Capítulo 1

Conceitos básicos

Este capítulo tem por objetivo definir engenharia de software e explicar a sua importância para o desenvolvimento de sistemas de software. Para que esse objetivo seja atingido, são definidos outros conceitos importantes, tais como sistemas, software, produtos e processos de software.

1.1 Sistemas de software

A palavra *sistema* é definida no dicionário da língua portuguesa [FER 86] como "um conjunto de elementos concretos ou abstratos entre os quais se pode encontrar alguma relação". Bruno Maffeo [MAF 92] define um sistema em termos gerais como:

> Um conjunto, identificável e coerente, de elementos que interagem coesivamente, onde cada elemento pode ser um sistema.

Nessas duas definições, fica evidente a noção de relação estrutural que deve existir entre as partes componentes do sistema. Pode-se traçar uma fronteira conceitual separando o sistema do resto do universo e tratar o que está em seu interior como uma entidade relativamente autônoma e identificável. Os elementos que constituem o sistema, isto é, aqueles que estão dentro de seu contorno conceitual, têm entre eles interações fortes, quando comparadas às interações entre estes e os elementos do resto do universo. As interações entre os elementos constituintes do sistema e os demais elementos do universo devem ser suficientemente fracas para que possam ser desprezadas, quando se deseja considerar o sistema isolado. Por exemplo, em um estudo sobre ecologia, pode-se definir o ecossistema de uma certa espécie de inseto que passa toda a sua vida em uma única árvore de uma determinada espécie. A fronteira conceitual, nesse caso, envolveria o inseto (fisiologia, anatomia, hábitos reprodutivos, ciclo de vida etc.), a árvore, a fauna que habita a árvore e a caracterização do hábitat da árvore (que inclui clima, solo, vegetação na vizinhança etc.), como mostra a Figura 1.1.

Figura 1.1: Ecossistema de um inseto.

Se, por outro lado, o ecossistema fosse a floresta Amazônica, então a fronteira conceitual deveria envolver a fauna, a flora, a ocupação humana na floresta e em volta dela e a utilização que os seres humanos fazem da floresta, podendo inclusive chegar a aspectos atmosféricos.

Por analogia, quando se trata de *sistemas de software*, busca-se identificar componentes do sistema que interagem entre si para atender necessidades específicas do ambiente no qual estão inseridos. Existe também a possibilidade de haver um componente (sistêmico) humano. A escolha de uma fronteira conceitual adequada é um passo decisivo para o êxito do processo de concepção do sistema, pois a determinação dessa fronteira permitirá a separação entre os componentes pertencentes ao sistema, cujas informações devem ser detalhadamente estudadas, e aqueles pertencentes ao ambiente externo, que só têm interesse quanto a sua interação com o sistema.

Para exemplificar o conceito de fronteira conceitual de um sistema de software, pode-se considerar um *sistema de reservas de passagem aérea* de uma determinada companhia.

Para esse sistema, a fronteira conceitual só englobaria a própria companhia aérea e os dados sobre vôos (disponibilidade, reserva, cancelamento etc.). Se, por outro lado, o sistema de software incluísse reservas de hotel, locação de carros, ofertas de pacotes turísticos e assim por diante, a fronteira conceitual deveria ser muito mais abrangente, envolvendo informações sobre hotéis, locadoras de carros e agências de turismo. Sistemas de software entregues ao usuário com a documentação que descreve como instalar e usar o sistema são chamados *produtos de software* [SOM 96].

Existem duas categorias de produtos de software: (1) sistemas genéricos, produzidos e vendidos no mercado a qualquer pessoa que possa comprá-los; e (2) sistemas específicos, encomendados especialmente por um determinado cliente. O produto de software consiste em:

1) *instruções* (programas de computador) que, quando executadas, realizam as funções e têm o desempenho desejados;

2) *estruturas de dados* que possibilitam às instruções manipular as informações de forma adequada;

3) *documentos* que descrevem as operações e uso do produto.

Sistemas de software são produtos lógicos, não suscetíveis aos problemas do meio ambiente. No começo da vida de um sistema, há um alto índice de erros, mas, à medida que esses erros são corrigidos, o índice se estabiliza [PRE 92]. Com a introdução de mudanças, seja para corrigir erros descobertos após a entrega do produto, seja para adaptar o sistema a novas tecnologias de software e hardware, ou ainda para incluir novos requisitos do usuário, novos erros são também introduzidos, e o software começa a se deteriorar.

A maioria dos produtos de software é construída de acordo com as necessidades do usuário e não montada a partir de componentes já existentes, pois não existem catálogos de componentes de software; é possível comprar produtos de software, mas somente como uma unidade completa, não como componentes separados que podem ser utilizados na confecção de novos sistemas. Todavia, esta situação está mudando rapidamente, com a disseminação do conceito de software reutilizável, que privilegia a reutilização de componentes de produtos de software já existentes [SOM 96]. Uma grande vantagem deste tipo de abordagem é a redução dos custos de desenvolvimento como um todo, pois um menor número de componentes terá de ser desenvolvido e validado. Outras vantagens dessa abordagem são o aumento na confiabilidade do sistema, pois os componentes a ser reutilizados já foram testados em outros sistemas, e a redução dos riscos de desenvolvimento, pois existe menos incerteza sobre os custos de reutilização de software se comparados aos custos de desenvolvimento.

O *processo de desenvolvimento de software* envolve o conjunto de atividades e resultados associados a essas atividades, com o objetivo de construir o produto de software. Existem três atividades fundamentais, comuns a todos os processos de construção de software, apresentadas na Figura 1.2. São elas:

1) *desenvolvimento*: as funcionalidades e as restrições relativas à operacionalidade do produto são especificadas, e o software é produzido de acordo com essas especificações;

2) *validação*: o produto de software é validado para garantir que ele faça exatamente o que o usuário deseja;

3) *manutenção*: o software sofre correções, adaptações e ampliações para corrigir erros encontrados após a entrega do produto, atender os novos requisitos do usuário e incorporar mudanças na tecnologia.

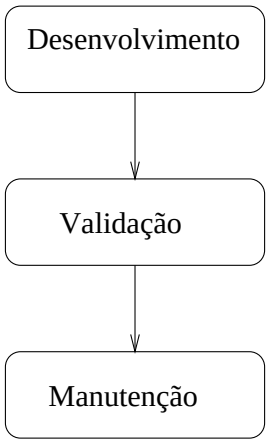

Figura 1.2: O processo de desenvolvimento de software.

O processo de desenvolvimento de software é complexo e envolve inúmeras atividades, e cada uma delas pode ser detalhada em vários passos a ser seguidos durante o desenvolvimento. *Modelos de processos* (também chamados *paradigmas de desenvolvimento*) especificam as atividades que, de acordo com o modelo, devem ser executadas, assim como a ordem em que devem ser executadas. Produtos de software podem ser construídos utilizando-se diferentes modelos de processos. No entanto, alguns modelos são mais adequados que outros para determinados tipos de aplicação, e a opção por um determinado modelo deve ser feita levando-se em consideração o produto a ser desenvolvido.

1.2 Engenharia de software

Durante as décadas de 60 e 70, o desafio primordial era desenvolver hardware que reduzisse o custo de processamento e de armazenamento de dados [PRE 92]. Durante a década de 80, avanços na microeletrônica resultaram em um aumento do poder computacional a um custo cada vez menor. Entretanto, tanto o processo de desenvolvimento como o software produzido ainda deixavam muito a desejar: cronogramas não eram cumpridos, custos excediam os previstos, o software não cumpria os requisitos estipulados e assim por diante. Portanto, o desafio primordial nas últimas duas décadas foi e continua sendo melhorar a qualidade e reduzir o custo do software produzido, através da introdução de disciplina no desenvolvimento, o que é conhecido como engenharia de software. Dessa forma, pode-se dizer que a engenharia de software é:

Uma disciplina que reúne metodologias, métodos e ferramentas a ser utilizados, desde a percepção do problema até o momento em que o sistema desenvolvido deixa de ser operacional, visando resolver problemas inerentes ao processo de desenvolvimento e ao produto de software.

O objetivo da engenharia de software é auxiliar no processo de produção de software, de forma que o processo dê origem a produtos de alta qualidade, produzidos mais rapidamente e a um custo cada vez menor. São muitos os problemas a ser tratados pela engenharia de software, pois tanto o processo quanto o produto de software possuem vários atributos que devem ser considerados para que se tenha sucesso, por exemplo, a complexidade, a visibilidade, a aceitabilidade, a confiabilidade, a manutenibilidade, a segurança etc. Por exemplo, para a especificação de sistemas de controle de tráfego aéreo e ferroviário, a confiabilidade é um atributo fundamental [FFO 00]. Já para sistemas mais simples, tais como os controladores embutidos em aparelhos eletrodomésticos, como lavadoras de roupa e videocassetes, o desempenho é o atributo mais importante a ser considerado.

A engenharia de software herda da engenharia o conceito de disciplina na produção de software, através de *metodologias*, que por sua vez seguem *métodos* que se utilizam de *ferramentas* automatizadas para englobar as principais atividades do processo de produção de software. Alguns métodos focalizam as funções do sistema; outros se concentram nos objetos que o povoam; outros, ainda, baseiam-se nos eventos que ocorrem durante o funcionamento do sistema.

Existem alguns princípios da engenharia de software que descrevem de maneira geral e abstrata as propriedades desejáveis para os processos e produtos de software. O desenvolvimento de software deve ser norteado por esses princípios, de forma que seus objetivos sejam alcançados.

1.3 Os princípios da engenharia de software

Existem vários princípios importantes e gerais que podem ser aplicados durante toda a fase de desenvolvimento do software [GUE 91]. Esses princípios se referem tanto ao processo como ao produto final e descrevem algumas propriedades gerais dos processos e produtos. O processo correto ajudará a produzir o produto correto, e o produto almejado também afetará a escolha do processo a ser utilizado. Entretanto, esses princípios por si sós não são suficientes para dirigir o desenvolvimento de software. Para aplicar esses princípios na construção de sistemas de software, o desenvolvedor deve estar equipado com as *metodologias* apropriadas e com os *métodos* e as *ferramentas* específicos que o ajudarão a incorporar as propriedades desejadas aos processos e produtos. Além disso, os princípios devem guiar a escolha das metodologias, métodos e ferramentas apropriados para o desenvolvimento de software. Alguns dos princípios a ser observados durante o desenvolvimento são descritos a seguir.

1.3.1 Formalidade

O desenvolvimento de software é uma atividade criativa e, como tal, tende a ser imprecisa e a seguir a "inspiração do momento" de uma maneira não estruturada. Através de um enfoque formal, pode-se produzir produtos mais confiáveis, controlar seu custo e ter mais confiança no seu desempenho. A formalidade não deve restringir a criatividade, mas melhorá-la através do aumento da confiança do desenvolvedor em resultados criativos, uma vez que eles são criticamente analisados à luz de uma avaliação formal. Em todo o campo da engenharia, o projeto acontece como uma seqüência de passos definidos com precisão. Em cada passo o desenvolvedor utiliza alguma metodologia que segue algum método, que pode ser formal, informal ou semiformal. Não há necessidade de ser sempre formal, mas o desenvolvedor tem de saber quando e como sê-lo. Por exemplo, se a tarefa atribuída ao engenheiro fosse projetar uma embarcação para atravessar de uma margem para a outra de um riacho calmo, talvez uma jangada fosse suficiente. Se, por outro lado, a tarefa fosse projetar uma embarcação para navegar através de um rio de águas turbulentas, na certa o engenheiro teria de fazer um projeto com especificações mais precisas para uma embarcação muito mais sofisticada. Finalmente, projetar uma embarcação para fazer uma viagem transatlântica demandaria maior formalidade na especificação do produto.

O mesmo acontece na engenharia de software. O desenvolvedor de software deve ser capaz de entender o nível de formalidade que deve ser atingido, dependendo da dificuldade conceitual da tarefa a ser executada. As partes consideradas críticas podem necessitar de uma descrição formal de suas funções, enquanto as partes mais bem entendidas e padronizadas podem necessitar de métodos mais simples. Tradicionalmente, o formalismo é usado somente na fase de programação, pois programas são componentes formais do sistema, com sintaxe e semântica totalmente definidas e que podem ser automaticamente manipulados pelos compiladores. Entretanto, a formalidade pode ser aplicada durante toda a fase de desenvolvimento do software, e seus efeitos benéficos podem ser sentidos na manutenção, reutilização, portabilidade e entendimento do software.

1.3.2 Abstração

Abstração é o processo de identificação dos aspectos importantes de um determinado fenômeno, ignorando-se os detalhes. Os detalhes a ser ignorados vão depender do objetivo da abstração. Por exemplo, para um telefone sem fio, uma abstração útil para o usuário seria um manual contendo a descrição dos efeitos de apertar os vários botões, o que permite que o telefone entre nos vários módulos de funcionalidade e reaja diferentemente às seqüências de comandos. Uma abstração útil para a pessoa ocupada em manter o telefone funcionando seria um manual contendo as informações sobre a caixa que deve ser aberta para substituir a pilha. Outras abstrações podem ser feitas para entender o funcionamento do telefone e as atividades necessárias para consertá-lo. Portanto, podem existir diferentes abstrações da mesma realidade, cada uma fornecendo uma *visão* diferente da realidade e servindo para diferentes objetivos. Quando requisitos de uma aplicação são analisados e especificados, desenvolvedores de software constroem modelos da aplicação proposta. Esses

modelos podem ser expressos de várias formas, dependendo do grau de formalidade desejado. As linguagens de programação, por exemplo, fornecem condições para que programas sejam escritos ignorando-se os detalhes, tais como o número de bits usado para representar números e caracteres ou o mecanismo de endereçamento utilizado. Isso permite que o programador se concentre no problema a ser resolvido e não na máquina. Os programas, por si sós, são abstrações das funcionalidades do sistema.

1.3.3 Decomposição

Uma das maneiras de lidar com a complexidade é subdividir o processo em atividades específicas, provavelmente atribuídas a especialistas de diferentes áreas. Podem-se separar essas atividades de várias formas, uma delas por tempo. Por exemplo, considere o caso de um médico-cirurgião que decide concentrar suas atividades cirúrgicas no período da manhã e seu atendimento aos pacientes no período da tarde, reservando as sextas-feiras para estudo e atualização. Essa separação permite o planejamento das atividades e diminui o tempo extra que seria gasto mudando de uma atividade para outra. No caso do software, pode-se separar, por exemplo, atividades relativas ao controle de qualidade do processo e do produto das atividades de desenvolvimento, como, por exemplo, especificação do projeto, implementação e manutenção, que são atividades bastante complexas. Além disso, cada uma dessas atividades pode ser decomposta, levando naturalmente à divisão das tarefas, possivelmente atribuídas a pessoas diferentes, com diferentes especialidades. A decomposição das atividades leva também à separação das preocupações. Por exemplo, pode-se lidar com a eficiência e correção de um dado produto de software separadamente, primeiro projetando-o de maneira cuidadosa e estruturada, de forma que garanta seu corretismo, e só então passando a reestruturá-lo parcialmente, de forma que melhore sua eficiência.

Além da decomposição do processo, também se pode decompor o produto em sub-produtos, definidos de acordo com o sistema que está sendo desenvolvido. Essa decomposição do produto traz inúmeras vantagens; por exemplo, permite que atividades do processo de desenvolvimento sejam executadas paralelamente. Além disso, dado que os componentes são independentes, eles podem ser reutilizados por outros componentes ou sistemas, e não haverá propagação de erros para outros componentes. A decomposição do produto pode ser feita, por exemplo, em termos dos objetos que povoam o sistema. Nesse caso, o produto será decomposto em um conjunto de objetos que se comunicam. Uma outra maneira de decompor o produto é considerando-se as funções que ele deve desempenhar. Nesse caso, o produto é decomposto em componentes funcionais que aceitam dados de entrada e os transformam em dados de saída. O objetivo maior, nos dois casos, é diminuir a complexidade.

1.3.4 Generalização

O princípio da generalização é importante pois, sendo mais geral, é bem possível que a solução para o problema tenha mais potencial para ser reutilizada. Também pode acontecer que através da generalização o desenvolvedor acabe projetando um componente que seja

utilizado em mais de um ponto do sistema de software desenvolvido, em vez de ter várias soluções especializadas. Por outro lado, uma solução generalizada pode ser mais custosa, em termos de velocidade de execução ou tempo de desenvolvimento. Portanto, é necessário avaliar os problemas de custo e eficiência para poder decidir se vale a pena desenvolver um sistema generalizado em vez de atender a especificação original. Por exemplo, supondo que seja necessário desenvolver um sistema de software para catalogar os livros de uma biblioteca pequena, em que cada livro tem um nome, autor, editor, data de publicação e um código específico. Além de catalogar os livros da biblioteca, deve ser possível fazer buscas baseadas na disponibilidade dos livros, por autor, por título, por palavras-chave etc. Em vez de especificar um conjunto de funcionalidades para o produto de software, envolvendo apenas essas funcionalidades, pode-se considerá-las como um caso especial de um conjunto mais geral de funcionalidades fornecidas por um sistema de biblioteca, como empréstimos, devoluções, aquisições etc. Esse sistema mais geral atenderia as necessidades do proprietário da pequena biblioteca e poderia interessar também a bibliotecas de médio porte, para as quais as outras funcionalidades são relevantes.

1.3.5 Flexibilização

O princípio da flexibilização diz respeito tanto ao processo como ao produto de software. O produto sofre constantes mudanças, pois em muitos casos a aplicação é desenvolvida enquanto seus requisitos ainda não foram totalmente entendidos. Isso ocorre porque o processo de desenvolvimento acontece passo a passo, de maneira incremental. Mesmo depois de entregue ao usuário, o produto pode sofrer alterações, seja para eliminar erros que não tenham sido detectados antes da entrega, seja para evoluir no sentido de atender as novas solicitações do usuário, seja para adaptar o produto de software às novas tecnologias de hardware e software. Os produtos são com freqüência inseridos em uma estrutura organizacional, o ambiente é afetado pela introdução do produto, e isso gera novos requisitos que não estavam presentes inicialmente.

O princípio da flexibilização é necessário no processo de desenvolvimento para permitir que o produto possa ser modificado com facilidade. O processo deve ter flexibilidade suficiente para permitir que partes ou componentes do produto desenvolvido possam ser utilizados em outros sistemas, bem como a sua portabilidade para diferentes sistemas computacionais.

A fim de alcançar esses princípios, a engenharia de software necessita de mecanismos para controlar o processo de desenvolvimento. Na próxima seção serão vistos três dos paradigmas mais utilizados no desenvolvimento de sistemas de software.

1.4 Paradigmas de engenharia de software

Paradigmas são modelos de processo que possibilitam: (a) ao gerente: controlar o processo de desenvolvimento de sistemas de software; e (b) ao desenvolvedor: obter a base para produzir, de maneira eficiente, software que satisfaça os requisitos preestabelecidos. Os paradigmas especificam algumas atividades que devem ser executadas, assim como a ordem em

que devem ser executadas. A função dos paradigmas é diminuir os problemas encontrados no processo de desenvolvimento do software, e, devido à importância desse processo, vários paradigmas já foram propostos. O paradigma é escolhido de acordo com a natureza do projeto e do produto a ser desenvolvido, dos métodos e ferramentas a ser utilizados e dos controles e produtos intermediários desejados. A seguir serão apresentados três dos paradigmas mais utilizados. São eles: *ciclo de vida clássico*, *evolutivo* e *espiral*.

1.4.1 Ciclo de vida clássico

É um paradigma que utiliza um método sistemático e seqüencial, em que o resultado de uma fase se constitui na entrada de outra. Devido à forma com que se dá a passagem de uma fase para outra, em ordem linear, esse paradigma também é conhecido como *cascata*. Cada fase é estruturada como um conjunto de atividades que podem ser executadas por pessoas diferentes, simultaneamente. Compreende as seguintes atividades, apresentadas na Figura 1.3: análise e especificação dos requisitos; projeto; implementação e teste unitário; integração e teste do sistema; e operação e manutenção. Existem inúmeras variações desse paradigma, dependendo da natureza das atividades e do fluxo de controle entre elas. Os estágios principais do paradigma estão relacionados às atividades fundamentais de desenvolvimento.

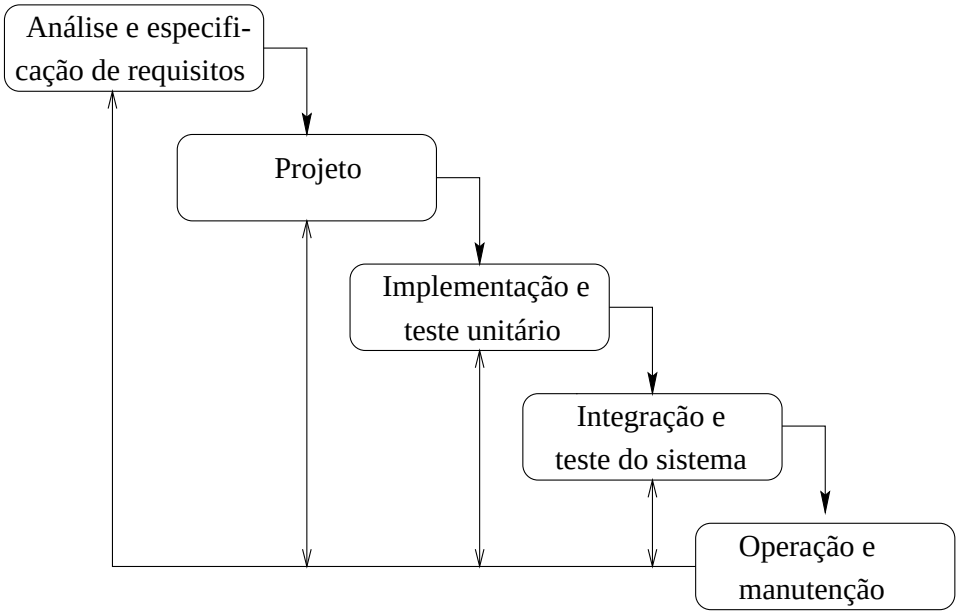

Figura 1.3: Os cinco passos do ciclo de vida clássico.

A opção pelo paradigma clássico vai depender da complexidade da aplicação. Aplicações simples e bem entendidas demandam menos formalidade; já aplicações maiores e mais complexas podem necessitar de uma maior decomposição do processo para garantir um melhor controle e obter os resultados almejados. Por exemplo, o desenvolvimento de uma aplicação

a ser utilizada por usuários não especialistas pode necessitar de uma fase na qual um material especial de treinamento é projetado e desenvolvido para tornar-se parte integrante do software; além disso, a fase em que o sistema entra em operação deve conter uma fase de treinamento. Por outro lado, se os usuários forem especialistas, a fase de treinamento pode não existir, sendo necessário somente o fornecimento de manuais técnicos. Outros detalhes podem ser fornecidos por telefone ou por um serviço de atendimento ao usuário.

Análise e especificação de requisitos

Durante essa fase do desenvolvimento, são identificados, através de consultas aos usuários do sistema, os serviços e as metas a ser atingidas, assim como as restrições a ser respeitadas. É, portanto, identificada a qualidade desejada para o sistema a ser desenvolvido, em termos de funcionalidade, desempenho, facilidade de uso, portabilidade etc. O desenvolvedor deve especificar *quais* os requisitos que o produto de software deverá possuir, sem especificar *como* esses requisitos serão obtidos através do projeto e da implementação. Por exemplo, ele deve definir quais funções o produto de software deverá desempenhar, sem dizer como uma certa estrutura ou algoritmo podem ajudar a realizá-las. Os requisitos especificados não devem restringir o desenvolvedor de software nas atividades de projeto e implementação; ele deve ter liberdade e responsabilidade para escolher a estrutura mais adequada, assim como para fazer outras escolhas relativas à implementação do software. O resultado dessa fase é um documento de especificação de requisitos, com dois objetivos: (1) o documento deve ser analisado e confirmado pelo usuário para verificar se ele satisfaz todas as suas expectativas; e (2) deve ser usado pelos desenvolvedores de software para obter um produto que satisfaça os requisitos.

Esse documento vai servir de instrumento de comunicação entre muitos indivíduos e, portanto, deve ser inteligível, preciso, completo, consistente e sem ambigüidades. Além disso, deve ser facilmente modificável, pois deve evoluir para acomodar a natureza evolutiva dos sistemas de software. As características da especificação podem variar, dependendo do contexto. Por exemplo, *precisão* pode significar formalidade para o desenvolvedor de software, ainda que especificações formais possam ser ininteligíveis para o usuário. Uma maneira de conciliar as necessidades do usuário e as do desenvolvedor é traduzir a especificação para um texto em língua natural e, talvez, complementar essa especificação com uma versão preliminar do manual do usuário, que descreva precisamente como o usuário deverá interagir futuramente com o sistema. Um outro produto da fase de análise e especificação de requisitos é o *plano de projeto*, baseado nos requisitos do produto a ser desenvolvido.

A fase de especificação de requisitos tem tarefas múltiplas e, para atingi-las, o desenvolvedor de software deve aplicar os princípios vistos na seção 1.3, especialmente *abstração*, *decomposição* e *generalização*. O software a ser produzido deve ser entendido e então descrito em diferentes níveis de abstração, dos aspectos gerais aos detalhes necessários. O produto deve ser dividido em partes, que possam ser analisadas separadamente. Uma possível lista do conteúdo do documento de especificação dos requisitos é a seguinte:

– *requisitos funcionais*: descrevem o que o produto de software faz, usando notações informais, semiformais, formais ou uma combinação delas;

– *requisitos não funcionais*: podem ser classificados nas categorias confiabilidade (disponibilidade, integridade, segurança), acurácia dos resultados, desempenho, problemas de interface homem–computador, restrições físicas e operacionais, questões de portabilidade etc.;

– *requisitos de desenvolvimento e manutenção*: incluem procedimentos de controle de qualidade — particularmente procedimentos de teste do sistema —, prioridades das funções desejadas, mudanças prováveis nos procedimentos de manutenção do sistema e outros.

Projeto

O projeto de software envolve a representação das funções do sistema em uma forma que possa ser transformada em um ou mais programas executáveis. Nessa fase, o produto é decomposto em partes tratáveis, e o resultado é um *documento de especificação do projeto*, que contém a descrição da arquitetura do software, isto é, o que cada parte deve fazer e a relação entre as partes. O processo de decomposição pode acontecer iterativamente e/ou pode ser feito através de níveis de abstração. Cada componente identificado em algum passo pode ser decomposto em subcomponentes. É comum distinguir entre projeto preliminar e detalhado, mas o significado desses termos pode variar. Para alguns, o projeto preliminar descreve a estrutura em termos de relações (por exemplo, *usa*, *é_composto_de*, *herda_de*), enquanto o projeto detalhado trata da especificação das interfaces. Outros usam esses termos para diferenciar entre a decomposição lógica (projeto em alto nível) e a decomposição física do programa em unidades de linguagem de programação. Outros, ainda, referem-se às principais estruturas de dados e aos algoritmos para cada módulo.

Implementação e teste unitário

Essa é a fase em que o projeto de software é transformado em um programa, ou unidades de programa, em uma determinada linguagem de programação. O resultado dessa fase é uma coleção de programas implementados e testados. A implementação também pode estar sujeita aos padrões da organização, que nesse caso pode definir o completo *layout* dos programas, tais como cabeçalhos para comentários, convenção para a utilização de nomes de variáveis e subprogramas, número máximo de linhas para cada componente e outros aspectos que a organização porventura achar importantes.

O teste unitário tem por objetivo verificar se cada unidade satisfaz suas especificações. Os testes são freqüentemente objeto de padronizações, que incluem uma definição precisa de um plano e critérios de testes a ser seguidos, definição do critério de completude (isto é, quando parar de testar) e o gerenciamento dos casos de teste. A depuração é uma atividade relacionada e também executada nessa fase. O teste das unidades é a principal atividade de controle de qualidade executada nessa fase, que ainda pode incluir inspeção de código para verificar se os programas estão de acordo com os padrões de codificação, estilo de programação disciplinada e outras qualidades do software, além da correção funcional (como, por exemplo, desempenho).

Integração e teste do sistema

Os programas ou unidades de programa são integrados e testados como um sistema completo para garantir que todos os seus requisitos sejam satisfeitos. A integração consiste na junção das unidades que foram desenvolvidas e testadas separadamente. Essa fase nem sempre é considerada separadamente da implementação; desenvolvimentos incrementais podem integrar e testar os componentes à medida que eles forem sendo desenvolvidos. Testes de integração envolvem testes das subpartes à medida que elas estão sendo integradas, e cada uma delas já deverá ter sido testada separadamente. Freqüentemente isso não é feito de uma só vez, mas progressivamente, em conjuntos cada vez maiores, a partir de pequenos subsistemas, até que o sistema inteiro seja construído. No final, são executados testes do sistema, e, uma vez que os testes tenham sido executados a contento, o produto de software pode ser colocado em uso. Padrões podem ser usados tanto na maneira como a integração é feita (por exemplo, *ascendente* ou *descendente*) como na maneira de projetar os dados de teste e documentar a atividade de teste. Depois de testado, o produto de software é entregue ao usuário.

Operação e manutenção

Normalmente esta é a fase mais longa do ciclo de vida. O sistema é instalado e colocado em uso. A entrega do software normalmente é feita em dois estágios. No primeiro, a aplicação é distribuída entre um grupo selecionado de usuários, para executar uma experiência controlada, obter *feedback* dos usuários e fazer alterações, se necessário, antes da entrega oficial. A manutenção é o conjunto de atividades executadas depois que o sistema é entregue aos usuários e consiste, basicamente, na correção dos erros remanescentes, que não foram descobertos nos estágios preliminares do ciclo de vida (manutenção corretiva), adaptação da aplicação às mudanças do ambiente (manutenção adaptativa), mudanças nos requisitos e adição de características e qualidades ao software (manutenção evolutiva).

Outras atividades

O paradigma clássico apresenta uma visão de desenvolvimento em fases. Algumas atividades, entretanto, são executadas antes que o ciclo de vida tenha início; outras, durante todo o ciclo de vida. Entre essas atividades estão: *estudo de viabilidade*, *documentação*, *verificação* e *gerenciamento*.

– **Estudo de viabilidade**:

  Esse estágio é crítico para o sucesso do resto do projeto, pois ninguém quer gastar tempo solucionando o problema errado. O conteúdo do estudo de viabilidade vai depender do tipo de desenvolvedor e do tipo de produto a ser desenvolvido. O objetivo dessa fase é produzir um *documento de estudo de viabilidade* que avalie os custos e benefícios da aplicação proposta. Para fazer isso, primeiro é necessário analisar o problema, pelo menos em nível global. Quanto melhor o problema for entendido, mais facilmente poderão ser identificados soluções alternativas, seus custos e potenciais benefícios para o usuário. Portanto, o ideal é fazer uma análise profunda do problema para produzir um estudo de viabilidade bem fundamentado. Na prática, entretanto, o estudo de

viabilidade é feito em um certo limite de tempo e sob pressão. Geralmente, o resultado desse estudo é uma oferta ao usuário potencial, e, como não se pode ter certeza de que a oferta será aceita, por razões econômicas não é possível investir muitos recursos nessa etapa. A identificação de soluções alternativas é baseada nessa análise preliminar, e, para cada solução, são analisados os custos e datas de entrega. Portanto, nessa fase é feita uma espécie de simulação do futuro processo de desenvolvimento, da qual é possível derivar informações que ajudem a decidir se o desenvolvimento vai valer a pena e, se for esse o caso, qual opção deve ser escolhida. O resultado do estudo de viabilidade é um documento que deve conter os seguintes itens: (a) a definição do problema; (b) soluções alternativas, com os benefícios esperados; e (c) as fontes necessárias, custos e datas de entrega para cada solução proposta.

– **Documentação**:

  Os produtos ou resultados de grande parte das fases são documentos; a mudança ou não de uma fase para outra vai depender desses documentos, e, normalmente, os padrões organizacionais definem a forma em que eles devem ser entregues.

– **Verificação e Validação**:

  Embora tenha sido dito que a verificação e a validação ocorrem em duas fases específicas (teste unitário e teste do sistema), elas são realizadas em várias outras fases. Em muitos casos, são executadas como um processo de controle de qualidade através de revisões e inspeções, com o objetivo de monitorar a qualidade do produto durante o desenvolvimento e não após a implementação. A descoberta e remoção de erros devem ser feitas o quanto antes para que a entrega do produto com erros seja evitada.

– **Gerenciamento**:

  A meta principal do gerenciamento é controlar o processo de desenvolvimento. Há três aspectos a ser considerados no gerenciamento. O primeiro diz respeito à *adaptação* do ciclo de vida ao processo, pois ele não deve ser tão rígido a ponto de ser aplicado exatamente da mesma forma a todos os produtos, indistintamente. Por exemplo, alguns procedimentos podem ser necessários para alguns produtos, mas excessivamente caros para aplicações mais simples. O segundo aspecto é a *definição de políticas*: como os produtos ou resultados intermediários vão ser armazenados, acessados e modificados, como as versões diferentes do sistema são construídas e quais as autorizações necessárias para acessar os componentes de entrada e saída do banco de dados do produto. Finalmente, o gerenciamento tem de lidar com todos os *recursos* que afetam o processo de produção de software, particularmente com os *recursos humanos*.

O paradigma do ciclo de vida clássico trouxe contribuições importantes para o processo de desenvolvimento de software, sendo as principais:

– o processo de desenvolvimento de software deve ser sujeito à disciplina, planejamento e gerenciamento;

– a implementação do produto deve ser postergada até que os objetivos tenham sido completamente entendidos.

Existem vários problemas com o paradigma clássico, sendo um deles a rigidez. O processo de desenvolvimento de software não é linear, envolve uma seqüência de iterações das atividades de desenvolvimento. Essas iterações estão representadas na Figura 1.3 pelas setas que retornam às atividades executadas anteriormente. O paradigma clássico, como proposto inicialmente, não contemplava essa volta às fases anteriores. Os resultados de uma fase eram "congelados" antes de se passar para a próxima fase. Dessa forma, o paradigma assumia que os requisitos e as especificações de projeto podiam ser "congelados" num estágio preliminar de desenvolvimento, quando o conhecimento sobre a aplicação pode ainda estar sujeito a mudanças.

Na prática, entretanto, todas essas fases se sobrepõem e fornecem informações umas para as outras. Durante a fase de projeto, podem ser identificados problemas com os requisitos; durante a implementação, podem surgir problemas com o projeto; durante a fase final do ciclo de vida, quando o software é instalado e posto em uso, erros e omissões nos requisitos originais podem ser descobertos, assim como erros de projeto e de implementação. Novas funcionalidades também podem ser identificadas, e modificações podem se tornar necessárias. Para fazer essas mudanças, pode ser necessário repetir alguns ou todos os estágios anteriores.

Todavia, a meta do paradigma clássico continua sendo tentar a linearidade, para manter o processo previsível e fácil de monitorar. Os planos são baseados nessa linearidade, e qualquer desvio é desencorajado, pois vai representar um desvio do plano original e, portanto, requerer um replanejamento.

Finalmente, nesse paradigma todo o planejamento é orientado para a entrega do produto de software em uma data única; toda a análise é executada antes do projeto e da implementação, e a entrega pode ocorrer muito tempo depois. Quando se cometem erros de análise e quando isto não é identificado durante as revisões, o produto pode ser entregue ao usuário com erros, depois de muito tempo e esforços terem sido gastos. Além disso, como o processo de desenvolvimento de sistemas complexos pode ser longo, demandando talvez anos, a aplicação pode ser entregue quando as necessidades do usuário já tiverem sido alteradas, o que vai requerer mudanças imediatas na aplicação.

Apesar de suas limitações, o paradigma do ciclo de vida clássico ainda é um modelo de processo bastante utilizado, especialmente quando os requisitos estão bem claros no início do desenvolvimento. Nas próximas seções, serão apresentados dois paradigmas alternativos que tentam sanar os problemas do ciclo de vida clássico, especialmente no que diz respeito ao "congelamento" dos requisitos.

1.4.2 O paradigma evolutivo

O paradigma evolutivo é baseado no desenvolvimento e implementação de um produto inicial, que é submetido aos comentários e críticas do usuário; o produto vai sendo refinado através de múltiplas versões, até que o produto de software almejado tenha sido desenvolvido.

As atividades de desenvolvimento e validação são desempenhadas paralelamente, com um rápido *feedback* entre elas. O paradigma evolutivo pode ser de dois tipos:

1) *Desenvolvimento exploratório*, em que o objetivo do processo é trabalhar junto do usuário para descobrir seus requisitos, de maneira incremental, até que o produto final seja obtido. O desenvolvimento começa com as partes do produto que são mais bem entendidas, e a evolução acontece quando novas características são adicionadas à medida que são sugeridas pelo usuário. O desenvolvimento exploratório é importante quando é difícil, ou mesmo impossível, estabeler uma especificação detalhada dos requisitos do sistema *a priori*. A Figura 1.4, adaptada de SOM 96, apresenta as atividades do desenvolvimento exploratório.

