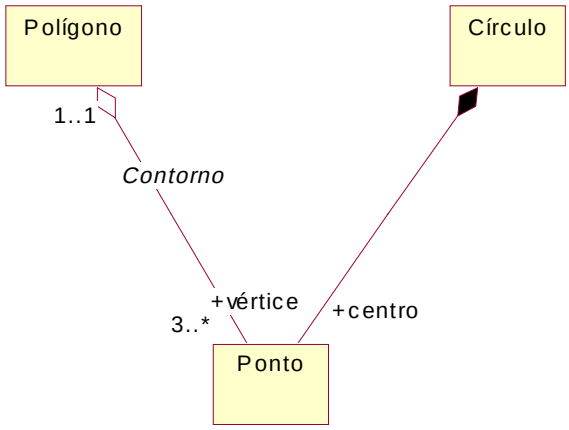

Figura 97 – Agregação versus composição

Na Figura 97, 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 `Contorno`. Isto faz sentido, por exemplo, se os pontos representarem entidades físicas, como pontos de um relevo, enquanto que o polígono representa uma curva de nível que passa por estes pontos.

[14] A declaração da classe parte pode estar aninhada ou não na declaração da classe todo.

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 desta classe podem usar (símbolo na UML #);
- **privado**: nenhuma outra classe pode usar diretamente (símbolo na UML -);
- **implementação**: declarado no corpo de uma operação.

Os seguintes critérios devem ser usados, para determinar o tipo de acesso.

- 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 estas classes como **amigas** da classe fornecedora.

Na Figura 98 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.

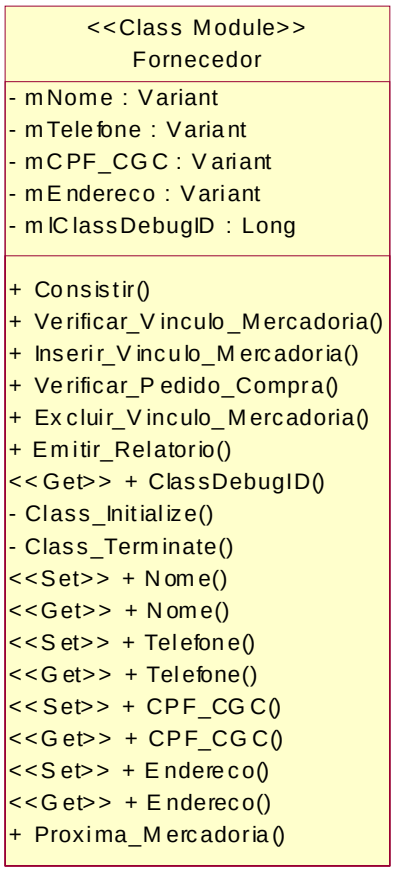

Figura 98 - Exemplo de controles de acesso

Note-se também na Figura 98 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 99) 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 99 – Exemplo de caso de uso detalhado no desenho

Fluxos complexos podem ser melhor descritos através de diagramas de estado (Figura 100). 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 101) e de controle.

