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.

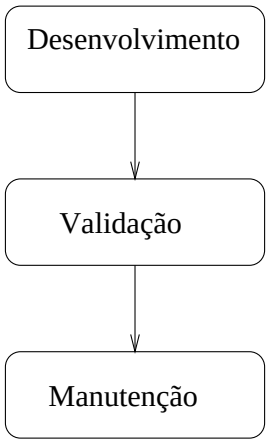

Figura 1.2: O processo de desenvolvimento de software.

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.

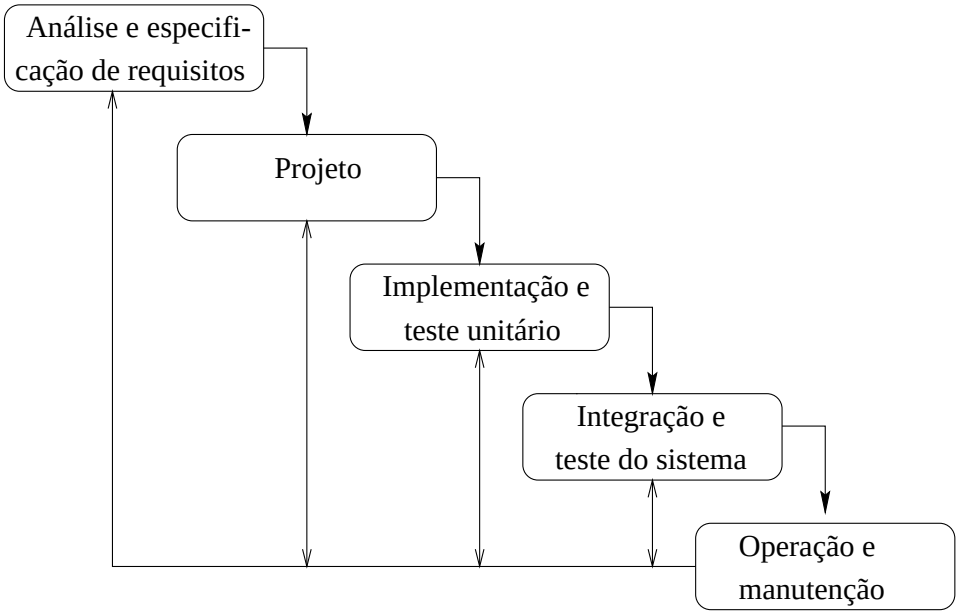

Figura 1.3: Os cinco passos do ciclo de vida clássico.

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.

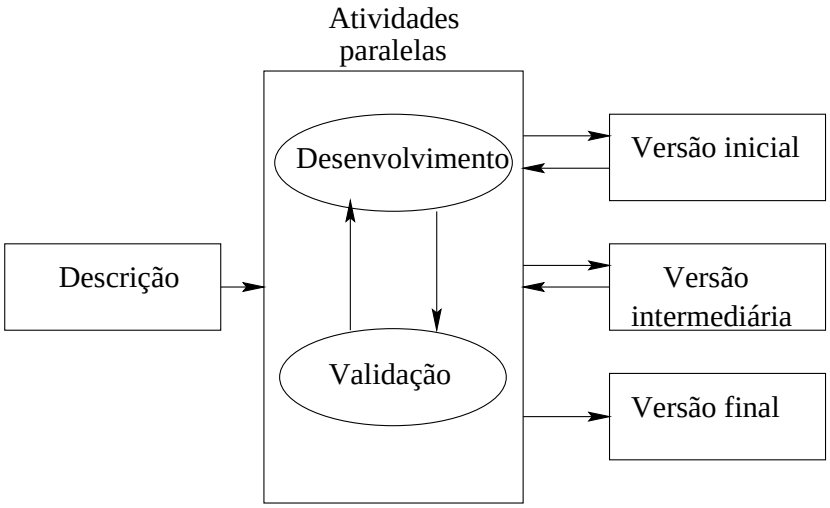

Figura 1.4: Desenvolvimento exploratório.

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.