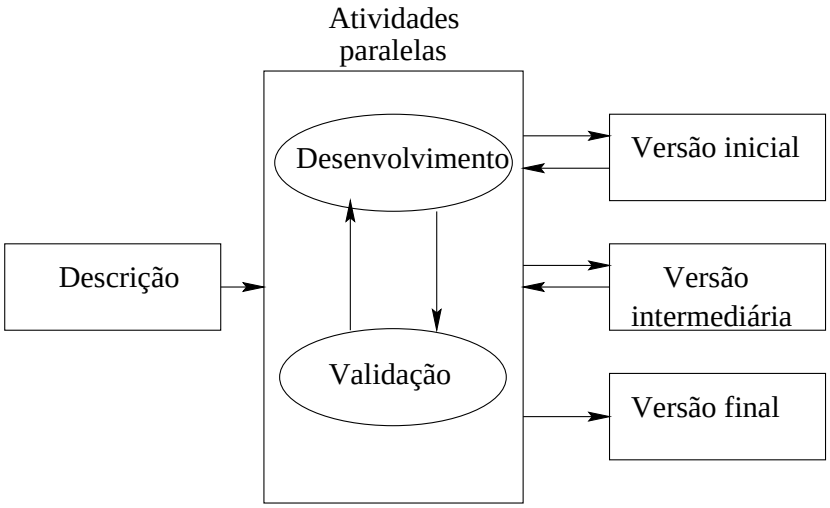

Figura 1.4: Desenvolvimento exploratório.

Primeiramente é desenvolvida uma versão inicial do produto, que é submetida a uma avaliação inicial do usuário. Essa versão é refinada, gerando várias versões, até que o produto almejado tenha sido desenvolvido.

2) *Protótipo descartável*, cujo objetivo é entender os requisitos do usuário e, conseqüentemente, obter uma melhor definição dos requisitos do sistema. O protótipo se concentra em fazer experimentos com os requisitos do usuário que não estão bem entendidos e envolve projeto, implementação e teste, mas não de maneira formal ou completa. Muitas vezes o usuário define uma série de objetivos para o produto de software, mas não consegue identificar detalhes de entrada, processamento ou requisitos de saída. Outras vezes, o desenvolvedor pode estar incerto sobre a eficiência de um algoritmo, a adaptação de um sistema operacional ou ainda sobre a forma da interação homem–máquina. Nessas situações, o protótipo pode ser a melhor opção. É um processo que possibilita ao desenvolvedor criar um modelo do software que será construído. Por um lado, o desenvolvedor pode perceber as reações iniciais do usuário e obter sugestões

para mudar ou inovar o protótipo; por outro, o usuário pode relacionar o que vê no protótipo diretamente com os seus requisitos. A Figura 1.5, adaptada de JAL 97, apresenta as atividades do paradigma do protótipo descartável.

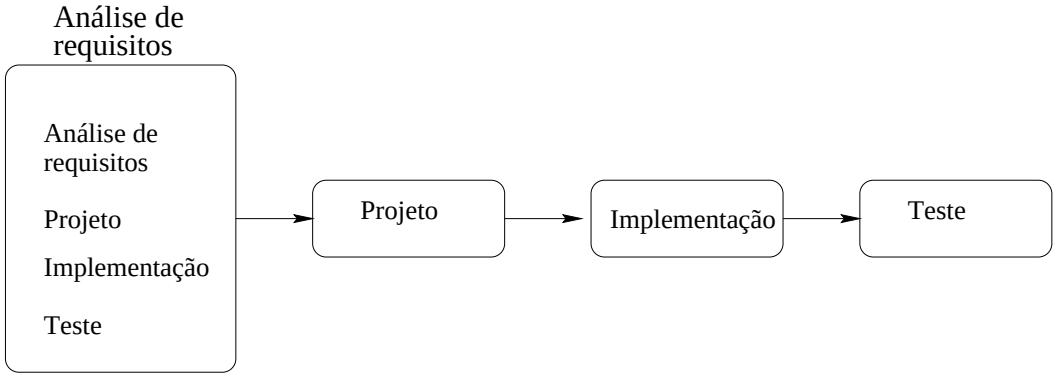

Figura 1.5: Protótipo descartável.

O desenvolvimento do protótipo descartável começa depois que uma versão preliminar da especificação de requisitos é obtida. Após o desenvolvimento do protótipo, é dada aos usuários a oportunidade de usá-lo e "brincar" com ele; baseados nessa experiência, eles dão opiniões sobre o que está correto, o que precisa ser modificado, o que está faltando, o que é desnecessário etc. Os desenvolvedores, então, modificam o protótipo para incorporar aquelas mudanças facilmente incorporáveis, e é dada aos usuários nova chance de utilizá-lo. Esse ciclo se repete até que os desenvolvedores concluam que o custo e o tempo envolvidos em novas mudanças não irão compensar as informações que porventura venham a ser obtidas. Baseados em todas as informações obtidas, os desenvolvedores, então, modificam os requisitos iniciais, com o objetivo de produzir a especificação de requisitos definitiva, que será usada para desenvolver o produto de software.

O paradigma evolutivo é normalmente mais efetivo do que o ciclo de vida clássico no desenvolvimento de produtos de software que atendam aos requisitos do usuário. Entretanto, sob a perspectiva de engenharia e do gerenciamento, o paradigma evolutivo apresenta os seguintes problemas:

- o processo não é visível, pois, como o desenvolvimento acontece de maneira rápida, não compensa produzir documentos que reflitam cada versão do produto de software. Entretanto, os gerentes de projeto precisam de documentos para medir o progresso no desenvolvimento;

- os sistemas são pobremente estruturados, pois as mudanças constantes tendem a corromper a estrutura do software. Portanto, a sua evolução provavelmente será difícil e custosa;

– quando um protótipo a ser descartado é construído, o usuário vê o que parece ser uma versão em funcionamento do produto de software, sem saber que o protótipo se mantém unido artificialmente e que, na pressa de colocá-lo em funcionamento, questões de qualidade e manutenção a longo prazo foram ignoradas. Quando informado de que o produto deve ser reconstruído, o usuário faz pressão para que algumas "pequenas" mudanças sejam feitas, para que o protótipo se transforme em produto final. Muitas vezes, o gerente de desenvolvimento de software cede;

– o desenvolvedor, muitas vezes, assume certos compromissos de implementação para garantir que o produto esteja funcionando rapidamente. Um sistema operacional ou uma linguagem inapropriados podem ser usados simplesmente porque estão disponíveis e são conhecidos; um algoritmo ineficiente pode ser implementado simplesmente para demonstrar as possibilidades do sistema. Depois de um certo tempo, o desenvolvedor pode se tornar familiarizado com essas escolhas e esquecer as razões pelas quais elas são inapropriadas. Uma escolha inapropriada pode se tornar parte integrante do sistema.

O desenvolvimento evolutivo é mais apropriado para sistemas pequenos, pois os problemas de mudanças no sistema atual podem ser contornados através da reimplementação do sistema todo quando mudanças substanciais se tornam necessárias. Uma outra vantagem deste tipo de abordagem é que os testes podem ser mais efetivos, visto que testar cada versão do sistema é provavelmente mais fácil do que testar o sistema todo no final.

1.4.3 O paradigma espiral

Também conhecido como paradigma de Boehm [BOE 91], foi desenvolvido para englobar as melhores características do ciclo de vida clássico e do paradigma evolutivo, ao mesmo tempo em que adiciona um novo elemento — a *análise de risco* — que não existe nos dois paradigmas anteriores.

Riscos são circunstâncias adversas que podem atrapalhar o processo de desenvolvimento e a qualidade do produto a ser desenvolvido. O paradigma espiral prevê a eliminação de problemas de alto risco através de um planejamento e projeto cuidadosos, em vez de tratar tanto problemas triviais como não triviais da mesma forma. Como o próprio nome sugere, as atividades do paradigma podem ser organizadas como uma espiral que tem muitos ciclos. Cada ciclo na espiral representa uma fase do processo de desenvolvimento do software. Portanto, o primeiro ciclo pode estar relacionado com o estudo de viabilidade e com a operacionalidade do sistema, isto é, com as funcionalidades e características do sistema e com o ambiente no qual será desenvolvido; o próximo ciclo pode estar relacionado com a definição dos requisitos, o próximo com o projeto do sistema e assim por diante. Não existem fases fixas neste paradigma. É durante o planejamento que se decide como estruturar o processo de desenvolvimento de software em fases. O paradigma, representado pela espiral da Figura 1.6, define quatro atividades principais representadas pelos quatro quadrantes da figura:

1) *Definição dos objetivos, alternativas e restrições*: os objetivos para a fase de desenvolvimento são definidos, tais como desempenho e funcionalidade, e são determinadas

alternativas para atingir esses objetivos; as restrições relativas ao processo e ao produto são também determinadas; um plano inicial de desenvolvimento é esboçado e os riscos de projeto são identificados. Estratégias alternativas, dependendo dos riscos detectados, podem ser planejadas, como, por exemplo, comprar o produto em vez de desenvolvê-lo.

2) *Análise de risco*: para cada um dos riscos identificados é feita uma análise cuidadosa. Atitudes são tomadas visando à redução desses riscos. Por exemplo, se houver risco de que os requisitos estejam inapropriados ou incompletos, um protótipo pode ser desenvolvido.

3) *Desenvolvimento e validação*: após a avaliação dos riscos, um paradigma de desenvolvimento é escolhido. Por exemplo, se os riscos de interface com o usuário forem predominantes, o paradigma de prototipagem evolutiva pode ser o mais apropriado.

4) *Planejamento*: o projeto é revisado, e a decisão de percorrer ou não mais um ciclo na espiral é tomada. Se a decisão for percorrer mais um ciclo, então o próximo passo do desenvolvimento do projeto deve ser planejado.

À medida que a volta na espiral acontece (começando de dentro para fora), versões mais completas do software vão sendo progressivamente construídas. Durante a primeira volta na espiral, são definidos objetivos, alternativas e restrições. Se a análise de risco indicar que há incerteza nos requisitos, a prototipagem pode ser usada para auxiliar o desenvolvedor e o usuário. Simulações e outros modelos podem ser usados para melhor definir o problema e refinar os requisitos. O processo de desenvolvimento tem início, e o usuário avalia o produto e faz sugestões de modificações. Com base nos comentários do usuário, ocorre então a próxima fase de planejamento e análise de risco. Cada volta na espiral resulta em uma decisão "continue" ou "não continue". Se os riscos forem muito grandes, o projeto pode ser descontinuado. Na maioria dos casos, o fluxo em volta da espiral continua, com cada passo levando os desenvolvedores em direção a um modelo mais completo do sistema e, em última instância, ao próprio sistema.

A diferença mais importante entre o paradigma espiral e os outros paradigmas é a análise de risco. O paradigma espiral possibilita ao desenvolvedor entender e reagir aos riscos em cada ciclo. Usa prototipagem como um mecanismo de redução de risco e mantém o desenvolvimento sistemático sugerido pelo ciclo de vida clássico. Incorpora, ainda, um componente iterativo que reflete o mundo mais realisticamente. Demanda uma consideração direta dos riscos envolvidos em todos os estágios do projeto e, se aplicado corretamente, reduz os riscos antes que eles se tornem problemáticos. Esse paradigma exige um especialista em análise de risco, e o sucesso em sua utilização reside exatamente no conhecimento desse especialista. Se um grande risco não for descoberto, problemas certamente ocorrerão. Os riscos podem ser diminuídos, ou mesmo sanados, através da descoberta de informações que venham a diminuir a incerteza com relação ao desenvolvimento.

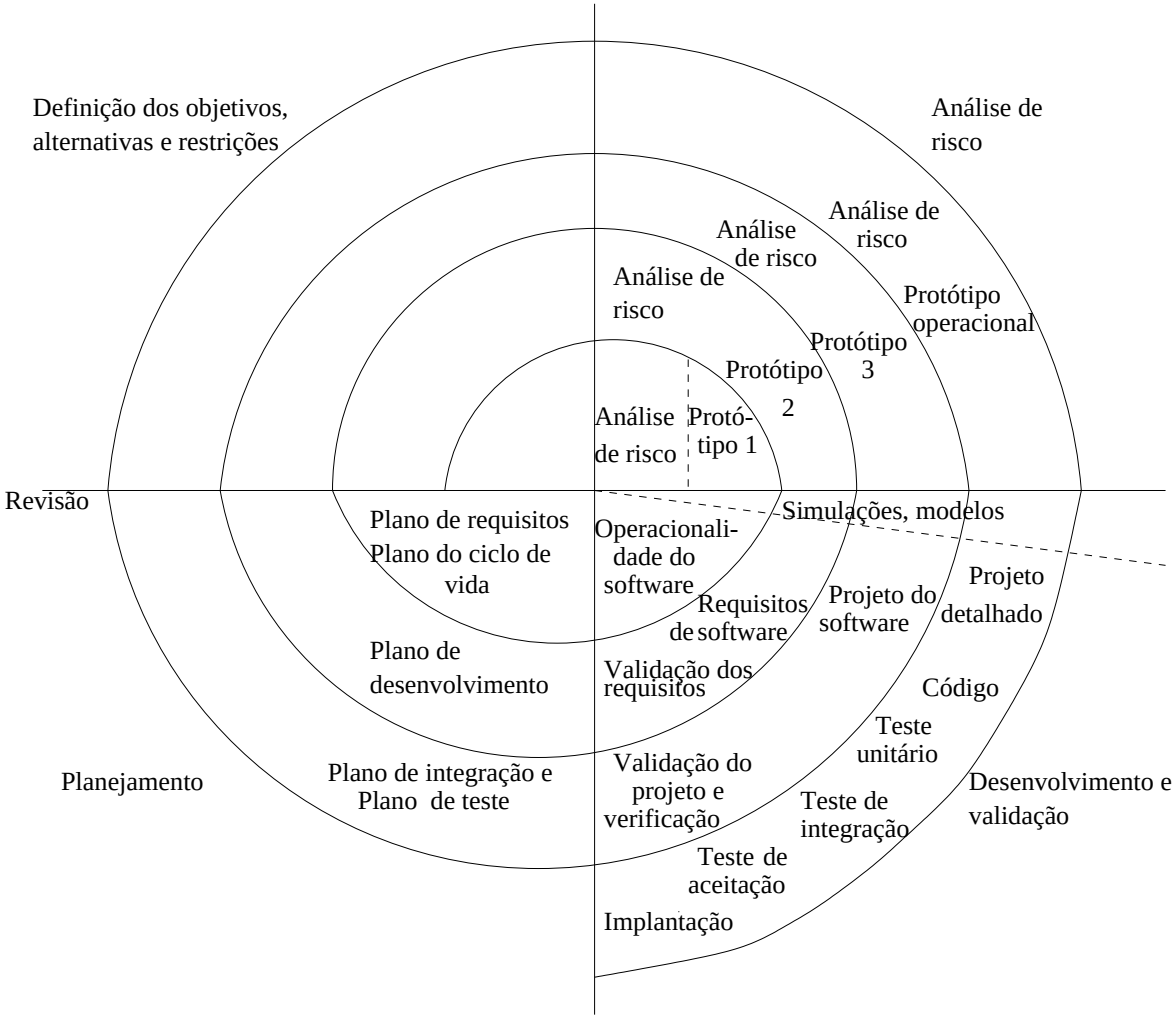

Figura 1.6: O paradigma espiral.

1.5 Engenharia de software influenciando e sendo influenciada por outras áreas dentro e fora da computação

A influência da engenharia de software pode ser notada em praticamente todas as áreas da computação e vice-versa. Entretanto, em algumas delas essa influência é mais óbvia. A seguir são apresentadas algumas dessas áreas.

A *teoria da computação* tem desenvolvido modelos que se tornaram ferramentas úteis para a engenharia de software, como, por exemplo, autômatos de estados finitos, que têm servido como base para o desenvolvimento de software. Autômatos têm sido utilizados, por exemplo, para especificação de sistemas operacionais, interfaces homem–computador e para a construção de processadores para tais especificações.

A *semântica formal* permite que se raciocine sobre as propriedades dos sistemas de software, e sua importância cresce proporcionalmente ao tamanho e complexidade do sistema que está sendo desenvolvido. Por exemplo, se os requisitos de um sistema de software que controla o fluxo em linhas de trem determinam que deve existir, no máximo, um trem em qualquer parte dos trilhos de uma determinada linha, então a semântica formal possibilita a produção de uma prova de que o software sempre terá esse comportamento [GEO 95].

As *linguagens de programação* são as principais ferramentas utilizadas no desenvolvimento de software e, por isso, têm uma influência muito grande no alcance dos objetivos da engenharia de software. Por outro lado, esses objetivos influenciam o desenvolvimento de linguagens de programação, visto que as linguagens mais modernas permitem a inclusão de características modulares, tais como a compilação independente e a separação entre especificação e implementação, para permitir o desenvolvimento de sistemas de software grandes, normalmente por uma equipe de desenvolvedores. Existem ainda outras linguagens que permitem o desenvolvimento de "pacotes", possibilitando a separação entre a interface e sua implementação, assim como bibliotecas de pacotes que podem ser usados como componentes no desenvolvimento de sistemas de software independentes.

O desenvolvimento dos *compiladores* modernos é feito de maneira modular e envolve basicamente dois componentes: análise da linguagem e geração do código propriamente dito. Essa decomposição permite a reutilização do segundo componente no desenvolvimento de outros compiladores. Essa técnica é usada na construção da família de compiladores GNU, por exemplo, para atender a diferentes arquiteturas e linguagens de programação de alto nível. A escrita do menor número possível de linhas de código só se torna viável através da reutilização de código, que é um conceito bastante utilizado em engenharia de software.

Os *sistemas operacionais* modernos fornecem ferramentas que possibilitam o gerenciamento da configuração do software, isto é, a manutenção e o controle das relações entre os diferentes componentes e versões do sistema de software. A grande vantagem do sistema operacional Unix [BOU 83] sobre seus antecessores é a sua organização como um conjunto de programas simples que podem ser combinados com grande flexibilidade, graças a uma "interface" comum (arquivos-texto).

Na área de *bancos de dados*, podem-se encontrar linguagens de consulta a bancos de dados, que permitem que aplicações usem os dados sem se preocupar com a representação interna deles. O banco de dados pode ser alterado para, por exemplo, melhorar o desempenho do sistema, sem nenhuma mudança na aplicação. Bancos de dados também podem ser usados como componentes de sistemas de software, visto que os primeiros já resolvem muitos dos problemas associados com o gerenciamento do acesso concorrente, por múltiplos usuários, a grandes quantidades de informações.

Sistemas multiagentes são sistemas complexos, cujo desenvolvimento envolve a decomposição do produto e a conseqüente separação das preocupações. Por exemplo, o desenvolvimento de um sistema multiagente para processamento de textos é um processo complexo, que pode ser decomposto em subprocessos (análise sintática, semântica, pragmática etc.) para resolver o problema em questão.

Sistemas especialistas são sistemas modulares, com dois componentes distintos: os fatos

conhecidos pelo sistema e as regras usadas para processar esses fatos, como, por exemplo, uma regra para decidir o curso de uma determinada ação. A idéia é que o conhecimento sobre uma determinada aplicação é dado pelo usuário, e os princípios gerais de como aplicar esse conhecimento a qualquer problema são fornecidos pelo próprio sistema.

Existem também outras áreas que têm tratado de problemas que não são específicos da engenharia de software, mas cujas soluções têm sido adaptadas a ela. Questões relativas ao gerenciamento, tais como estimativas de projeto, cronograma, planejamento dos recursos humanos, decomposição e atribuição de tarefas e acompanhamento do projeto, assim como questões pessoais envolvendo contratação e atribuição da tarefa certa à pessoa certa, estão diretamente relacionadas à engenharia de software. A ciência do gerenciamento estuda essas questões, e muitos dos modelos desenvolvidos nessa área podem ser aplicados à engenharia de software.

Outra área cujos estudos têm contribuído para a engenharia de software é a engenharia de sistemas, que estuda sistemas complexos, partindo da hipótese de que certas leis governam o comportamento de qualquer sistema complexo, que por sua vez é composto de muitos componentes com relações complexas. O software é normalmente um componente de um sistema muito maior, e, portanto, técnicas de engenharia de sistemas podem ser aplicadas no estudo desses sistemas.

1.6 Comentários finais

Em muitos casos, os paradigmas podem ser combinados de forma que os "pontos fortes" de cada um possam ser utilizados em um único projeto. O paradigma espiral já faz isso diretamente, combinando prototipagem e elementos do ciclo de vida clássico em um paradigma evolutivo. Entretanto, qualquer um desses pode servir como alicerce no qual outros paradigmas podem ser integrados. O processo sempre começa com a determinação dos objetivos, alternativas e restrições, que algumas vezes é chamada de obtenção de requisitos preliminar. Depois disso, qualquer caminho pode ser tomado. Por exemplo, os passos do ciclo de vida clássico podem ser seguidos se o sistema puder ser completamente especificado no começo. Se os requisitos não estiverem muito claros, um protótipo pode ser usado para melhor defini-los. Usando o protótipo como guia, o desenvolvedor pode retornar aos passos do ciclo de vida clássico (projeto, implementação e teste). Alternativamente, o protótipo pode evoluir para um sistema, retornando ao paradigma em cascata para ser testado. A natureza da aplicação é que vai determinar o paradigma a ser utilizado, e a combinação de paradigmas só tende a beneficiar o processo como um todo.

1.7 Exercício

1) Um gerente-geral de uma cadeia de lojas de presentes acredita que o único objetivo da construção de um protótipo é entender os requisitos do usuário e que depois esse protótipo será descartado. Portanto, ele acha bobagem gastar tempo e recursos em

algo que será desprezado mais tarde. Considerando essa relutância, resolva as seguintes questões:

(a) Compare brevemente o protótipo descartável com o desenvolvimento evolutivo, de forma que o gerente compreenda o que um protótipo pode significar.

(b) O gerente pensa em implementar o sistema, implantá-lo e testá-lo em uma loja e, depois, se obtiver sucesso, instalá-lo nas outras cinco lojas da cadeia. Diga qual método de prototipagem deve ser usado e justifique sua escolha.

Capítulo 2

Extração de requisitos

A palavra requisito é definida no dicionário da língua portuguesa [FER 86] como condição necessária para a obtenção de certo objetivo ou para o preenchimento de certo fim. Em se tratando de requisitos de software, o termo se refere aos requisitos que o produto a ser desenvolvido deve possuir [CHR 92]. Os problemas que os desenvolvedores de software são chamados para resolver são geralmente complexos, e o entendimento do problema pode ser uma tarefa bastante árdua. Se o produto de software não existir, então fica ainda mais difícil entender a natureza do problema e, conseqüentemente, o que o produto deve fazer exatamente.

Extração de requisitos, também chamada de engenharia de requisitos, é o processo de transformação das idéias que estão na mente dos usuários (a entrada) em um documento formal (saída) [PAN 97]; essa transformação só é possível através da determinação dos objetivos do produto e das restrições para a sua operacionalidade, através de uma análise do problema, documentação dos resultados e verificação do entendimento do problema. A saída do processo de extração de requisitos é um *documento de especificação dos requisitos*, que descreve *o que* o produto a ser desenvolvido deverá fazer, sem entretanto descrever *como* deve ser feito. Idealmente, esse documento deve ser completo e consistente; entretanto, a entrada para esse processo não tem nenhuma dessas propriedades, isto é, não é completa nem consistente. Como conseqüência, o processo de extração de requisitos não pode ser totalmente formal e, portanto, não pode ser totalmente automatizado.

Durante a extração de requisitos, o foco é o entendimento do produto a ser desenvolvido e de seus requisitos. Quanto mais complexo for o produto, mais difícil se torna o processo. O princípio da decomposição, visto na seção 1.3.3, ajuda a lidar com a complexidade. Os requisitos podem ser tanto funcionais, descrevendo um serviço ou função, como não funcionais, descrevendo, por exemplo, restrições ao processo de desenvolvimento ou ao tempo de resposta do sistema. O processo de extração de requisitos consiste nos seguintes passos, representados na Figura 2.1:

– *entendimento do domínio*: nessa fase, os desenvolvedores devem entender o domínio da aplicação o mais completamente possível;

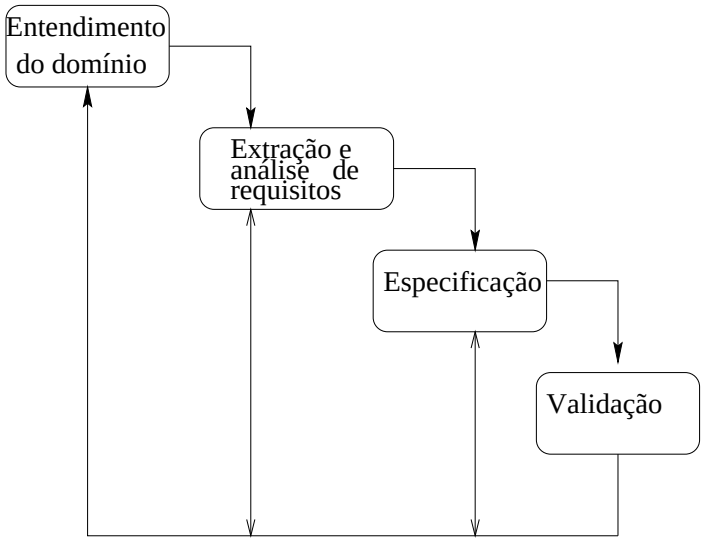

Figura 2.1: Processo de extração de requisitos.

- *extração e análise de requisitos*: nessa etapa acontece a descoberta, revelação e entendimento dos requisitos, através de interação com o(s) usuário(s); são feitas a classificação e organização dos requisitos, assim como a determinação de suas prioridades, resolução de inconsistências e conflitos e a descoberta de omissões;

- *especificação dos requisitos*: nessa etapa ocorre o armazenamento dos requisitos em uma ou mais formas, incluindo língua natural, linguagem semiformal ou formal, representações simbólicas ou gráficas;

- *validação dos requisitos*: nessa etapa é feita a verificação dos requisitos, visando determinar se estão completos e condizentes com as necessidades e desejos do usuário.

Embora possa parecer que o processo é uma seqüência linear dessas atividades, elas não podem ser totalmente separadas e executadas seqüencialmente; todas são intercaladas e executadas repetidamente. No mundo real, existe uma sobreposição e um *feedback* entre as atividades, representados na Figura 2.1 pelas setas que retornam às fases anteriores. Portanto, pode acontecer que algumas partes do produto sejam analisadas e especificadas enquanto outras ainda estão sendo analisadas. Além disso, a fase de validação pode revelar problemas com a especificação, o que pode acarretar um retorno à fase de análise e, conseqüentemente, à fase de especificação; problemas com o entendimento do domínio exigem um retorno a essa atividade.

As necessidades do usuário mudam à medida que o ambiente no qual o sistema funciona muda.

Com freqüência, o próprio documento de especificação de requisitos e o processo de extração dão novas idéias aos usuários sobre as suas necessidades e sobre as funções do sistema. Portanto, mudanças nos requisitos acontecem na maioria dos sistemas complexos. Embora

muitas delas sejam devidas a mudanças das necessidades dos usuários, outras advêm da interpretação incorreta dos requisitos do produto a ser desenvolvido. Requisitos incompletos, incorretos ou mal entendidos são as causas mais freqüentes da baixa qualidade, ultrapassagem dos custos previstos e atraso na entrega do produto de software.

Para lidar com a complexidade, a maioria das técnicas e ferramentas existentes se concentra na etapa de especificação e utiliza ferramentas para representar a informação e ajudar a enxergar o produto de software como uma série de abstrações. Este capítulo descreve algumas das técnicas mais utilizadas para extração e análise dos requisitos, visto que, indubitavelmente, essa é uma etapa crucial para o desenvolvimento de um produto de software que cumpra integralmente os requisitos do usuário. O capítulo 3 descreve os modelos utilizados para especificação de requisitos.

2.1 Dificuldades no processo de extração de requisitos

Como já mencionado anteriormente, o processo de extração de requisitos é impreciso e difícil, o que torna a sua total automatização, se não impossível, pouco provável. Pode-se utilizar como exemplo de análise de requisitos a seguinte descrição de um sistema hospitalar, feita por um usuário:

> Gostaria que fosse construído um sistema para monitorar a temperatura e a pressão de pacientes da UTI, que deverão ficar ligados *on-line* à rede de computadores do hospital, que é formada por um computador principal e vários terminais que monitoram os pacientes. Se a temperatura ou pressão do paciente lida pelo terminal se tornarem críticas, o computador principal deverá mostrar uma tela de alerta com um histórico das medidas realizadas para o paciente. Um aviso sonoro deve ser ativado nesse caso. A verificação da pressão é feita comparando-se a pressão do paciente com um valor padrão de pressão (máximo e mínimo) a ser digitado pelo responsável e verificando-se se a pressão medida está dentro dos parâmetros considerados normais para o paciente (valores próximos ao máximo e mínimo são permitidos). Temos vários sistemas *on-line* no computador e todos devem rodar ao mesmo tempo.

Para esse sistema, podem-se apontar as seguintes funções: monitorar temperatura e pressão; apresentar uma tela de alerta com o histórico de medidas; e providenciar um aviso sonoro de temperatura e pressão críticas. Como restrições podem-se considerar: o sistema deve ser *on-line*; o sistema deve rodar ao mesmo tempo que outros, necessitando, portanto, de controle de concorrência; e o aviso deve ser sonoro.

Existem várias dificuldades a ser contornadas no processo de extração de requisitos; algumas são mais óbvias, outras são implícitas. Todavia, todas devem ser levadas em consideração para que o processo de extração de requisitos atinja seus objetivos. Algumas dessas dificuldades são descritas a seguir:

1) *Falta de conhecimento do usuário das suas reais necessidades e do que o produto de software pode lhe oferecer.*

Muitas vezes os usuários, no momento da extração dos requisitos, têm somente uma vaga noção do que precisam e do que um produto de software pode lhes oferecer. O processo de extração de requisitos ajuda os usuários a explorar e entender suas reais necessidades, especialmente no que diz respeito às diferenças entre o que eles *querem* e o que *precisam*. Através de interações com o desenvolvedor, os usuários passam a entender as restrições que podem ser impostas ao produto de software pela tecnologia ou pelas práticas da própria empresa, assim como as alternativas, tanto tecnológicas quanto operacionais, e as escolhas que podem ser necessárias quando dois requisitos não podem ser totalmente satisfeitos. Dessa forma, os usuários passam a compartilhar com o desenvolvedor uma visão dos problemas e dos tipos de soluções possíveis, sentindo-se informados durante o processo e tornando-se, também, responsáveis pelo sucesso do projeto. Similarmente, os desenvolvedores de software que participaram do processo de extração de requisitos adquirem a confiança de que estão resolvendo o problema certo e de que este é viável sob o aspecto técnico e humano.

2) *Falta de conhecimento do desenvolvedor do domínio do problema.*

   Muitas vezes os desenvolvedores não têm conhecimento adequado do domínio, o que os leva a tomar decisões erradas. É imprescindível, portanto, que os desenvolvedores reconheçam e façam um esforço para entender o domínio da maneira mais completa possível, sendo os usuários a melhor e mais completa fonte de conhecimento.

3) *Domínio do processo de extração de requisitos pelos desenvolvedores de software.*

   Os desenvolvedores podem dominar o processo de extração, não ouvindo o que os usuários têm a dizer e forçando suas próprias visões e interpretações. Dessa forma, um clima de insatisfação é criado, o que pode resultar numa participação menos efetiva por parte dos usuários e em respostas menos completas às perguntas dos desenvolvedores. Estes, por sua vez, podem tomar decisões erradas devido à falta de entendimento das reais necessidades dos usuários. Os requisitos, quando são mal entendidos, podem mudar freqüentemente, resultando em demoras ou esforços desperdiçados nas fases de projeto e implementação. O resultado é um custo maior, atraso no planejamento e, algumas vezes, projetos cancelados.

4) *Comunicação inadequada entre desenvolvedores e usuários.*

   Os usuários podem estar cientes de suas necessidades, mas ser incapazes de expressá-las apropriadamente. Também pode ocorrer um mal-entendido entre usuários e desenvolvedores por atribuírem significados diferentes a termos comuns, pois ambos vêm de mundos diferentes, com vocabulários e *jargões* diferentes. Por exemplo, palavras tais como *implementação* têm significado diferente para o desenvolvedor e para o usuário: para o desenvolvedor, significa a escrita do código de programa e, para o usuário, a colocação do produto de software em funcionamento na organização. No exemplo do sistema hospitalar, a sentença "Se a temperatura ou pressão do paciente lida pelo terminal se tornarem críticas", encontrada na declaração dos requisitos do usuário,

pode indicar, para um profissional da área médica, uma temperatura maior que 40 graus centígrados. E para os desenvolvedores do sistema de software, o que isso poderá significar? Finalmente, a forma de comunicação normalmente utilizada entre desenvolvedores e usuários (língua natural) é inerentemente *ambígua*. Entretanto, ela é utilizada na extração de requisitos porque, usualmente, essa é a única linguagem comum a desenvolvedores e usuários. Outras formas de comunicação, como, por exemplo, diagramas e linguagens artificiais, também podem e devem ser utilizadas algumas vezes. Podem-se apontar algumas ambigüidades na declaração de requisitos relativa ao sistema hospitalar, como, por exemplo, a ambigüidade presente na sentença "Se a temperatura ou pressão do paciente lida pelo terminal se tornarem críticas, o computador principal deverá mostrar uma tela de alerta com um histórico das medidas realizadas para o paciente. Um aviso sonoro deve ser ativado nesse caso". Essa declaração pode levar a duas interpretações: (1) o terminal ativará um aviso sonoro; e/ou (2) o computador principal ativará um aviso sonoro. Finalmente, existe o problema das omissões, isto é, descrições que parecem triviais e por isso são omitidas, podendo trazer conseqüências graves para o desenvolvimento do sistema. Entre outras omissões da declaração de requisitos do sistema hospitalar, podem-se apontar as omissões presentes na frase "A verificação da pressão é feita comparando-se a pressão do paciente com um valor padrão de pressão (máximo e mínimo) a ser digitado pelo responsável e verificando-se se a pressão medida está dentro dos parâmetros considerados normais para o paciente (valores próximos ao máximo e mínimo são permitidos)". Foram omitidas, por exemplo, as seguintes informações: (a) Quais são os valores possíveis para máximo e mínimo a ser digitados pelo usuário? (b) O que ocorre se o usuário digitar o valor máximo menor que o mínimo? (c) E se o intervalo fornecido pelo usuário estiver fora de um valor normal para pressão? (d) O que significa valores próximos?

5) *Dificuldade de o usuário tomar decisões.*

   Os usuários podem ter muita dificuldade para decidir sobre algumas questões, pois podem não entender as conseqüências das decisões, ou as alternativas possíveis. Ademais, podem ter necessidades ou perspectivas diferentes sobre um produto de software a ser construído; geralmente os usuários se preocupam com atributos de alto nível, como usabilidade e confiança, e os desenvolvedores, com questões de baixo nível, como utilização de recursos, algoritmos etc.

6) *Problemas de comportamento.*

   A extração de requisitos é um processo social, e algumas vezes existem conflitos e ambigüidades nos papéis que os usuários e desenvolvedores desempenham no processo de extração. Cada usuário pode assumir que é responsabilidade de algum outro usuário contar ao desenvolvedor algum aspecto dos requisitos, e o resultado disso é que ninguém o conta. O desenvolvedor pode assumir que o usuário é um especialista no domínio e que dará toda a informação necessária, e o usuário pode assumir que o desenvolvedor fará as perguntas apropriadas para obter o conhecimento do domínio necessário.

Adicionalmente, o desenvolvimento de um produto de software para uma empresa normalmente resulta em uma expectativa ou medo de que a instalação do produto vá exigir uma mudança total no comportamento dos indivíduos, muitas vezes acarretando a perda do emprego. Isso pode causar omissão de informações aos desenvolvedores.

7) *Questões técnicas.*

Os problemas a ser resolvidos pelos produtos de software estão se tornando cada vez mais complexos, e os requisitos para esses sistemas são baseados em conhecimentos cada vez mais detalhados sobre o domínio do usuário. Além disso, os requisitos mudam com o tempo, e deve-se tomar cuidado para evitar que se termine o processo com uma série de requisitos considerados obsoletos no momento em que o processo de extração estiver terminado. As tecnologias de software e hardware mudam rapidamente, e os avanços tecnológicos podem tornar um requisito, anteriormente considerado caro ou complexo demais, totalmente possível. Existem muitas fontes de requisitos, e pode acontecer, por exemplo, que os gerentes exijam que certas tarefas sejam executadas ou que certas restrições sejam respeitadas, sem que o pessoal de suporte tenha conhecimento desse fato. Finalmente, a natureza do sistema impõe restrições no processo de extração de requisitos. Um novo sistema muito semelhante a vários outros previamente construídos pelo mesmo grupo de desenvolvimento pode se beneficiar dos esforços de extração de requisitos anteriores, assim como do retorno dado pelos usuários do sistema anterior. Já um sistema totalmente novo requer um esforço de extração de requisitos consideravelmente maior. Extração de requisitos para um produto de software típico depende fortemente de pesquisa de mercado, exame dos produtos competidores e algum tipo de comunicação com uma amostra de usuários típicos. Por outro lado, um produto de software que passa por uma série de versões através dos anos necessita de um processo de extração de requisitos contínuo para identificar defeitos na versão corrente e descobrir novos requisitos para melhorá-lo.