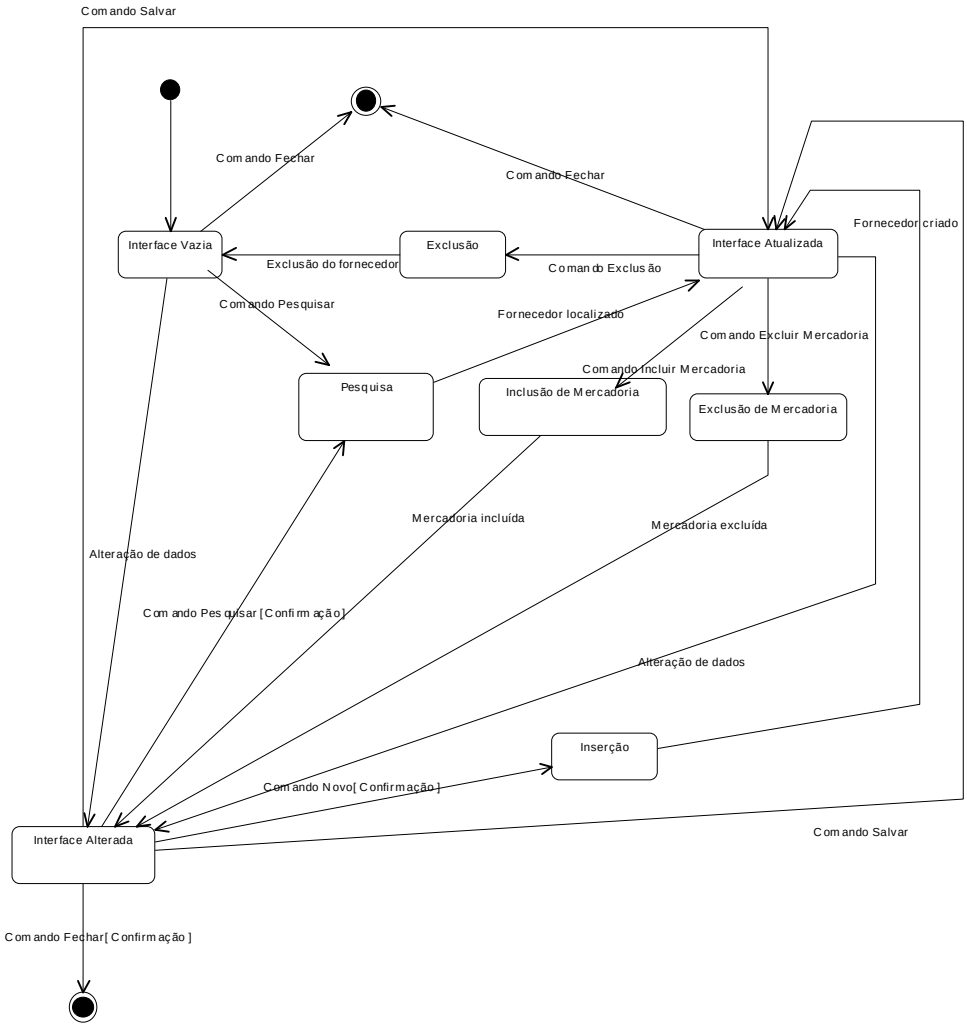

Figura 100 – Exemplo de diagrama de estados de caso de uso

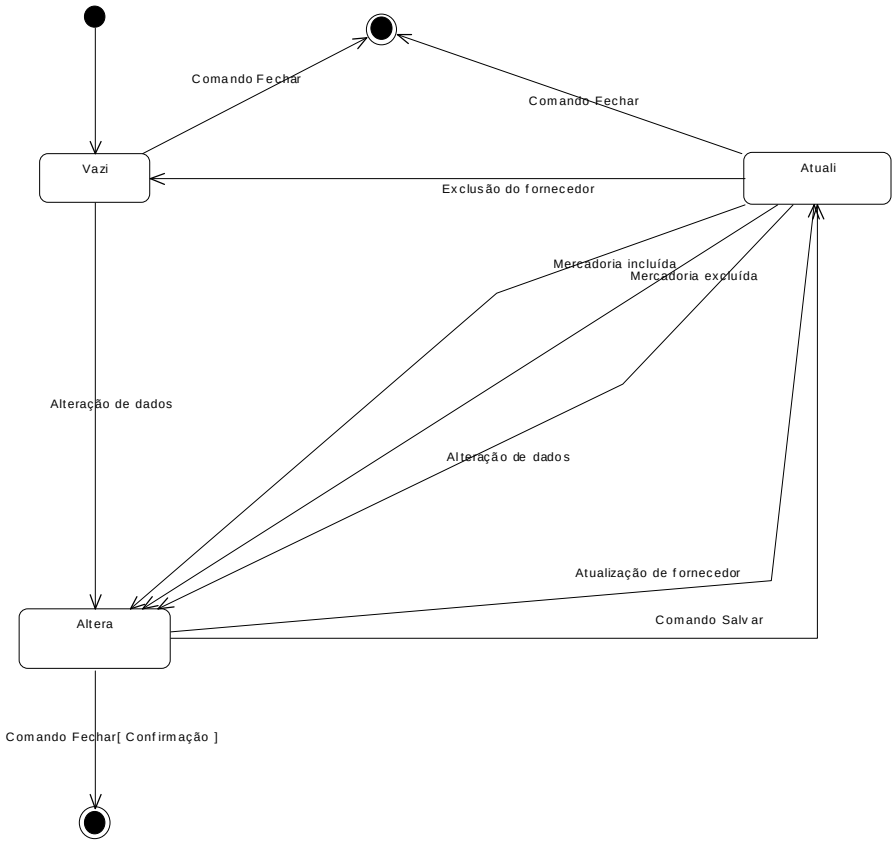

Figura 101 – Exemplo de diagrama de estado de classe de interface de usuário

O modelo de desenho deve mostrar a realização dos casos de uso através de diagramas de seqüência e colaboração. Os diagramas de seqüência (Figura 102) devem ser preferidos quando se quer enfatizar o ordenamento temporal das mensagens, enquanto que os diagramas de colaboração (Figura 103) devem ser usados para 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 102. 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.

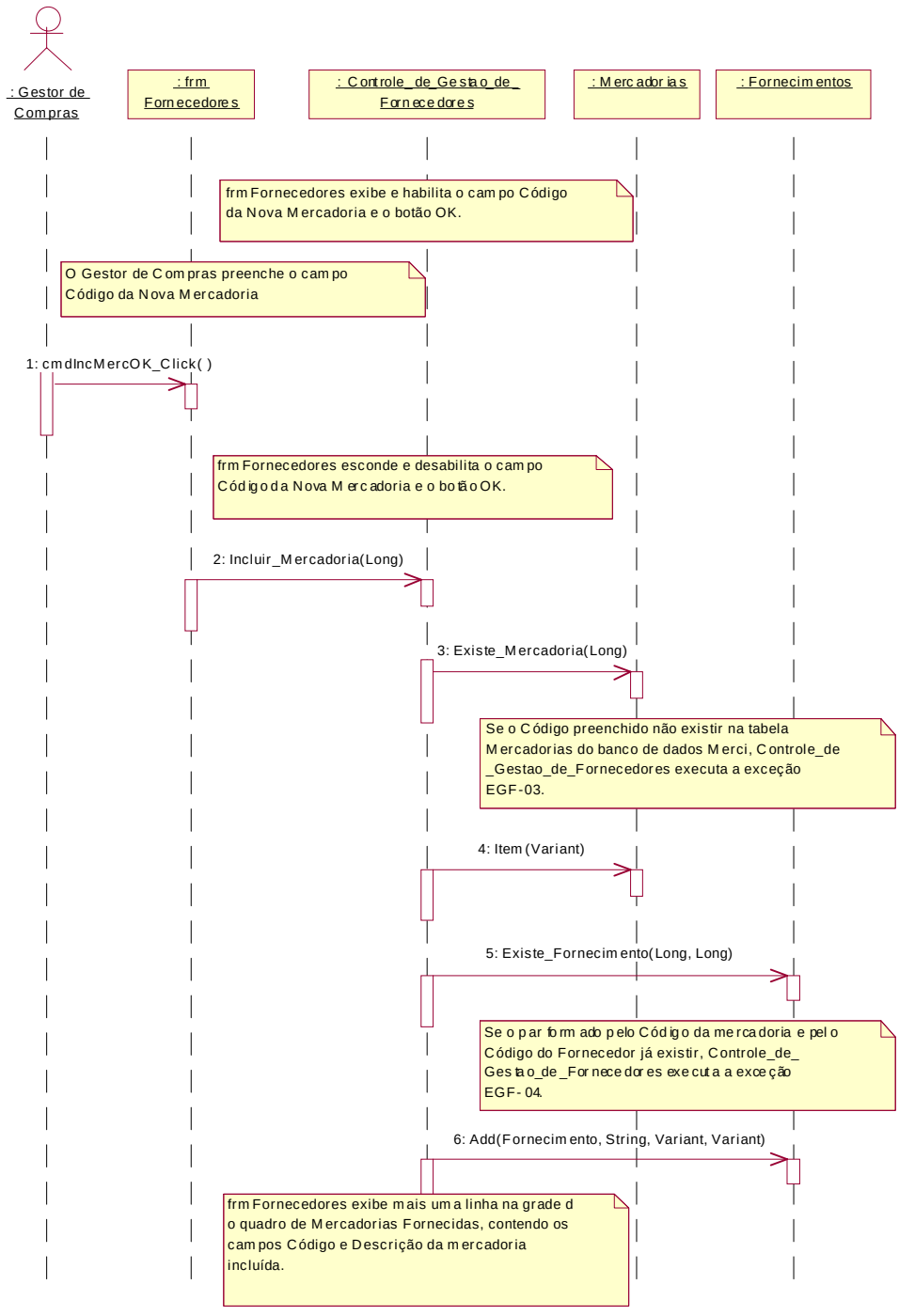

Figura 102 – Exemplo de diagrama de seqüência para realização de um caso de uso

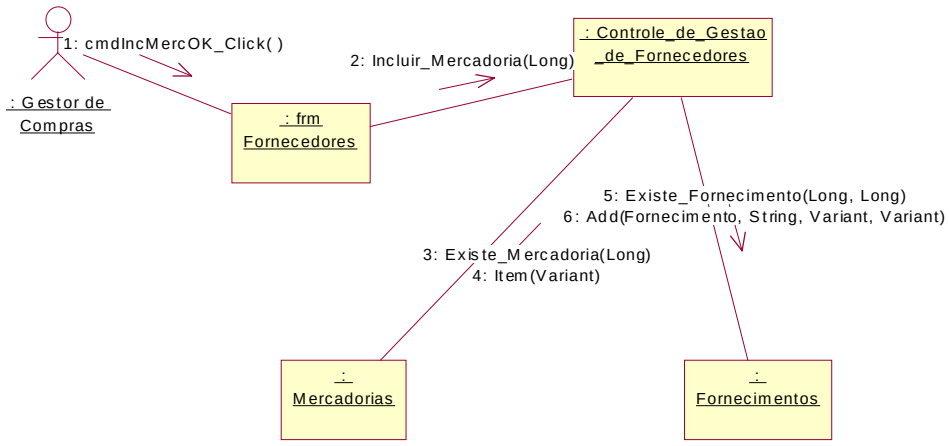

Figura 103 – Exemplo de diagrama de colaboração para realização de um caso de uso

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. Estas 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 Merci.
2	Vendas	Implementar os casos de uso relacionados com a Gestão de Vendas, completando as funções do Merci.