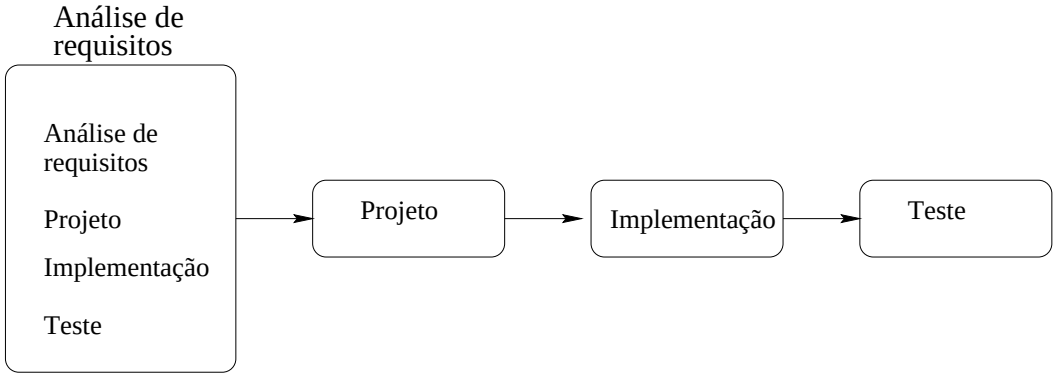

Figura 1.5: Protótipo descartável.

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.

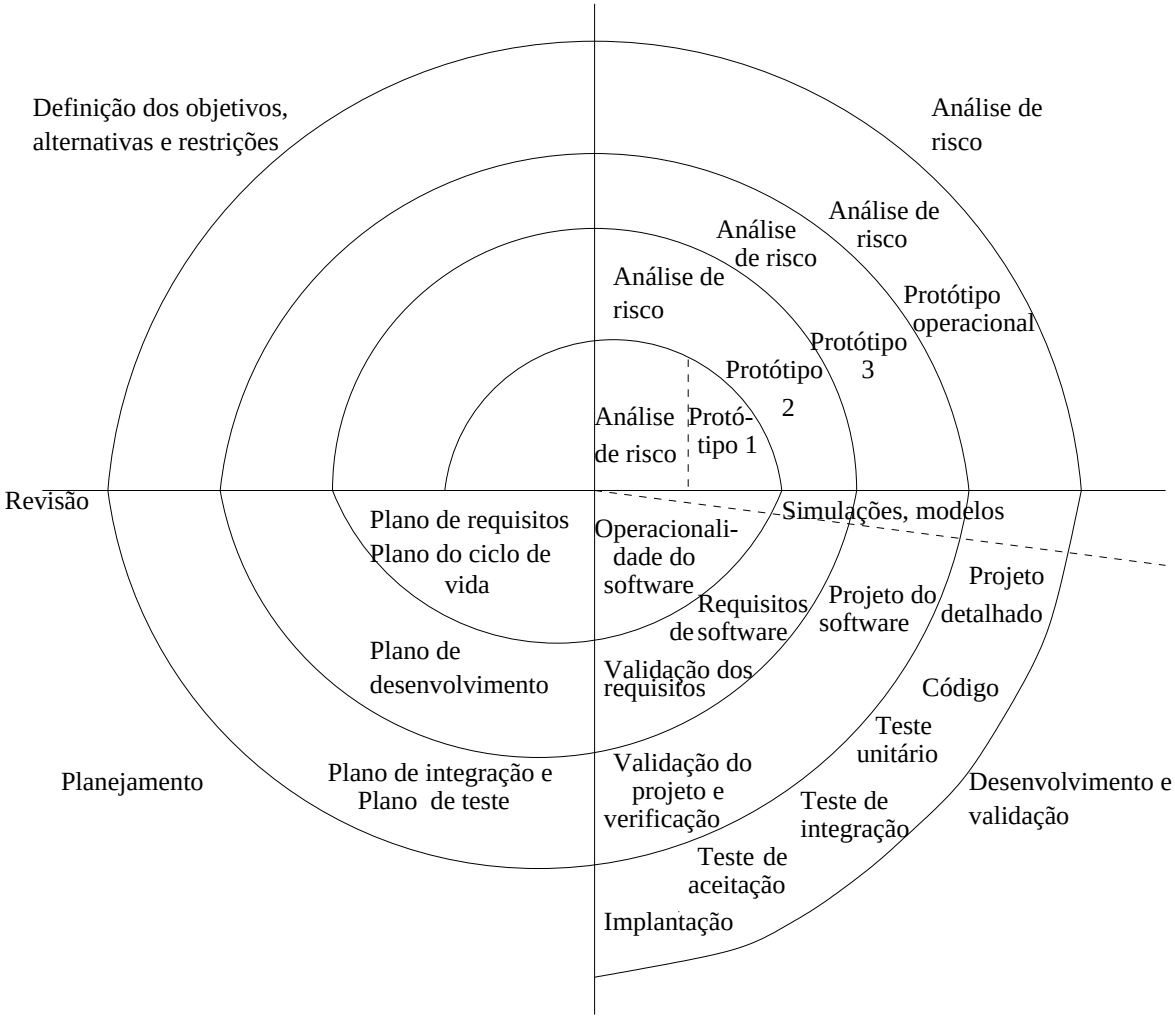

Figura 1.6: O paradigma espiral.