A identificação das dificuldades e problemas pode servir como ponto inicial para as questões a ser discutidas durante a aplicação das técnicas de extração de requisitos. As solicitações ou requisições do usuário podem ser classificadas de acordo com algumas características, que podem auxiliar no processo de extração de requisitos. São elas: a freqüência da requisição, a previsibilidade da solicitação e a atualização da informação. A lista abaixo exemplifica diversas requisições feitas por vários usuários.

1) Desejo receber diariamente uma lista das compras feitas no dia anterior. O relatório deve estar disponível até as 12 horas.

2) Quando a quantidade em estoque de um item for menor que o estoque de segurança, emita um pedido de compra para o item. Esse pedido de compra deve ser gerado até o final do expediente.

3) Qual é o valor do pedido de compra número 34923? O fornecedor precisa de confirmação e está ao telefone agora.

4) Qual é o total de pedidos feitos ao fornecedor X no período de março a agosto deste ano? Os dados precisam estar disponíveis amanhã.

5) Quantas vezes, nos últimos seis meses, o fornecedor X faltou ao seu compromisso quanto à data de entrega? Preciso dessa informação agora.

6) Necessito de um relatório do percentual de compras feitas em microempresas. A informação será necessária numa reunião de conselho no próximo mês.

7) Forneça-me o nome e telefone de funcionários que conheçam a língua francesa, tenham tido treinamento fora do país e sejam solteiros. A lista deve ser classificada por tempo de serviço. Quero essa informação agora.

Podem-se analisar as solicitações da seguinte forma:

– Qual é a freqüência da solicitação do cliente?

  Solicitação programada, como em 1. Nesse caso, a solicitação foi detectada durante a extração e análise dos requisitos e faz parte do produto de software.

  Solicitação disparada por um evento, como em 2. Esse tipo de solicitação também deve ter sido prevista para fazer parte do produto.

  Requisição eventual, como nas solicitações de 3 a 7. Neste caso, o produto desenvolvido deve ter flexibilidade suficiente para comportar tais solicitações. É necessário analisar o volume de tais solicitações durante a construção do produto de software.

– Quão previsível é a natureza da solicitação?

  Previsíveis. As solicitações com periodicidade definida ou disparadas por eventos, como em 1 e 2, são previsíveis por definição. Algumas solicitações eventuais também podem ser previsíveis, como é o caso da solicitação 3.

  Imprevisíveis. Solicitações eventuais, em que variam os elementos de dados e/ou os processamentos necessários para atender a solicitação, como nos casos de 4 a 7.

– Quão atuais devem ser os dados?

  Atualização imediata. É necessário que os dados sejam atualizados a cada transação.

  Atualização adiada. É suficiente que os dados sejam atualizados ao final de um período de tempo predeterminado.

A partir dessas informações, é possível avaliar a complexidade e o custo do processamento. Por exemplo, quando a solicitação é imprevisível e o resultado da informação precisa ser imediato, o custo de processamento é maior. Como exemplo pode-se considerar a solicitação 7. Quando a solicitação é previsível e o resultado não precisa ser imediato, o custo de processamento é menor, como, por exemplo, nas solicitações 1 e 2.

2.2 Participantes na extração de requisitos

A extração de requisitos pode envolver um número maior ou menor de pessoas, dependendo da complexidade e dos objetivos do produto de software a ser desenvolvido. O desenvolvedor, também chamado de engenheiro de requisitos, é o responsável pela produção dos requisitos e lidera o processo. Um dos conhecimentos imprescindíveis a um desenvolvedor é a habilidade de empregar um processo sistemático na extração de requisitos. Ele é freqüentemente auxiliado por outros desenvolvedores de software, por especialistas em documentação e pelo pessoal de apoio. Os usuários potenciais do produto são também envolvidos.

Por exemplo, se o produto de software for um "novo e melhor" processador de textos, um número significativo de usuários de processadores de textos já existentes no mercado deverá participar do processo de extração de requisitos. Eles poderão responder perguntas sobre o que gostam ou desgostam nos processadores que utilizam e sobre as características que desejam que estejam presentes no novo produto. Por outro lado, se o produto não tiver precedentes, será mais difícil extrair seus requisitos detalhados. Uma pesquisa de mercado pode ajudar a identificar a necessidade do sistema e seus requisitos gerais, mas os requisitos detalhados podem ter de surgir de uma série de protótipos, testes e avaliações com os usuários. Seja qual for o produto a ser desenvolvido, nenhuma pessoa sozinha consegue descobrir quais são seus requisitos. Devem existir sempre vários participantes no processo para que a extração de requisitos seja bem-sucedida.

2.3 Técnicas para extração e análise de requisitos

A extração e análise dos requisitos devem fornecer informações completas e consistentes para que a atividade de especificação seja desempenhada a contento. A maior dificuldade é obter toda a informação necessária, e a maior fonte de informações são as pessoas e os documentos relacionados. Dado que a quantidade de informações é muito grande, elas precisam ser organizadas para que se possa avaliar a sua consistência e completude, além de se resolver eventuais contradições sobre informações advindas de fontes diversas.

As técnicas de extração e análise de requisitos visam superar as várias dificuldades inerentes ao processo. Algumas tratam das dificuldades de comunicação, enquanto outras tratam de dificuldades técnicas ou de comportamento humano. Algumas são de alto nível no sentido de que são técnicas amplas para o processo de extração de requisitos; outras são de baixo nível, pois fornecem táticas específicas para a extração de detalhes sobre uma determinada parte do produto ou um usuário específico. Nenhuma técnica por si só é suficiente para projetos reais. O desenvolvedor deve ser capaz de escolher um conjunto de técnicas que melhor se adaptem ao produto a ser desenvolvido.

Existem alguns procedimentos genéricos, que devem fazer parte de qualquer processo de extração de requisitos. São eles:

- *Perguntar*. Identificar a pessoa apropriada, como o usuário do produto de software, e perguntar quais são os requisitos.

– *Observar e inferir.* Observar o comportamento dos usuários de um produto existente (manual ou automático) e então inferir suas necessidades a partir do seu comportamento.

– *Discutir e formular.* Discutir com os usuários suas necessidades e, juntamente com eles, formular um entendimento comum dos requisitos.

– *Negociar a partir de um conjunto-padrão.* Começar com um conjunto-padrão de requisitos ou características, negociar com os usuários quais dessas características serão incluídas, excluídas ou modificadas.

– *Estudar e identificar problemas.* Investigar os problemas para identificar os requisitos que podem melhorar o produto. Por exemplo, se o produto for muito lento, ele pode necessitar de um sistema de monitoramento complexo para identificar quais os requisitos que alterariam o sistema. Para um produto com milhões de usuários, uma pesquisa estatística através de questionários pode ser necessária para identificar problemas significativos.

– *Supor.* Quando não existe acesso ao usuário, ou para a criação de um produto inexistente, é preciso usar a intuição para identificar características ou funções que o usuário possa desejar. Por exemplo, se o produto a ser construído for um "novo e melhor" processador de textos para competir com outros produtos semelhantes, será muito útil estudar os produtos existentes para identicar seus pontos fracos. Se o objetivo for criar um produto sem precedentes, então a suposição pode ser muito útil; perguntar e discutir com usuários não seria apropriado, pois, no momento da extração de requisitos, eles ainda não terão sido identificados.

Nas próximas seções, serão descritas algumas das técnicas mais utilizadas na extração e análise de requisitos. As técnicas de extração de requisitos podem ser divididas em informais e formais. As técnicas informais são baseadas em comunicação estruturada e interação com o usuário, questionários, estudo de documentos etc. O modelo do problema e o do produto são construídos na mente dos desenvolvedores, que podem fazer uso de notações informais, que são traduzidas diretamente para o documento de especificação de requisitos. São exemplos de técnicas informais o *Joint Application Design (JAD)*, *brainstorming*, *entrevistas* e *PIECES*.

Já as técnicas formais pressupõem a construção de um modelo conceitual do problema sendo analisado, ou de um protótipo do produto de software a ser construído. Para a construção do modelo conceitual, geralmente é utilizado o princípio da decomposição, visto na seção 1.3.3, e são produzidas estruturas representando alguns aspectos do problema. São exemplos de modelos conceituais o modelo funcional, o modelo de dados e o modelo de objetos, que serão discutidos no capítulo 3. Quando a prototipagem é utilizada, o problema é analisado e os requisitos são entendidos através da interação com os usuários, a partir de um protótipo do produto. Neste capítulo serão descritas as técnicas informais e a prototipagem, amplamente utilizadas na análise e extração de requisitos de software. A modelagem conceitual, que será vista em detalhes no capítulo 3, embora não seja capaz de modelar todos

os aspectos do problema, pode e deve ser usada em conjunto com técnicas informais para focalizar aspectos específicos do software a ser construído.

2.3.1 Entrevistas

As entrevistas acontecem através de uma série de encontros com os usuários. Nos primeiros encontros, os usuários geralmente explicam o seu trabalho, o ambiente no qual atuam, as suas necessidades etc. [PAN 97]. Entretanto, entrevistar não é somente fazer perguntas; é uma técnica estruturada, que pode ser aprendida e na qual os desenvolvedores podem ganhar proficiência com o treino e a prática. Ela requer o desenvolvimento de algumas habilidades sociais gerais, habilidade de ouvir e o conhecimento de uma variedade de táticas de entrevista. Um entrevistador competente pode ajudar a entender e explorar os requisitos do produto de software, superando muitos dos problemas vistos na seção 1.1. A entrevista consta de quatro fases: identificação dos candidatos, preparação, condução da entrevista e finalização [RAG 94].

1) *Identificação dos candidatos para entrevista*

   A extração de requisitos através de entrevista começa com a identificação das pessoas a ser entrevistadas. Normalmente começa com o próprio financiador do projeto ou com os usuários do produto a ser desenvolvido. A extração de requisitos pode envolver a entrevista de muitas pessoas, mas não é necessário que todas sejam identificadas antes de começarem as entrevistas. Em cada entrevista é possível descobrir outras pessoas que devem ser entrevistadas, fazendo perguntas como "Com quem mais eu deveria conversar?" ou "Quem mais deverá usar o produto?". Também é preciso considerar as pessoas que não serão usuárias do produto, mas que irão interagir com os usuários. Essas interações poderão ser mudadas ou mesmo interrompidas depois que o produto for instalado, e assim podem-se minimizar os efeitos negativos dessas mudanças. Pode-se perguntar, por exemplo, "Quem mais interage com você?".

2) *Preparação para uma entrevista*

   Existem duas atividades principais no preparo de uma entrevista: agendar entrevistas com as pessoas envolvidas e preparar uma lista de questões. As entrevistas devem sempre ser agendadas com antecedência, por uma questão de cortesia e para que os entrevistados possam se preparar. Devem-se deixar claros os objetivos da entrevista e a sua duração; é preciso fornecer aos entrevistados material relevante para que possam se preparar de acordo. Os usuários devem ser relembrados da entrevista um ou dois dias antes; isso pode ajudar a garantir que eles realmente se preparem com antecedência. As entrevistas são algumas vezes gravadas, mas certas pessoas podem se sentir constrangidas com esse fato, o que pode atrapalhar a qualidade da informação obtida; portanto, é necessário pedir permissão ao entrevistado com antecedência para gravar a entrevista. Também deve ser preparada com antecedência uma lista de questões aos entrevistados. As entrevistas são usadas para extrair informações detalhadas sobre os

requisitos do produto; portanto, no momento de fazer as perguntas, o desenvolvedor já deve ter algumas idéias gerais sobre o tipo de produto a ser construído, e essas idéias deverão guiá-lo no preparo das questões. Por outro lado, não é possível preparar todas as questões com antecedência; as informações obtidas durante a entrevista abrirão espaço para novas perguntas, que serão formuladas à medida que a entrevista avançar. As perguntas devem seguir uma ordem lógica, geralmente devem ser agrupadas por assuntos relacionados. Finalmente, é preciso decidir quanto tempo dedicar a cada assunto.

3) *Condução da entrevista*

No início da entrevista, o entrevistador se apresenta ao entrevistado (no caso de ainda não se conhecerem). A seguir, o entrevistador faz uma breve revisão dos objetivos da entrevista, isto é, por que ela está acontecendo, que destino terá a informação obtida, os tipos de assunto que serão abordados, o tempo alocado a cada assunto etc. Durante essa revisão, é possível julgar quanto o entrevistado está preparado; raramente a falta de preparo do entrevistado pode levar ao adiamento da entrevista. Apesar de a entrevista ser baseada em questões já preparadas, existem habilidades e estratégias para comunicação oral que podem ser usadas para aumentar a qualidade da informação recebida. É preciso estar alerta para o fato de que a primeira resposta para a pergunta pode não estar necessariamente completa e correta; além disso, pode ser expressa numa linguagem desconhecida para o entrevistador. Nesse caso, a melhor alternativa é resumir, refrasear e mostrar as implicações do que o entrevistador está ouvindo, de forma que o entrevistado possa confirmar o seu entendimento. A sumarização é útil durante a entrevista toda e não só no final. Ela ajuda a confirmar o entendimento e pode trazer generalizações úteis e abstrações de alto nível. Existe ainda o problema da falta de conhecimento técnico por parte do entrevistado, o que normalmente não lhe dá uma boa idéia das implicações de um determinado requisito. É importante que se expliquem essas implicações para o usuário, que pode então decidir se é isso mesmo o que ele quer. De tempos em tempos, é útil fazer comentários com o entrevistado, além das questões sobre os requisitos de software, tais como "Estamos indo bem?", "Esquecemos de alguma coisa?", ou "Gastamos tempo suficiente nessa questão?". Questões assim ajudam o entrevistador a se certificar de que o processo está correndo dentro do esperado. Existem alguns tipos de questões de caráter geral que quase sempre são usados nas entrevistas, como, por exemplo, "Por que este produto está sendo desenvolvido?", "O que você espera dele?", "Quem são os outros usuários desse sistema?". Questões de caráter geral encorajam respostas não reprimidas e podem extrair uma grande quantidade de informação. Elas podem ser muito úteis quando não se conhece o suficiente sobre o produto para poder perguntar questões mais detalhadas. Por outro lado, questões específicas são úteis quando é preciso informar o usuário sobre um aspecto particular e forçar uma resposta detalhada ou precisa. Cuidado deve ser tomado para não induzir a resposta, como, por exemplo, "O relatório de vendas deveria ser produzido semanalmente?". Perguntas com respostas do tipo "sim" ou "não" permi-

tem que o entrevistado responda sem que precise de muito tempo para pensar; dessa forma, dependendo da personalidade do entrevistado, o entrevistador pode terminar com a sua visão sobre os requisitos e não com a visão do usuário.

Como já mencionado, os requisitos de software são geralmente muito complexos, e o usuário pode não possuir um entendimento completo de suas necessidades. Isso normalmente significa que uma única pergunta sobre um determinado tópico pode não produzir uma resposta completa ou significativa. Devem-se, portanto, explorar os tópicos com questões que os abordem de diferentes direções, ou em diferentes níveis de abstração. Deve-se também formular perguntas que subam o nível quando o entrevistado começar a se concentrar em detalhes, ou em uma única solução para o problema. Quando o entrevistado diz que uma função específica é necessária, pode-se fazer uma série de perguntas, tais como "Qual é o objetivo disso?", "Como o objetivo será obtido?". Durante a entrevista, são mudados os tópicos ou contextos das questões, cabendo ao entrevistador se certificar de que o entrevistado está entendendo o contexto no qual cada questão está sendo formulada. Por exemplo, se o entrevistador faz uma pergunta sobre o formato de um determinado dado, a resposta pode depender de se o contexto é uma discussão sobre dados de entrada ou de saída. A mudança de contexto com muita freqüência prolonga a entrevista e aumenta a confusão e, portanto, deve ser evitada. Durante a entrevista, o entrevistador deve estar preparado para erros de comunicação. É importante que ele verifique periodicamente se esses erros existem, reconhecendo-os e corrigindo-os. Alguns tipos de erros mais comuns são:

- Erros de observação: quando estão observando um determinado fenômeno, pessoas diferentes se concentram em diferentes aspectos e podem "ver" coisas diferentes.
- Erros de memória: o entrevistado pode estar confiando demais na lembrança de informações específicas, e a memória humana pode falhar.
- Erros de interpretação: o entrevistador e o entrevistado podem estar interpretando palavras comuns de maneira diferente, tais como "pequena quantidade de dados" ou "caracteres especiais".
- Erros de foco: o entrevistador pode estar pensando de maneira ampla, enquanto o entrevistado pode estar pensando de maneira restrita (ou vice-versa), o que afeta o nível de abstração na discussão daquele tópico.
- Ambigüidades: há ambigüidades inerentes à maioria das formas de comunicação, especialmente a língua natural.
- Conflitos: entrevistador e entrevistado podem ter opiniões conflitantes sobre um determinado problema, e a tendência é registrar seu próprio ponto de vista e não o que o entrevistado está dizendo.
- Fatos que simplesmente não são verdadeiros: o entrevistado pode dar informações que ele assume como fatos verdadeiros, mas que, na verdade, são só a sua opinião; o entrevistador deve se certificar sobre a veracidade desses fatos com outras fontes, especialmente aqueles nos quais se baseará para tomar decisões significativas.

4) *Finalização da entrevista*

A entrevista pode terminar quando todas as questões tiverem sido feitas e respondidas, quando o tempo alocado tiver se esgotado, ou quando o entrevistador sentir que o entrevistado está exausto. É importante reservar cinco ou dez minutos para sumariar e consolidar a informação recebida, descrevendo os principais tópicos adequadamente explorados, assim como aqueles que necessitam de informação adicional. As próximas ações a ser tomadas devem ser explicadas, incluindo a oportunidade para o entrevistado revisar e corrigir um resumo escrito da entrevista. Finalmente, deve-se agradecer o entrevistado pelo tempo e esforço dedicados.

Após a finalização da entrevista, há algumas poucas atividades a ser executadas. Como cortesia, é comum enviar ao entrevistado um agradecimento por escrito. A atividade mais importante posterior à entrevista é a produção de um resumo escrito, com o objetivo de reconhecer ou reordenar os tópicos discutidos e consolidar a informação obtida. Essa atividade também ajuda a descobrir ambigüidades, informação conflitante ou ausente. Se a entrevista tiver produzido informações estatísticas ou baseadas em fatos relatados de memória pelo entrevistado, elas devem ser confirmadas com fontes confiáveis. Finalmente, o entrevistador deve revisar os procedimentos utilizados para preparar e conduzir a entrevista, com o objetivo de encontrar maneiras de melhorar o processo no futuro.

2.3.2 Brainstorming

Brainstorming é uma técnica básica para geração de idéias. Ela consiste em uma ou várias reuniões que permitem que as pessoas sugiram e explorem idéias sem que sejam criticadas ou julgadas.

Entre os participantes da sessão (desenvolvedores e usuários), existe um *líder* cujo papel é fazer com que a sessão comece, sem restringi-la. A sessão consiste em duas fases: *geração de idéias* e *consolidação*. Na fase de geração, os participantes são encorajados a fornecer quantas idéias puderem, sem que haja discussão sobre o mérito delas. A técnica pode ser útil na geração de uma ampla variedade de visões do problema e na formulação do problema de diferentes maneiras.

Na fase de consolidação, as idéias são discutidas, revisadas e organizadas. O *brainstorming* é especialmente útil no começo do processo de extração de requisitos, pois a ausência de crítica e julgamento ajuda a eliminar algumas das dificuldades inerentes ao processo; a técnica estimula o pensamento imaginativo e, com isso, ajuda os usuários a se tornarem cientes de suas necessidades. Também evita a tendência a limitar o problema muito cedo e, para alguns tipos de pessoa, fornece uma interação social mais confortável do que algumas técnicas de grupo mais estruturadas. Uma vantagem adicional da técnica é a facilidade com que pode ser aprendida, com muito pouco investimento. Por outro lado, por ser um processo relativamente não estruturado, pode não produzir a mesma qualidade ou nível de detalhe de outros processos.

1) *Geração de idéias*

A preparação para uma sessão de *brainstorming* requer a identificação dos participantes, a designação do líder, o agendamento da sessão com todos os participantes e a preparação da sala de encontro. Os participantes são normalmente os usuários, além dos desenvolvedores do software. A saída da sessão depende das idéias geradas pelos participantes; portanto, é essencial incluir pessoas com conhecimento e especialidades apropriados. O líder abre a sessão falando sobre o problema de um modo geral, e os participantes, então, podem gerar novas idéias para expressar o problema. O processo continua enquanto novas idéias estiverem sendo geradas. Existem quatro regras a ser seguidas nessa fase de geração: (1) é terminantemente proibido criticar as idéias, pois os participantes devem se sentir totalmente confortáveis para expressar qualquer idéia; (2) idéias não convencionais ou estranhas são encorajadas, pois elas freqüentemente estimulam os participantes a irem em direções imprevisíveis, o que pode levar a soluções criativas para o problema; (3) o número de idéias geradas deve ser bem grande, pois quanto mais idéias forem propostas, maior será a chance de aparecerem boas idéias; e (4) os participantes também devem ser encorajados a combinar ou enriquecer as idéias de outros, e, para isso, é necessário que todas as idéias permaneçam visíveis a todos os participantes. Existem várias possibilidades para permitir que todo o material esteja visível o tempo todo; o método a ser escolhido vai depender do equipamento disponível na sala do encontro. O líder ou um "escrivão" é designado para registrar todas as idéias numa lousa branca ou de papel. À medida que cada folha de papel é preenchida, ela é colocada de forma que todos os participantes possam vê-la. Os participantes vão até o papel para registrar suas novas idéias. Várias folhas de papel menores são utilizadas e colocadas no meio da mesa, de forma que todos os participantes tenham acesso a elas. Quando uma nova idéia é proposta, ela é adicionada a qualquer uma das folhas. A fase de geração pode terminar de duas maneiras: (1) se o líder acreditar que não estão sendo geradas idéias suficientes, o encontro pode ser terminado; o grupo se retira e continua em outra ocasião; (2) se tiverem sido geradas e registradas idéias suficientes, o líder pode passar para a próxima fase.

2) *Consolidação das idéias*

A fase de consolidação permite que o grupo organize as idéias de maneira que possam ser mais bem utilizadas. É nessa fase que as idéias são avaliadas. Numa primeira etapa, as idéias são revisadas com o objetivo de esclarecê-las. Pode ser necessário refrasear algumas das idéais de forma que possam ser mais bem entendidas por todos os participantes. Durante essa fase, é comum que duas ou mais idéias sejam consideradas iguais, então elas podem ser combinadas e reescritas para capturar a sua essência original. Em seguida, os participantes podem concordar em que algumas das idéias são muito esquisitas para ser utilizadas e então são descartadas. As idéias remanescentes são então discutidas, com o objetivo de classificá-las em ordem de prioridade. Como se trata de requisitos de software, é freqüentemente necessário identificar aqueles absolutamente essenciais, aqueles que são bons, mas não essenciais, e aqueles que poderiam ser apro-

priados para uma versão subseqüente do produto de software. Depois da sessão, o líder ou outra pessoa que tenha sido designada produz um registro das idéias remanescentes, juntamente com suas prioridades ou outros comentários relevantes nessa fase de consolidação.

2.3.3 PIECES

Um dos grandes problemas para um desenvolvedor inexperiente é determinar como realmente começar. Não está claro quais as perguntas a ser feitas para extrair os requisitos do usuário. A técnica PIECES ajuda a resolver esse problema fornecendo um conjunto de categorias de questões que podem ajudar o analista a estruturar o processo de extração de requisitos. PIECES é uma sigla para seis categorias de questões a ser levadas em consideração: *desempenho* (ou *performance*), *informação e dados*, *economia*, *controle*, *eficiência* e *serviços*. Em cada categoria, existem várias questões que o desenvolvedor deve explorar com os usuários. A técnica pode ser adaptada para incluir questões iniciais ou básicas que sejam especialmente relevantes para o tipo de produto de software a ser construído. Como no caso das entrevistas, ela ajuda a lidar com dificuldades de articulação dos problemas e comunicação. A técnica PIECES é mais proveitosa na análise de produtos de software já existentes, sejam manuais ou automatizados [RAG 94]. A técnica pode também ser adaptada para domínios de aplicação específicos. Com a experiência, pode-se elaborar um conjunto de questões detalhadas para ajudar a garantir uma extração de requisitos bem fundamentada para novos produtos de software ou para produtos a ser melhorados. A seguir, são descritas as seis categorias de questões abordadas pela técnica PIECES:

1) *Desempenho*

   O desempenho de um sistema é usualmente medido de duas maneiras: (a) pelo número de tarefas completadas em uma unidade de tempo (*throughput*), tal como o número de pedidos processados no dia; e (b) pelo tempo de resposta, ou seja, a quantidade de tempo necessária para executar uma única tarefa. Para extrair os requisitos de desempenho, é preciso fazer perguntas que ajudem a identificar as tarefas que o produto deverá executar e então identificar o *throughput* ou tempo de resposta para cada tipo de tarefa. Durante a análise de um produto de software já existente, é possível descobrir se os usuários experientes já sabem onde existem problemas de desempenho.

2) *Informação e dados*

   Faz parte da natureza dos produtos de software o fornecimento de dados ou informações úteis para a tomada de decisão. Para ser mais efetivo, o software deve fornecer acesso ao tipo certo de informação, nem de mais nem de menos, no tempo certo e em forma utilizável. Esses pontos devem ser explorados com o usuário. Se os usuários tendem a não utilizar o produto, isso pode ser um sintoma de que informações erradas estão sendo fornecidas. Se eles o utilizam, mas expressam frustração, isso pode significar que o sistema apresenta muita informação, ou o faz de uma forma diferente daquela

que o usuário necessita. O sistema pode fornecer informação na forma de um relatório diário que seria necessário somente mensalmente, ou em um relatório mensal que seria necessário diariamente. Outras vezes, o relatório pode conter informação relevante, mas é uma tarefa entediante ter de consultar um relatório de cem páginas várias vezes ao dia; isso sugere que um acesso *on-line* pode ser melhor do que um relatório impresso.

3) *Economia*

Questões relacionadas ao custo de usar um produto de software são sempre importantes, e, de um modo geral, existem dois fatores de custo inter-relacionados que podem ser considerados no desenvolvimento de um sistema de software: nível de serviço e capacidade de lidar com alta demanda. O nível de serviço é a medida do desempenho do sistema (*throughput*, tempo de resposta, ou ambos). Para alguns produtos, a demanda varia consideravelmente de minuto a minuto, ou de hora em hora, mas os usuários gostariam de ter um nível de serviço ou desempenho relativamente estáveis. Isso pode ser conseguido embutindo-se no produto a capacidade de lidar com a alta demanda necessária nas horas de pico. A capacidade de lidar com a alta demanda em um produto de software pode significar ter processadores adicionais, unidades de disco ou conexões de rede, ou o projeto de estruturas de dados internas para armazenar informações de tamanho ou complexidade não previsíveis de tempos em tempos. Essa capacidade de lidar com alta demanda pode ser cara, e, portanto, essas questões devem ser discutidas com os usuários; um completo entendimento da carga esperada e do nível de serviço necessário ao produto ajudará os desenvolvedores a tomar decisões que levem em conta o nível de serviço e a capacidade de lidar com alta demanda.

4) *Controle*

Os sistemas são normalmente projetados para ter desempenho e saídas previsíveis. Quando o sistema se desvia do desempenho esperado, algum controle deve ser ativado para tomar ações corretivas. Em sistemas de tempo real, o controle é exercido diretamente pelo software. Segurança é um tipo de controle importante para alguns produtos de software; o acesso ao sistema pode ser restrito a certos usuários ou a certas horas do dia. O acesso a algumas informações também pode ser restrito a certos usuários, assim como o tipo de acesso (por exemplo, somente leitura ou leitura e escrita). Outro tipo de controle é a auditoria, ou seja, a habilidade de ver, monitorar ou reconstruir o comportamento do sistema, durante ou depois da execução do processo. A extração de requisitos deve abordar essas questões de controle cuidadosamente, pois, do contrário, pode ser construído um sistema que fornece pouco controle ou controle em excesso. No primeiro caso, o processo pode fugir de controle e, no segundo, o excesso pode impedir que o trabalho seja executado.

5) *Eficiência*

Não é sempre garantido que a energia e os recursos aplicados a uma tarefa realmente produzam trabalho útil; algumas vezes há uma perda. Eficiência é a medida dessa

perda, normalmente definida como a relação entre os recursos que resultam em trabalho útil e o total dos recursos gastos. Eficiência é diferente de economia; para melhorar a economia do processo, a quantidade total de recursos utilizados deve ser reduzida; para melhorar a eficiência, a perda no uso desses recursos deve ser reduzida. Existem muitas maneiras de melhorar a eficiência de um produto de software. Durante a extração de requisitos, podem-se explorar essas oportunidades de melhoria da eficiência com os usuários. Algumas ineficiências podem ser caracterizadas como redundâncias desnecessárias, como, por exemplo, coletar o mesmo dado mais de uma vez, armazená-lo em espaços múltiplos ou computar um determinado valor mais de uma vez. Ineficiências podem resultar do uso de algoritmos e estruturas de dados pobres. Uma interface pobre pode ocasionar perda de tempo do usuário.

6) *Serviços*

Um produto de software fornece serviços aos usuários, e pode ser muito útil pensar em termos de serviços durante o processo de extração de requisitos. Os usuários respondem perguntas sobre que tipos de serviços eles precisam que o produto realize e como esses serviços devem ser fornecidos. O novo produto de software pode também prestar serviços a outros produtos de software, e é preciso saber que interfaces serão necessárias entre esses dois produtos. Todos esses tipos de questões ajudarão a extrair os requisitos funcionais principais do sistema.

2.3.4 JAD

Joint Application Design (JAD) [AUG 91] é uma técnica para promover cooperação, entendimento e trabalho em grupo entre usuários e desenvolvedores [RAG 94]. Ela facilita a criação de uma visão compartilhada do que o produto de software deve ser, e, através da sua utilização, os desenvolvedores ajudam os usuários a formular problemas e explorar soluções, e os usuários ganham um sentimento de envolvimento, posse e responsabilidade para com o sucesso do produto.

A técnica JAD consta de quatro princípios básicos:

1) *dinâmica de grupo*, com a utilização de sessões de grupo facilitadas para aumentar a capacidade dos indivíduos;

2) *uso de técnicas visuais* para aumentar a comunicação e o entendimento;

3) *manutenção do processo organizado e racional*;

4) *utilização de documentação-padrão*, que é preenchida e assinada por todos os participantes de uma sessão.

A técnica JAD consta de duas etapas principais: planejamento e projeto; a primeira lida com extração e especificação de requisitos, e a segunda, com projeto de software. Dado que

este capítulo é dedicado à extração de requisitos, somente a primera etapa será tratada. Cada etapa consiste, por sua vez, em três fases: *adaptação*, *sessão* e *finalização*.

A fase de adaptação consiste na preparação para a sessão. Isso inclui organizar a equipe, adaptar o processo JAD ao produto a ser construído e preparar o material. A fase de sessão consiste em um ou mais encontros estruturados, envolvendo desenvolvedores e usuários. É durante esses encontros que os requisitos são desenvolvidos e documentados. A fase de finalização é dedicada a converter a informação da fase de sessão em sua forma final, em um documento de especificação de requisitos de software. Há seis tipos de participantes, embora nem todos participem de todas as fases.

O *líder da sessão* é responsável pelo sucesso do esforço, sendo o facilitador dos encontros. Ele deve estar familiarizado com todos os aspectos do JAD, ter habilidade para gerenciar encontros, além de experiência suficiente na área da aplicação para ser capaz de planejar e entender as várias tarefas da técnica JAD e as suas saídas. Embora todos os participantes necessitem de treinamento nos processos envolvidos na técnica JAD, o líder da sessão deve ser especialmente competente, com bom relacionamento pessoal e qualidades gerais de liderança. Através da prática e da experiência, líderes de sessão desenvolvem habilidades para entender e facilitar a dinâmica de grupo, iniciar e manter o foco das discussões, reconhecer quando os encontros estão saindo da meta original e trazê-los de volta para o objetivo inicial, lidar eficientemente com personalidades e comportamentos diferentes dos participantes e permanecer entusiasmados ao longo de encontros demorados e difíceis.

O *engenheiro de requisitos* é o participante diretamente responsável pela produção dos documentos de saída das sessões JAD. Além disso, deve ser um desenvolvedor experiente para poder entender questões técnicas e detalhes que são discutidos durante as sessões. Engenheiros de requisitos devem ser selecionados também pela habilidade de organizar idéias e expressá-las com clareza. Eles devem saber usar as ferramentas de software necessárias, tais como aquelas para produção de documentos ou ferramentas de prototipagem de software.

O *executor* é o responsável pelo produto sendo construído e tem duas principais responsabilidades no processo. A primeira é dar aos outros participantes uma visão dos pontos estratégicos do produto de software a ser construído, tais como o porquê de ele estar sendo construído e como a empresa espera melhorar com a utilização do novo produto. A segunda responsabilidade é tomar decisões executivas, tais como alocação de recursos, que podem afetar os requisitos e o projeto do novo produto.

Os *representantes dos usuários* são as pessoas na empresa que irão utilizar o produto de software. Durante a extração de requisitos, os representantes são freqüentemente gerentes ou pessoas-chave dentro da empresa; elas tendem a ter uma melhor visão do sistema todo e de como ele será usado. Os representantes dos usuários devem ser selecionados de acordo com o conhecimento de suas próprias necessidades dentro da empresa, o entendimento de como seu departamento interage com outros departamentos e algum conhecimento de produtos de software.

Os *representantes de produtos de software* são pessoas que estão bastante familiarizadas com as capacidades dos produtos de software. Seu papel é ajudar os usuários a entender o que é razoável ou possível que o novo produto faça. Em alguns casos, isso envolve esclarecer

o usuário sobre as tecnologias existentes; em outros, os representantes podem ajudar os usuários a entender as conseqüências de escolher um ou outro caminho para resolução do problema, quando existem dois ou mais caminhos que parecem igualmente satisfatórios do ponto de vista do usuário, mas que diferem em custos ou complexidade do ponto de vista da implementação.

O *especialista* é a pessoa que pode fornecer informações detalhadas sobre um tópico específico. Um especialista da comunidade de usuários, por exemplo, pode ser a pessoa que usa um determinado tipo de relatório, ou que é responsável pela execução de um determinado tipo de pedido. Nesse caso, ninguém mais na empresa conheceria os requisitos para esse tipo de pedido ou relatório. Um especialista da comunidade de desenvolvedores poderia ser alguém que conhecesse os detalhes da rede interna da empresa, tais como protocolos de comunicação. A participação dessa pessoa seria solicitada durante a definição dos aspectos da rede do novo sistema.

1) *A fase de adaptação*

Como a técnica JAD fornece uma estrutura geral para extração de requisitos, para ser mais efetiva ela deve ser adaptada a cada produto de software a ser desenvolvido. Isso é responsabilidade do líder da sessão, com a ajuda de um ou dois desenvolvedores. Esta seção consta dos seguintes passos:

Conduzir a orientação. No momento em que o executor autoriza a extração de requisitos, a finalidade do novo produto de software já foi de alguma forma discutida. Normalmente isso ocorre na comunidade de usuários, pois estes são os primeiros a reconhecer a necessidade potencial para o novo produto. O primeiro passo do líder da sessão e dos desenvolvedores de software é obter um entendimento do que foi conseguido até agora, de que tipo de produto está sendo discutido e, se tiverem sido tomadas decisões, de quais foram elas. Esse tipo de atividade requer pequenos encontros com um ou mais usuários e talvez um encontro com o executor. O líder da sessão e o engenheiro de requisitos podem também necessitar de familiarização com a empresa ou departamento para o qual o produto será construído. Um organograma da companhia pode ajudar na identificação das pessoas-chave que realmente irão contribuir para o JAD.

Organizar o grupo. A seguir, o líder da sessão seleciona os participantes. O executor também pode ter identificado alguns participantes, mas o líder da sessão tem a responsabilidade final de se certificar de que todas as pessoas necessárias foram identificadas e convidadas. O líder da sessão tem também de preparar os participantes para a sessão. Além de informar a data, hora e lugar da sessão, o líder dá aos participantes uma lista de perguntas e pede que pensem sobre ela antes da sessão. As questões são escolhidas visando aos requisitos de alto nível que serão abordados na fase de sessão, como, por exemplo, objetivos, benefícios esperados, restrições, adaptados ao produto específico a ser desenvolvido. Solicita-se que os participantes abordem as questões de acordo com as suas perspectivas; por exemplo, os usuários abordam restrições do ponto de vista

comercial, e os representantes de produtos de software, do ponto de vista tecnológico. Também se solicita que os participantes tomem notas para serem trazidas à sessão.

Ajustar o processo. O líder da sessão usa sua experiência e julgamento para adaptar o processo JAD ao produto a ser construído, como, por exemplo, decidir quanto tempo e quantos encontros serão necessários para a fase de sessão. Também inclui ajustar o formato geral dos documentos JAD às necessidades do produto de software a ser construído.

Preparar material. O líder da sessão faz os arranjos logísticos necessários para a sessão, incluindo a reserva e organização da sala do encontro. Recursos visuais, tais como transparências em branco, canetas de marcação, lousa de papel etc., também são providenciados. Para facilitar o andamento da sessão, o líder pode preparar várias transparências ou escrever na lousa de papel com antecedência. Isso inclui uma mensagem de boas-vindas, uma agenda, uma revisão do processo de extração de requisitos de acordo com a técnica JAD, uma revisão das categorias de requisitos de alto nível, questões sobre o escopo do produto e os formulários JAD em branco para registrar informações, decisões e perguntas.