Tabela 122 - 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 122)

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 123 – Exemplo de especificação de liberação

De preferência, as liberações são divididas de forma que cada liberações implemente classes inteiras e casos de uso inteiros. Em certos casos, é admissível que uma liberações implemente apenas algumas das operações de uma classe ou algum dos roteiros de um caso de uso, quando isto 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 123) 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, onde 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 esta 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 desta 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. Este 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 disto, 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 estãos contido no padrão para Descrição do Desenho do Software.

3 Técnicas

3.1 Visão geral

As técnicas tratadas aqui em detalhe dizem respeito 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 estes 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 disto, a participação dos usuários geralmente é muito menor do que nos fluxos de Requisitos e mesmo de Análise. Um exceção, naturalmente, pode ser representada pelas 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 estes 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 deste 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 freqüente;
- 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 através 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 124 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 124 - 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, maior o esforço 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 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-se 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.

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 freqüência eles executam estas tarefas;
- em que condições eles executam estas tarefas;

No caso de produtos em evolução, as versões correntes devem ser avaliadas. Nesta avaliação, é preciso identificar o que a versão atual faz, e como ela 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. Isto é feito, por exemplo, desabilitando-se as opções inválidas em cada estado das interfaces. Com isto, apenas um subconjunto das opções ficará habilitado e disponível, em um dado momento. Isto 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. Esta confirmação é ainda mais necessária quando o produto não oferecer meios para que o usuário desfaça 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 estes usuários, a necessidade de apontamento para utilizar cardápios e outros itens reduz a produtividade.

O controle da interação deve estar sempre nas mãos do usuários. Por exemplo, ao preencher formulários interativos, os usuários devem poder editar os campos na ordem que desejarem, embora uma ordem padrão de deslocamento entre campos deva ser oferecida. Deve-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 deste conjunto. Esta 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. Isto 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 através 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. Isto 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 subrotinas 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 da 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 estes ambientes deve ter à mão os modelos UML destas 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 portável 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** ("*patterns*") de desenho, geralmente apresentados na forma de modelos de classes, juntamente com instruções de utilização e sugestões de implementação. Geralmente este 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 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, através de mecanismos de montagem e desmontagem destes objetos. A desmontagem dos objetos traduz os dados contidos nestes 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. Requer-se 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.

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.

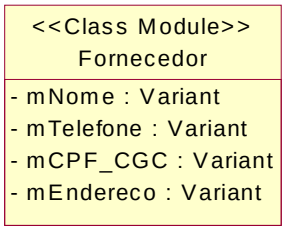

Figura 104 - Exemplo de classe persistente

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

Figura 105 - Definição de tabela para classe persistente

mNome	mTelefone	mCPF_CGC	mEndereco	FornecedorId

Figura 106 - Tabela correspondente a classe persistente

A Figura 104 mostra um exemplo de classe persistente, que é traduzida na Figura 106, através do comando SQL mostrado na Figura 105. Este comando DDL[15], expresso na linguagem ANSI SQL, foi gerado automaticamente por uma ferramenta de modelagem.

[15] Data Description Language.

3.4.2.2 Representação dos relacionamentos

A representação dos relacionamentos seguirá as 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.