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;

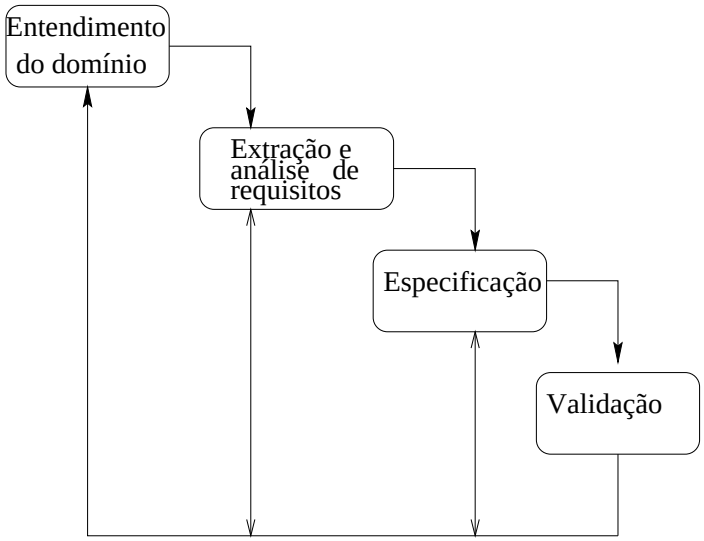

Figura 2.1: Processo de extração de requisitos.

- *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.

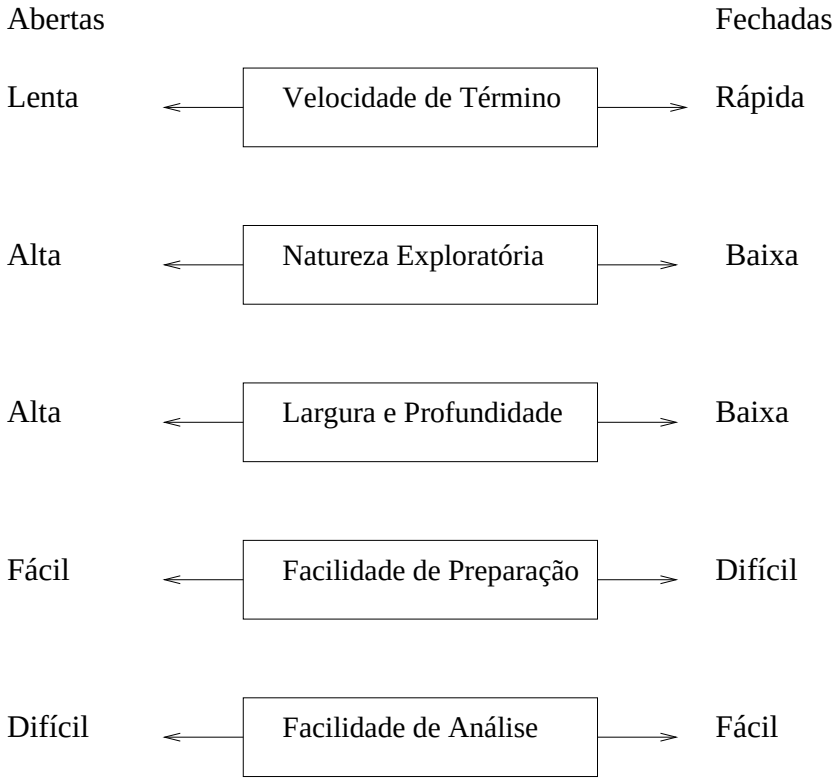

Figura 2.2: Prós e contras das questões abertas e fechadas.

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.

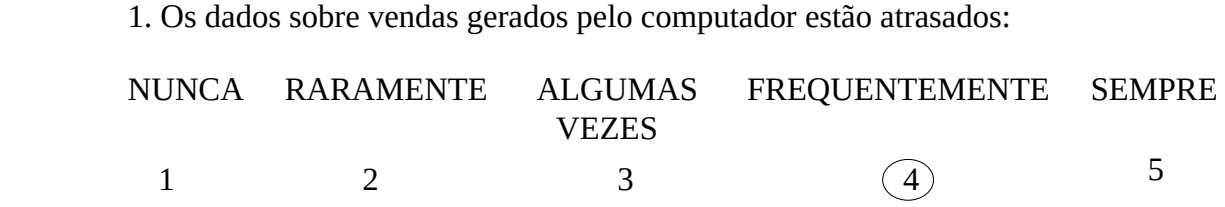

Figura 2.3: Uso de escala no questionário.

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.