2) *A fase de sessão*

A fase de sessão consiste em um ou mais encontros do grupo para definir os requisitos de alto nível para o novo sistema, assim como seu escopo. Todos os participantes trazem idéias e visões diferentes do produto para a sessão, e, através de discussões cuidadosas e facilitadas, essas idéias e visões são apresentadas, analisadas e refinadas, de forma que, no final da sessão, todas as pessoas estejam de acordo.

Conduzir orientações. A sessão começa com o líder e o executor dando boas-vindas aos participantes. Todos os participantes são apresentados. O executor faz um breve resumo do esforço feito até o momento e descreve as expectativas dos participantes em relação à sessão. O líder dá, então, uma visão geral do processo JAD, incluindo o tempo a ser gasto em cada tarefa. Entretanto, isso não é um curso de treinamento detalhado. À medida que uma nova tarefa começa, o líder fornece informações mais detalhadas sobre ela. Isso inclui o motivo da tarefa, os papéis dos participantes, como a tarefa é executada e como as saídas são registradas e formatadas.

Definir requisitos de alto nível. O líder facilita a discussão do grupo, cuja função é extrair requisitos de alto nível. Cinco grandes tópicos são abordados:

1) Objetivos: qual é a razão para a construção desse produto de software; qual será a sua finalidade?
2) Benefícios esperados: quais benefícios (quantificáveis ou não; tangíveis ou intangíveis) irão advir do uso deste produto?
3) Estratégias e considerações futuras: como esse produto pode ajudar na organização, no futuro; como ele poderá ser um avanço estratégico ou competitivo?

4) Restrições e suposições: quais as restrições do produto que está sendo construído (recursos, estrutura organizacional, padrões, leis); quais as restrições para o projeto do sistema?

5) Segurança, auditoria e controle: existem requisitos de segurança internos ou externos para o produto e seus dados; serão necessários auditorias ou controles?

Tipicamente, para começar a discussão, o líder faz perguntas gerais (preparadas com antecedência) para cada um desses tópicos. À medida que os requisitos são identificados pelos participantes, eles são registrados pelo analista em lousas brancas ou transparências, que permanecem disponíveis durante a sessão. Os participantes discutem, refinam e julgam esses requisitos.

Delimitar o escopo do sistema. A discussão gera um grande número de requisitos. O próximo passo é começar a organizá-los e entrar num acordo sobre o escopo do produto a ser construído. É interessante identificar quem realmente vai usar o produto e quais as principais funções que o produto ajudará a executar. Também é importante identificar funcionalidades que estão fora do escopo do sistema. O objetivo é delimitar o escopo, de forma que o produto seja abrangente o suficiente para atingir seus objetivos, mas não tão grande que seja excessivamente custoso ou complexo para construir. Nessa etapa, os recursos visuais podem ser bastante úteis. Por exemplo, os nomes das tarefas podem ser escritos sobre dispositivos magnéticos, que podem ser projetados numa lousa branca e unidos através de setas, representando o fluxo dos dados. À medida que a discussão prossegue, o formato do sistema muda e os dispositivos magnéticos podem ser movidos para mostrar a evolução do sistema. Nesse ponto, a parte de extração de requisitos da etapa de planejamento está essencialmente completa.

Documentar questões e considerações. Durante a sessão, aparecem questões que afetam os requisitos do produto, mas para as quais nenhum dos participantes pode ter a informação necessária ou autoridade para resolver. É importante que essas questões sejam documentadas e resolvidas. Algumas vezes aparecem considerações que não afetam o processo JAD corrente, mas que podem afetar a maneira como o produto será construído ou utilizado. Essas considerações também devem ser documentadas para futura referência. O processo JAD especifica a forma do documento para registrar essas questões e considerações. Cada questão é atribuída a uma pessoa para ser resolvida em uma data específica. Outras considerações, não incluídas no formulário, são geralmente registradas na simples forma de uma lista.

Concluir a fase de sessão. O líder conclui a sessão revisando a informação coletada e as decisões tomadas. Cada participante tem a oportunidade de expressar preocupações sobre os requisitos remanescentes. O líder conduz essa discussão de forma que todos adquiram um senso de posse e de responsabilidade para com os requisitos documentados. A conclusão da sessão de forma positiva garante contribuições futuras de todos os participantes.

3) *A fase de finalização*

O objetivo principal dessa fase é transformar as transparências, anotações da lousa de papel e outros documentos escritos na fase de sessão em documentos de especificação dos requisitos de software. Os desenvolvedores trabalham em tempo integral durante essa fase, auxiliados pelo líder da sessão. Essa fase consta de três etapas distintas:

Completar o documento. A organização normalmente tem um formato fixo para o documento, embora possa ser adaptado para um determinado produto de software. Os desenvolvedores são responsáveis pela tradução das saídas da sessão em um documento que esteja de acordo com esse formato.

Revisar o documento. Depois de os desenvolvedores terem produzido um documento completo, é dada a oportunidade a todos os participantes da sessão de revisarem e comentarem o documento. Normalmente uma cópia do documento pode ser dada a cada participante, pedindo que façam comentários por escrito. Se houver comentários substanciais dos revisores, um novo encontro é marcado para que sejam discutidos. Todos os participantes da sessão original são convidados, de forma que as mudanças no documento sejam feitas de comum acordo.

Obter a aprovação do executor. Depois de os desenvolvedores terem revisado o documento para refletir sobre os comentários dos revisores, o líder da sessão submete o documento à aprovação do executor. Essa aprovação dá um caráter formal ao documento e encerra o processo de extração de requisitos. Todos os participantes recebem, então, uma cópia do documento final.

2.3.5 Prototipagem

Em algumas situações, os usuários podem entender e expressar melhor as suas necessidades através da comparação com um produto de software que sirva de referência. Quando tal produto não existe, a prototipagem pode ser usada para criar um produto que ilustre as características relevantes [RAG 94]. Através do exame do protótipo, os usuários podem descobrir quais são as suas reais necessidades. O processo de prototipagem começa com um estudo preliminar dos requisitos do usuário. A seguir, começa um processo iterativo de construção do protótipo e avaliação junto dos usuários. Cada repetição permite que o usuário entenda melhor seus requisitos, inclusive as implicações dos requisitos articulados nas iterações anteriores. Eventualmente, um conjunto final de requisitos pode ser formulado, e o protótipo descartado. A prototipagem é benéfica somente se o protótipo puder ser construído substancialmente mais rápido que o sistema real; por isso, é interessante utilizar alguma das muitas ferramentas desenvolvidas para facilitar essa atividade. A prototipagem é usada para extrair e entender requisitos; ela é seguida pelo processo estruturado e gerenciado de construção do sistema propriamente dito, como descrito na seção 1.4.2. Quando usada apropriadamente, a prototipagem pode ser muito útil para superar várias dificuldades inerentes ao processo de extração de requisitos, especialmente as dificuldades de comunicação e de articulação de necessidades pelo usuário.

2.3.6 Questionário

Questionários são uma técnica de extração de requisitos informal. Existem muitas semelhanças entre questionários e entrevistas e eles podem ser usados conjuntamente[KEN 88]. Os questionários podem ser usados para elucidar informações que não ficaram claras na entrevista, ou podem ser elaborados a partir do que foi descoberto na entrevista. Entretanto, cada uma dessas técnicas tem suas funções específicas e nem sempre é necessário ou desejável utilizar as duas.

Os questionários, à primeira vista, podem parecer uma maneira rápida de obter informações sobre como os usuários acessam o sistema, que problemas tem encontrado para executar o seu trabalho, ou ainda sobre o que eles esperam do novo sistema a ser implantado. Embora com os questionários seja possível obter uma quantidade muito grande de informações sem perder tempo com as entrevistas face a face, a elaboração de questionários úteis demanda muito tempo. Existem algumas circunstâncias em que o uso de questionários se torna particularmente interessante. São elas:

- As pessoas que precisam ser entrevistadas estão dispersas, como por exemplo em várias filiais de uma mesma empresa.
- Existe um número muito grande de pessoas envolvidas no projeto do sistema e é importante saber que proporção de um determinado grupo, como por exemplo gerentes, aprovam ou desaprovam uma determinada característica do sistema proposto.
- Um estudo exploratório precisa ser feito para obter opiniões antes que seja dado um direcionamento específico ao projeto.
- Deseja-se descobrir problemas com o sistema atual que serão aprofundados nas entrevistas que se seguirão.

A maior diferença entre as questões usadas nas entrevistas e aquelas usadas em questionários é que as entrevistas permitem uma interação para sanar qualquer problema de entendimento entre o entrevistador e o entrevistado. Na entrevista, o engenheiro de requisitos tem oportunidade de reformular a questão, de explicar um termo ambígüo, de mudar o rumo da entrevista e, geralmente, de controlar o contexto. Quase nada disso é possível com questionários. Portanto, as perguntas que compõem o questionário devem ser claras, o fluxo do questionário deve ser pertinente, as dúvidas devem ser antecipadas pelo elaborador do questionário e a sua aplicação deve ser planejada em detalhes.

As perguntas usadas no questionário podem ser de dois tipos: *abertas* e *fechadas*. Como o próprio nome diz, as questões abertas deixam as possiblidades de resposta em aberto. Exemplos desse tipo de questão são “Descreva os problemas que porventura você esteja experimentando com os relatórios de saída”, ou ainda “Por que você acha que os manuais do usuário para o sistema contabilidade não funcionam?”. Quando questões abertas são utilizadas é importante que se antecipe as repostas, para que seja possível interpretá-las corretamente. Por exemplo, as respostas para a pergunta “O que você acha do sistema?”

podem ser amplas demais para que seja possível interpretá-las e compará-las. Se o objetivo for obter *feedback* sobre o sistema, as questões podem ser elaboradas em termos da satisfação versus insatisfação dos usuários com o sistema. Além disso, as perguntas podem mencionar algumas características em particular que sejam de interesse. As questões abertas são especialmente úteis quando se deseja obter opiniões de alguns grupos sobre um determinado aspecto do sistema, seja produto ou processo, ou quando é impossível enumerar todas as possíveis respostas para uma determinada pergunta. Esse tipo de pergunta também é útil em sistuações exploratórias, quando por exemplo o engenheiro de requisitos não é capaz de determinar com precisão quais os problemas com o sistema atual.

As questões fechadas limitam as opções de resposta, como por exemplo "Os dados sobre vendas são normalmente entregues com atraso", onde as possibilidades de resposta são "concordo" ou "discordo". Questões fechadas devem ser usadas quando o engenheiro de requisitos é capaz de enumerar todas as possibilidades para a pergunta e quando todas as questões são mutuamente exclusivas. As questões fechadas também devem ser usadas quando se deseja saber a opinião de um grande número de pessoas, visto que a análise e a correta interpretação das questões abertas, respondidas por um grande número de pessoas, é uma tarefa quase impossível. Os prós e contras na utilização de questões abertas e fechadas estão resumidos na Figura 2.2.

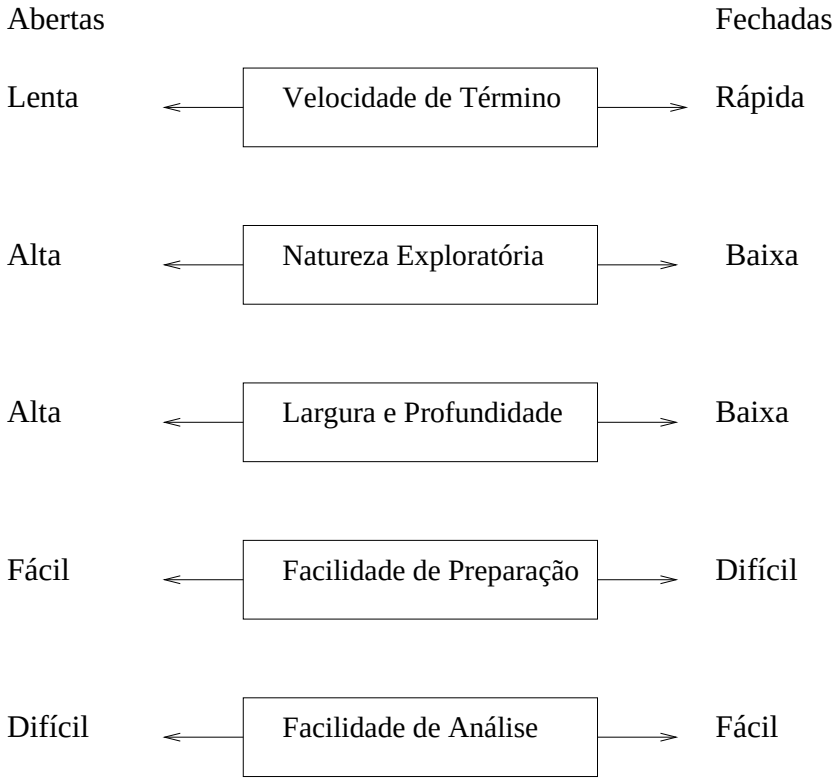

Figura 2.2: Prós e contras das questões abertas e fechadas.

Assim como nas entrevistas, a linguagem empregada nos questionários é muito importante

para que se obtenha os resultados desejados. Mesmo quando o engenheiro de requisitos possui um conjunto padrão de perguntas relacionadas ao desenvolvimento de sistemas, muitas vezes é importante que essas questões sejam reescritas na terminologia própria da empresa ou do negócio no qual o sistema será inserido. Por exemplo, se a empresa usa o termo "unidade" em vez de "departamento", a incorporação do termo apropriado auxiliará no correto entendimento do significado da pergunta. Para ter certeza de que está usando o termo apropriado, o engenheiro de requisitos pode fazer um teste com um grupo piloto, solicitando-lhe que preste especial atenção às palavras utilizadas.

Existem alguns conselhos úteis para a escolha do vocabulário apropriado para o questionário:

- Usar a linguagem de quem vai responder o questionário sempre que possível, mantendo as perguntas simples, claras e curtas.
- Ser específico, mas não exageradamente.
- Fazer a pergunta certa para a pessoa certa.
- Ter certeza de que as questões estão tecnicamente corretas antes de incluí-las no questionário.

Elaboração do Questionário

Embora o objetivo do questionário seja descobrir atitudes, crenças, comportamentos e características que podem alterar substancialmente o trabalho dos usuários, eles nem sempre estão motivados para respondê-lo. Um questionário bem elaborado, com perguntas relevantes, pode ajudar a lidar com a resistência das pessoas. Existem alguns aspectos estilísticos na confecção do questionário que podem auxiliar na melhoria das respostas obtidas, assim como guias para a ordenação do conteúdo de forma a obter melhores resultados.

Com relação ao formato do questionário, deve-se deixar espaço em branco suficiente para as respostas, assim como amplo espaço nas margens. Outra boa prática na confecção de questionários é solicitar que seja feito um círculo em volta da resposta ou do número, se uma escala estiver sendo utilizada, como é exemplificado na Figura 2.3.

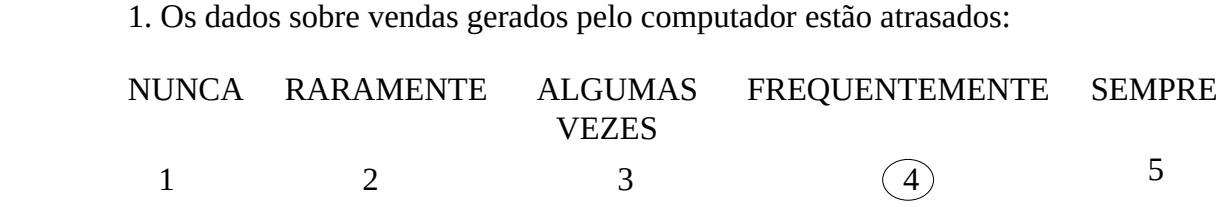

Figura 2.3: Uso de escala no questionário.

Os objetivos devem guiar a elaboração do questionário. Por exemplo, se o objetivo for consultar tantos funcionários quantos possível para identificar uma lista de problemas com o

sistema atual, talvez seja melhor utilizar um questionário que possa ser lido automaticamente pelo computador. Se, por outro lado, deseja-se obter respostas elaboradas, então é necessário calcular o espaço necessário para cada resposta. Pode ser necessário utilizar tanto respostas escritas quanto numéricas e talvez utilizar uma terceira pessoa para digitar as respostas. Muitas vezes questionários respondidos diretamente são mais simples de ser entendidos do que aqueles que podem ser lidos automaticamente pelo computador.

O questionário deve ter um estilo consistente e as intruções devem ser colocadas no mesmo lugar nas várias seções. Letras maiúsculas e minúsculas devem ser usadas nas questões e nas respostas somente maiúsculas, como exemplificado na Figura 2.3. Uma decisão crucial na elaboração do questionário é a ordem em que as perguntas devem aparecer; para isso os objetivos do questionário devem ler levados em consideração, assim como a função de cada pergunta na obtenção desses objetivos.

Deve-se levar em conta também como a pessoa que está respondendo o questionário se sentirá a respeito da ordem e do lugar específico de uma determinada questão, e se esta é a situação desejada. As questões mais importantes devem vir primeiro e devem lidar com assuntos que as pessoas que estão sendo questionadas julguem importantes. Elas devem sentir que, através das suas respostas, podem causar mudanças ou que suas respostas podem gerar algum impacto. Essa técnica torna as pessoas rapidamente envolvidas com o processo de extração de requisitos. As questões de conteúdo semelhante e relacionado devem estar próximas no questionário, como por exemplo as questões que lidam com o usuário final.

As associações prováveis devem ser antecipadas pelo elaborador do questionário, que por sua vez deve usá-las na definição da ordem das questões. Por exemplo, se a pergunta fosse "Quantos subordinados você tem?", provavelmente deveriam existir outras perguntas sobre outras relações formais dentro da empresa. Além disso, a pessoa poderia associar a estrutura organizacional formal com a informal e a inclusão de perguntas sobre as relações informais também seria pertinente. As questões que podem gerar controvérsias devem ser deixadas para depois. Por exemplo, se o engenheiro de requisitos achar que questões sobre quais tarefas deve ser automatizadas são polêmicas, deve deixá-las para o final.

Aplicação do Questionário

A decisão sobre quem responderá o questionário é tomada em conjunto com a definição dos seus objetivos. A utilização de um grupo piloto ajuda a determinar que tipo de representação é necessária e, portanto, que pessoas devem responder ao questionário. As pessoas que compõm esse grupo são normalmente escolhidas dentre os representantes de posições ou tarefas dentro da empresa, ou ainda devido a algum interesse epecial no sistema proposto.

Um número suficiente de pessoas deve ser incluído para que a amostra seja considerada razoável, mesmo se alguns questionários não forem retornados ou se algumas folhas de resposta forem preenhidas incorretamente.

Existem várias possibilidades para aplicar o questionário e a escolha é normalmente determinada pela situação da empresa. Algumas opções para a aplicação do questionário são:

1. Todos respondem ao questionário ao mesmo tempo no mesmo lugar.

2. Os questionários são entregues pessoalmente e depois de preenchidos são recolhidos.

3. Os questionários são colocados a disposição dos funcionários e depois de preenchidos, colocados pelo próprio funcionário em algum lugar predeterminado.

4. Os questionários são envidados por correio eletrônico ou correio normal juntamente com prazo e instruções sobre para qual endereço devem ser retornados.

Cada uma dessas opções tem vantagens e desvantagens. Quando o questionário é aplicado ao mesmo tempo, economiza-se tempo e o engenheiro de requisitos pode controlar melhor a situação, garantindo que todas as pessoas recebam as mesmas instruções e que todos os questionários sejam devolvidos. Uma desvantagem da aplicação simultânea do questionário é que nem todos os empregados selecionados podem estar disponíveis no horário marcado para aplicação do questionário. Outra desvantagem é que alguns funcionários podem ficar irritados com o fato de terem sido convocados para responder o questionário, deixando outras tarefas urgentes e importantes por fazer.

O engenheiro de requisitos também pode garantir uma alta taxa de retorno entregando pessoalmente o questionário, mas se o número de pessoas for muito grande ou se elas estiverem dispersas, o tempo para fazer a distribuição pode se tornar um problema. A confidencialidade também pode ser um problema visto que o engenheiro poderá saber quem está respondendo o questionário.

Quando os questionários são colocados a disposição, a taxa de resposta torna-se um pouco menor, porque as pessoas podem se esquecer do questionário, perdê-lo ou ignorá-lo propositalmente. Entretanto, essa forma de aplicação do questionário permite que o anonimato das pessoas seja mantido e pode resultar em respostas mais francas. Uma maneira de melhorar a taxa de resposta é instalar uma caixa-resposta na mesa de um funcionário e pedir a ele que anote o nome das pessoas que devolveram o questionário respondido.

A taxa de resposta para o último método é notadamente baixa, mas é importante que pessoas fisicamente distantes da empresa sejam incluídas na extração de requisitos, pois provavelmente terão pontos de vista diferentes sobre o sistema a ser construído.

2.4 Comentários finais

Existem várias ferramentas para auxiliar na extração de requisitos. Muitas delas têm por objetivo propiciar condições para que as pessoas possam trabalhar juntas, sem que necessariamente estejam na mesma sala ou prédio. Ferramentas de videoconferência são um exemplo. Com estações de trabalho configuradas e em rede, os participantes de uma sessão de *brainstorming*, por exemplo, poderiam permanecer em seus escritórios e, ainda assim, ser vistos e ouvidos por todos os outros participantes. As idéias poderiam ser digitadas pelos participantes individuais ou por um "escrivão", com todos vendo imediatamente as idéias à medida que fossem sendo digitadas na tela da sua estação. Existem também ferramentas

para prototipagem e para produção de documentos, mas sua efetividade é ainda incerta. Algumas pessoas acreditam que as ferramentas podem ser úteis primeiramente na fase de consolidação, que envolve a edição e a reordenação das idéias. Se isso for feito *on-line*, o grupo terá a oportunidade de evoluir para a lista final de idéias durante a sessão.

2.5 Exercícios

1) A Editora ABC trabalha com diversos autores que escrevem livros que ela publica. Alguns autores escrevem apenas um livro, enquanto outros escrevem muitos; além disso, alguns livros são escritos em conjunto por diversos autores. Mensalmente é enviado às livrarias um catálogo com o nome dos livros lançados e seus respectivos autores. Esse catálogo é organizado por assunto para facilitar a divulgação. Informações sobre a cota de cada livraria são modificadas a cada três meses, de acordo com a média de compra no trimestre, e então uma carta é enviada à livraria anunciando a nova cota em cada assunto e os descontos especiais que lhe serão concedidos para compras em quantidades maiores. Aos autores dos dez livros mais vendidos no ano, a Editora ABC oferece prêmios. A festa de premiação é anunciada com dez dias de antecedência, através de publicação em jornal dos dez livros mais vendidos, com seus respectivos autores.

   (a) Indique ambigüidades, omissões e jargões (se houver).

   (b) Elabore um questionário baseado nos problemas encontrados no item a.

   (c) Apresente uma lista de funções e restrições.

2) Considere um sistema de controle para um salão de beleza e estética, que tem como funcionalidades básicas o agendamento dos clientes e alguns relatórios estatísticos. Escolha e aplique um método para extração de requisitos e faça o relatório contendo:

   (a) plano de extração de requisitos;

   (b) justificativa para escolha do método utilizado;

   (c) descrição sucinta do sistema;

   (d) objetivos e restrições do sistema.

3) Descreva os quatro passos envolvidos no processo de extração de requisitos.

4) Descreva as principais dificuldades da extração de requisitos.

5) Descreva as dificuldades da extração de requisitos abordadas pela técnica JAD.

6) Descreva as dificuldades da extração de requisitos abordadas pela técnica de *brainstorming*.

7) Descreva os principais passos e protocolos da técnica de entrevista.

8) Forneça exemplos dos tipos de questões que devem ser preparados com antecedência para uma entrevista visando à extração de requisitos.

9) Descreva os vários tipos de erros que podem ocorrer em uma entrevista e explique como corrigi-los.

10) Explique como a técnica PIECES pode melhorar uma entrevista.

11) Explique os seis tipos de questões que compõem a sigla PIECES e dê exemplos dos tipos de questões que podem ser feitos para extrair requisitos nessas seis categorias.

Capítulo 3

Modelagem de Casos de Uso

3.1 Introdução

No contexto do software, a fase de especificação de requisitos tem o objetivo de representar a idéia do usuário a respeito do sistema que será desenvolvido. Essa representação é feita através do documento de requisitos, que deve compreender a necessidade real dos usuários e suas expectativas, dimensionar a abrangência do sistema e delimitar o seu escopo.

Em geral, os requisitos podem ser vistos como condições ou capacidades necessárias para resolver um problema ou alcançar um objetivo. Para a elicitação dessas capacidades, os processos normalmente estruturam a fase de especificação de requisitos nas quatro etapas seguintes: (i) identificação do domínio do problema; (ii) identificação das funcionalidades esperadas pelo sistema (objetivos); (iii) definição de restrições existentes para o desenvolvimento; (iv) identificação dos atributos de qualidade desejados.

As funcionalidades especificadas para o sistema e os seus atributos de qualidade recebem um tratamento diferenciado no documento de especificação de requisitos. Por essa razão, os requisitos são classificados em dois tipos [SOM 96]: (i) **requisitos funcionais**, composto pelas funcionalidades especificadas; e (ii) **requisitos não funcionais**, que materializam os atributos de qualidade do sistema. Apesar de não representarem funcionalidades diretamente, os requisitos não-funcionais podem interferir na maneira como o sistema deve executá-las.

De acordo com a norma ABNT/ISO 9126, os requisitos não-funcionais de um software podem ser classificados em seis grupos, de acordo com a sua característica principal: (i) **completude** (do inglês *functionality*), que quantifica a grau de satisfação das necessidades do cliente; (ii) **confiabilidade**, que identifica a capacidade do software em manter a sua integridade após a ocorrência de falhas não controladas; (iii) **usabilidade**, que identifica a facilidade de se compreender o funcionamento e operação do software; (iv) **eficiência**, que identifica a capacidade do software em desempenhar as suas atividades de forma adequada em relação ao tempo e aos recursos alocados; (v) **manutenibilidade**, que identifica a capacidade do software em sofrer modificações; e (vi) **portabilidade**, que identifica a capacidade de adaptação do software quando transferido para outros ambientes e/ou plataformas.

Em UML, a notação utilizada para a representação dos requisitos do sistema o diagrama de casos de uso. Pelo fato da UML ser uma linguagem unificada, os processos que a utilizam evidenciam a importância dos casos de uso. Esses artefatos, que representam o objetivo do sistema, é a base para todo o processo de desenvolvimento, sendo o ponto de ligação entre o cliente e toda a equipe de desenvolvimento.

A modelagem de casos de uso é uma técnica que auxilia o entendimento dos requisitos de um sistema computacional através da criação de descrições narrativas dos processos de negócio, além de delimitar o contexto do sistema computacional [Lar97]. Um **caso de uso** é uma narrativa que descreve uma seqüência de eventos de um agente externo (ator) usando o sistema para realizar uma tarefa. Casos de uso são utilizados para descrever os requisitos funcionais, que indicam o que o sistema deverá ser capaz de fazer, sem entrar em detalhes sobre como essas funcionalidades serão materializadas. Como pode ser visto na Figura 3.1, o modelo de casos de uso é um dos principais artefatos oriundos da fase de especificação de requisitos e serve de entrada para a fase de análise. Casos de uso bem estruturados denotam somente o comportamento essencial do sistema ou subsistema e não podem ser nem muito gerais, a ponto de prejudicar o seu entendimento, nem muito específicos, a ponto de detalhar como as tarefas serão implementadas.

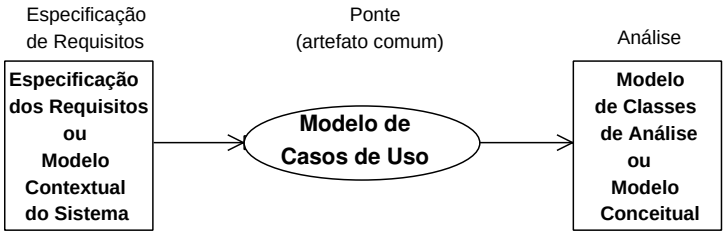

Figura 3.1: Casos de Uso na Especificação dos Requisitos.

Além de capturar o conhecimento de clientes e futuros usuários do sistema sobre seus requisitos, casos de uso também são úteis para definir cronogramas e auxiliam na elaboração dos casos de teste do sistema.

A fim de fixar os conceitos apresentados neste capítulo, será utilizado o exemplo de um sistema de controle de bibliotecas, descrito a seguir.

Enunciado: Sistema de Controle de Bibliotecas

Um sistema de controle de bibliotecas é um sistema computacional usado para controlar o empréstimo e a devolução de exemplares de uma biblioteca. O usuário pode fazer um empréstimo de um exemplar durante um certo período e, ao final desse tempo, ou o exemplar deve ser devolvido, ou o empréstimo deve ser renovado.

O atendente é um funcionário que interage com os usuários e com o sistema de

controle da biblioteca através de um terminal. As principais características do sistema são listadas a seguir:

1. Um usuário do sistema, que pode ser um aluno, um professor ou um outro funcionário da universidade, pode reservar publicações e também cancelar reservas previamente agendadas.
2. Um usuário deve estar devidamente cadastrado no sistema para usar os seus serviços. O sistema é operado pelo atendente da biblioteca, que também é um funcionário da universidade.
3. Um usuário pode emprestar exemplares previamente reservados ou não. Se foi feita uma reserva, ela deve ser cancelada no momento do seu empréstimo.
4. No caso da devolução de um exemplar em atraso, existe uma multa que deve ser paga. Essa multa é calculada com base no número de dias em atraso. Além disso, se o exemplar estiver atrasado por mais de 30 dias e se o usuário não for um professor, além de pagar a multa, o usuário é suspenso por um período de 2 meses.
5. Um exemplar da biblioteca pode ser bloqueado/desbloqueado por um professor por um período de tempo. Nesse caso, o exemplar fica disponível numa estante, podendo ser consultado por usuários da biblioteca, mas não pode ser emprestado.
6. O período de empréstimo é variável, dependendo do tipo de usuário (aluno, funcionário ou professor).
7. A manutenção dos dados do acervo da biblioteca é feita pela bibliotecária, que também é funcionária da universidade. Essa funcionária é responsável pela inclusão de novos exemplares, exclusão de exemplares antigos e pela atualização dos dados dos exemplares cadastrados.

Os exemplares podem ser livros, periódicos, manuais e teses. As publicações são identificadas pelo seu número do tombo, além de outras características como o título, a editora e o número da edição correspondente.

A biblioteca só empresta suas obras para usuários cadastrados. Um usuário é identificado através de seu número de registro. Outras informações relevantes são seu nome, instituto/faculdade a que pertence e seu tipo (aluno/funcionário/professor).

3.2 Casos de Usos

Um **caso de uso** é uma descrição de um processo de negócio relativamente longo com um começo, meio e fim. Ele representa as principais funcionalidades que o sistema deve oferecer de maneira observável por seus interessados externos, isto é, os seus **atores**. Os casos de uso, na verdade, representam funções no nível do sistema e podem ser utilizados para

visualizar, especificar, construir e documentar o comportamento esperado do sistema durante o levantamento e análise de seus requisitos. A definição formal de caso de uso, segundo Larman [Lar97], é: "um conjunto de seqüências de ações que um sistema desempenha para produzir um resultado observável, de valor para um ator específico".

Por se preocupar unicamente com o problema e utilizar conceitos do domínio, a utilização de casos de uso também facilita a comunicação entre os usuários finais, os especialistas no domínio e os desenvolvedores do sistema.

Um exemplo de caso de uso do sistema de bibliotecas seria o Emprestar Exemplar. A Figura 3.2 apresenta a representação gráfica desse caso de uso, de acordo com a linguagem UML. O caso de uso Emprestar Exemplar pode ser descrito da seguinte forma:

> **Caso de Uso**: Emprestar Exemplar
> **Atores**: Usuário, Atendente, Sistema de Cadastro.
> **Descrição**: Este caso de uso representa o processo de empréstimo de um ou vários exemplares da biblioteca. O empréstimo se inicia com a solicitação feita pelo usuario ao atendente. Em seguida, através de um terminal, o atendente solicita ao sistema o empréstimo dos respectivos exemplares.

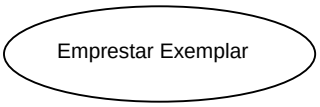

Figura 3.2: Caso de Uso Emprestar Exemplar em UML.

3.3 Atores e Papéis

Um **ator** é uma entidade externa ao sistema computacional que participa de um ou mais casos de uso. Essa participação normalmente é baseada na geração de eventos de entrada ou no recebimento de alguma resposta do sistema. De uma maneira geral, atores representam entidades interessadas, que podem interagir diretamente com o sistema.

A representação dos atores se dá a partir do papel que eles representam, como por exemplo, usuario, atendente, aluno, etc. Em geral, atores podem ser: (i) papéis que pessoas representam nos casos de uso, (ii) dispositivos de hardware mecânicos ou elétricos, ou até mesmo (iii) outros sistemas computacionais.

A interação entre um ator e o sistema acontece através de troca de mensagens e em UML, é representada por um relacionamento de associação (uma linha contínua). Em relação aos atores, A Figura 3.3 apresenta as duas maneiras de representá-los: (i) na forma de um boneco; ou (ii) como uma classe com o estereótipo $<< actor >>$.

No exemplo do sistema de bibliotecas, podemos identificar inicialmente dois atores: o atendente e o usuário. Na Figura 3.4 podem ser vistas as associações que indicam a participação desses atores no caso de uso Emprestar Exemplar.

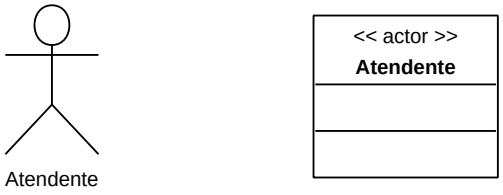

Figura 3.3: Representação de Atores em UML.

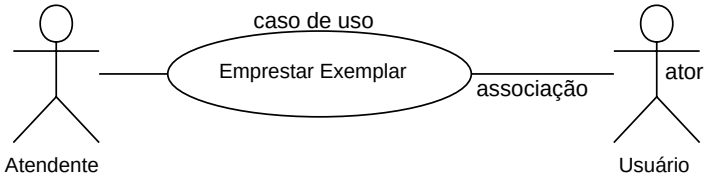

Figura 3.4: Caso de Uso Emprestar Exemplar com seus atores.

Como visto anteriormente, os atores são usados para representar qualquer entidade externa com a qual o sistema interage. Dessa forma, a interferência do tempo em um sistema, por exemplo, pode ser representada através de um ator chamado tempo, associado aos seus respectivos casos de uso. De modo particular no sistema de bilbiotecas, o cancento de uma reserva após um período caracteriza casos de uso associados ao ator tempo. Outra solução é não representar o tempo explicitamente, o considerando como um componente interno do sistema. Nesse caso, o caso de uso inicia a si próprio numa determinada hora e por isso é associado apenas aos demais atores interessados no seu funcionamento. Por exemplo, os sistemas e interessados que recebem a saída gerada por ele.

3.4 Fluxo de Eventos

O **fluxo de eventos** de um caso de uso é uma seqüência de comandos declarativos que descreve as etapas de sua execução, podendo conter desvios de caminhos e iterações. Esse fluxo deve especificar o comportamento de um caso de uso de uma maneira suficientemente clara para que alguém de fora possa compreendê-lo facilmente. Em outras palavras, ele deve permanecer focado no domínio do problema e não em sua solução. Além disso, esse fluxo deverá incluir como e quando o caso de uso inicia e termina, quais são os atores interessados no sistema e quando e como o caso de uso interage com eles.

O fluxo de eventos de um caso de uso é composto por: um (i) fluxo básico e (ii) zero ou mais fluxos alternativos. Enquanto o **fluxo básico** descreve a seqüência de passos mais esperada para a execução da funcionalidade do caso de uso, os **fluxos alternativos** descrevem desvios do fluxo básico. Esses fluxos podem ser especificados de diversas maneiras, incluindo descrição textual informal, texto semi-formal (através de assertivas lógicas), pseudocódigo, ou ainda uma combinação dessas maneiras.

A seguir, é mostrado o fluxo de eventos do caso de uso Emprestar Exemplar. O fluxo básico é descrito através de pseudocódigo, enquanto os fluxos alternativos são descritos informalmente, por meio de descrição textual. Para facilitar a leitura do caso de uso, será adotada a seguinte convenção: cada frase que descreve uma ação deve iniciar com o nome "sistema", ou o nome do ator que a executa. Em seguida, deve-se indicar o verbo da ação e finalmente o complemento da frase. Quando apropriado, o complemento da frase deve referenciar o paciente da ação. Sendo assim, o formato sugerido para cada frase é o seguinte:
$< sistema/ator\_agente > + < verbo > + < complemento\_frase > + < sistema/ator\_paciente >$.