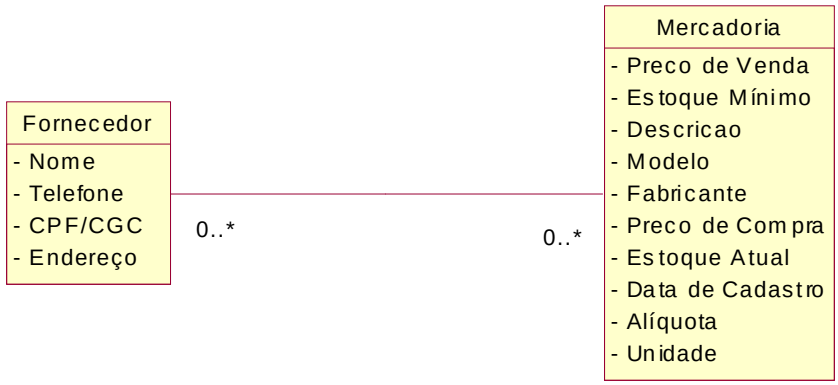

Figura 107 - Exemplo de classes de análise persistentes com relacionamento

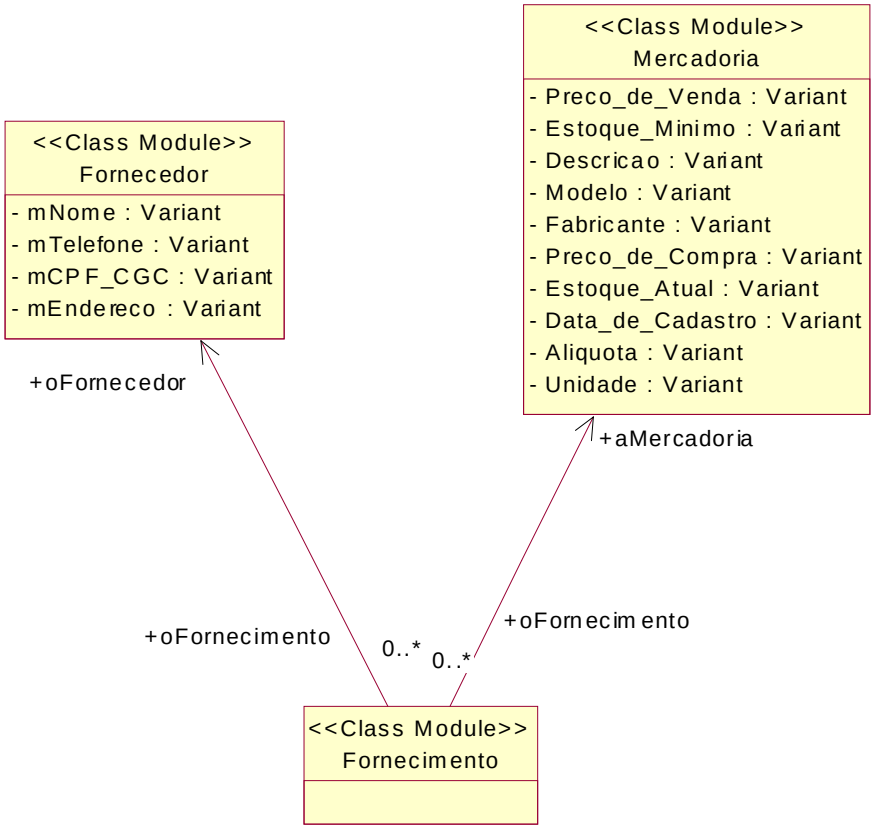

Figura 108 - Exemplo de classes de desenho persistentes com relacionamentos

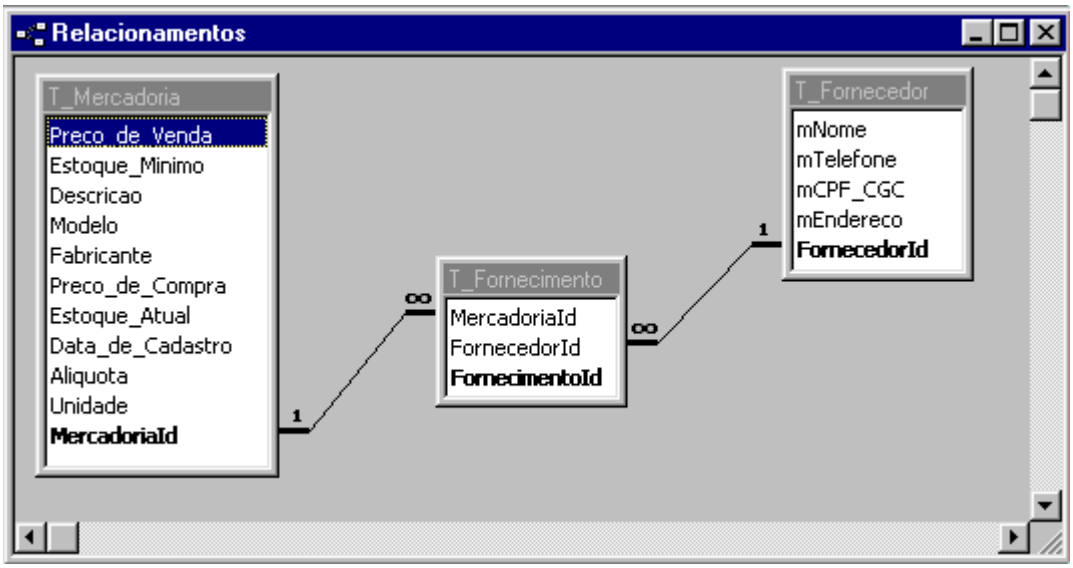

Figura 109 - Tabelas construídas no Microsoft Access

As classes do modelo de análise da Figura 107 foram convertidas, no modelo de desenho, para a Figura 108. Um relacionamento **m:n** foi substituído por uma classe intermediária. A Figura 109 mostra as tabelas correspondentes em um sistema de gestão[16] 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 110 - Exemplo de classes persistentes com relacionamento de herança

[16] 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 111 - Tabelas correspondentes a classes com herança

numero_da_venda	quantidade	preco_total

Figura 112 - 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 isto, as tabelas geradas a partir de modelos orientados a objetos geralmente estarão na terceira forma normal, de forma automática. Entretanto, tabelas podem ser partidas por questões de desempenho. Quando há necessidade de desnormalizar, os mecanismos de tradução entre paradigmas devem ocultar estes 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 através de um mecanismo contido em um pacote lógico específico. Quando objetos devam ser armazenados no banco de dados, as classes deste 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 deste sistema. Esta linguagem é um dialeto da linguagem SQL, cujos detalhes variam conforme o fabricante.

Além disto, os fabricantes oferecem componentes de acesso a bancos de dados, dos quais estas 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.

Pode-se colocar nos mecanismos de tradução lógica adequada para manter a consistência. Esta 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 através 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 este protocolo, as partes de uma transação são efetuadas de forma provisória. Só ao final da transação as modificações se tornam definitivas ("*comitted*"), 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 de fabricante, embora exista um esforço de padronização delas na linguagem SQL.

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 isto, 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, leiaute);
- 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 mias fáceis. As interfaces principais devem conter apenas os campos mais usados, colocando-se os demais em interfaces secundárias, acionadas apenas quando necessário.

1.4 Questões de memorização

O armazenamento e restauração de informação na memória humana é cansativo e sujeito 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 este 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. Uma interface só deve permitir o fechamento de uma tarefa complexa quando todas as subtarefas necessárias forem completadas com êxito.

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 destes. Um cardápio, por outro lado, apresenta diretamente as opções possíveis: o usuário tem apenas que reconhecer a que lhe interessa.

Para os usuários noviços, uma estratégia baseada no reconhecimento dos objetos e funções facilita o aprendizado, pois não é necessário decorar previamente listas de comandos. Isto 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 isto, 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 possíveis 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 funções das interfaces são metáforas de ferramentas não informatizadas do domínio da aplicação (Tabela 125).

Tarefa	Metáfora usual
Edição de texto	Máquina de escrever
Editoração de texto	Tipografia
Edição gráfica bidimensional	Ferramentas de desenho
Animação gráfica tridimensional	Estúdio de cinema

Tabela 125 - 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 caráter em um editor de texto, que este caráter 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 ações deles, 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 126).

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 126 - Exemplos de realimentação

A tolerância quanto aos tempos de resposta varia muito com a complexidade percebida das tarefas. Tarefas simples e freqüentes 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 melhor tolerada se os usuários receberem indicadores apropriados de estado. Neste 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 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" dão 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 deve 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, estas 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 127 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 127 - 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 conseqüência, ações aparentemente semelhantes podem ter resultados diferentes, dependendo do modo em que a interface está. De preferência, as interfaces não devem possuir modos. Quando isto 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 128 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 128 - 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; 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 ações deles, 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 isto, existem diversos graus de reversibilidade que devem ser considerados como alternativas de desenho.

O desenho da reversibilidade 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, estes devem ter o destaque necessário. Utilizar em excesso mecanismos de atração eqüivale 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 no máximo três tipos e quatro tamanhos de fonte em uma mesma tela;
- utilizar de preferência fontes com serifa, que são mais fáceis de ler;
- 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 uso deles, 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-as para áreas de fundo de textos de cor clara;

- utilizar cores de fundo e de primeiro plano contrastantes, desde que não sejam ambas 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 esta 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, deve-se observar as seguintes diretrizes:

- comandos concisos;
- mensagens concisas;
- ícones fáceis de reconhecer;
- 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.

Estas diferenças afetam as taxas de desempenho dos usuários. O perfil de erros mais freqüentes 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 destes mecanismos pode ser elevado.

Um grupo importante de diferenças individuais decorre de diferenças de experiência entre os usuários. Por exemplo, os usuários noviços 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 freqüência, 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 confirmação devem ser simplificados, evitando-se que contribuam para perda de tempo.