Fluxo Básico de Eventos:

1. O usuário solicita empréstimo de um ou mais exemplares de publicações (livro, periódico, tese ou manual), fornecendo o seu código e os exemplares desejados;
2. O atendente solicita o empréstimo ao sistema, fornecendo o código do usuário;
3. Para cada exemplar a emprestar:
   - 3.1 O atendente fornece o número de registro do exemplar.
   - 3.2 O sistema valida o usuário e verifica o seu *status* ("Normal" ou "Suspenso") através de seu número de registro.
   - 3.3 O sistema verifica se o exemplar pode ser emprestado pelo usuário em questão;
   - 3.4 Se o *status* do usuário for "Normal" e o exemplar estiver disponível:
     - 3.4.1. O sistema verifica se a publicação do exemplar está reservada. Se estiver reservada:
       - A. O sistema cancela a reserva, passando o número de tombo da publicação
     - 3.4.2. O sistema calcula o período do empréstimo, que depende do tipo de usuário - 7 dias para alunos e 15 para professores
     - 3.4.3. O sistema registra o empréstimo do exemplar;
     - 3.4.4. O sistema atualiza seu banco de dados com a informação de que o exemplar não irá se encontrar na biblioteca até completar o período.

Fluxos Alternativos

Fluxo Alternativo 1:

No Passo 3.4, se o usuário estiver suspenso, este é informado de sua proibição de retirar exemplares e o empréstimo não é realizado.

Fluxo Alternativo 2:

No Passo 3.4, se todas as cópias da publicação estiverem emprestadas ou reservadas, o sistema informa ao atendente que não será possível realizar o empréstimo daquele exemplar. Se tiver outros exemplares para emprestar, vá para o Passo 3.1 do fluxo básico.

Como pôde ser visto neste exemplo, os fluxos alternativos são executados quando condições especiais pré-definidas são detectadas durante a execução do fluxo básico. Neste caso, a execução do fluxo básico é interrompida e o fluxo alternativo correspondente à condição é executado. No exemplo acima, se o usuário não estiver suspenso e o exemplar não estiver disponível, (Passo 3.4 do fluxo básico), o segundo fluxo alternativo é executado. Se for bem sucedido, ao final deste, a execução é retomada a partir do Passo 3.1 do fluxo básico (próximo exemplar). Também é possível que um fluxo alternativo leve ao encerramento precoce do fluxo de eventos, como é o caso do primeiro fluxo alternativo do exemplo acima.

3.5 Cenários

Um **cenário** é uma seqüência de comandos/ações simples, que representa uma execução específica do fluxo de eventos, com todas as decisões e iterações já conhecidas de antemão. Um cenário representa uma interação ou execução de uma instância de um caso de uso. O fluxo de eventos de um caso de uso produz um **cenário primário**, que representa a funcionalidade básica do caso de uso, e zero ou mais **cenários secundários**, que descrevem desvios do cenário primário.

O cenário primário de um caso de uso é escrito supondo que tudo dá certo e ilustra uma situação típica de sucesso. Normalmente o cenário primário de um caso de uso corresponde à execução dos passos de seu fluxo básico quando nenhum desvio é tomado. Cenários secundários representam situações menos comuns, incluindo aquelas em que um erro ocorre. Cenários secundários que representam situações de erro também são conhecidos como **cenários excepcionais**.

Tipicamente, para cada cenário derivado de um fluxo de eventos, deve ser possível responder as quatro questões seguintes: (i) como o cenário começa? (ii) o que causa o término do cenário? (iii) quais respostas são produzidas pelo cenário? (iv) existem desvios condicionais no fluxo de eventos?

Seguindo o nosso exemplo, o cenário primário do caso de uso Emprestar Exemplar pode ser descrito da seguinte forma:

1. O usuário José chega à biblioteca para tomar emprestado um exemplar do livro *São Bernardo*. José apresenta o exemplar e informa o seu código pessoal, que é "A55";
2. O atendente solicita o empréstimo, fornecendo o código "A55" para José;
3. O atendente olha o exemplar entregue e fornece o número do registro ao sistema;
4. O sistema verifica que José está com *status* "Normal";
5. O sistema verifica que o exemplar está disponível para locação;
6. O sistema calcula que o exemplar pode ficar emprestado a José por 7 dias;
7. O sistema registra o empréstimo;

8. O sistema atualiza o registro, finalizando o empréstimo;
9. O atendente entrega o exemplar a José;
10. José vai embora.

Cenários secundários são descritos de maneira similar, mas levando em consideração desvios do cenário primário. Por exemplo, no Passo 3 do cenário primário, o código do produto pode ser inválido e, nesse caso, o sistema deve indicar um erro. De forma análoga, no Passo 5 o exemplar pode não estar disponível e então a transação de empréstimo do exemplar em questão deve ser cancelada. O primeiro cenário alternativo poderia ser descrito da seguinte forma:

1. O usuário José chega à biblioteca para tomar emprestado um exemplar do livro *São Bernardo*. José apresenta o exemplar e informa o seu código pessoal, que é "A55";
2. O atendente solicita o empréstimo, fornecendo o código "A55" para José;
3. O atendente olha o exemplar entregue e fornece o número do registro ao sistema;
4. O sistema verifica que José está com *status* "Suspenso";
5. O sistema avisa ao atendente que José não pode realizar o empréstimo e cancela a operação;
6. O atendente informa a José;
7. José vai embora.

3.6 Formato e Convenções para Casos de Uso

Formato de Caso de Uso

Além do fluxo de eventos, a descrição de um caso de uso pode incluir informações que aumentem a rastreabilidade dos requisitos e facilitem a comunicação entre clientes e a equipe de desenvolvimento do sistema [Coc00]. Não existem padrões na indústria ou na literatura sobre quais informações devem ser incluídas na descrição de um caso de uso; autores normalmente preferem indicar uma lista de ítens que podem ser relevantes e sugerem que sejam escolhidos de acordo com a utilidade de cada um para um determinado projeto de desenvolvimento [FS97]. O livro de Cockburn [Coc00] apresenta diversos formatos com diferentes informações e estilos de descrição.

Neste livro adotaremos o seguinte formato para a descrição de casos de uso, baseado no RUP [JBR99]:

1. **Nome do Caso de Uso**

(a) **Breve Descrição**
...texto...

(b) **Atores**
...texto...

(c) **Pré-condições**:
...texto...

(d) **Pós-condições**:
...texto...

(e) **Requisitos Especiais (requisitos não-funcionais)**:
...texto...

(f) **Referência Cruzada (requisitos mapeados)**:
...lista de requisitos...

2. **Fluxo de Eventos**

(a) **Fluxo Básico**
...passos do cenário...

(b) **Fluxo Alternativo 1**
...passos do cenário...

(c) **Fluxo Alternativo 2**
...passos do cenário...
...

(d) **Fluxo Alternativo N**
...passos do cenário...

No modelo acima, um caso de uso é identificado por seu nome. Este é seguido por (i) uma descrição sucinta, similar às que foram apresentadas na Seção 3.2; (ii) a lista dos atores que participam desse caso de uso; (iii) um conjunto de **pré-condições**, predicados que devem ser verdadeiros para que o cenário seja executado; (iv) um conjunto de **pós-condições**, predicados que devem ser verdadeiros ao final da execução do cenário; (v) uma lista de requisitos especiais, requisitos não-funcionais que o sistema deve apresentar durante a execução do caso de uso, como desempenho e confiabilidade (Seção 3.1); e (vi) uma referência ao requisito (ou conjunto de requisitos) satisfeito(s) pelo caso de uso. A especificação dos fluxos de eventos segue o formato apresentado na Seção 3.4.

3.7 Diagramas de Casos de Uso

Um **diagrama de casos de uso** é uma representação gráfica que mostra um conjunto de casos de uso, atores e seus relacionamentos para um determinado sistema. Ele é um diagrama que contextualiza o sistema no ambiente no qual ele se insere e possui quatro elementos básicos: (i) atores; (ii) casos de uso; (iii) interações; e (iv) fronteira do sistema. A

Figura 3.5 apresenta o diagrama de casos de uso para o sistema da biblioteca. Esse diagrama inclui o caso de uso Emprestar Exemplar apresentado na Figura 3.4, além de três casos de uso novos: Devolver Exemplar, Reservar Publicação e Cancelar Reserva.

Além do seu papel central para a modelagem dos requisitos funcionais de um sistema, os diagramas de casos de uso também são muito utilizados para facilitar a compreensão de sistemas legados, por meio de engenharia reversa [BRJ99].

Como pode ser visto na Figura 3.5, a fronteira do sistema é representada graficamente através de uma linha ao redor dos casos de uso. Essa delimitação explícita do contexto representa a forma como o sistema interage com as entidades externas associadas a ele, que são representadas pelos atores.

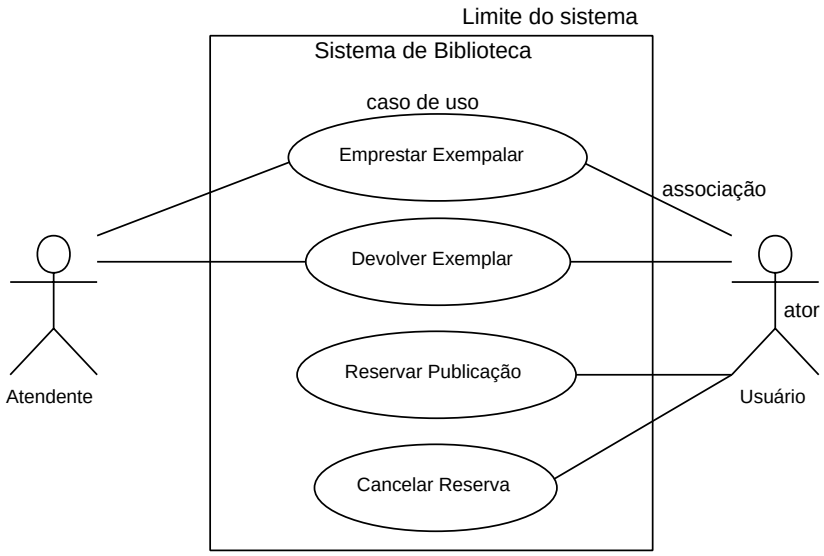

Figura 3.5: Diagrama de Casos de Uso para o Sistema de Biblioteca.

3.8 Relacionamentos entre Casos de Uso

Além das associações entre os casos de uso e seus atores, em um diagrama de casos de uso é possível definir outros relacionamentos, tanto entre os casos de uso, quanto entre os atores. A linguagem UML define três tipos de relacionamentos, que aumentam o valor semântico do diagrama. São eles: (i) generalização (herança); (ii) inclusão ($<< include >>$); e (iii) extensão ($<< extend >>$). Esses relacionamentos são utilizados com a finalidade principal de fatorar tanto o comportamento comum entre os casos de uso, quanto as variações desses comportamentos.

A seguir, as Seções 3.8.1, 3.8.2 e 3.8.3 apresentam cada um desses relacionamentos em maiores detalhes.

3.8.1 Generalização

O relacionamento de generalização entre casos de uso é similar à generalização entre classes, isto é, o caso de uso derivado herda tanto o significado do caso de uso base, quanto o seu comportamento, que normalmente é estendido e especializado. Por exemplo, na Figura 3.6 (a), o caso de uso Emprestar Exemplar é responsável por efetuar o empréstimo de um exemplar. Podemos definir dois casos de uso derivados, Emprestar sem Renovação e Renovar Empréstimo, de tal forma que ambos se comportem igual ao caso de uso Emprestar Exemplar, exceto pelo fato de explicitarem o tipo específico de empréstimo que é feito. Os dois casos de uso derivados podem então ser usados em qualquer lugar onde o caso de uso Emprestar Exemplar é esperado.

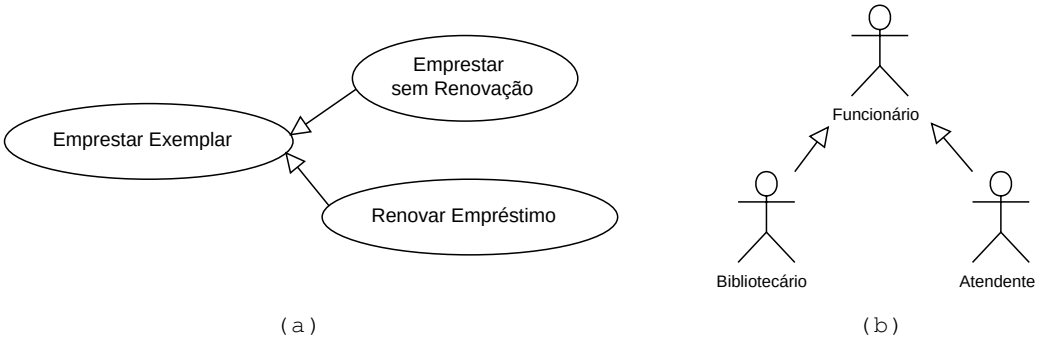

Figura 3.6: Exemplo de Generalização no Diagrama de Casos de Uso.

Num diagrama de casos de uso, o relacionamento de generalização também pode ser usado entre atores, significando que um ator desempenha os mesmos papéis de um outro ator, podendo ainda desempenhar papéis extras. A Figura 3.6 (b) mostra um exemplo de generalização entre atores, onde os atores Bibliotecário e Atendente são derivados de um ator base chamado Funcionário. Isso significa que todos os papéis desempenhados pelo ator Funcionário podem ser desempenhados por qualquer ator derivado, seja ele um Bibliotecário ou um Atendente.

3.8.2 Inclusão

Um relacionamento de inclusão (<< *include* >>) entre casos de uso significa que o caso de uso base sempre incorpora explicitamente o comportamento de outro caso de uso em um ponto específico. Sendo assim, mesmo que o caso de uso incluído não esteja associado a nenhum ator, ele é executado como parte do caso de uso que o inclui. Costuma-se usar um relacionamento de inclusão para evitar a descrição de um mesmo conjunto de fluxo de eventos, através da fatoração do comportamento comum.

Um relacionamento de inclusão é representado em UML como uma dependência cuja seta aponta para o caso de uso incluído, isto é, o caso de uso base "depende" do caso de uso incluído, significando que se o caso de uso incluído for modificado, o caso de uso base também

deve ser revisto. Essa dependência recebe o estereótipo << *include* >>, como mostrado na Figura 3.7. No exemplo da Figura 3.7, o caso de uso Emprestar Exemplar explicita o fato da validação do usuário fazer parte da sua lógica de negócio. Dessa forma, sempre que o caso de uso Emprestar Exemplar é executado, o caso de uso Validar Usuário é executado.

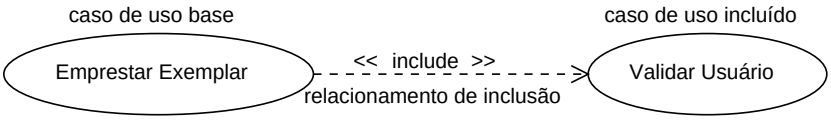

Figura 3.7: Exemplo de Inclusão entre Casos de Uso.

3.8.3 Extensão

Um relacionamento de extensão (<< *extend* >>) entre casos de uso significa que o caso de uso base incorpora implicitamente o comportamento de outro caso de uso num ponto específico. No relacionamento de extensão, diferentemente do que ocorre no relacionamento de inclusão, o caso de uso extensor só tem seu comportamento incorporado pelo caso de uso base em algumas circunstâncias específicas. As condições necessárias para que o caso de uso base possa ser estendido devem ser especificadas explicitamente através de pontos pré-estabelecidos na especificação. Esses pontos são chamados de **pontos de extensão**.

Um relacionamento de extensão é representado em UML como uma dependência cuja seta aponta para o caso de uso base (não para o caso de uso extensor), isto é, o caso de uso extensor "depende" do caso de uso base. Essa dependência possui o estereótipo << *extend* >>, como mostrado no exemplo da Figura 3.8.

O relacionamento de extensão é utilizado para modelar a parte do caso de uso que o usuário considera como sendo um comportamento condicional do sistema. Desta forma, é possível, por exemplo, separar o comportamento opcional do comportamento obrigatório, ou até mesmo expressar um subfluxo separado que é executado apenas em circunstâncias específicas. A Figura 3.8 apresenta o caso de uso Emprestar Exemplar, que é estendido pelo caso de uso Cancelar Reserva.

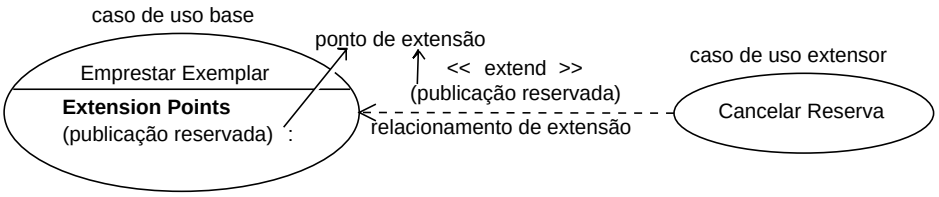

Figura 3.8: Exemplo de Extensão entre Casos de Uso.

Agora que já conhecemos como casos de uso podem se relacionar entre si, o fluxo de eventos do caso de uso Emprestar Exemplar, que já foi definido na Seção 3.4, poderia explicitar

o local onde outros casos de uso são executados. Para isso, serão utilizados os relacionamentos de << *include* >> e << *extend* >> A versão refinada desse fluxo básico é apresentada a seguir:

Fluxo Básico de Eventos:

1. O usuário solicita empréstimo de um ou mais exemplares de publicações (livro, periódico, tese ou manual), fornecendo o seu código e os exemplares desejados;
2. O atendente solicita o empréstimo ao sistema, fornecendo o código do usuário;
3. Para cada exemplar a emprestar:
   3.1 O atendente fornece o número de registro do exemplar.
   3.2 O sistema valida o usuário e verifica o seu *status* ("Normal" ou "Suspenso") através de seu número de registro. (<< *include* >> Validar Usuário);
   3.3 O sistema verifica se o exemplar pode ser emprestado pelo usuário em questão;
   3.4 Se o *status* do usuário for "Normal" e o exemplar estiver disponível:
      3.4.1. O sistema verifica se a publicação do exemplar está reservada. Se estiver reservada (publicação reservada):
         A. O sistema cancela a reserva, passando o número de tombo da publicação (<< *extend* >> Cancelar Reserva)
      3.4.2. O sistema calcula o período do empréstimo, que depende do tipo de usuário - 7 dias para alunos e 15 para professores (<< *include* >> Calcular Tempo de Empréstimo)
      3.4.3. O sistema registra o empréstimo do exemplar;
      3.4.4. O sistema atualiza seu banco de dados com a informação de que o exemplar não irá se encontrar na biblioteca até completar o período.

Neste exemplo, `(publicação reservada)` é um ponto de extensão. Um caso de uso pode ter vários pontos de extensão, que podem inclusive aparecer mais do que uma vez. Em condições normais, o caso de uso Emprestar Exemplar é executado independentemente da satisfação dessas condições. Pporém, se o usuário solicitar o empréstimo de um exemplar que está reservado por ele, então o ponto de extensão `(publicação reservada)` é ativado e o comportamento do caso de uso Cancelar Reserva é então executado. Em seguida, o fluxo de controle continua do lugar de onde foi interrompido (Passo 3.4.2).

3.9 Método para a Modelagem de Casos de Uso

Quando a modelagem de casos de uso é utilizada para melhorar o entendimento dos requisitos funcionais do sistema computacional, é importante que haja a participação ativa dos clientes/usuários do sistema. O usuário entende como o sistema será usado e esse conhecimento

é capturado na forma de casos de uso. Esses casos de uso são carregados através das fases de análise, projeto, implementação, testes e manutenção do sistema, servindo como guia e base para o seu entendimento. A Figura 3.9 mostra como os casos de uso são realizados através das fases de análise e projeto. O modelo conceitual da análise refina os casos de uso e as classes do modelo de projeto, por sua vez, refinam o modelo conceitual da análise. Em geral, durante essas transformações o número de classes aumenta.

O estereótipo $<< trace >>$ entre a fase de análise e a fase de especificação de requisitos, mostrado na Figura 3.9, indica qual conjunto de elementos da análise corresponde à especificação do caso de uso Emprestar Exemplar. Similarmente para o estereótipo $<< trace >>$ entre as fases de análise e projeto. Além de possibilitar o rastreamento entre os artefatos das fases do desenvolvimento do software, o relacionamento de dependência traz informações importantes que auxiliam o gerenciamento da evolução do sistema. Conforme a notação UML, o fato do modelo de análise depender do modelo de casos de uso, implica que se o caso de uso Emprestar Exemplar for alterado, o modelo de análise correspondente deve ser revisto. A mesma semântica também é válida entre os modelos de projeto e de análise.

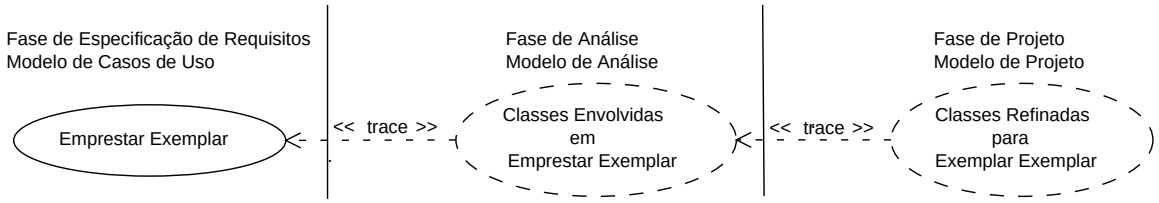

Figura 3.9: Realizações dos Casos de Uso em Modelos Diferentes.

Nesta seção são apresentados três métodos para a construção do modelo de casos de uso, partindo de uma especificação inicial informal dos requisitos do sistema, chegando até a descrição gráfica de seus fluxos de eventos. Apesar de suas características particulares, os métodos expostos aqui são complementares e podem ser utilizados em conjunto, a fim de maximizar a eficácia da elicitação e representação dos requisitos. As abordagens mostradas são:

1. **Identificação de Casos de Uso Baseada em Atores.** Essa técnica se baseia na identificação das funcionalidades requeridas por cada um dos interessados no sistema. O ponto forte dessa abordagem é o seu caráter intuitivo, que motiva a sua popularização na literatura.

2. **Identificação de Casos de Uso Baseada em Atributos.** Essa abordagem enfatiza a necessidade de analisar as informações relevantes para cada entidade conceitual do sistema, isto é, os atributos identificados nas descrições textuais. A partir daí, são identificadas as funcionalidades relativas ao gerenciamento e atualização dessas informações, tais como cadastro e manutenção de dados.

3. **Identificação de Casos de Uso Baseada em Análise de Domínio.** O objetivo principal dessa abordagem é a identificação dos requisitos inerentes ao domínio do

sistema. Por serem normalmente comuns a outros sistemas do mesmo domínio, esses requisitos representam as funcionalidades mais propícias à reutilização.

Cada um desses métodos é composto por uma seqüência de passos que estruturam a modelagem dos requisitos de um sistema através de casos de uso. Além da utilização desses métodos, é bom ter sempre em mente as seguintes dicas que orientam a especificação de casos de uso:

1. Um caso de uso não diz nada sobre o funcionamento interno do sistema, isto é, o sistema é visto como uma caixa preta;
2. Casos de uso são parte do domínio do problema e não da solução;
3. Um caso de uso diz como atores interagem com o sistema e como o sistema responde;
4. Um caso de uso é sempre iniciado ou por um ator, ou por outro caso de uso do qual faça parte;
5. Um caso de uso oferece um resultado observável, sob o ponto de vista do ator;
6. Um caso de uso é completo, isto é, ele possui um começo, um meio e um fim;
7. O fim de um caso de uso é indicado quando o seu resultado observável é obtido pelo ator;
8. Podem ocorrer várias interações entre os atores e os casos de uso, durante a execução dos fluxos de eventos.

O sistema de controle de bibliotecas, descrito no início do capítulo, terá seus casos de uso identificados e especificados em maiores detalhes. Para isso, serão utilizadas as três abordagens de modelagem de casos de uso apresentados anteriormente.

3.9.1 Identificação de Casos de Uso Baseada em Atores

Nesse método, o primeiro passo para a identificação dos casos de uso é identificar os atores que irão interagir com o sistema. Como visto na Seção 3.3, esses atores podem ser pessoas ou outros sistemas externos com os quais o sistema especificado interage. Para ajudar na tarefa de descoberta desses atores, existem algumas perguntas a serem respondidas [SW01]. A seguir, são apresentadas as sete principais questões, respondidas de acordo com o sistema da biblioteca.

1. Quem opera o sistema?

   Resp.: O sistema pode ser operado pelo atendente ou pelo bibliotecário.

2. Quem é responsável pela sua administração?

Resp.: A administração do sistema fica por conta do bibliotecário.

3. Quem é responsável pela manutenção dos seus dados?

   Resp.: A manutenção dos dados é feita pelo atendente (dados de usuários) e pelo bibliotecário (dados de usuários e do acervo).

4. Quem necessita das suas informações?

   Resp.: As informações são úteis para o usuário (alunos, professores e funcionários), para o atendente e para o bibliotecário.

5. Quem oferece informações para o sistema?

   Resp.: As informações podem ser oferecidas pelo usuário (informações pessoais), pelo atendente (informações pessoais) e pelo bibliotecário (informações sobre o acervo).

6. Os outros sistemas utilizam algum dado/processamento do sistema especificado?

   Resp.: O sistema contábil necessita de informações sobre o valor de todas as multas pagas pelos usuários.

7. Acontece algo automaticamente/periodicamente no sistema?

   Resp.: Sim. Uma reserva pode ser cancelada automaticamente, caso o empréstimo não tenha sido efetuado no período estipulado.

Os atores identificados com as respostas dadas foram os seguintes: (i) atendente; (ii) usuário; (iii) aluno; (iv) professor; (v) funcionário; (vi) bibliotecário; (vii) sistema contábil; e (viii) tempo.

Numa biblioteca pequena, é possível que vários desses papéis sejam desempenhados pela mesma pessoa. Em cada papel, esta pessoa age de forma diferente e também espera respostas diferentes do sistema. Dado que os atores já foram descobertos, é possível iniciar a especificação dos casos de uso, a partir do que cada um dos atores espera do sistema. Para isso, para cada um dos atores identificados, devemos considerar seis pontos importantes, que serão apresentados a seguir. As respostas elaboradas se referem ao ator usuário do exemplo da biblioteca.

1. Quais tarefas o ator deseja que o sistema realize?

   Resp.: O usuário deseja "emprestar um exemplar", "devolver um exemplar", "reservar um exemplar", e "cancelar reserva".

2. Quais informações o ator deve fornecer para o sistema?

   Resp.: O usuário pode fornecer as seguintes informações: nome, endereço, título da publicação, número de tombo da publicação, número de registro do exemplar.

3. Existem eventos que o ator deve comunicar ao sistema?

   Resp.: O usuário pode comunicar uma possível mudança de endereço.

4. O ator precisa ser informado de alguma coisa importante pelo sistema?

   Resp.: O usuário deve ser informado quando uma publicação reservada por ele possuir algum exemplar disponível para ser emprestado.

5. O ator é responsável por iniciar ou terminar a execução do sistema?

   Resp.: Sim, nas funcionalidades: 'consultar uma publicação", "reservar um exemplar", e "cancelar reserva".

6. O sistema armazena informações? O ator necessita manipulá-las, isto é, ler, atualizar ou apagar?

   Resp.: Sim. O usuário pode desejar saber os últimos exemplares alugados por ele, ou ainda atualizar seus dados pessoais.

Com base nessas respostas, podemos identificar os seguintes casos de usos: (i) Manter Dados Usuário. O usuário deve ser cadastrado na biblioteca, fornecendo informações sobre seu nome, endereço, e tipo de vínculo com a universidade; (ii) Emprestar Exemplar. O usuário pode emprestar exemplares da biblioteca; (iii) Devolver Exemplar. O usuário pode devolver exemplares emprestados a ele; (iv) Reservar Publicação. O usuário pode reservar um número de exemplares de uma determinada publicação; (v) Cancelar Reserva. O usuário pode cancelar uma reserva feita por ele; (vi) Contactar Usuário. O usuário pode ser contactado quando uma publicação reservada por ele estiver disponível; e (vii) Consultar Histórico Usuário. O usuário pode consultar os últimos exemplares emprestados a ele.

Considerando o atendente, o bibliotecário e o professor, podemos identificar outros casos de uso: (i) Manter Dados Publicação. O bibliotecário pode gerenciar os dados das publicações do sistema; (ii) Manter Dados Exemplar. O bibliotecário pode gerenciar os dados dos exemplares de cada pubicação; (iii) Consultar Histórico Biblioteca. Antes de comprar novos exemplares, uma informação útil ao bibliotecário é saber quais publicações são mais populares; (iv) Bloquear Exemplar. Um professor pode bloquear o empréstimo de exemplares específicos; e (v) Desbloquear Exemplar. Os exemplares bloqueados para empréstimo podem voltar a ser emprestados (desbloqueados);

3.9.2 Identificação de Casos de Uso Baseada em Atributos

Uma outra forma de identificar os casos de uso é considerar os possíveis atributos das entidades do sistema alvo. Por exemplo, uma publicação pode ter associados a ela: um título, o gênero da publicação (técnica, não-técnica), a data de lançamento, etc. Pensando nesses atributos, é possível identificar casos de uso relacionados com funcionalidades de consulta e manutenção. Por exemplo: (i) Consultar Publicação., que representa a consulta da disponibilidade dos exemplares de uma determinada publicação.

De forma análoga, analisando a entidade usuário seria possível identificar o caso de uso Consultar Usuário, uma vez que o atendente pode encontrar o registro de um determinado usuário, informando um dos seus dados pessoais.

3.9.3 Identificação de Casos de Uso Baseada em Análise de Domínio

Analisando o domínio do problema, é possível identificar entidades e funcionalidades mais propícias a serem reutilizadas posteriormente.

A seguir, para cada uma das quatro etapas da análise de domínio, relatamos o produto final obtido para o sistema da biblioteca.

1. **Estudo da viabilidade do domínio.** Esta etapa consistiu na identificação e seleção dos domínio relacionados para o sistema em questão. Foi identificada uma grande semelhança entre sistemas de bibliotecas e sistemas de locação. Além disso, também julgou-se que o domínio do sistema analisado possui características de sistemas de livraria. Baseado nesses domínios, constituiu-se o domínio dos "sistemas de empréstimo de publicações", que agrega as características dos domínios de: (i) sistemas de locação; e de (ii) sistemas de livraria.

2. **Planejamento do domínio.** Em relação à análise de risco relativa ao nosso exemplo, não foram identificados riscos sérios de projeto. Essa decisão se baseou no fato de se tratar de um domínio amplamente conhecido e sem a necessidade de se utilizar tecnologias imaturas.

3. **Contextualização do domínio.** Nesta etapa foi avaliada a contextualização do domínio do ponto de vista do sistema em particular. Com o intuito de acompanhar a tendência especificada no domínio do sistema, é desejável existir um módulo de consulta e reserva pela Internet.

4. **Aquisição do conhecimento do domínio.** Esta fase é responsável pela elicitação final e representação das informações e requisitos relacionados ao domínio. Os artefatos produzidos nesta etapa são descrições textuais semelhantes aos fluxos de eventos dos casos de uso, apresentados na Seção 3.4. Por esse motivo, essa atividade deve ser desempenhada normalmente em conjunto, tanto pelo especialista no domínio, quanto pelo engenheiro de requisitos e os artefatos produzidos aqui servirão de base para a especificação dos casos de uso do sistema.

   No contexto do nosso exemplo, foram identificadas algumas características inerentes aos sistemas de bibliotecas. O conhecimento dessas características foi decorrente da análise dos domínios identificados anteriormente. As principais características encontradas são enumeradas a seguir:

   (a) Uma biblioteca normalmente disponibiliza vários ítens distintos para serem emprestados, não apenas material impresso;

   (b) No caso da devolução ser atrasada, normalmente é cobrada uma multa proporcional ao tempo de atraso (Cobrar Multa) e o usuário pode ser suspenso temporária ou permanentemente ou (Suspender Usuário e Cancelar Suspensão);

   (c) Durante o cadastro de usuários, pode ser conveniente consultar instituições de proteção ao crédito (ator Sistema de Crédito);

Após identificar os casos de uso do sistema utilizando os métodos apresentados anteriormente, o próximo passo é detalhar as suas especificações. Sendo assim, as atividades e conceitos mostrados a partir da Seção 3.9.4 se baseiam apenas nas particularidades do negócio e nos casos de uso identificados, independentemente da abordagem (ou combinação de abordagens) utilizada para tal.

3.9.4 Construção de um Glossário e Termos

Um glossário contém a definição de todos os conceitos utilizados na especificação e modelagem do sistema, que possam comprometer o seu entendimento [Lar97]. Sendo assim, a definição de um glossário busca tanto definir termos desconhecidos, quanto esclarecer conceitos aparentemente similares. O principal benefício desse esclarecimento da terminologia adotada é a melhoria da comunicação, reduzindo os riscos de desentendimento entre os interessados do sistema.

Um entendimento consistente dos termos utilizados no domínio do desenvolvimento é extremamente importante durante as fases de desenvolvimento [Lar97]. Apesar de ser criado originalmente na fase de elicitação de requisitos e planejamento do projeto, esse glossário deve ser refinado continuamente em cada ciclo iterativo. Dessa forma, no decorrer do desenvolvimento, novos termos podem ser adicionados e as definições anteriores podem ser atualizadas.

Uma simplificação do modelo de definição de glossários proposto por Larman [Lar97] é mostrado na Tabela 3.1. O campo `termo` representa o conceito cujo significado é definido no campo `comentário`. Além dessas informações, esse modelo possibilita a especificação de `informações adicionais` referentes ao campo. Essas informações adicionais podem ser utilizadas, por exemplo, para contextualizar o termo descrito em relação à fase do desenvolvimento que ele foi identificado.

Tabela 3.1: Modelo Sugerido para a definição do Glossário.

Termo	Comentário	Inf. Adicionais
Entidade a ser definida	Explicação descritiva	Informações adicionais

No contexto do exemplo da biblioteca, é interessante que seja estabelecida uma distinção entre alguns termos, a fim de evitar ambigüidade nas descrições dos casos de uso. Esse glossário é mostrado na Tabela 3.2, que define os termos (Publicação e Exemplar).

3.9.5 Levantamento Inicial dos Casos de Uso

Uma vez que os casos de uso fundamentais foram identificados, podemos refiná-los descrevendo-os com maiores detalhes. Por exemplo, o caso de uso Devolver Exemplar envolve as seguintes questões que devem ser considerados: (i) a devolução está atrasada? (ii) o exemplar está danificado?

Tabela 3.2: Glossário dos Termos da Biblioteca.

Termo	Comentário	Inf. Adicionais
Publicação	nome coletivo para todos os exemplares de uma determinada publicação. Essa abstração de tipo é utilizada para realizar as operações de consulta e reserva.	(i) fase de requisitos; (ii) exigência do cliente.
Exemplar	cópia individual de uma publicação que pode ser emprestada pelo usuário. Essa é a abstração de tipo que representa o objeto físico.	(i) fase de requisitos; (ii) exigência do cliente.

Todas essas questões sugerem uma revisão dos casos de uso identificados até agora, gerando a seguinte lista parcial de casos de uso:

[**Caso #1**] Reservar Publicação. O usuário pode reservar um número de exemplares de uma determinada publicação.

[**Caso #2**] Cancelar Reserva. O usuário pode cancelar uma reserva feita por ele.

[**Caso #3**] Contactar Usuário. O usuário pode ser contactado quando uma publicação reservada por ele estiver disponível.

[**Caso #4**] Emprestar Exemplar. O usuário pode emprestar exemplares da biblioteca.

[**Caso #5**] Devolver Exemplar. O usuário pode devolver exemplares emprestados a ele.

[**Caso #6**] Cobrar Multa. Multa cobrada nos casos onde há devoluções em atraso.

[**Caso #7**] Bloquear Exemplar. Um professor pode bloquear o empréstimo de exemplares específicos.

[**Caso #8**] Desbloquear Exemplar. Os exemplares bloqueados para empréstimo podem voltar a ser emprestados (desbloqueados).

[**Caso #9**] Suspender Usuário. Usuários podem ser suspensos para locação.

[**Caso #10**] Cancelar Suspensão. Usuários suspensos provisoriamente podem voltar a realizar empréstimos.

[**Caso #11**] Manter Dados Usuário. O usuário deve ser cadastrado na biblioteca, fornecendo informações sobre seu nome, endereço, e tipo de vínculo com a universidade.

[**Caso #12**] Manter Dados Publicação. O bibliotecário pode gerenciar os dados das publicações do sistema.

[**Caso #13**] Manter Dados Exemplar. O bibliotecário pode gerenciar os dados dos exemplares de cada pubicação.

[**Caso #14**] Consultar Usuário. O atendente pode encontrar o registro de um determinado usuário informando um dos seus dados pessoais.

[**Caso #15**] Consultar Publicação. O usuário pode consultar a disponibilidade dos exemplares de uma determinada publicação.

[**Caso #16**] Consultar Histórico Usuário. O usuário pode consultar os últimos exemplares emprestados a ele.

[**Caso #17**] Consultar Histórico Biblioteca. Antes de comprar novos exemplares, uma informação util ao bibliotecário é saber quais publicações são mais populares.