2 Estilos de interação

2.1 Janelas

2.1.1 Visão geral

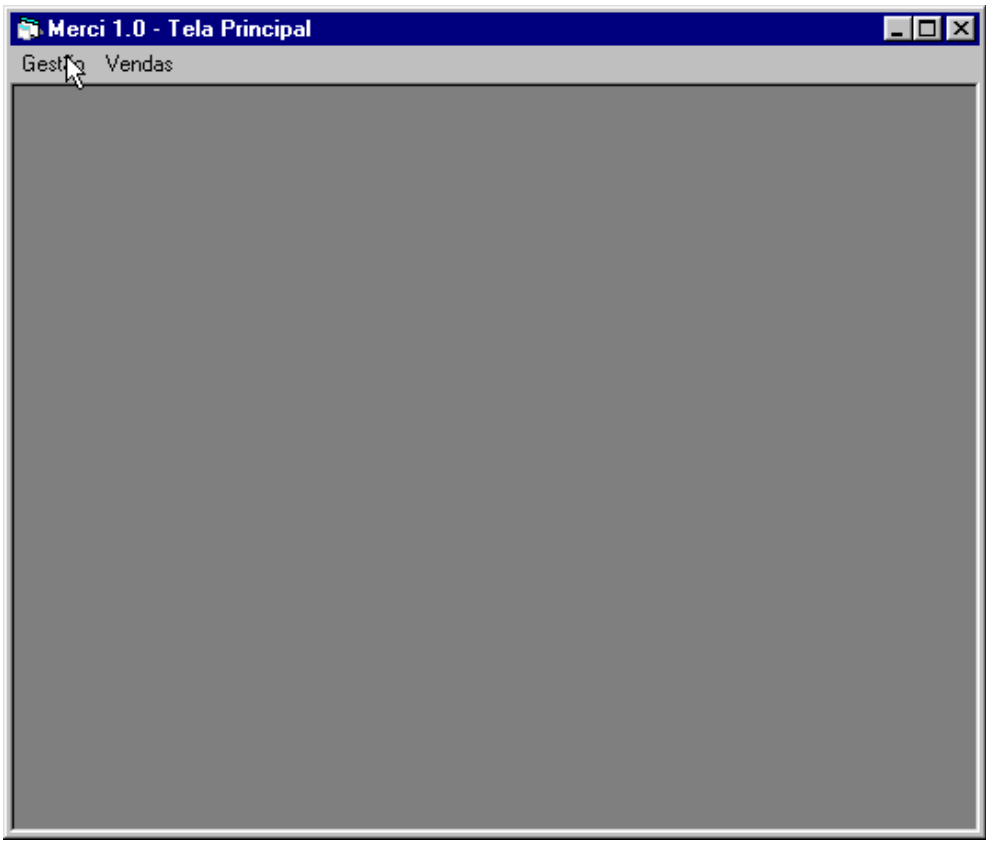

Figura 113 - Exemplo de janela primária

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 113) aparecem na abertura dos aplicativos, e através delas invocadas as janelas secundárias. Janelas secundárias (Figura

114) podem tomar formas especializadas, como caixas de diálogo. Janelas partidas (Figura 115) apresentam visões diferentes das mesmas estruturas: por exemplo, resumo e detalhes.

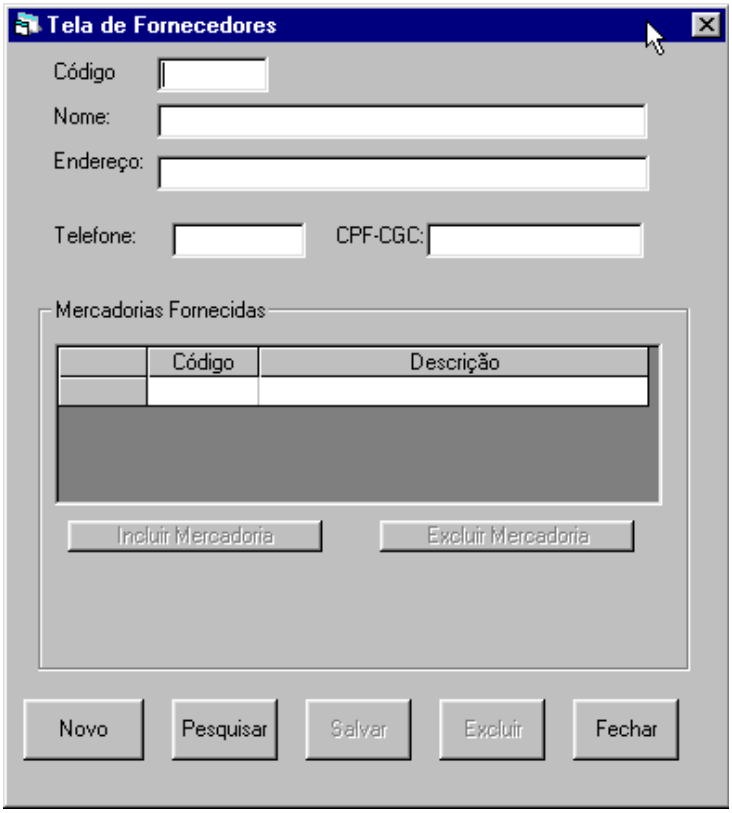

Figura 114 - Exemplo de janela secundária

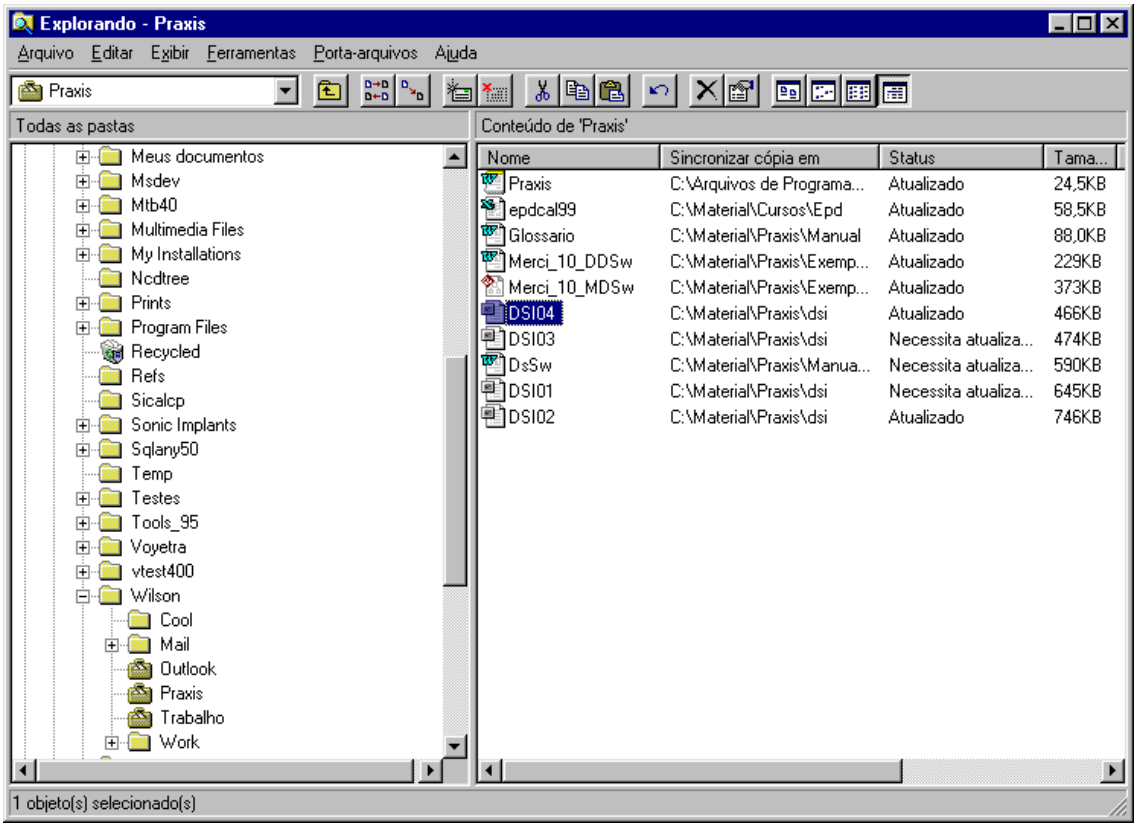

Figura 115 - Exemplo de janela partida

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 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 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 116 - 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*"[17]) apresentam um conjunto de opções que são mutuamente exclusivas. A opção selecionada é graficamente destacada das demais (Figura 116). Como todas as opções são simultaneamente exibidas, o que ocupa muito espaço de uma janela, este 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 exclusivas. As opções selecionadas são representadas visualmente por um sinal como X ou √. Na Figura 117, a interface da Figura 116 foi redesenhada para permitir a impressão conjunta de vários tipos de relatórios. Este estilo é adequado apenas quando o número de opções é relativamente pequeno, pelo espaço que ocupa na interface.