Refinamento de Casos de Usos Relacionados

Uma vez que uma lista de casos de uso tenha sido obtida, você pode unir e refinar aqueles que são similares e definir várias versões, cenários e variantes para cada um deles. Por exemplo, os casos de uso #14 e #15 são parte dos casos de uso #11 e #12, respectivamente. As diversas atividades de manutenção: consulta, cadastro, alteração e exclusão podem ser descritas como fluxos do mesmo caso de uso. Outra possibilidade é definí-los como extensões do caso de uso mais geral, utilizando o relacionamento de $<< extend >>$, como mostrado na Figura 3.10 para o caso de uso Manter Dados Publicação. Apesar das várias possibilidades de especificação de um diagrama de casos de uso, existe uma relação de compromisso (do inglês *tradeoff*) entre a complexidade dos casos de uso e o número excessivo de casos de uso em um sistema.

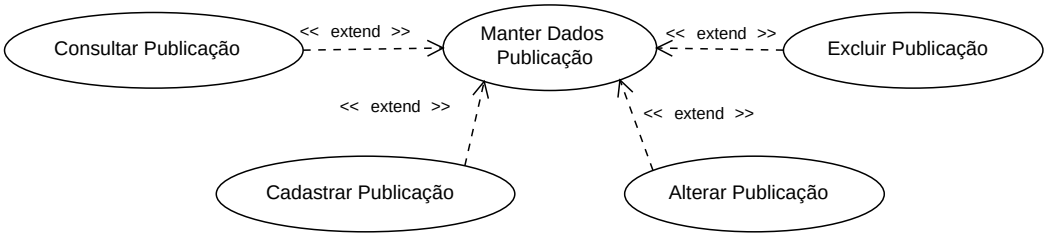

Figura 3.10: Modelagem Alternativa do Caso de Uso Manter Dados Publicação.

3.9.6 Descrição de Casos de Usos

Cada caso de uso da lista preliminar deve receber um nome único e, em seguida, deve ser atribuído a ele um comentário expandido que proporcione um entendimento mais detalhado. Por exemplo, o caso de uso #5 (Devolver Exemplar) pode expandir a sua descrição inicial, de modo a envolver os atores interessados na sua execução (usuário e atendente). Uma descrição mais completa do caso de uso Devolver Exemplar é apresentada a seguir:

Caso de Uso: Devolver Exemplar.
Atores: Usuário e Atendente.

Descrição: Este caso de uso representa o processo de devolução de empréstimo de um ou vários exemplares da biblioteca. A devolução se inicia com a solicitação feita pelo usuario ao atendente. Em seguida, através de um terminal, o atendente solicita ao sistema a devolução dos respectivos exemplares. Se a devolução estiver em atraso e o usuário não for um professor, é cobrada uma multa e o usuário pode ser suspenso.

3.9.7 Gerenciamento de Casos de Uso Complexos

Para promover o reuso e tornar mais claras as especificações dos casos de uso do sistema, é útil usar os relacionamentos entre casos de uso descritos na Seção 3.7. Por exemplo, o caso de uso Cancelar Reserva, identificado na Seção 3.9.5, descreve em detalhes como um usuário cancela uma reserva feita para uma determinada publicação. Por outro lado, o caso de uso Emprestar Exemplar pode envolver o cancelamento de uma reserva como parte do processo de empréstimo. Neste caso, ao invés de copiarmos o caso de uso Cancelar Reserva para o caso de uso Emprestar Exemplar, podemos dizer que Cancelar Reserva estende Emprestar Exemplar através do relacionamento $<< extend >>$.

É possível usar também o relacionamento de inclusão, visto na Seção 3.8.3. Por exemplo, sempre que o empréstimo é realizado ou quando um exemplar é devolvido ou bloqueado, o usuário deve ser validado para verificar a sua possibilidade de executar o serviço solicitado (caso de uso Validar Usuário). Com o intuito de explicitar a reutilização desde o modelo de casos de uso, pode-se dizer que o caso de uso Validar Usuário é incluído pelos casos de uso Emprestar Exemplar, Devolver Exemplar e Bloquear Exemplar.

Pacotes

Para controlar a complexidade do modelo, à medida que o número de casos de usos cresce, se faz necessário utilizar o conceito de módulos. A modularização é implementada em UML através do conceito de pacotes. Um **pacote** agrupa um conjunto de entidades UML relacionadas. Vários critérios diferentes podem ser utilizados na hora de definir como os casos de uso serão empacotados. Por exemplo, você pode agrupar casos de uso que interagem com o mesmo ator, ou aqueles que estabelecem relacionamentos de inclusão, extensão ou generalização (casos de uso mais acoplados). Como exemplo de uma abordagem mista, a Figura 3.11 apresenta a visão modularizada dos casos de uso do sistema de bibliotecas. Enquanto os Pacotes 1 e 2 agrupam os casos de uso baseado nos atores, os Pacotes 3 e 4 agrupam pela semelhança das suas funcionalidades. A Figura 3.11 também representam uma hierarquia de casos de uso, agrupando os casos de uso principais juntos e colocando os menos importantes em pacotes secundários (sub-pacotes). Dessa forma, o pacote de nível mais alto, chamado de *top-level*, representa o modelo completo do sistema.

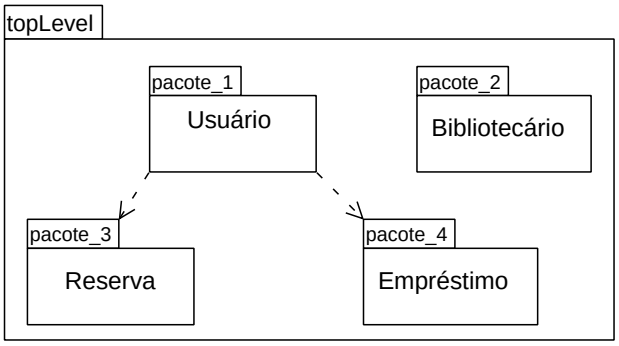

Figura 3.11: Pacotes de Casos de Uso.

3.9.8 Descrições Formais de Casos de Usos

A definição de assertivas é uma maneira muito utilizada para a formalização das especificações dos casos de uso. Esse princípio se baseia na técnica de projeto por contrato(do inglês *design by contract*), definido por Bertrand Meyer [Mey91]. Sua filosofia principal é a definição de restrições que devem ser satisfeitas antes (pré-condições) ou após (pós-condições) a execução da funcionalidade especificada. As pré- e pós-condições são adicionadas à especificação dos casos de uso, conforme o modelo mostrado na Seção 3.6.

Uma **pré-condição** de um caso de uso descreve restrições no sistema antes de um caso de uso iniciar. Por exemplo, antes do caso de uso Devolver Exemplar começar, o usuário deve estar cadastrado na biblioteca e deve ter emprestado o exemplar que está sendo devolvido. Além disso, a biblioteca deve estar aberta. Um caso de uso só tem o compromisso de executar corretamente nos cenários onde as suas pré-condições são satisfeitas. Cada um desses cenários podem ter uma versão mais detalhada das suas pré-condições.

As **pós-condições** de um caso de uso descrevem o estado do sistema e possivelmente dos atores, depois que o caso de uso foi completado. Elas são verdadeiras para todos os cenários do caso de uso, embora cada cenário possa descrever suas próprias pós-condições mais detalhadamente. Por exemplo, ao final do caso de uso Devolver Exemplar, a situação do usuário deve estar regularizada. Além disso, uma multa pode ou não ter sido cobrada, dependendo do tipo de usuário e das condições da devolução.

3.9.9 Diagrama de Casos de Uso do Sistema da Biblioteca

Utilizando os conceitos apresentados na Seção 3.7, a Figura 3.12 mostra o diagrama de casos de uso do sistema de gerenciamento de bibliotecas. Lembrando que dependendo da decisão do analista, poderiam ser produzidas outras versões corretas do diagrama.

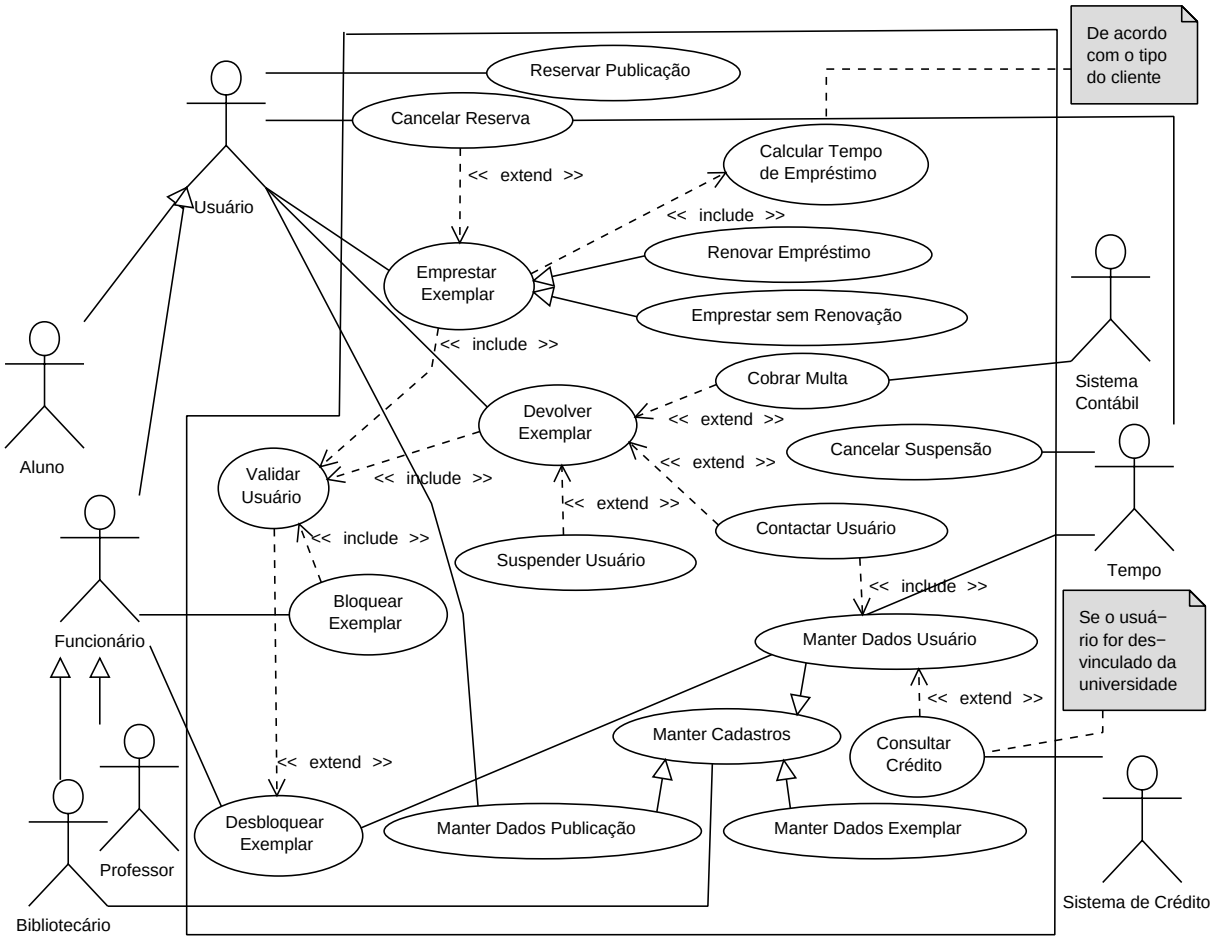

Figura 3.12: Diagrama de Casos de Uso do Sistema da Biblioteca.

3.9.10 Diagrama de Atividades para Fluxo de Eventos

Diagramas de atividades possuem elementos adicionais que tornam possível a representação de desvios condicionais e execução concorrente de atividades. Dessa forma, todos os fluxos de eventos de um caso de uso (básico e alternativos) podem ser representados graficamente através de um único diagrama. Um diagrama de atividades é bastante similar a uma máquina de estados na qual os estados correspondem a atividades e as transições são, em sua maioria, vazias.

A Figura 3.13 apresenta um diagrama de atividades contendo os principais elementos que podem ser encontrados em um diagrama de atividades. O fluxo de eventos descrito pelo diagrama começa no **estado inicial**, indicado pelo círculo preto na parte superior da figura, e termina no **estado final**, indicado pelos dois círculos concêntricos, o menor preto e o maior branco, na parte inferior da figura 3.13.

Um **desvio** ou **ramificação** (do inglês *decision*) é uma estrutura utilizada para especificar caminhos alternativos de execução. Como pode ser visto na Figura 3.13, ele é representado

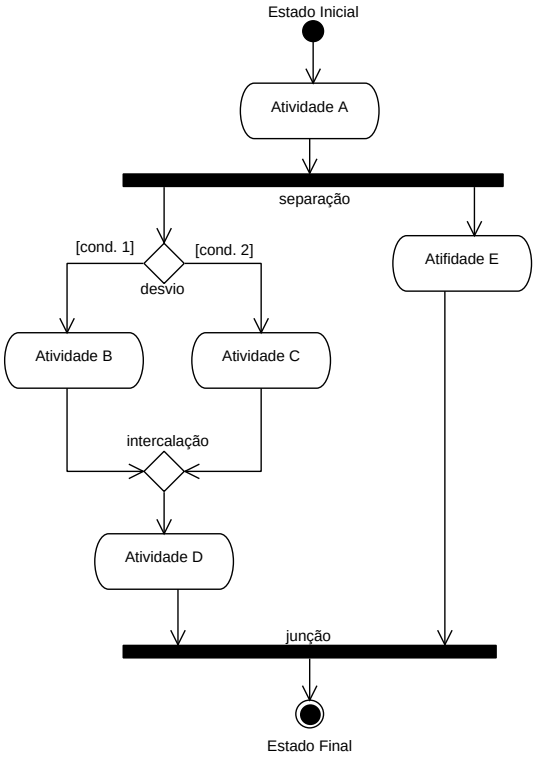

Figura 3.13: Um Exemplo Simples de Diagrama de Atividades.

por um losango que possui uma transição de entrada e várias saídas distintas. A seleção da saída correta se dá a partir da resolução de expressões booleanas, definidas em cada uma das saídas. Um desvio é similar a um comando condicional de uma linguagem de programação. Na Figura 3.13, o único desvio existente escolhe a Atividade B se a condição [cond1] for verdadeira e a Atividade C se a condição [cond2] for verdadeira (relembrando: as condições devem ser disjuntas). Uma **intercalação** (do inglês *merge*), que é opcional, indica o final da execução de um bloco condicional iniciado por um desvio. Apesar de também ser representado por um losango, a intercalação é complementar ao desvio, possuindo múltiplas transições de entrada e apenas uma de saída. Ao término do bloco condicional, independentemente de qual atividade tenha sido executada (B ou C), a Atividade D é executada.

Uma das características mais importantes de um diagrama de atividades é a facilidade de se representar atividades que são executadas de forma concorrente. Para indicar que dois subfluxos do fluxo de eventos devem executar concorrentemente, usa-se o símbolo de **separação** (do inglês *fork*), que é representado por uma barra sólida (horizontal ou vertical) com exatamente um fluxo de entrada e vários fluxos de saída. Essas saídas indicam subfluxos que são executados em paralelo quando a transição é acionada.

Sempre que iniciamos uma execução concorrente, é necessário indicar o ponto da execução onde a concorrência termina. Nesse local acontece o que chamamos de sincronização, per-

manecendo assim até que todos os caminhos de execução finalizem. Para representar o fim da concorrência em um diagrama de atividades, utiliza-se o símbolo de **junção** (do inglês *join*), também conhecido como união. Sua representação também se dá a partir de uma barra sólida, porém, com duas ou mais entradas e uma única saída. Um detalhe importante é o balanceamento entre separações e junções. Isso significa que o número de fluxos que saem de uma separação deve ser necessariamente o mesmo número de fluxos que entram na junção correspondente.

Na Figura 3.13, logo depois que a Atividade A termina, dois subfluxos concorrentes são executados; um contendo apenas a Atividade E e o outro contendo as atividades B, C e D. Esses subfluxos se juntam imediatamente antes do estado final ser alcançado.

A Figura 3.14 apresenta um diagrama de atividades correspondente ao fluxo de eventos do caso de uso Emprestar Exemplar. Neste diagrama são representados dois fluxos alternativos. O primeiro é iniciado quando o usuário está suspenso para empréstimo. O segundo por sua vez, é ativado quando apesar do cliente estar apto a realizar empréstimos, o exemplar desejado encontra-se indisponível para empréstimo. Essa indisponibilidade pode ser conseqüência de um bloqueio por parte de um professor.

Além de oferecer uma maneira gráfica de representar o fluxo de eventos de um caso de uso, diagramas de atividade também são úteis para descrever algoritmos seqüenciais complicados e especificar o comportamento de aplicações paralelas e concorrentes.

3.9.11 Diagramas de Interação de Sistema

As interações entre os atores e o sistema podem ser representadas através de **diagramas de interação de sistema**. A principal característica desses diagramas é a representação do sistema como uma caixa-preta. Assim, a semântica proporcionada pelos diagramas de interação de sistema descreve o que o sistema faz sem se preocupar com o como é feito [Lar97]. A representação da interação do sistema pode ser feita tanto através de um **diagrama de seqüência**, quanto através de um **diagrama de colaboração**, que são diagramas dinâmicos da UML. Esses diagramas, descritos a seguir, serão detalhados no Capítulo 5.

De uma maneira geral, os diagramas de interação de sistema são modelos que ilustram eventos entre atores e o sistema. Ele mostra, para um cenário particular de um caso de uso, os eventos que os atores geram e a ordem com que esses eventos são executados. Dessa forma, a ênfase desses diagramas está nos eventos que cruzam a fronteira do sistema [Lar97].

A principal diferença entre os diagramas de seqüência e os diagramas de colaboração está na ênfase dada por cada um deles. Enquanto os **diagramas de seqüência** enfatizam a ordem em que os participantes da interação se comunicam, os **diagramas de colaboração** enfatizam as conexões entre os participantes.

A Figura 3.15 mostra um diagrama de seqüência de sistema com as interações em um cenário do caso de uso Emprestar Exemplar, onde o usuário realiza o empréstimo de um exemplar com sucesso. O diagrama apresenta os atores do sistema da Videolocadora e o objeto (:SistemaBiblioteca) que representa o próprio sistema computacional.

Em um diagrama de seqüência, os participantes da interação aparecem no topo do di-

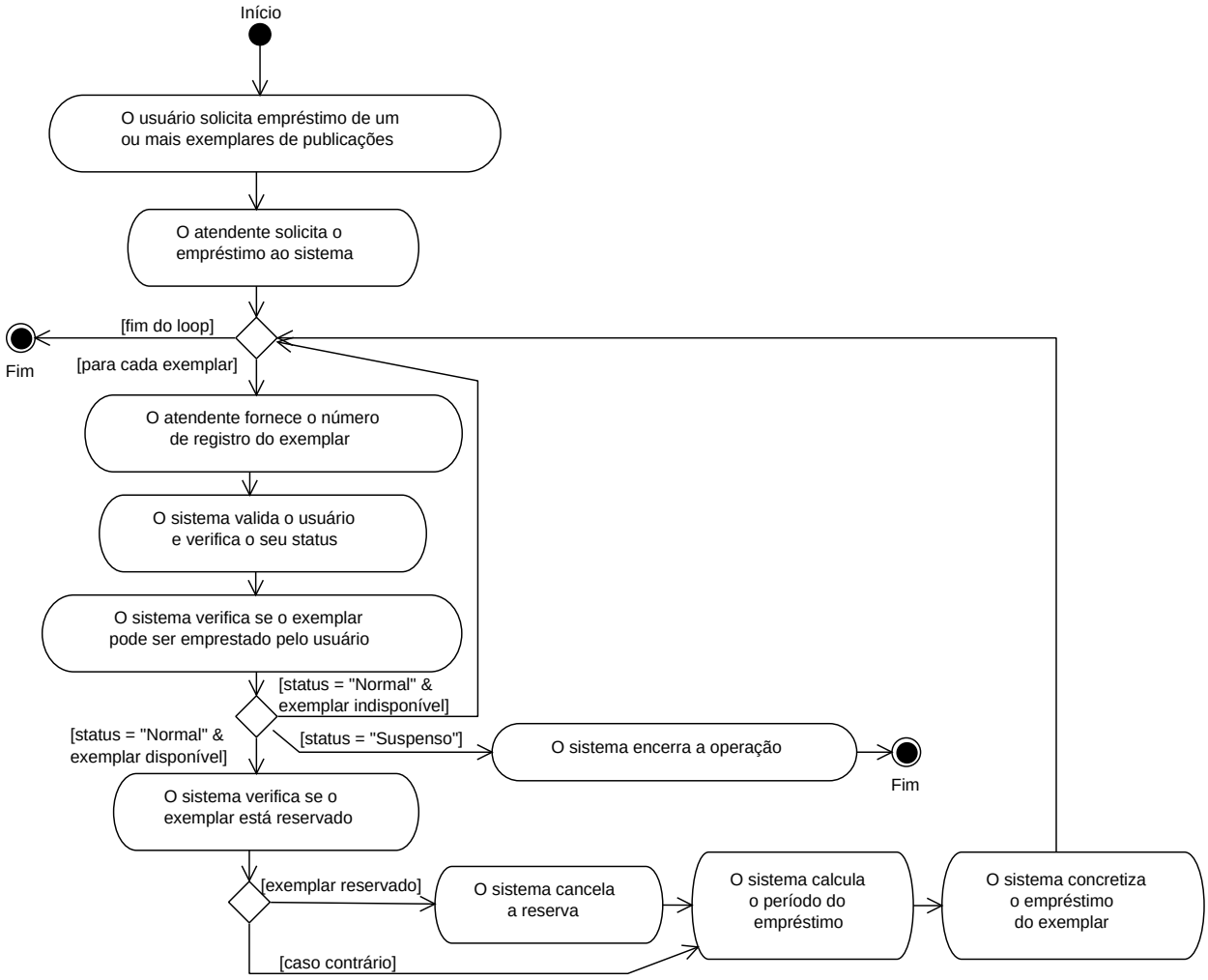

Figura 3.14: Diagrama de Atividades para o Caso de Uso Emprestar Exemplar.

agrama, como retângulos. Os nomes associados aos participantes indicam seus tipos e são sublinhados para ressaltar que eles são instâncias de classes, isto é, objetos e não classes.

As mensagens entre os participantes são representadas pelas setas entre as linhas verticais abaixo dos mesmos. Além de representar as mensagens em si com sua origem e destino, os diagramas de seqüência enfatizam a ordem cronológica entre todas as mensagens da interação. Essa ordem é definida através da posição que a mensagem ocupa na linha de tempo do objeto (linha vertical tracejada). Por exemplo, na Figura 3.15, "Emprestar Exemplar" ocorre antes de "Registrar empréstimo", já que se encontra em uma posição mais alta no diagrama.

Além de enviar mensagens para outros participantes, um participante pode enviar uma mensagem para si mesmo ("auto-referenciação") para indicar que está realizando algum trabalho interno. Na figura, um exemplo disso é a mensagem "Validar usuário", que o

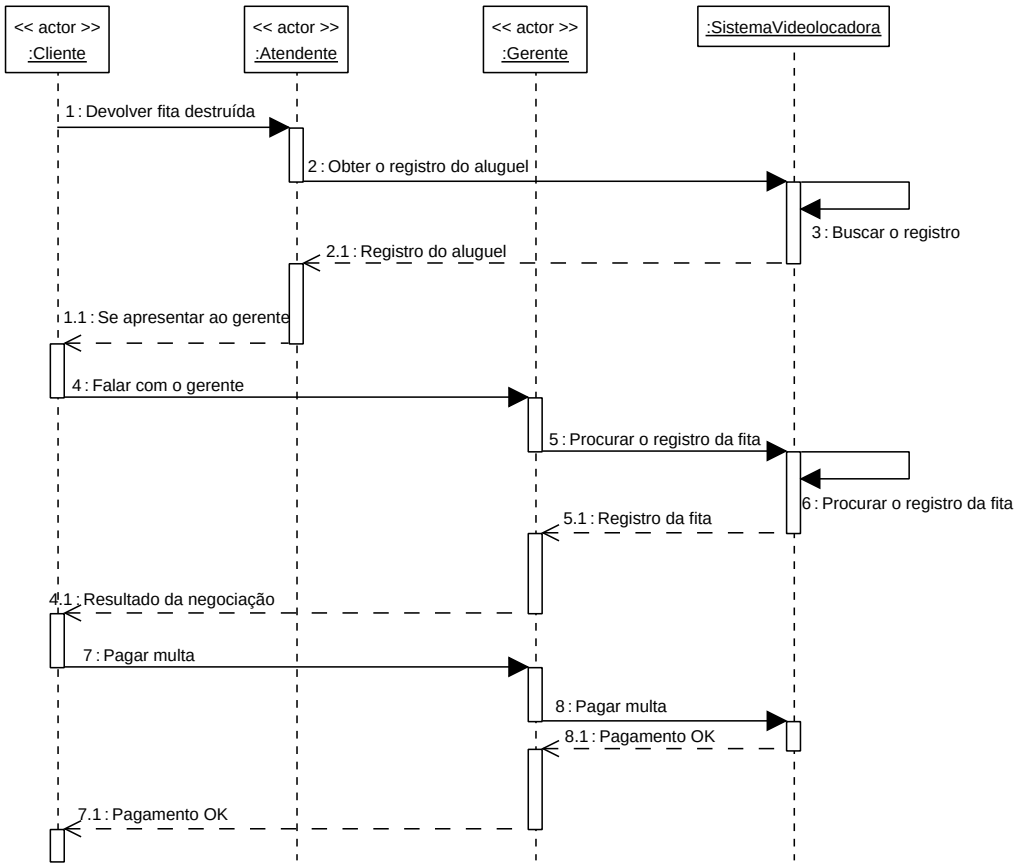

Figura 3.15: Diagrama de Seqüência de Sistema para um Cenário de Emprestar Exemplar.

objeto :SistemaBiblioteca envia para si mesmo quando recebe uma mensagem "Emprestar exemplar", advinda do objeto :Atendente.

Você pode também construir um diagrama de colaborações de sistema (Figura 3.16) para uma melhor visualização das associações entre os diversos objetos (no caso, atores e o sistema). Por exemplo, na Figura 3.16, é fácil ver que o bibliotecário e o atendente extraem informações do sistema não se comunicam entre si.

Em um diagrama de colaboração, os participantes da interação são representados da mesma maneira como aparecem em um diagrama de seqüência, porém, não é necessário dispor os objetos de uma maneira pré-definida. Associações entre os objetos são indicadas explicitamente no diagrama através de linhas que ligam dois participantes. As mensagens, por sua vez, são representadas por setas. Diferentemente do que ocorre em diagramas de seqüência, nos quais a ordem entre as mensagens é explícita, num diagrama de colaboração ela só pode ser representada através do artifício de enumeração das mensagens.

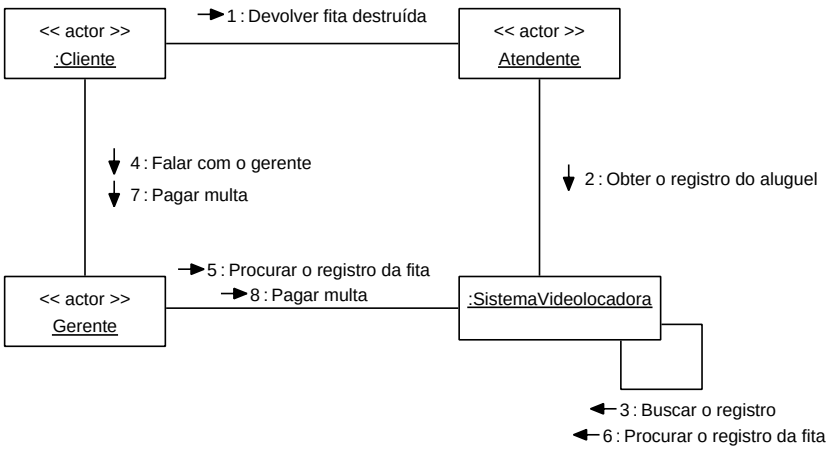

Figura 3.16: Diagrama de Colaboração de Sistema para um Cenário de Emprestar Exemplar.

3.9.12 Início da Análise

Nos diagramas anteriores, o sistema foi tratado de forma monolítica (como uma caixa preta); a fase de análise é responsável por particionar/identificar os objetos que compõem o sistema internamente. Nessa fase, os componentes do sistema são identificados gradativamente a partir dos casos de uso e representados num **diagrama de classes** UML. Como apresentado na Figura 3.17, esse diagrama é construído a partir da identificação das principais entidades conceituais do sistema e da especificação da maneira como elas se relacionam. Cada entidade, que em OO é conhecida como **classe**, encapsula os seus dados através de **atributos** e o seu comportamento através de **operações**.

A identificação das classes do sistema baseia-se principalmente na análise do domínio do negócio, definido na especificação dos casos de uso. No contexto do sistema para controle de bibliotecas, nós sabemos que a entidade "publicação" faz parte do seu domínio. Além disso, sabemos que esses livros são tipos de "item emprestável". A decisão de modelar o conceito de publicação separado do conceito de item emprestável se baseia na possibilidade da biblioteca disponibilizar outros ítens para empréstimo, como por exemplo, DVDs, CDs de música , etc., como mostrado na Seção 3.9.3. As operações `emprestar()` e `devolver()`, comuns a todos os ítens emprestáveis, são identificadas na classe ItemEmprestável.

O diagrama de classes do sistema da biblioteca mostrado na Figura 3.17 é parcial. No Capítulo 3 veremos uma forma sistemática de construir um diagrama de classes mais elaborado, a partir das especificações dos casos de uso.

3.10 Resumo

Os requisitos de um software representam a idéia do usuário a respeito do sistema que será desenvolvido. Em geral, os requisitos podem ser vistos como condições ou capacidades ne-

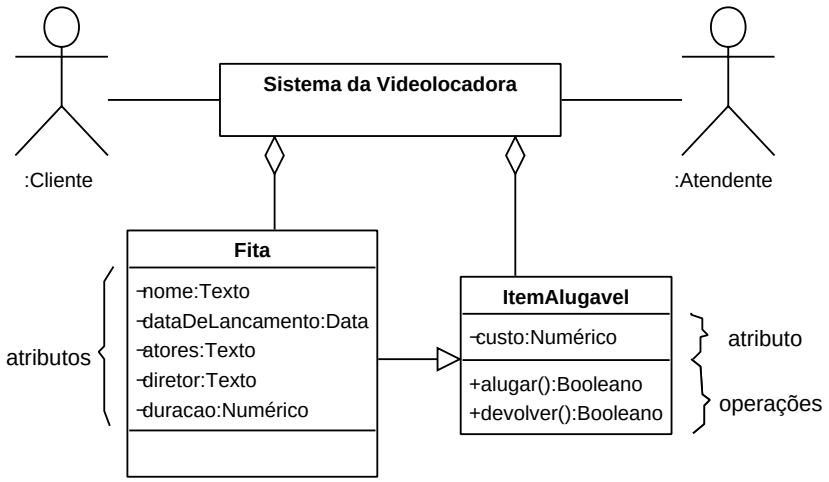

Figura 3.17: Modelo de Classes Preliminar para o Sistema da Biblioteca.

cessárias para resolver um problema ou alcançar os objetivos pretendidos. Durante a fase de especificação dos requisitos é feita uma distinção clara entre as funcionalidades especificadas para o sistema (requisitos funcionais) e os atributos de qualidade desejados, que são materializados pelos requisitos não-funcionais. Apesar de não representarem funcionalidades diretamente, os requisitos não-funcionais podem interferir na maneira como o sistema deve executá-las.

Os requisitos de um sistema são representados através de um documento, conhecido como documento de requisitos. Esse documento deve registrar a necessidade real dos usuários e suas expectativas, dimensionar a abrangência do sistema e delimitar o seu escopo. Em UML, essa representação é feita através do conceito de casos de uso.

A modelagem de casos de uso é uma técnica que auxilia o entendimento dos requisitos de um sistema computacional através da criação de descrições narrativas dos processos de negócio (casos de uso), além de delimitar claramente o contexto do sistema computacional (atores).

Existem três maneiras de representar a forma como dois casos de uso se relacionam entre si: (i) relacionamento de inclusão ($<< include >>$), que indica a inclusão do comportamento de um caso de uso em outro; (ii) relacionamento de extansão ($<< extend >>$), que indica uma inclusão condicional; e (iii) relacionamento de generalização/especialização, que representa o conceito de subtipo.

O comportamento de um caso de uso é detalhado através da descrição de fluxos de eventos. Esses fluxos definem uma seqüência de comandos declarativos, incluindo desvios condicionais e iterações. Instâncias desses fluxos, onde as decisões e iterações são pré-definidas são conhecidas como cenários de uso. Em UML, a forma indicada para representar graficamente os fluxos é através de diagramas de atividades. Já os cenários de uso são normalmente representados através de diagramas de seqüência ou diagramas de colaboração. Por ocultar os detalhes internos do sistema e se preocupar unicamente com a representação da interação do

sistema com o meio externo, os diagramas de seqüência e colaboração feitos durante a especificação dos casos de uso são classificados como diagramas "de sistema", isto é, diagramas de seqüência de sistema e diagramas de de colaboração de sistema, respectivamente.

Com os casos de uso especificados, o próximo passo é detalhar o sistema internamente. Do ponto de vista da análise orientada a obetos, esse detalhamento consiste na identificação das classes internas que compõem o sistema e do relacionamento e regras de interação entre elas.

3.11 Exercícios

1. Dado o seguinte enunciado de um sistema de Ponto de Venda (PDV):

   > Um sistema de ponto de vendas é um sistema computacional usado para registrar vendas e efetuar pagamentos. Ele inclui componentes de hardware, como um computador e um *scanner* de código de barras, além dos componentes de software para o controle do sistema. Nesse exemplo, estamos interessados na compra e pagamento de produtos. Os requisitos básicos de funcionamento do sistema são nove: (i) registrar os ítens vendidos em cada venda; (ii) calcular automaticamente o total de uma venda, incluindo taxas; (iii) obter e apresentar as informações sobre cada produto mediante a leitura de seu código de barras; (iv) reportar ao estoque a quantidade de cada produto vendido quando a venda é completada com sucesso; (v) registrar cada venda completada com sucesso; (vi) exigir que o atendente forneça sua senha pessoal para que possa operar o sistema; (vii) o sistema deve ter um mecanismo de armazenamento de memória estável; (viii) receber pagamentos em dinheiro ou cartão de crédito; e (ix) emitir mensalmente o balanço do estoque.

   (a) Classifique os nove requisitos do sistema de pontos de venda entre funcionais e não-funcionais. Os requisitos não-funcionais devem ser categorizados de acordo com a nomenclatura ABNT/ISO 9126, apresentada na Seção 3.1 .

   (b) Especifique o diagrama de casos de uso do sistema;

   (c) Identifique os relacionamentos entre os diversos casos de uso do sistema usando $<< include >>$ e $<< extend >>$.

   (d) Descreva um dos casos de uso do sistema usando o formato da Seção 3.6.

2. Dado o seguinte enunciado de um sistema de Controle de Videolocadora:

   > Um sistema de controle para uma videolocadora tem por objetivo automatizar o processo de locação e devolução de fitas de vídeo. Deve-se manter um controle dos cadastros de clientes e seus respectivos dependentes e também um controle sobre o acervo de fitas e sua movimentação.

Os clientes podem executar operações que envolvem locação, devolução e compra de fitas. Caso a fita não seja devolvida no prazo previsto, uma multa será cobrada. Caso o cliente perca ou danifique uma fita alugada, ele deve ou pagar uma multa equivalente ao preço de uma fita nova, ou comprar uma nova fita para substituir a que foi danificada.

(a) Especifique o diagrama de casos de uso do sistema;

(b) Identifique os relacionamentos entre os diversos casos de uso do sistema usando $<< include >>$ e $<< extend >>$.

(c) Atualize o diagrama de casos de uso especificado para contemplar as seguintes restrições, adicionadas ao sistema:

- O cliente *VIP* pode alugar um número ilimitado de fitas; caso contrário, o número máximo de fitas é limitado a três.
- O pagamento pode ser efetuado no ato da locação ou da devolução e pode ser feito em dinheiro, com cartão de crédito, ou através de "cheque-vídeo", que é comprado antecipadamente. Se pagar com "cheque-vídeo", o cliente recebe um desconto especial.

3. Considere os seguintes requisitos de um sistema para gestão de um estacionamento:

- O controle é efetuado com base na placa do veículo, que deve estar cadastrado no sistema.
- Na entrada do estacionamento deve existir um funcionário inserindo os números das placas no sistema, que registra automaticamente a data e a hora de início.
- Se o veículo não estiver cadastrado, o funcionário deve cadastrá-lo. O cadastro é feito em duas etapas: (i) cadastro do responsável; e (ii) associação do veículo ao responsável.
- O funcionário também pode editar os dados do responsável ou alterar os veículos associados a ele.
- Na saída do estacionamento, o funcionário registra novamente o número da placa do veículo. Nesse momento, o sistema calcula o valor a pagar pelo serviço.
- O estacionamento tem clientes avulso e clientes mensalistas. Para os clientes mensalistas, o sistema deve oferecer a possibilidade de adicionar o valor na conta do usuário, que é paga mensalmente.
- O gerente do estacionamento consulta diariamente um relatório do sistema. Em algumas situações, o gerente poderá desempenhar as funções de atendimento, no entanto, apenas o gerente pode solicitar o relatório.

Construa um diagrama de casos de uso para esse sistema. Tente aplicar os relacionamentos de $<< include >>$ e $<< extend >>$ no seu modelo.