Figura 117 - Exemplo com botões de checar

2.2.2.4 Cardápios permanentes

Figura 118 - Cardápio permanente

Os cardápios permanentes ("*pull-down menus*") são sempre visíveis, sendo geralmente localizados na parte superior da interface (Figura 118). O cardápio é inteiramente exibido quando seu título é

[17] 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 119). Eles são geralmente usados para acionar uma operação aplicável ao objeto selecionado. Apenas as funções aplicáveis a este objeto podem ser acionadas, ajudando o usuário a escolher ações válidas. Além disto, estes cardápios reduzem a movimentação do mouse, tornando a interação mais rápida e menos cansativa.

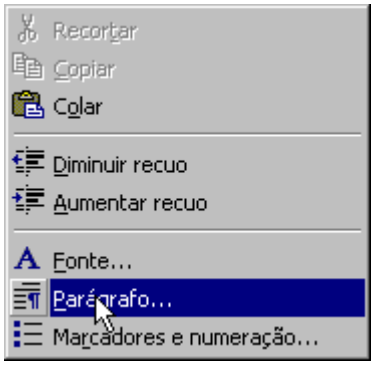

Figura 119 - Exemplo de cardápio instantâneo

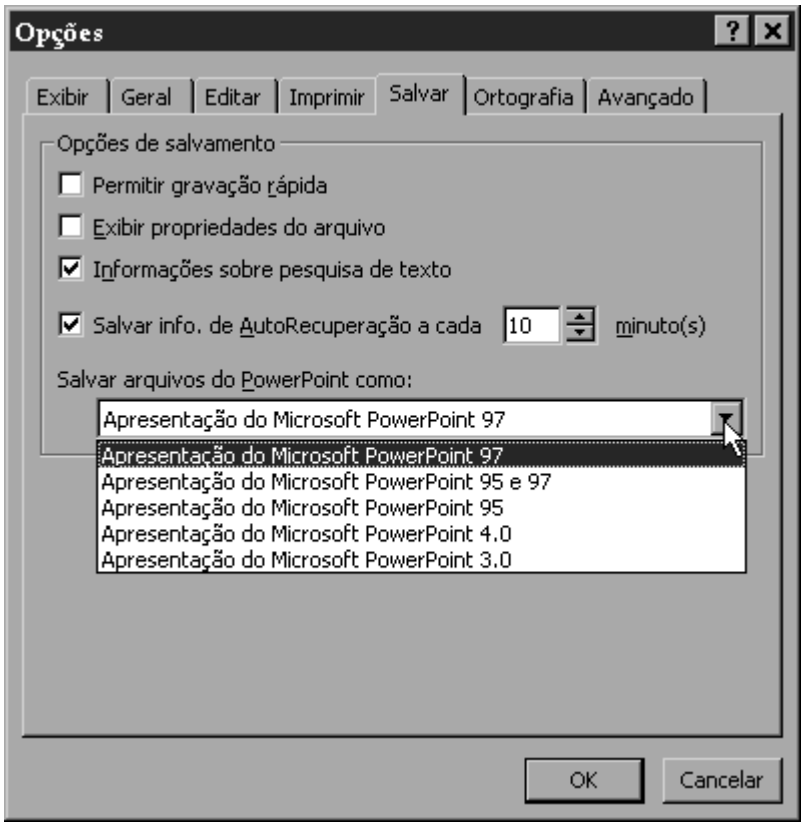

Figura 120 - Exemplo de interface com botões de checar e cardápio de opção

2.2.2.6 Cardápios de opção

Os cardápios de opção (Figura 120) mostram normalmente um único item. Os demais itens, que são mutuamente exclusivos, aparecem quando o mouse é pressionado sobre um indicador de expansão. Uma vez selecionado um item desta lista, ele passa a ser o único exibido. Este 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 121), 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. Este tipo de cardápio permite organizar de forma hierárquica cardápios com grande número de opções.

Figura 121 - Cardápio em cascata

2.2.2.8 Cardápios de paleta

Nos cardápios de paleta, as opções (mutuamente exclusivas) são representadas por ícones. Estes cardápios são muitos usados em aplicativos gráficos. Os ícones podem representar atributos que se quer aplicar ao objeto corrente, como cores (Figura 122), ou ferramentas (Figura 123), que são arrastadas até a área de trabalho.

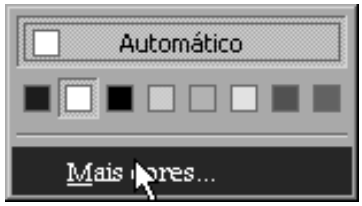

Figura 122 - Exemplo de cardápio de paleta de cores

Figura 123 - Exemplo de cardápio de paleta usado como caixa de ferramentas

2.2.2.9 Cardápios de hiperligações

As hiperligações, encontradas em documentos de hipertexto, e muito popularizadas pela WWW, podem ser usadas como itens de cardápio (Figura 124).

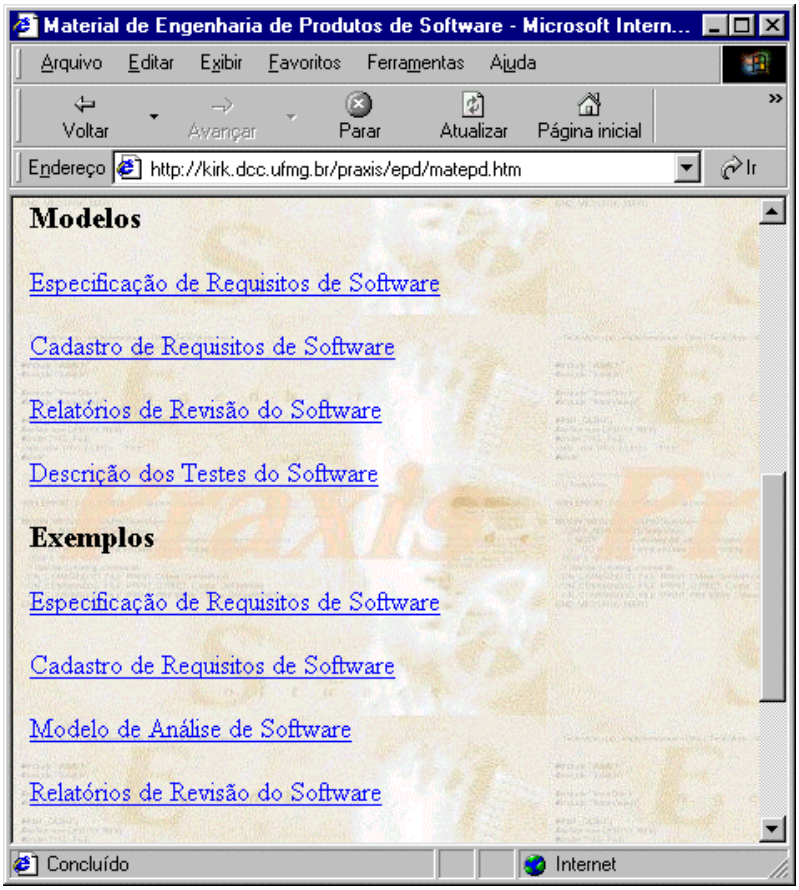

Figura 124 - Exemplo de cardápio de hiperligações

2.2.2.10 Cardápios dinâmicos

Os cardápios dinâmicos (lista de arquivos na parte inferior do cardápio da Figura 125) 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.

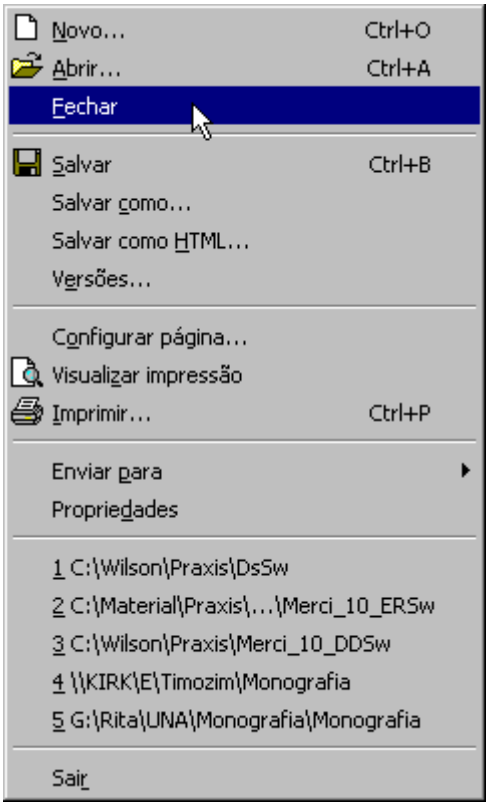

Figura 125 - Exemplo de cardápio com parte dinâmica

2.2.2.11 Mapas de imagem

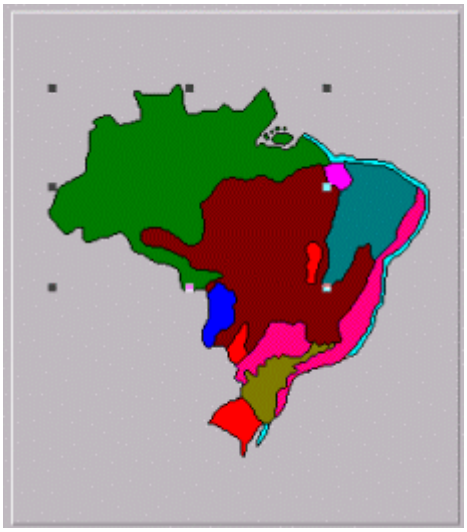

Figura 126 - Exemplo de mapa de imagem

Os mapas de imagem são cardápios nos quais os itens são constituídos por partes de uma imagem. Estes mapas são uma forma eficiente de interação em aplicativos gráficos, sendo também muito utilizados na WWW. Na Figura 126, cada região do mapa representa um bioma[18] diferente. Um clique sobre a representação de um bioma leva, neste aplicativo, a uma página com informação sobre as respectivas aves.

[18] 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. Estas 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 freqüência de uso ou por importância relativa).
- Usar para os itens do cardápio rótulos breves e significativos para o domínio de aplicação. A natureza gramatical deste rótulos deve ser consistente para todos os itens.
- Oferecer recursos aceleradores, como atalhos e macros. Pelo menos as funções de uso mais freqüente devem ser acessíveis através de atalhos.

Os seguintes aspectos dos cardápios devem ser usados de forma consistente, de preferência baseando-se 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 seqüência 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 129 - Tipos de campos de formulário

Os campos (Tabela 129) devem ser preenchidos inicialmente com um valor padrão, sempre que possível, 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 obter leiaute e conteúdo consistentes e visualmente atraentes:

- título claro e significativo;
- campos ordenados e logicamente agrupados;
- 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 ser eliminadas;
- existem campos não utilizados, que devam ser eliminados;
- todas as informações necessárias estão dispostas em campos próprios;
- há anotações extra nas margens ou 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 específicas o suficiente 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 usadas 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 127), 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 através da digitação dos primeiros caracteres do texto de um item.

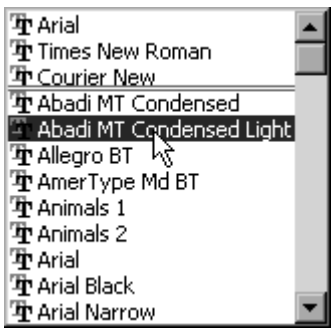

Figura 127 – Exemplo de caixa de lista

2.4.2.2 Caixas de entrada

Caixas de entrada são muito utilizadas para entrada e edição de texto (Figura 128). A caixa pode permitir uma única linha ou múltiplas linhas, e pode ter barras de rolagem para acomodar textos maiores.

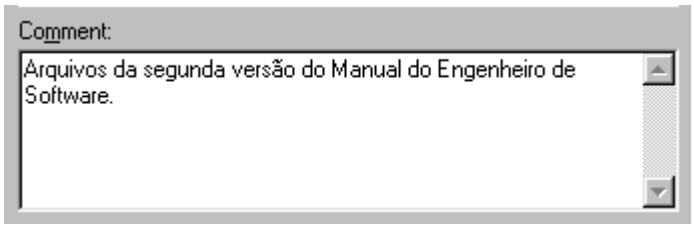

Figura 128 – Caixa de entrada

2.4.2.3 Caixas de mensagem

Caixas de mensagem são usadas 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 129).

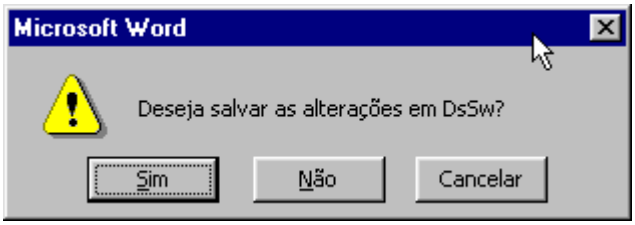

Figura 129 - Exemplo de caixa de mensagem modal

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 130) ou não modais (Figura 131).