4. Modele o diagrama de casos de uso de um sistema de gerenciamento de hotel descrito a seguir, seguindo as diretrizes propostas na Seção 3.9:

   Um grupo de empresários deseja que sua equipe desenvolva um sistema para gerenciar reservas e ocupações de apartamentos em uma rede de hotéis. O sistema será utilizado para controlar serviços internos de cada hotel e para a comunicação entre hotéis da rede de forma que seja possível que uma unidade da rede faça consultas sobre a disponibilidade de vagas em outras unidades da mesma cidade ou região. Os serviços básicos a ser considerados são:

   - Entrada para cadastro de cliente (nome, endereço, e-mail, data de chegada, data de saída, classificação do cliente, documento);
   - Consultas, reservas e cancelamento de reserva através da Web;
   - Caso uma reserva não seja cancelada e nem ocupada no prazo especificado, é cobrada uma tarifa de *no show*, correspondente a 30% do valor da diária;
   - Cadastro de apartamento: tipo de apartamento (suíte, standard, duplo, ar-condicionado), cidade ou local;
   - Cadastro de salas e auditório;
   - Cadastro de despesas;
   - Serviços adicionais são também incluídos no sistema: telefone, TV paga, acesso à internet, "frigobar", lavanderia, serviço de lanche e café da manhã;
   - Conexão para consultas e reservas de vagas em outros hotéis do grupo;
   - Controle de ocupação de apartamentos (reservado ou entrada do hóspede);
   - Controle de ocupação de salas e auditório;
   - Controle de limpeza dos apartamentos;
   - Descontos para clientes VIP e grupos;
   - Recebimento de pagamento (tipo de pagamento cheque, dinheiro, cartão, parcelado, moeda estrangeira);
   - Emissão de nota fiscal (podendo ser separado por ítens: hospedagem, restaurante, lavanderia, etc);
   - Emissão da fatura parcial (somente para consulta);
   - Emissão de relatórios contábeis;
   - Emissão de relatórios de ocupação;
   - Emissão de relatórios de hóspedes em débito;
   - Relatórios parciais de consulta;
   - Gerar relatórios estatísticos (média de dias que o cliente hospeda, gastos médios, ítens mais consumidos nos restaurantes);
   - Esse sistema deve ser interligado a um sistema contábil, que é responsável pelo pagamento dos serviços consumidos no hotel.

Capítulo 4

Planejamento de projetos de software

O sucesso no desenvolvimento de software depende, em grande parte, da qualidade técnica e administrativa dos projetos. Este capítulo trata de um conjunto de atividades que têm como objetivo melhorar a qualidade através da melhoria administrativa e técnica dos projetos.

O plano de projeto oficializa estimativas feitas para custos, prazos e recursos do projeto, permitindo que gerentes possam acompanhar e controlar o processo e o produto de software a partir de dados quantitativos. O plano não é estático e deve evoluir junto com os progressos alcançados no desenvolvimento e, por trabalhar com previsões para o futuro, só pode ser avaliado parcialmente enquanto o desenvolvimento não tiver sido concluído. A comparação entre o que foi planejado e o que realmente ocorre no projeto permite ao gerente controlar de forma mais eficiente o processo de desenvolvimento.

O primeiro plano para o projeto de software consiste em duas partes [TOM 89]. A primeira tem como objetivo determinar as necessidades específicas do usuário (definição e análise do escopo do projeto). As técnicas de extração de requisitos vistas no capítulo 2 são bastante úteis para a realização dessa primeira parte do plano. A segunda parte determina como implementar o sistema para suprir essas necessidades; inclui o desenvolvimento do sistema, validação, instalação, treinamento e operação. Os modelos de custo vistos no capítulo 3 são bastante úteis nessa fase.

O planejamento deve ser um processo iterativo no qual a definição do escopo do projeto é o primeiro estágio da iteração. As atividades envolvidas no planejamento do desenvolvimento de software são:

(a) determinação de objetivos e restrições do projeto:
   - observação dos requisitos do usuário;
   - elaboração da declaração de objetivos e restrições;

(b) estudo de viabilidade:
   - elaboração da lista de alternativas;
   - elaboração de estimativas de custo, tempo e recursos;

- determinação dos riscos;
- análise de custo–benefício;

(c) organização do projeto:

- organização do desenvolvimento;
- organização da equipe;
- programação do projeto.

4.1 Objetivos e restrições

Como já foi visto no capítulo 2, a primeira tarefa do desenvolvedor de software é determinar a fronteira conceitual e, em seguida, o escopo do projeto de software, ou seja, seus objetivos e restrições. Para a construção de alguns sistemas, os desenvolvedores partem de definições vagas sobre o sistema (também chamadas de declaração dos requisitos do usuário) e devem gastar tempo colhendo e especificando informações sobre problemas do usuário para poder definir os objetivos e restrições do projeto de desenvolvimento do sistema de software. Esse tipo de situação ocorre, em geral, em aplicações comerciais. Uma boa parte do esforço de desenvolvimento é gasta determinando-se a natureza do problema e suas soluções. Outro tipo de desenvolvimento de sistema é o que lida com declaração de requisitos que inclui um método preciso, necessário para resolver o problema. Esse tipo de declaração é comum nas áreas matemática, científica e técnica. É comum, também, quando o usuário tem experiência em desenvolvimento de software ou quando sua experiência técnica o leva a dar detalhes da solução. A ênfase da avaliação da declaração, nesse caso, deve estar na verificação de se o método indicado pelo usuário é o melhor para resolver o problema.

Para se definir objetivos e restrições, são considerados os requisitos que o usuário apresenta de maneira explícita e empregadas as técnicas de extração de requisitos para obter informações mais precisas. Essa atividade auxiliará nas estimativas de esforço (custo) e tempo do projeto, que é outra atividade do planejamento. Dependendo do desempenho exigido para uma determinada função, o esforço estimado pode ser sensivelmente alterado, o que influenciará diretamente no custo e no prazo do projeto. Portanto, deve-se determinar o escopo do projeto, ou seja, o que será feito e quais critérios determinam o sucesso do projeto.

A declaração do escopo do problema e a sua análise são vitais para o sucesso do desenvolvimento do projeto. Isso evita que o desenvolvimento resolva um problema diferente daquele que o usuário tem. O escopo serve de guia para as estimativas do projeto e permite que se verifique se o produto que está sendo produzido resolverá o problema como um todo ou, se não, em que grau o fará. A definição do escopo também formaliza e testa o nível de entendimento entre desenvolvedores e usuários.

4.1.1 Os requisitos do usuário

Freqüentemente o usuário descreve o problema em termos vagos, como, por exemplo, "Eu desejo automatizar os serviços de escritório", ou então "Eu desejo um sistema que calcule o imposto a pagar". Nesses dois casos, o usuário não deu nenhuma pista sobre o escopo do problema, suas restrições e outras informações indispensáveis para o início do desenvolvimento do sistema. Técnicas de extração de requisitos, como as apresentadas no capítulo 2, devem ser empregadas para a obtenção dessas informações. Além disso, os requisitos do usuário devem ser analisados de maneira cuidadosa para que sejam determinadas suas funções (o que é para o sistema fazer) e suas restrições (as propriedades que restringem o desenvolvimento do sistema), de forma que o plano de desenvolvimento tenha uma base sólida.

Os clientes descrevem suas necessidades, e o desenvolvedor deve considerar essa descrição como ponto de partida para a definição dos objetivos e restrições e, baseando-se nessas informações, traçar o plano de projeto. Além disso, pode ser utilizada uma estratégia para a extração de requisitos, visando à melhor compreensão do problema e à constatação das dificuldades previstas para o projeto. Sob o ponto de vista do planejamento, podem-se considerar os requisitos do usuário avaliando-se, entre outras coisas, a complexidade do sistema e o seu custo. O cliente deve compreender e aprovar os objetivos e restrições definidos para o projeto, de forma que um contrato entre ele e o desenvolvedor possa ser estabelecido.

4.1.2 Declaração de objetivos e restrições do projeto

É o primeiro produto no ciclo de vida do desenvolvimento de software. A declaração de objetivos e restrições do projeto (DORP) pode ser utilizada como um contrato entre o cliente e o desenvolvedor. Esse documento deve organizar informações obtidas durante os primeiros contatos com o cliente e seu conteúdo deve ser resumido. Para isso, são avaliados a função e o desempenho do software e elaborado o documento que, entre outras informações, pode conter:

- a definição dos objetivos do projeto, na qual são descritas e avaliadas as funcionalidades do sistema a ser desenvolvido. Em alguns casos, as funções devem ser refinadas para melhorar a precisão das estimativas de custo e prazo;
- as restrições e/ou delimitações, que identificam os limites impostos ao software pelo hardware, tais como memória disponível, outros sistemas existentes ou ainda limite de recursos;
- os critérios de seleção, que fornecem um guia para a seleção de alternativas de solução para o problema. Exemplos de critérios de seleção são facilidade de acesso e disponibilidade de apoio ao treinamento;
- as interfaces com outros sistemas com os quais o software a ser desenvolvido vai interagir (hardware, software, dispositivos de entrada/saída etc.), pois isso implicará algum esforço e influenciará o custo e o prazo do projeto;

– o desempenho esperado, considerando-se fatores tais como requisitos de processamento e tempo de resposta. Podem-se considerar, como exemplos de medida de desempenho, o número de usuários simultâneos, a quantidade de clientes e o tempo máximo de resposta necessário;

– a confiabilidade, que é um requisito difícil de se especificar nessa fase inicial do projeto. A natureza do projeto pode determinar considerações que garantam a confiabilidade necessária; por exemplo, o impacto de uma falha num sistema que monitora pacientes é muito maior do que o de uma falha num sistema que controla estoque.

Além disso, pode ser definido um nome para o sistema a ser desenvolvido e enumeradas as idéias preliminares para resolver o problema.

A seguir, é apresentado um exemplo de um *sistema de venda de sementes*; a partir da declaração de requisitos e de entrevistas com o usuário, foi elaborada uma declaração de objetivos e restrições do sistema a ser construído.

> *Sementes & Companhia é uma empresa que comercializa sementes para jardinagem. Ela tem um sistema manual de manufatura de mercadorias que atendia plenamente a demanda; no entanto, a expansão de mercado acabou apontando falhas tanto no setor de vendas como no setor de produção.*

Empregando-se a técnica de entrevistas, obtiveram-se as seguintes informações:

– Sobre a empresa:

Nunca teve um sistema computadorizado; tem forte vínculo com métodos de trabalho manual utilizados; necessitará longo tempo de treinamento e adaptação; tem desconfiança em relação às mudanças.

– Sobre os funcionários:

Vários tipos de usuários; total inexperiência no uso de computadores.

– Problemas com as funcionalidades do sistema:

Falta de eficiência para lidar com produtos manufaturados no estoque.

Erros no controle de estoque de matérias-primas. A elaboração de uma receita (manufatura do produto) depende do estoque de várias matérias-primas que a compõem. A ausência de estoque de qualquer uma delas fará com que o pedido fique pendente até que a matéria-prima seja comprada.

Atraso na atualização de informações sobre produção.

Erros e dificuldade na obtenção de informações, gerando descontentamento de clientes.

Excesso de burocracia na manipulação e encaminhamento de pedidos, gerando atrasos, descontentamento de clientes e sobrecarga de trabalho para os funcionários.

Problemas com maus pagadores por não se ter disponíveis informações de crédito do cliente.

Considerando-se esses dados, foram elaborados questionários e realizadas outras entrevistas com os usuários do sistema, obtendo-se dados que resultaram na seguinte *declaração de objetivos e restrições do projeto*:

Projeto: Sistema de venda de sementes.

Problemas do sistema atual: São muitos os problemas que tornaram o sistema manual ineficiente. Entre eles podem-se citar: (a) a atualização dos registros de estoque não acompanha o ritmo da produção, ocasionando problemas no setor de manufatura e de vendas por telefone; (b) a produção não acompanha o ritmo das vendas; (c) o tempo consumido no processamento de pedidos é tão grande que o pessoal do armazém precisa exceder a jornada de trabalho.

Objetivos do projeto: O objetivo principal do sistema a ser desenvolvido é agilizar a comunicação entre os departamentos da empresa, viabilizando seu sistema de compras, produção e comercialização. Assim sendo, o sistema deve:

- controlar pedidos: cadastrar pedido (verificar crédito do cliente, consultar preço do produto e efetivar pedido); controlar pedidos da fila de espera (pedidos que não puderam ser atendidos imediatamente); e gerar ordens de serviço;
- controlar estoque de matéria-prima: atualizar e conferir estoque (permitir um controle rígido do emprego da matéria-prima); gerar ordem de compra (para matérias-primas que atingiram estoque mínimo); e gerar relatórios de controle de estoque;
- controlar estoque de produtos manufaturados: atualizar e conferir estoque de produtos manufaturados; gerar ordem de produção; e gerar relatório de controle de estoque;
- controlar contas a receber;
- agilizar produção;
- fazer composição de custos: calcular preço do produto de acordo com o preço das matérias-primas que o compõem;
- emitir fatura.

Restrições do projeto

Os pedidos recebidos até o meio-dia devem estar prontos para entrega até o início do expediente do dia seguinte.

Os relatórios de controle de estoque devem ser diários.

A composição de custos de um produto deve ser refeita toda vez que houver alteração no preço de alguma das matérias-primas que o compõem.

As faturas devem ser arquivadas para uso futuro.

Critérios de aceitação

Facilidade de utilização, pois os usuários têm total inexperiência no uso de computador.

Facilidade de manutenção, pelo mesmo motivo do item anterior.

Idéias preliminares

Colocar código do produto no próprio produto para auxiliar sua localização no depósito e reduzir enganos de estocagem.

Produzir "guias de busca" de matérias-primas para agilizar o processo de manufatura dos produtos.

Criar uma interface semelhante à atual para minimizar o problema da inexperiência dos usuários no uso de computadores.

Informatizar o sistema de vendas e de produção.

4.2 Estudo de viabilidade

Parte do estudo de viabilidade é fazer a análise dos requisitos do sistema para definir várias alternativas de solução para o projeto do sistema. Essas alternativas devem ser investigadas mais apuradamente, de forma que possam ser ordenadas por preferência em uma recomendação que é o resultado do estudo de viabilidade. Para se verificar a viabilidade de cada alternativa, usualmente é necessário refiná-las. A análise dos requisitos tenta determinar características de soluções aceitáveis, ferramentas, bem como facilidades e pessoas disponíveis para desenvolver a solução. Requisitos freqüentemente são conflitantes ou não são economicamente viáveis considerando-se os recursos disponíveis. Muitos requisitos podem e devem ser negociados. Deve-se iniciar esse estudo indicando soluções alternativas para o problema. Uma lista de alternativas deve ser considerada e, num processo de refinamento, deve-se ir excluindo aquelas que, pela ordem, não são viáveis tecnicamente, operacionalmente e economicamente. As alternativas que sobrarem serão consideradas viáveis.

– Viabilidade técnica.

  Esse estudo deve gerar um conjunto de alternativas tecnicamente viáveis através do exame de implicações técnicas dos requisitos e da verificação da viabilidade de cada requisito e de possíveis conflitos técnicos com outros requisitos. Por exemplo, se o desenvolvimento for interno, só poderão ser consideradas alternativas que necessitem apenas do conhecimento técnico da equipe.

– Viabilidade operacional ou organizacional.

  São considerações a respeito da rejeição do usuário a alguma alternativa tecnicamente viável. Por exemplo, o cliente poderia se opor ao desenvolvimento do sistema por terceiros, ou à compra de outro equipamento, exigindo a utilização dos recursos existentes na empresa.

– Viabilidade econômica.

  Nesse ponto, deve-se estimar o custo tanto operacional como de desenvolvimento de cada alternativa restante. Calculam-se também economias de custo e/ou aumentos de receita em comparação com o sistema existente (se houver). Podem-se selecionar, a partir desse estudo, alternativas viáveis sob todos os aspectos já considerados.

– Pode-se considerar, também, um estudo da viabilidade legal da alternativa.

A quantidade de tempo que se deve gastar com o estudo de viabilidade depende do projeto. Por exemplo, é bastante razoável gastar um dia para avaliar as alternativas para modificar o formato de um relatório. Já para desenvolver um novo sistema de contabilidade, será necessário gastar semanas. O desenvolvimento de um novo pacote que custará milhões de dólares com certeza pede um estudo mais apurado das possíveis soluções e, portanto, pode levar meses ou até anos.

4.2.1 Lista de alternativas

Inicialmente, o grau de funcionalidade do sistema deve ser examinado. Nesse caso, podem-se utilizar, como técnicas para gerar alternativas, o delineamento de diferentes fronteiras de automação e a técnica de *brainstorming*, vista no capítulo 2. Os aspectos funcionais de cada alternativa devem ser verificados e pontuados pela complexidade de implementação. Se duas funcionalidades têm a mesma prioridade de implementação e a mesma prioridade de negociação, a mais simples (menos complexa) é a melhor.

Um conjunto de alternativas será gerado a partir da aplicação das técnicas acima. Deve-se verificar, então, a viabilidade técnica de cada alternativa, rejeitando-se as que não forem viáveis. Por exemplo, se um sistema exigir tempo de resposta de três a quatro segundos, as alternativas voltadas a lote (*batch*) devem ser ignoradas. As alternativas tecnicamente viáveis podem ser apresentadas ao usuário para verificar se ele rejeita alguma delas (viabilidade operacional). A partir desse ponto, deve-se passar para o estudo da viabilidade econômica.

Para o *sistema de venda de sementes*, podem-se considerar as seguintes alternativas viáveis para sua solução, muito embora várias outras pudessem ser geradas, bastando, por exemplo, modificar as fronteiras de automação.

– *Alternativa 1*: manter o funcionamento atual da fábrica, aumentando o número de funcionários e o tamanho das instalações, para atender a demanda crescente.

– *Alternativa 2*: desenvolver um sistema informatizado para agilizar o funcionamento do sistema atual, mantendo seu modo de operação. Esse sistema terá as seguintes funções:

  F1: controlar estoque de matéria-prima e de produtos manufaturados;

  F2: controlar pedidos;

  F3: controlar contas a receber;

  F4: produzir guia de busca;

  F5: compor os custos dos produtos manufaturados;

  F6: emitir fatura.

– *Alternativa 3*: instalar um sistema integrado, totalmente informatizado, envolvendo os subsistemas de produção, controle de estoque e vendas. Esse sistema conterá todas as funções da alternativa 2 e mais a função controlar produção das máquinas. As ordens de serviço irão diretamente para as máquinas disponíveis.

Para cada uma das alternativas técnica e operacionalmente viáveis, devem-se fazer estimativas de custo (em geral em termos de mão-de-obra e tempo de desenvolvimento), benefício e recursos e determinar os riscos, preparando um estudo de *custo–benefício* para cada alternativa. Dessa forma, será possível quantificar a viabilidade econômica e justificar o fato de se descartar algumas das alternativas. As alternativas que sobrarem, por serem viáveis, devem ser apresentadas ao cliente, incluindo-se considerações sobre vantagens e desvantagens de cada uma. Baseando-se nas vantagens e desvantagens das alternativas viáveis, deve-se apresentar também uma *recomendação* da melhor solução para o problema. Para a alternativa recomendada, pode-se acrescentar um estudo de custo–benefício detalhado.

4.2.2 Estimativas

O processo de estimativa para projetos de software está diretamente relacionado com a decomposição do produto e do processo de desenvolvimento (princípios discutidos no capítulo 2). No início do desenvolvimento, têm-se poucos detalhes e, portanto, só é possível fazer uma estimativa grosseira; mas, à medida que o desenvolvimento progride e mais detalhes se tornam conhecidos, as estimativas se tornam mais precisas. Isso deve ser bem compreendido para garantir uma melhor qualidade das estimativas de esforço, tempo e recursos contidas no planejamento do projeto de software. As estimativas iniciais em geral são baseadas em experiências de desenvolvimentos anteriores, quando foram registrados as estimativas feitas em cada fase e o valor real obtido. Nas fases que se sucedem durante a construção do software, os valores estimados podem ser registrados, de forma que, ao se terminar o projeto, seja possível verificar se o processo de estimativa permitiu que os resultados intermediários das estimativas convergissem para o valor real conforme o projeto foi avançando.

O processo de estimativa faz uso não somente de perícia e conhecimento, mas também de um conjunto de regras que podem ser encontradas em métodos de estimativas. Para obter bons resultados, um método de estimativa deve:

– ter a primeira estimativa entre ±30% do valor real (estimativa feita nos primeiros estágios de desenvolvimento);

– ter definida uma faixa de valores (erro-padrão de estimativa) que garanta que em pelo menos 68% das vezes o valor estimado esteja compreendido nessa faixa;

– permitir refinamentos da estimativa durante o desenvolvimento do sistema (reestimar ao final de cada fase, incluindo novas informações);

– ser fácil de utilizar;

– ter ferramentas de suporte e documentação.

Os modelos usados para estimar são bastante imprecisos; entretanto, essas distorções podem e devem ser compensadas de alguma forma, pois, do contrário, o método produzirá resultados impróprios no planejamento de recursos, implicando, em geral, valores subestimados, o que gera problemas tais como ultrapassagem de custos, descumprimento de prazos e produtos de baixa qualidade. As variações normais nas estimativas podem ser contornadas utilizando-se fatores múltiplos que influenciem o tamanho e a complexidade do sistema e utilizando-se alguma maneira independente para verificar os resultados. A imprecisão nas estimativas feitas nas fases iniciais do projeto é grande, pois pouco se sabe sobre o produto a ser desenvolvido. Conforme o projeto avança, mais se passa a conhecer sobre o produto, e o grau de imprecisão tende a diminuir.

De acordo com DeMarco [DEM 82], os desenvolvedores de software reconhecidamente não são bons estimadores por não saberem exatamente o que é estimativa, não fazerem previsões adequadas para contrabalançar o efeito de distorções, não saberem lidar com os problemas políticos que dificultam o processo de estimativa e não basearem as estimativas em desempenhos passados. Um projeto normal de software requer estimativas no início do projeto e, periodicamente, daí por diante. Isso consome pouco tempo do gerente (3% do tempo gasto no projeto), o que explica, em parte, a sua falta de especialização no assunto.

4.2.3 Estimativa de custo

A estimativa de custo faz parte da fase de planejamento de qualquer projeto de engenharia. A diferença é que na engenharia de software o custo principal é o esforço, ou seja, o custo de mão-de-obra; assim, para se calcular o custo do software, é necessário dimensionar o trabalho para desenvolvê-lo. Em geral, esse trabalho é expresso pelo número de pessoas trabalhando numa unidade de tempo, tal como pessoas-mês ou pessoas-ano. O trabalho que uma pessoa consegue fazer num mês pode ser traduzido em número de horas de trabalho num mês. Boehm [BOE 81] sugere que se utilize o valor de 152 horas por mês (máximo de horas produtivas num mês) como parâmetro para o cálculo do esforço. Assim, se para um projeto forem estimadas 40 pessoas-mês, tem-se um trabalho equivalente a 6.080 horas (152 x 40), que deverão ser distribuídas no prazo estimado para o projeto. Pode-se afirmar que o esforço está diretamente relacionado à produtividade, que é medida pela quantidade de trabalho

realizada por uma unidade de esforço. Assim, um exemplo de medida de produtividade é a quantidade de linhas de código/pessoas-mês.

Existem vários métodos que podem ser utilizados para se estimar o custo do desenvolvimento e a vida útil de um sistema. Em geral, o custo pode representar o custo monetário ou o esforço necessário para desenvolver e manter o sistema. Para estabelecer essas estimativas, pode-se decompor o produto e o processo e, então, utilizar a opinião de especialistas que, baseados em experiências de projetos anteriores, são capazes de estimar o esforço e o tempo de desenvolvimento para o novo projeto. Outra maneira de se estabelecer uma estimativa de custo é utilizar modelos empíricos (como os discutidos no capítulo 3), que poderão ou não necessitar da decomposição do produto e do processo. Apesar de os modelos empíricos serem bastante utilizados (usualmente fazem parte das ferramentas automatizadas para previsão de custos), eles não são diretamente transportáveis para qualquer empresa. Assim, os modelos empíricos podem ser utilizados como base inicial em uma empresa até que ela possa criar seu próprio modelo.

Sendo assim, quando se deseja estabelecer estimativas para um software novo, em geral é necessário utilizar *técnicas de decomposição*. Nesse caso, o processo de estimativa começa com uma declaração definida do escopo do software, e a partir dela tenta-se decompor o software em pequenas subfunções que podem ser estimadas individualmente. Podem-se considerar dois tipos de decomposição: (1) decomposição do produto para estimar o número de linhas de código, utilizando-se a métrica *LOC* (*lines of code*), ou estimando a funcionalidade através da métrica *FP* (*function points*); e (2) decomposição do processo considerando-se as atividades de cada etapa da engenharia de software, dependendo do paradigma utilizado.

LOC se refere à estimativa do número de linhas de código executáveis de um software, e FP se refere aos pontos de função que são determinados estimando-se o número de entradas, saídas, arquivos de dados, consultas e interfaces externas, bem como valores de ajuste de complexidade. A partir da decomposição do produto utilizando-se qualquer métrica (LOC, FP etc.), pode-se estimar um valor otimista (o), um valor mais provável (m) e um valor pessimista (p) para cada subfunção e então utilizar métodos de estimativa existentes para calcular o valor previsto para a variável a ser estimada [LON 87]. Entre os modelos disponíveis, pode-se citar o modelo PERT, que na sua forma mais simplificada utiliza o valor otimista e o valor pessimista para o cálculo da variável estimada, como segue:

$$Ve_i = \frac{o_i + p_i}{2}.$$

Essa estimativa deverá ser corrigida, por um erro-padrão de estimativa, para garantir que em pelo menos 68% das vezes o valor real esteja dentro do esperado (Ve_i) corrigido pelo erro.

Uma forma mais sofisticada do modelo PERT considera os valores otimista, pessimista e mais provável dos componentes (subfunções) do sistema, e o valor esperado pode ser calculado por uma média ponderada, como segue:

$$Ve_i = \frac{o_i + 4m_i + p_i}{6}$$

em que Ve_i é a variável de estimativa e o_i, m_i e p_i são os valores otimista, mais provável e pessimista dos componentes do sistema. Nesse caso, a margem de erro também deve ser indicada. A estimativa global, incluindo todos os n componentes de um sistema, será:

$$Ve = \sum_{i=1}^{n} (Ve_i).$$

A Tabela 4.1 lista o valor otimista, o mais provável e o pessimista para o tamanho em LOC das funções que compoẽm a alternativa 2 do projeto para o *sistema de venda de sementes*. Esses valores podem ter sido obtidos a partir de um banco de dados sobre projetos, ou através da opinião de especialistas, e foram utilizados para o cálculo do valor esperado para o tamanho das funções do sistema aplicando-se a técnica PERT, que calcula a média ponderada.

Funções	**Otimista**	**Mais provável**	**Pessimista**	**Esperado**
F1	1.400	2.000	2.800	2.033
F2	3.500	5.400	6.300	5.233
F3	1.500	2.300	3.700	2.400
F4	5.000	6.300	7.500	6.283
F5	1.000	1.500	1.900	1.483
F6	2.300	2.880	3.175	2.833

Tabela 4.1: Tamanho estimado para as funções da alternativa 2.

Considerando-se o tamanho estimado para cada um dos subsistemas ou funções de alternativas de solução para o projeto, pode-se utilizar um modelo de custo e estimar o esforço e tempo necessários para o desenvolvimento do projeto. Por exemplo, o tamanho total estimado para a alternativa 2 é S = 20.265 (20 KLOC), e, se o modelo básico de Boehm, visto no capítulo 3, for utilizado, o esforço total estimado será E = 56 pessoas-mês. Essa é apenas uma estimativa grosseira do custo do projeto, visto que apenas o tamanho do sistema foi considerado.

As estimativas podem ser refinadas estimando-se o esforço das funções em cada etapa da engenharia de software. Para isso, é necessário considerar o paradigma a ser utilizado para o desenvolvimento do sistema. Boehm sugere uma distribuição do esforço total entre as fases do ciclo de vida do desenvolvimento, apresentada na Tabela 4.2. Deve-se observar que o custo de manutenção é adicional, não considerado no custo calculado por Boehm.

A porcentagem de esforço depende, entre outros fatores, do tipo de sistema e do ambiente de desenvolvimento. Para a alternativa 2 do *sistema de venda de sementes*, pode-se considerar que a Tabela 4.3 representa a distribuição esperada do esforço nas diversas fases do desenvolvimento. Para se calcular o custo monetário da alternativa, o total de cada fase pode ser multiplicado pela taxa relativa ao custo em pessoas-mês (pm) para a fase, calculando-se o custo monetário de cada fase. Essa estimativa permite programar a utilização dos recursos financeiros. Por exemplo, a fase de planejamento/análise dos requisitos requer quatro

Etapa	% de esforço
Planejamento/análise dos requisitos	6 a 8
Projeto	16 a 18
Implementação	48 a 68
Teste/integração	16 a 34

Tabela 4.2: Proporção de esforço gasto nas fases de desenvolvimento.

pessoas-mês. Se o custo por pessoa-mês nessa fase é de $ 230,00, o custo total dessa fase é de $ 920,00. Calculando-se o custo de cada uma das outras fases, chega-se ao custo total da alternativa.

Etapa	Esforço estimado
Planejamento/análise dos requisitos	3 pm
Projeto	9 pm
Implementação	29 pm
Teste/integração	15 pm

Tabela 4.3: Distribuição do esforço para a alternativa 2.

4.2.4 Estimativa de tempo

Após estimar o esforço necessário para construir o produto, pode-se utilizar esse valor para estimar o tempo, ou seja, o prazo para o desenvolvimento do projeto. Evidentemente, o tempo de desenvolvimento depende da produtividade da equipe. Quanto maior a produtividade, menor é o tempo de desenvolvimento. Se o modelo de Boehm for utilizado para estimar o tempo de desenvolvimento da alternativa 2, tem-se que $T_d = 10$ meses. Se for necessário estimar o tempo que será gasto em cada fase do desenvolvimento, pode-se utilizar a distribuição de tempo sugerida por Boehm, apresentada na Tabela 4.4. Da mesma maneira que a Tabela 4.2 pode ser utilizada para estimar o esforço nas diferentes fases, a Tabela 4.4 pode ser utilizada para estimar a quantidade de tempo nas diferentes fases. Essa estimativa é muito útil na programação do projeto.

Etapa	% de tempo
Planejamento/análise dos requisitos	10 a 40
Projeto	19 a 38
Implementação	19 a 38
Teste/integração	18 a 34

Tabela 4.4: Proporção de tempo gasto nas fases de desenvolvimento.

4.2.5 Estimativa de recursos

Devem ser estimados os recursos necessários para o desenvolvimento do projeto e para a operação do produto de software. Os principais recursos são: humano, de software e de hardware.

As estimativas de recursos humanos devem fazer uma previsão das habilidades exigidas, disponibilidade, duração das tarefas e data em que esse recurso deve estar disponível. O número de pessoas só pode ser determinado após a estimativa de esforço ter sido efetuada. Para projetos pequenos (uma pessoa-ano ou menos), uma única pessoa pode executar todas as etapas da engenharia de software, consultando especialistas quando necessário. Para projetos grandes, deve-se organizar uma equipe. O número de pessoas na equipe pode ser estimado dividindo-se o esforço total pelo tempo de desenvolvimento. No exemplo do *sistema de venda de sementes*, pode-se prever que o número de pessoas na equipe é igual ao esforço dividido pelo tempo de desenvolvimento, ou seja, 56 pessoas-mês/10 meses, o que resulta em uma equipe de seis pessoas.

As estimativas de hardware e software devem prever a descrição, disponibilidade, duração do uso e início da utilização desse tipo de recurso. Recursos de hardware a ser estimados durante o planejamento podem ser relativos ao hardware utilizado durante o desenvolvimento ou ao hardware em que o produto será instalado, ou ainda a outros equipamentos necessários para o desenvolvimento e operação do sistema de software. Os recursos de software a ser estimados são aqueles necessários ao desenvolvimento e gerenciamento do projeto de software. Deve-se considerar um conjunto de ferramentas que auxiliem nas diversas tarefas de engenharia de software. Essas ferramentas são denominadas ferramentas CASE (*computer-aided software engineering* — engenharia de software auxiliada por computador) e auxiliam em várias atividades da engenharia de software, entre elas: desenvolvimento, planejamento, gerenciamento de projetos, programação, integração e testes e construção de protótipos. Outro tipo de recurso de software é aquele necessário para a operação do sistema. Nessa classe estão os softwares básicos e aplicativos.

A proporção dos recursos gastos em cada etapa do ciclo de vida depende da natureza do projeto. Se um sistema envolve grande e complexo arquivo de dados, os recursos devem se concentrar no projeto; as etapas de implementação e testes exigirão menos recursos. Se o sistema for utilizar um arquivo existente para a programação de produção, as fases de implementação e o treinamento de operadores vão requerer mais recursos. Em sistemas de apoio à decisão, se o usuário não souber direito quais são as informações necessárias, como serão usadas e com que freqüência, o estudo de viabilidade e a especificação de requisitos do sistema exigirão mais recursos.

4.2.6 Estimativa de benefícios

Os benefícios podem ser tangíveis ou intangíveis. Benefícios tangíveis, tais como redução de custo operacional do sistema e/ou aumento de lucro, são quantificáveis. Um cliente pode esperar aumentar o lucro porque mais trabalho será feito na mesma quantidade de tempo. Um estudo de custo–benefício detalhado indicará o lucro. Os benefícios tangíveis são:

aumento de receita, redução de custo, aperfeiçoamento de serviços ao cliente e atendimento de exigências inflexíveis. Benefícios intangíveis não são facilmente quantificáveis. Exemplos desse tipo de benefício são: moral da equipe, melhoria do processo de tomada de decisão, melhoria na documentação e facilidade de uso do sistema. É adequado pedir auxílio ao usuário, pois ele, melhor do que ninguém, conhece os benefícios possíveis, mas nem sempre é possível estimar todos os benefícios, já que muitos deles são baseados em fatos futuros que, não ocorrendo, modificam a estimativa feita. Os benefícios considerados em partes do sistema podem afetar outras partes, e nem sempre é possível estimar quais partes e quanto serão afetadas. Além disso, os benefícios baseados em novas tecnologias não podem ser previstos. Voltando ao exemplo do *sistema de venda de sementes*, podem-se considerar as seguintes estimativas para as alternativas de solução já apresentadas:

- *Alternativa 1*: a rotina de trabalho preestabelecida não será alterada; contudo, o aumento do número de funcionários tornará desnecessário o pagamento de horas-extras e permitirá que as operações de atualização de estoque não se atrasem em relação ao ritmo de trabalho das unidades produtivas. O custo de investimento para implantação dessa alternativa será de $ 10.400,00, referentes a reformas e ampliação das dependências da empresa. O custo operacional previsto será de $ 23.560,00 anuais, gastos com salários e encargos de três novos funcionários. A economia esperada será de $ 12.070,00 ao ano, correspondentes a economias hoje pagas em horas extras a funcionários.

- *Alternativa 2*: o custo de investimento previsto para essa alternativa é de $ 22.930,00, referentes ao desenvolvimento de software. O custo operacional será de $ 20.420,00 ao ano, necessários à manutenção do hardware, pagamento de funcionário para manutenção de software e outros gastos. A economia anual esperada é de $ 30.880,00.

- *Alternativa 3*: para essa alternativa, estima-se um custo de $ 52.900,00, que correspondem a gastos com aquisição de hardware e desenvolvimento de software. O custo operacional esperado é o mesmo da alternativa anterior, ou seja, $ 20.420,00 ao ano, e o benefício estimado é de $ 39.500,00 ao ano.

4.2.7 Análise de risco

A análise de risco do desenvolvimento de um sistema de software deve identificar as partes que apresentam as maiores dificuldades no desenvolvimento e apontar os riscos e as ações que devem ser tomadas para contornar as causas desses riscos (isto é, devem-se estabelecer os passos a ser seguidos para que se possa administrar os riscos detectados). Devem-se, também, estabelecer mecanismos para avaliar o progresso do desenvolvimento e a organização do pessoal que construirá o produto. Fatores que podem provocar o encerramento do projeto devem ser tratados antes do início do desenvolvimento.

Algumas considerações que podem ser feitas sobre os riscos no contexto da engenharia de software são: quais riscos podem fazer com que o projeto do software fracasse? Como

a mudança nos requisitos do cliente, nos computadores a que se destina o software, nas tecnologias de desenvolvimento e nas entidades ligadas ao projeto afetará o sucesso global e o cumprimento do cronograma? Quais métodos e ferramentas devem ser usados para o desenvolvimento do sistema? Quantas pessoas devem ser envolvidas? Quanta ênfase deve ser dada à qualidade?

Os riscos são medidos pelo *grau de incerteza das estimativas* estabelecidas para custo, prazo e recursos. Se o escopo de um projeto for mal estabelecido ou se os requisitos do sistema estiverem sujeitos a mudanças, a incerteza das estimativas torna-se elevada, aumentando, assim, os riscos de fracasso do projeto. Segundo Boehm [BOE 91], a determinação dos riscos envolve as seguintes atividades: *identificação*, *análise* e *priorização*.

- *Identificação*: essa atividade tem como objetivo produzir uma lista de fatores de risco específicos que podem comprometer o sucesso do projeto. Os fatores de risco podem estar relacionados ao produto (por exemplo, complexidade do produto), ao processo de desenvolvimento (por exemplo, ambigüidade na especificação) e ao negócio (por exemplo, construir um produto para o qual não existe mercado).

- *Análise*: essa atividade tem como objetivo determinar a probabilidade da ocorrência de cada fator de risco e o impacto caso o risco ocorra. Para tal, deve-se quantificar a probabilidade percebida de ocorrência de um risco. Quanto ao impacto do risco, três fatores podem ser considerados: a natureza, o alcance e o tempo de ocorrência do risco.

- *Priorização*: essa atividade tem como objetivo ordenar os fatores de risco identificados e analisados. Os riscos devem ser, então, ponderados em função do possível impacto percebido sobre o projeto e depois colocados em ordem de probabilidade de ocorrência.

4.2.8 Análise de custo–benefício

Todo investimento tem um risco, e o desenvolvimento de um sistema é um investimento. Quanto maior o risco, maior deverá ser o benefício. Tanto para benefícios tangíveis como para intangíveis, devem-se indicar prioridades para a sua obtenção, como, por exemplo, obrigatório, importante mas negociável, ou opcional. Essas prioridades facilitam a análise comparativa de alternativas e a negociação de requisitos, quando necessária.

Quando são comparados os benefícios econômicos com os custos do projeto, pode-se ver o benefício econômico como lucro. Só há duas maneiras de aumentar o lucro: aumentando a receita e/ou diminuindo o custo. Dessa forma, o benefício econômico pode ser calculado em termos de receita e custos, como segue:

$$lucro = receita - custo$$

em que:

lucro	→	benefício econômico;
receita	→	dinheiro que flui para a organização;
custo	→	dinheiro que flui da organização.

Análise de custo

O custo do software varia com sua funcionalidade e características de qualidade. Um cliente pode desejar um sistema com muitas características mas acabar optando por um sistema simples, devido à rapidez de sua construção ou pelo custo em proporção aos benefícios. Deve-se especificar o valor a ser gasto para o desenvolvimento e utilização do sistema novo. Assim, o custo para a construção do novo sistema pode ser dividido em investimento (custo fixo) e custo operacional (custo variável, que depende da utilização que será feita do software).

Os custos são baseados nas estimativas de esforço e/ou tempo para as atividades e na quantidade e qualidade de recursos. Isso inclui recursos para: desenvolvimento, conversão do sistema antigo (manual ou automatizado) no novo e operação do sistema novo.

Os *custos de desenvolvimento* ocorrem apenas uma vez e são considerados investimento de capital. Os itens abaixo podem ser considerados os que mais afetam o custo de desenvolvimento.

- Pessoal: analistas, programadores, operadores, pessoal administrativo etc.
- Equipamentos: tempo de máquina, espaço em disco, instrumentos e equipamentos novos.
- Software: ferramentas CASE.
- Materiais: discos, fitas, publicações, papéis etc.
- Despesas gerais: apoio administrativo, espaço, luz etc.
- Despesas externas: consultoria, treinamento especial etc.

Os *custos operacionais* iniciam-se com a instalação e continuam durante a vida útil do sistema; são considerados despesas. Podem-se levar em conta os seguintes itens para estimar os custos operacionais:

- Custo de hardware: tempo de residência, espaço de memória, operações de E/S.
- Custo de pessoal: operador, administrador, programador (manutenção).
- Materiais: formulários, discos.
- Despesas gerais: aluguéis, auditoria, serviços externos.

Outros custos também devem ser estimados; entre eles podem-se citar:

- Custo de instalação: inclui custos das atividades associadas com a integração do software ao complexo de facilidades, equipamentos, pessoal e procedimentos do sistema operacional do usuário.
- Custo de treinamento: inclui treinamento do usuário, do pessoal de operação, do pessoal de preparação de dados e do pessoal da manutenção.

– Custo de conversão: quando se estuda a possibilidade de uma nova configuração do computador, o custo da conversão pode ser um item crítico, que pode incluir conversão de programas, conversão de base de dados, documentação, teste de validação e aceitação.

– Custo de documentação: a quantidade de documentação produzida para um sistema pode ser associada ao tamanho do produto. Deve-se estimar o esforço necessário para escrever, revisar e especificar a documentação.

Análise de benefício

É importante calcular a proporção dos benefícios e o custo do projeto. Os benefícios são medidos avaliando-se a melhoria nos negócios dos clientes, a remoção de um problema, assim como a exploração de uma oportunidade. A análise de benefício compara pares de requisitos do produto e benefício para verificar se eles são consistentes e realistas. Essa atividade pode usar informações quantitativas sobre as alternativas para ordená-las e negociá-las, se necessário. Cada requisito de uma alternativa deve refletir benefícios e, se isso não ocorrer, deve-se considerar a possibilidade de o requisito ser supérfluo ou de os benefícios correspondentes não terem sido apropriadamente determinados.

Retorno do investimento

Cada parte do projeto rende seus próprios benefícios, acarreta seus próprios custos e também exige recursos próprios; dessa forma, a enumeração de custos, benefícios e recursos dos componentes do projeto ajuda a decidir quais partes do projeto devem ser realizadas, em que ordem devem ser implantadas e quais devem ser canceladas ou adiadas no caso de faltarem recursos. Além disso, facilita o processo de estimativa total de custo e benefício do projeto.

Sendo o dinheiro aplicado no desenvolvimento de um sistema considerado um investimento, deve-se prever em quanto tempo o cliente recuperará o dinheiro aplicado. O custo estimado para o sistema pode ser considerado alto ou não, dependendo do valor do benefício esperado e do tempo para o retorno do investimento. Deve-se dar preferência a investimentos em que o retorno é mais rápido e maior no início.

O valor estimado do benefício deve ser projetado para o futuro, de forma que se estime quando o benefício acumulado cobrirá o investimento feito no desenvolvimento do projeto. Pode-se utilizar a seguinte fórmula para o cálculo do valor futuro do dinheiro:

$$F = P(1+i)^n$$

em que:

F → valor futuro do dinheiro;
P → valor presente do dinheiro;
i → taxa de juros;
n → número do período da aplicação.

Um investimento em desenvolvimento tem chance de ser um bom negócio quando o retorno se dá em aproximadamente três anos e se o tempo de vida do sistema for longo o suficiente para que haja tempo de se recuperar o investimento e ter lucro. Entretanto, se o valor presente líquido for igual a zero, será melhor aplicar o dinheiro num investimento de menor risco.

Considerando-se o custo–benefício e o risco de cada alternativa para o projeto do *sistema de venda de sementes*, pode-se fazer uma análise das alternativas, como segue:

- *Alternativa 1*: a principal vantagem nesse caso é que não haverá necessidade de treinar pessoal para novos procedimentos. O investimento necessário à implantação dessa alternativa é pequeno. Em contrapartida, essa alternativa tem como fator de risco o fato de o problema inicial voltar a ocorrer caso a expansão de mercado continue. Além disso, não há previsão de recuperação do investimento.

- *Alternativa 2*: a vantagem dessa alternativa é que, com a automatização das funções básicas (que foram incluídas nessa alternativa), os sistemas de produção e de controle de estoque serão agilizados, resolvendo, assim, os principais problemas do cliente. Além disso, a expansão dos negócios da empresa não afetará essa solução. O investimento não é alto, e estima-se que o retorno se dê por volta de três anos. Podem-se apontar como desvantagens, nesse caso, os riscos inerentes ao desenvolvimento e os custos com aquisição de equipamentos e com treinamento.

- *Alternativa 3*: a comunicação direta da linha de montagem com os sistemas de controle (estoque e finanças) trará como vantagem bastante agilidade no processo produtivo e de vendas da empresa. Podem-se citar como fatores de risco nessa alternativa o custo e a modificação radical nos processos de praticamente todos os setores da empresa. O investimento é 2,3 vezes maior que o da alternativa 2, e o benefício apenas 1,3 vez maior.

Recomendações: a segunda alternativa irá suprir as necessidades operacionais da empresa sem, no entanto, modificar muito radicalmente sua estrutura de funcionamento. Essa alternativa permite que os negócios da empresa possam se expandir sem que haja necessidade de reinvestimento no sistema. Além disso, o retorno do investimento se dará após 2,5 anos (bastante razoável), e, durante a sua vida útil (cinco anos), o sistema dará um lucro de $ 37.783,00 – $ 22.930,00 = $ 13.473,00. Sendo assim, recomenda-se essa alternativa para solucionar os problemas da empresa. Detalhes do estudo de custo–benefício realizado são apresentados a seguir.

Estudo de custo–benefício detalhado para alternativa 2

Custo do desenvolvimento

Pessoal (56 pm)	$ 16.300,00
Equipamentos	$ 5.800,00
Materiais	$ 500,00
Despesas gerais	$ 330,00
	$ 22.930,00

Custo operacional do sistema novo (anual)

Hardware (manutenção)	$ 3.600,00
Mão-de-obra	$ 14.420,00
Materiais	$ 1.560,00
Despesas gerais	$ 840,00
	$ 20.420,00

Receita adicional esperada com a implantação do sistema
$ 2.573,33 por mês = $ 30.880,00 por ano

Economia anual de custo
Benefício = $ 30.880,00 – $ 20.420,00 = $ 10.460,00 por ano.

Retorno do investimento

A Tabela 4.5 mostra o resultado do estudo de retorno de investimento para a alternativa 2, considerando o benefício (valor futuro) igual para todos os anos e a taxa de juros a 12% ao ano. O benefício estimado hoje será desvalorizado; assim, o valor de $ 10.460,00 por ano estimado corresponderá a um valor de $ 7.071,00 após três anos. O retorno do investimento é esperado para a metade do segundo ano. A rigor, o retorno do investimento deve ser determinado por uma análise mais detalhada, considerando-se outras possibilidades de investimento.

Ano	Benefício	Taxa	Valor pres.	Valor pres. acum.
1	$ 10.460,00	1,12	$ 9.339,00	$ 9.339,00
2	$ 10.460,00	1,25	$ 8.368,00	$ 17.707,00
3	$ 10.460,00	1,40	$ 7.471,00	$ 25.178,00
4	$ 10.460,00	1,57	$ 6.662,00	$ 31.840,00
5	$ 10.460,00	1,76	$ 5.943,00	$ 37.783,00

Tabela 4.5: Retorno do investimento para a alternativa 2.

4.3 Organização do projeto

Após completar as estimativas para o projeto e obter a aceitação do cliente, para poder iniciar o desenvolvimento do sistema, deve-se decompor cada fase do desenvolvimento em atividades (a ser realizadas por apenas uma pessoa), selecionar e organizar as pessoas que farão parte da equipe e, finalmente, atribuir as atividades às pessoas da equipe. A organização do projeto exige que se estime a duração das atividades, assim como a definição das que devem ser realizadas em seqüência e das que podem ser realizadas paralelamente. Além disso, devem-se conhecer as habilidades necessárias para realizar cada atividade.

4.3.1 As atividades do desenvolvimento

As informações que os gerentes necessitam para julgar o progresso do desenvolvimento do sistema e para atualizar estimativas de custo e cronograma do projeto somente podem ser obtidas através de documentos que descrevem o trabalho que está sendo feito. Quando a especificação do sistema é feita em estágios, como os discutidos no capítulo 3, esses documentos são constituídos pelas especificações produzidas em cada estágio.

Segundo Somerville [SOM 96], quando o projeto é planejado, uma série de marcos deve ser estabelecida, e cada marco é o ponto final de alguma atividade do processo de engenharia de software. Deve-se, então, escolher um paradigma ou alguma outra estrutura para o desenvolvimento do sistema, desde que cada tarefa a ser executada seja definida, estimada, documentada e transmitida de etapa para etapa do desenvolvimento de software, isto é, deve-se determinar uma divisão do trabalho de desenvolvimento de software em partes gerenciáveis. Essa divisão nem sempre é simples, pois o produto do desenvolvimento de software não é um objeto físico, nenhum material o constitui ou envolve, como é o caso de outros processos de engenharia. Muitos paradigmas, com diferentes divisões de trabalho, têm sido propostos sob a denominação de ciclo de vida do software, como apresentado na seção 1.4. A escolha do paradigma de desenvolvimento tem um forte impacto no planejamento. As estimativas de custos, recursos e tempo podem ser baseadas na decomposição do processo em atividades ou tarefas, e, nesse caso, o paradigma escolhido determinará o primeiro nível de decomposição e influenciará a escolha das atividades para produção dos resultados desejados.

4.3.2 Organização da equipe

A organização de pessoas em equipes para desenvolvimento de projetos deve levar em conta a diferença de personalidade dos vários membros da equipe e como o papel da pessoa na equipe afeta sua personalidade. Deve-se ter a preocupação de recrutar as pessoas certas, sejam de dentro ou de fora da organização. Pessoas da organização que se tornaram disponíveis em outros projetos ou que estão alocadas, mas com previsão de disponibilidade, poderão ser recrutadas. Nesse caso, tanto o trabalho como a personalidade da pessoa já são conhecidos, e é mais fácil prever se ela é adequada ao projeto. O recrutamento de pessoas de

fora da organização apresenta mais riscos e gastos, pois será necessário selecionar e treinar uma pessoa até que ela se torne produtiva. Boehm [BOE 81] considera diferentes opções para aplicação de recursos humanos em um projeto que exija n pessoas trabalhando durante k anos, entre elas:

- n indivíduos são designados para m tarefas funcionais diferentes. Ocorre pouco trabalho combinado. A coordenação cabe a um gerente de software que pode ter outros projetos com que se preocupar;
- n indivíduos são designados para m tarefas funcionais diferentes, com $m < n$, de forma que equipes informais sejam estabelecidas. Um chefe de equipe pode ser designado, e a coordenação das equipes fica sob a responsabilidade de um gerente;
- n diferentes indivíduos são organizados em t equipes. Cada equipe tem uma organização específica e tem a ela atribuídas uma ou mais tarefas funcionais. A coordenação é controlada tanto pela equipe quanto pelo gerente. Estudos indicam que esse tipo de organização é o mais produtivo.

A composição da equipe pode ter como núcleo desenvolvedores com experiência (sênior), pessoal técnico e engenheiros substitutos. Além disso, deve-se especificar o pessoal auxiliar, como especialistas, pessoal de apoio e bibliotecário.

4.3.3 Programação de projeto

A programação de projeto tem como objetivo organizar as atividades de desenvolvimento em uma seqüência coerente. O objetivo dessa organização é equilibrar recursos de pessoal, hardware e software, de maneira que sejam usados da melhor forma possível. Depois de estimados os recursos necessários para o projeto de software, o cronograma do projeto especifica como e quando esses recursos devem estar disponíveis. Além disso, deve-se especificar, também, quem será responsável pelas atividades do ciclo de vida do sistema. Segundo DeMarco [DEM 82], a determinação de um cronograma para o projeto de software pode ser vista a partir de duas perspectivas:

1) Uma data final para a entrega do sistema já foi estabelecida de forma irrevogável. Nesse caso, o esforço deverá ser distribuído dentro desse espaço de tempo.
2) Limites cronológicos aproximados são discutidos, mas a data final para a entrega é estabelecida pela equipe de engenharia de software. O esforço é distribuído para se tirar o melhor proveito dos recursos, e uma data final é definida após cuidadosa análise.

A fixação de prazos para projetos tenta responder a uma série de perguntas. Por exemplo: como se relaciona o tempo cronológico com o esforço humano? Quais tarefas e paralelismos devem ser esperados? Quais marcos de referência podem ser usados para mostrar o progresso? Como o esforço é distribuído ao longo do processo de engenharia de software?

Existem métodos disponíveis para determinação de prazos? Como representar fisicamente o cronograma e como rastrear o progresso?

O cronograma do projeto envolve a divisão do trabalho total do projeto em atividades separadas, com estimativas do tempo necessário para completar cada uma dessas atividades. Usualmente, algumas dessas atividades são realizadas paralelamente. O cronograma do projeto deve coordenar essas atividades paralelas e organizar o trabalho de forma que o esforço seja aplicado adequadamente. Deve-se considerar que todo estágio de desenvolvimento está sujeito a problemas, que trabalhadores individuais podem falhar ou se afastar do projeto, que o hardware pode quebrar e que o suporte de software e hardware pode não estar disponível quando necessário.

O cronograma do projeto é usualmente representado como um conjunto de diagramas mostrando a divisão do trabalho, a dependência entre atividades e a alocação da equipe. Ele pode ser gerado automaticamente, a partir do banco de dados do projeto, utilizando-se uma ferramenta automatizada para gerenciamento de projetos. Informações como as listadas na Tabela 4.6 serão necessárias para a elaboração do cronograma. Essa tabela se refere às atividades envolvidas na implementação e teste do *sistema de venda de sementes* e apresenta uma estimativa de tempo para cada atividade (tarefa) e a precedência desejada entre elas.

Tarefa	**Descrição**	**Semanas**	**É precedida por**
A	1-3 Criar telas	3	nenhuma
B	3-4 Implementar cadastrar cliente	5	A e G
C	3-6 Implementar estoque matéria-prima	4	A e G
D	3-7 Implementar estoque produto	5	A e G
E	4-8 Implementar verificar crédito	3	B
F	1-2 Criar banco de dados	5	nenhuma
G	2-3 Inicializar base de dados teste	1	F
H	7-11 Implementar guia de busca	9	D
I	6-10 Implementar comprar matéria-prima	5	C
J	8-12 Implementar controlar pedidos	4	E
L	3-5 Implementar controlar contas a receber	3	A e G
M	5-9 Implementar compor custos de produto	5	L
N	9-12 Implementar emitir fatura	3	M
O	12-13 Integrar subsistema vendas	4	J e N
P	10-13 Integrar subsistema estoque	3	I
Q	11-13 Integrar subsistema produção	3	H
R	13-14 Realizar o teste do sistema	4	O, P e Q

Tabela 4.6: Tarefas para implementação, integração e teste do sistema.

O diagrama PERT/CPM, visto no capítulo 3, pode ser utilizado para representar o cronograma referente aos dados da Tabela 4.6. A Figura 4.1 representa todas as informações da tabela, além das folgas, tempo mais cedo e tempo mais tarde de cada atividade. Pelo

diagrama é possível saber quais são as atividades do caminho crítico (F, G, D, H, Q, R). Essas atividades têm folga igual a zero; assim, elas não podem ser atrasadas para que o projeto seja realizado em 27 semanas.

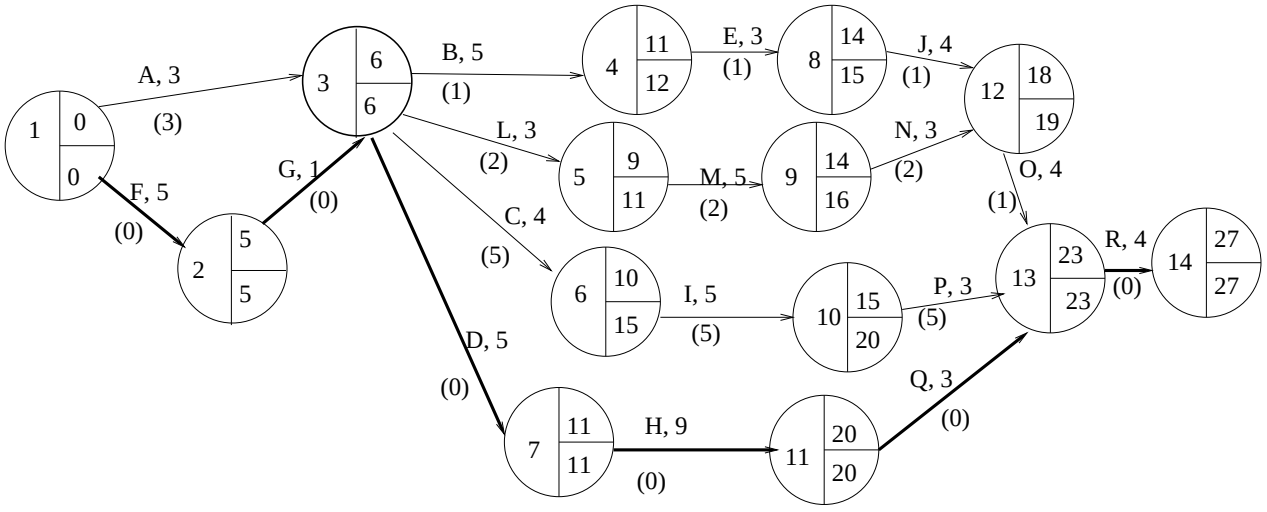

Figura 4.1: PERT/CPM para o sistema de venda de sementes.

Como já foi visto no capítulo 3, outra maneira de expor o relacionamento entre recursos e tarefas é o *diagrama de barras* ou *gráfico de Gantt*. Para cada atividade, esse diagrama indica a data prevista de início e de término e a pessoa responsável pela atividade. O diagrama de barras pode especificar a duração de cada tarefa (atividade), o que será feito e o executor de cada tarefa. Pode ser usado para controle do projeto marcando-se o tempo estimado e o tempo gasto em cada tarefa. Pode ser utilizado, também, para registrar o acompanhamento do projeto, acrescentando-se ao diagrama um outro tipo de barra, que representa as datas de início e término reais da atividade. A Figura 4.2 apresenta o diagrama de barras do *sistema de venda de sementes*, correspondente aos dados da Tabela 4.6. Pode-se observar que da sexta até a décima oitava semana deve-se alocar a maior parte dos recursos para que o projeto termine no prazo. Se não houver recursos suficientes para cobrir esse cronograma, por exemplo, se forem alocados programadores em número insuficiente, será necessário recalcular o tempo.

4.4 Comentários finais

Grande parte do trabalho envolvido no planejamento de sistemas de software depende da obtenção de dados quantitativos sobre o produto e o processo; dessa forma, o sucesso do projeto de construção do produto de software depende da obtenção e registro correto dos requisitos do sistema, além da manutenção de um banco de dados com informações sobre projetos passados. O estabelecimento do plano do projeto é um processo iterativo que deve convergir para valores próximos dos reais quando o projeto se aproxima da data de entrega.

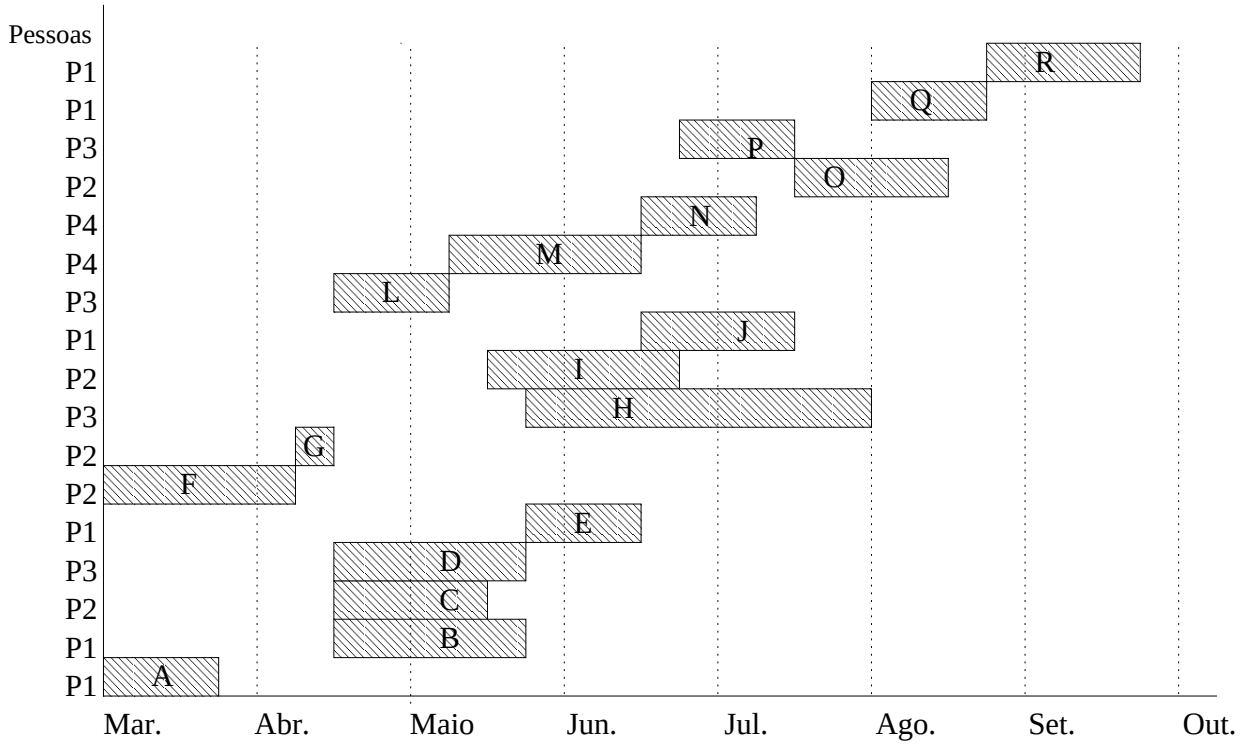

Figura 4.2: Diagrama de barras para o sistema de venda de sementes.

Quando se utilizam modelos para planejamento de projetos, fica evidente que seria extremamente útil que esses modelos fossem construídos utilizando-se ferramentas computadorizadas. Entre outros motivos, pode-se considerar o fato de que o planejamento é um processo iterativo e, conseqüentemente, requer revisões constantes e reorganização do cronograma. Os gráficos devem ser refeitos periodicamente e, além disso, todos os modelos utilizados no desenvolvimento do sistema devem poder se relacionar.

4.5 Exercícios

1) Uma empresa de desenvolvimento de software foi contratada para construir um software para um sistema de segurança residencial. Liste as atividades que ela deverá realizar para elaborar o plano inicial do projeto.

2) Uma indústria farmacêutica está planejando e especificando os requisitos de um sistema em tempo real para controle de um processo químico. A tabela a seguir mostra as atividades para as fases de planejamento e extração de requisitos, com a duração expressa em semanas.

Ativ.	Descrição	Duração
1-2	Levantamento das necessidades do usuário	3
1-3	Análise funcional	5
1-4	Estudo de um modelo	2
2-3	Especificação dos requisitos do usuário	8
2-5	Análise de requisitos	6
3-6	Desenvolvimento do modelo funcional	5
3-7	Desenvolvimento do modelo de dados	5
4-3	Desenvolvimento do modelo de comportamento	1
5-8	Elaboração do plano de desenvolvimento	5
5-9	Revisão dos requisitos	3
6-10	Validação do modelo funcional	2
7-10	Validação do modelo de dados	1
8-11	Validação do modelo de comportamento	3
9-11	Verificação dos requisitos	3
10-11	Verificação do plano de desenvolvimento	1

(a) Desenhe o diagrama PERT determinando os tempos mais cedo e mais tarde, assim como o caminho crítico.

(b) O que aconteceria se o tempo de duração da atividade "desenvolvimento do modelo funcional (3-6)" fosse reduzido para quatro semanas? E para três semanas? E para duas semanas?

3) Desenvolva um diagrama de barras para o problema da questão anterior.

4) Considerando as estimativas de tamanho para a alternativa 2 do *sistema de venda de sementes*, apresentadas na Tabela 4.1, o custo de $ 0,80 por linha de código e a produtividade média de 360 LOC/pm, resolva as seguintes questões:

(a) Para qual função o estimador apresentou maior grau de incerteza na estimativa de tamanho?

(b) Estime o esforço (pm) necessário para cada função e o esforço total para o desenvolvimento do sistema.

(c) Estime o custo monetário para o desenvolvimento do sistema.

(d) O tamanho esperado de cada função da Tabela 4.1 foi calculado utilizando-se média ponderada. Se a técnica PERT em sua forma mais simplificada fosse utilizada, qual seria o custo do sistema?

5) Usando a tabela de estimativa de esforço, a seguir, para cada fase de desenvolvimento e considerando a taxa de mão-de-obra dada por pessoas-mês e o esforço dado por pessoas-mês, resolva as questões que se seguem.

(a) Em que fase se estima que o esforço será maior?

(b) Compare o resultado obtido com o da questão anterior. As estimativas convergem para o mesmo resultado? Pode-se concluir que as estimativas estejam corretas?

Funções	**Análise (pm)**	**Projeto (pm)**	**Implem. (pm)**	**Teste (pm)**
F1	0,5	1,0	2,4	2,1
F2	1,0	2,5	5,5	5,0
F3	0,5	1,3	2,7	2,6
F4	1,2	3,0	6,8	6,0
F5	0,3	0,9	2,0	1,8
F6	0,5	1,3	2,6	2,5
Taxa ($)	400,00	360,00	250,00	280,00

6) Suponha que uma empresa de software esteja avaliando a possibilidade de desenvolver um software que vai concorrer com alguns outros similares disponíveis no mercado. Sabe-se que 12.000, 17.500 e 21.300 são, respectivamente, as estimativas otimista, mais provável e pessimista para o número de linhas do sistema. Sabe-se também que o número médio de erros nesse tipo de sistema é de um erro a cada 1.250 linhas de código (após a entrega do sistema). Considerando que na empresa o custo por linha de código é de $ 1,70 e que a produtividade é de 73 LOC/pm:

(a) Estime o custo, o esforço e o número de erros esperados.

(b) Estime o número de pacotes/mês e o preço unitário para venda do produto, de forma que o investimento seja recuperado em três anos. Sabe-se que o preço de seus concorrentes é, em média, $ 71,00.

7) Considere as funções para o *sistema de venda de sementes* como listadas no exercício 5. Considere, também, que sejam propostas as seguintes alternativas de solução para o problema:

- Solução 1: implementar apenas as funções F1, F2, F4 e F6.
- Solução 2: implementar apenas as funções F1, F2, F3, F4 e F6.
- Solução 3: implementar todas as funções.

(a) Calcule o custo do projeto para cada solução.

(b) Considerando o número máximo de seis pessoas trabalhando ao mesmo tempo no projeto, estime o tempo necessário para entrega do produto para cada solução.

(c) Se o recurso for de $ 13.000,00, qual será a melhor solução?

(d) Qual será a melhor solução se os recursos forem irrestritos mas o tempo não puder ultrapassar um mês?

8) Uma empresa de exportação deve decidir entre desenvolver ou comprar um software para controle de seu sistema de remessas. Sabe-se que existem pessoas disponíveis para realizar o trabalho, que projetos desenvolvidos na empresa têm custo médio de $ 1,60/LOC e que o custo de um pacote é de $ 2.500,00.

   (a) Que estimativas a empresa deveria considerar antes de decidir entre desenvolver ou comprar?

   (b) Em quais casos seria melhor comprar e em quais seria melhor desenvolver?

9) Sabendo-se que o investimento num sistema é de $ 14.200,00 e que a economia mensal está estimada em $ 560,00 com sua implantação, resolva:

   (a) Calcule o tempo (em anos) necessário para o retorno do investimento.

   (b) Esse sistema é economicamente viável? Por quê?

   (c) Considerando que seja estimada uma vida útil de quatro anos para esse sistema, ele geraria algum lucro?

Bibliografia

[ALA 98] Alagar, V. S. e Periyasamy, K. *Specification of software systems*. Springer, 1998.

[AUG 91] August, J. H. *Joint Application Design: the group session approach to systems Design*. Englewood Cliffs, N. J.: Prentice-Hall, 1991.

[BOE 81] Boehm, B. W. *Software engineering economics*. Prentice-Hall, 1981.

[BOE 91] _____. *Software risk management: principles and pratices.* Ieee Transactions Software Engineering, vol. 18, nº 1, jan., 1991, pp. 32-41.

[BOO 86] Booch, G. *Object-oriented development.* Ieee Transactions Software Engineering, vol. SE-12, nº 2, fev., 1986.

[BRJ99] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language user guide*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1999.

[BOU 83] Bourne, S. R. *The UNIX system*. International Computer Science Series, 1983.

[CHI 90] Chikofsky, E. J. e Cross III, J. H. *Reverse engineering and design recovery: a taxonomy*. Ieee Software, 1990, pp. 13-7.

[CHR 92] Christel, G. M. e Kang, K. C. *Issues in requirements elicitation*. Technical Report CMU/SEI-92-TR-12, ESC-TR-92012, set., 1992.

[COA 90] Coad, P. e Yourdon, E. *Object-oriented analysis*. Prentice-Hall, 1990.

[Coc00] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[COL 94] Coleman, D. et al. *OO developing: the fusion method*. Prentice-Hall, 1994.

[DEM 79] DeMarco, T. *Structured analysis and system specification*. Prentice-Hall, 1979.

[DEM 82] _____. *Controlling software projects*. Prentice-Hall, 1982.

[ELM 94] Elmasri, R. e Navathe, S. *Fundamentals of database systems*, 2ª ed. Benjamin/Cummings, 1994.

[FFO 00] Farines, J.; Fraga, J. S. e Oliveira, R. S. *Sistemas de tempo real*. São Paulo: Escola de Computação, jul., 2000.

[FER 86] Ferreira, A. B. H. *Novo dicionário da língua portuguesa*. Nova Fronteira, 1986.

[FS97] Martin Fowler and Kendall Scott. *UML distilled: applying the standard object modeling language*. Addison-Wesley Longman Ltd., Essex, UK, UK, 1997.

[GAN 82] Gane, T. e Sarson, C. *Structured systems analysis*. McDonnell Douglas, 1982.

[GEH 86] Gehani, N. e McGettrick, A. D. *Software specification techniques*. Addison-Wesley Publishing Company, 1986.

[GEO 95] George, C. et al. *The raise development method.* Prentice-Hall, 1995.

[GUE 91] Guezzi, C. e Jazayeri, M. *Fundamentals of software engineering.* Prentice-Hall, 1991.

[GUT 85] Guttag, J. V.; Horning, J. J. e Wing, J. M. *An overview of the larch family of specification languages*. Ieee Software, 2(5), set., 1985, pp. 24-36.

[INC 89] Ince, D. C. *Software engineering*. Chapman and Hall, 1989.

[JAC 83] Jackson, M. A. *System development*. Prentice-Hall, 1983.

[JBR99] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The unified software development process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[JAl 97] Jalote, P. *An integrated approach to software engineering*, 2ª ed. Springer-Verlag, 1997.

[JON 90] Jones, C. B. *Systematic software development using VDM*, 2ª ed. Prentice-Hall, 1990.

[KEN 88] Kendall, K. e Kendall, J. E. *Systems analysis and design*. Prentice-Hall, 1988.

[Lar97] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*. Prentice Hall, 1997.

[LON 87] Londeix, B. *Cost estimation for software development.* Addison-Wesley, 1987.

[MAF 92] Maffeo, B. *Engenharia de software e especificação de sistemas*. Campus, 1992.

[MCM 84] McMenamin, S. e Palmer, J. *Essential systems analysis*. Prentice-Hall, 1984.

[Mey91] Bertrand Meyer. *Design by contract. In Advances in Object-Oriented Software Engineering*. Prentice Hall, Englewood Cliffs, N.J., 1991.

[MOO 90] Moore, A. P. *The specification and verified decomposition of system requirements using CSP*. Ieee Transactions Software Engineering, 16(9), set., 1990, pp. 932-48.

[NAU 69] Naur, P. e Randell, B. (eds.). *Software engineering: a report on a conference sponsored by the NATO Science Committee*. Nato, 1969.

[PAG 80] Page-Jones, M. *The practical guide to structured systems design*. Yourdon Press, 1980.

[PAN 97] Pankaj, J. *An integrated approach to software engineering*, 2ª ed. Springer, 1997.

[PRE 92] Pressman, R. *Software engineering: a practitioner's approach*, 3ª ed. Mc-Graw Hill, 1992.

[PUT 78] Putnam, L. *A general empirical solution to the macro software sizing and estimating problem*. Ieee Transactions Software Engineering, vol. 4, nº 4, 1978, pp. 345-61.

[RAG 94] Ragham, S.; Zelesnik, G. e Ford, G. *Lecture notes on requirements elicitation*. Pittsburgh, Pensilvânia: Software Engineering Institute–Carnegie Mellon University, 1994.

[RUM 91] Rumbaugh, J. et al. *Object-oriented modeling and design*. Prentice-Hall, 1991.

[SW01] Geri Schneider and Jason P. Winters. *Applying use cases (2nd ed.): a practical guide*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[SNE 95] Sneed, H. M. *Planning the reengineering of legacy systems*. Ieee Software, jan., 1995.

[SOM 96] Somerville, I. *Software engineering*. Addison-Wesley, 1996.

[TOM 89] Tomayko, J. e Hallman, H. K. *Software project management*. SEI, 1989.

[WAR 85] Ward, P. T. e Mellor, S. J. *Structured development for real-time systems*. Yourdon Press, 1985.

[WAR 81] Warnier, J. D. *Logical construction of systems*. Van Nostrand Reinhold, 1981.

[WET 84] Wetherbe, J. *Systems analysis and design: traditional, structured and advanced concepts and techniques*. St. Paul, Minn.: West Publishing, 1984.

[WIE 97] Wiest, J. e Levy, F. *A management guide to PERT/CPM*, 2ª ed. Prentice-Hall, 1997.

[YOU 90] Yourdon, E. N. *Modern structured analysis*. Prentice-Hall, 1990.

[ZAV 96] Zave, P. e Jackson, M. "Four dark corners of requirements engineering", X Simpósio Brasileiro de Engenharia de Software. São Carlos-SP, 14-18 out., 1996.