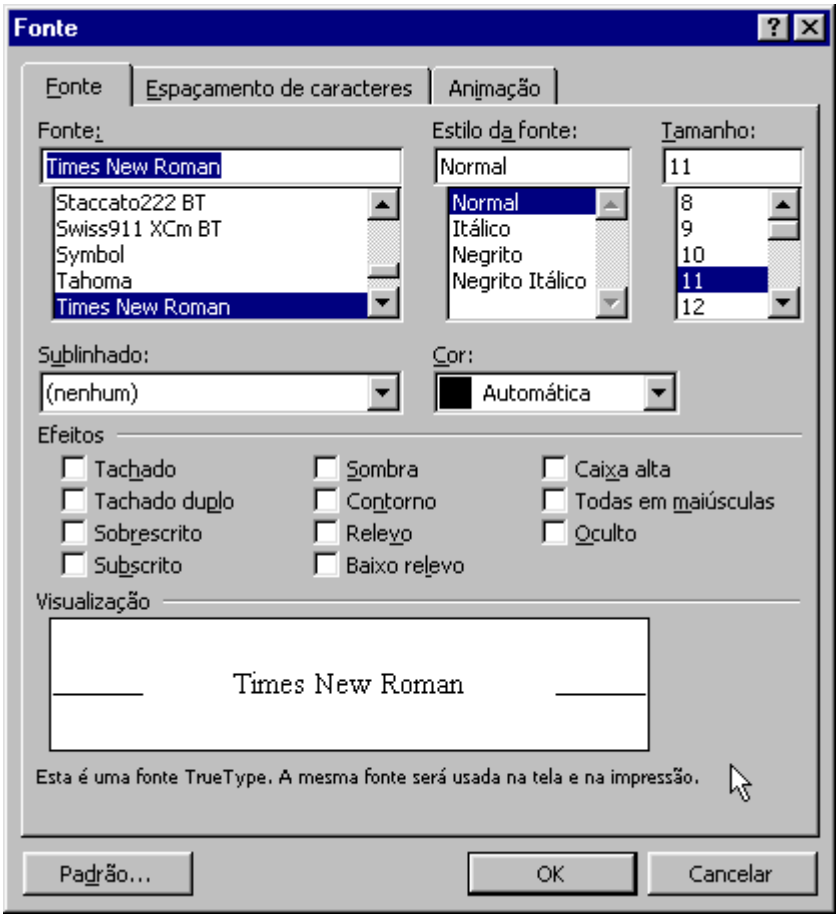

Figura 130 - Exemplo de caixa de diálogo modal

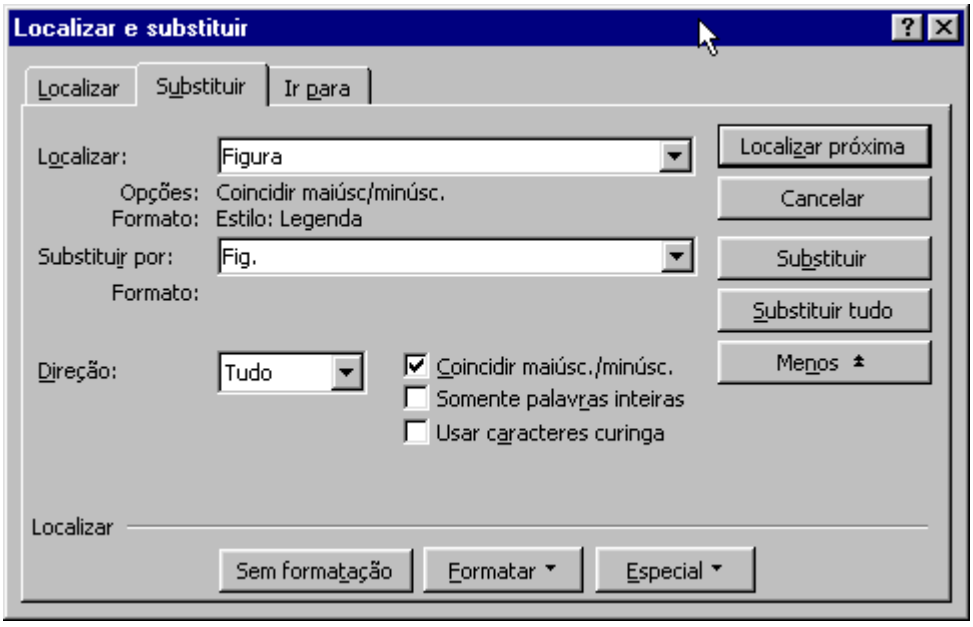

Figura 131 - Exemplo de caixa de diálogo não modal

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 as posições e alinhamentos dos elementos, de uma caixa para outra;
- 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 132) 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 de se lembrar precisamente da sintaxe empregada.

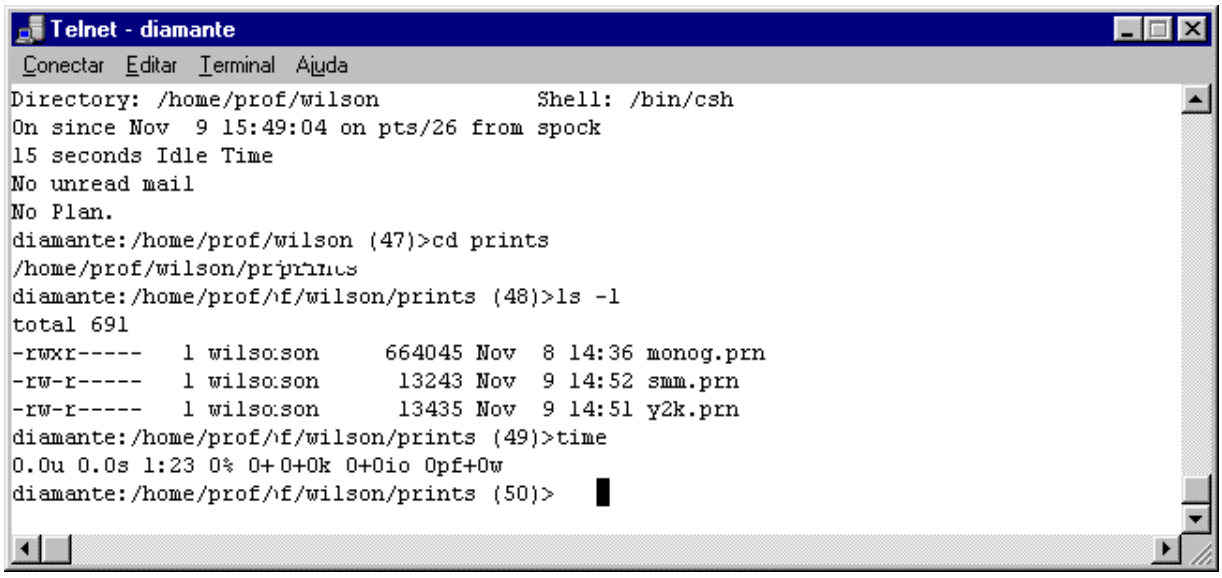

Figura 132 - Exemplo de linguagem de comando

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 ordem deles deve ser consistente de um comando para outro;
- os nomes devem ser específicos, significativos e distinguíveis, baseados em terminologia consistente;
- a abreviação regras de comandos deve obedecer a regras consistentes;
- a correção de erros de digitação deve ser fácil e rápida, sem ter de 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. Estas 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 tão simples quanto 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 130) 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 130 - 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** - usado para sistemas de atendimento telefônico, combinados com síntese de voz.
- **Reconhecimento de voz** - usado geralmente para comandos em situações em que as mãos estão ocupadas. Com a evolução da tecnologia, tendem a ser usados em muitas aplicações que atualmente são baseadas em teclados e janelas.