

CAPÍTULO

11

CONCEITOS E PRINCÍPIOS DE ANÁLISE

CONCEITOS E PRINCÍPIOS DE ANÁLISE

CHAVE

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

273

PANORAMA

durante a engenharia de sistemas (Cap. 10). No entanto, é necessário olhar mais atentamente para o papel do software — a fim de entender os requisitos específicos que precisam ser satisfeitos para construir software de alta qualidade. Esse é o serviço da análise de requisitos de software. Para realizar o serviço adequadamente, você deve seguir um conjunto de conceitos e princípios subjacentes.

Quem faz? Geralmente, um engenheiro de software realiza a análise de requisitos. No entanto, para aplicações comerciais complexas, um "analista de sistemas" — treinado nos aspectos de negócio do domínio de aplicação — pode realizar a tarefa.

Por que é importante? Se você não analisa, é altamente provável que construa uma solução de software muito elegante que resolve o problema errado. O resultado é:

A engenharia de requisitos de software é um processo de descobrimento, refinamento, modelagem e especificação. Os requisitos do sistema e o papel atribuído ao software — estabelecidos inicialmente pelo engenheiro de sistemas — são refinados em detalhes. Modelos dos dados, informação e fluxo de controle, bem como comportamentos operacionais necessários, são criados. Soluções alternativas são analisadas e um modelo de análise completo é criado. Donald Reifer [REI94] descreve o processo de engenharia de requisitos de software do seguinte modo:

Engenharia de requisitos é o uso sistemático de princípios, técnicas, linguagens e ferramentas comprovadas para a análise, documentação, evolução continuada das necessidades do usuário e especificação do comportamento externo de um sistema para satisfazer as necessidades do usuário, que sejam efetivas em termos de custo. Note que como todas as disciplinas de engenharia, a engenharia de requisitos não é conduzida de um modo esporádico, aleatório ou sujeito a azares, mas, ao contrário, é o uso sistemático de abordagens comprovadas.

Tanto o engenheiro de software quanto o cliente assumem um papel ativo na engenharia de requisitos de software — conjunto de atividades que é frequentemente referido como *análise*. O cliente tenta reformular uma descrição, às vezes nebulosa, em nível de sistema dos dados, função e comportamento em detalhes concretos. O desenvolvedor age como inquisidor, consultor, solucionador de problemas e negociador.

perda de tempo e dinheiro, frustração pessoal e clientes insatisfeitos.

Quais são os passos? Requisitos de dados, funcionais e comportamentais são identificados pela elicitação da informação do cliente. Requisitos são refinados e analisados para avaliar sua clareza, completude e consistência. Uma especificação que incorpora um modelo do software é criada e depois validada tanto pelos engenheiros de software quanto pelos clientes/usuários.

Qual é o produto do trabalho? Uma representação efetiva do software deve ser produzida em consequência da análise de requisitos. Como os requisitos do sistema, os requisitos do software podem ser representados usando um protótipo, uma especificação ou até um modelo simbólico.

Como garanto que fiz corretamente? Os produtos de trabalho da análise de requisitos de software devem ser revistos quanto à clareza, completude e consistência.

Citação

"Esta sentença contradiz a si mesmo — não, na verdade ela não o faz."
Douglas Hofstadter

11.1 ANÁLISE DE REQUISITOS

Análise de requisitos é uma tarefa de engenharia de software que vence o espaço entre a engenharia de requisitos, no nível de sistemas, e o projeto de software (Fig. 11.1). As atividades de engenharia de requisitos resultam na especificação das características operacionais do software (função, dados e comportamento), indicam a interface do software com outros elementos do sistema e estabelecem restrições que o software deve satisfazer. A análise de requisitos permite ao engenheiro de software (algumas vezes chamado de *analista*, nesse papel) refinar a atribuição do software e construir modelos dos domínios de dados, funcional e comportamental que serão tratados pelo software. A análise de requisitos fornece ao projetista de software uma representação da informação, função e comportamento, que podem ser traduzidos para os projetos dos dados, arquitetura, interface e para o nível de componentes. Finalmente, a especificação de requisitos fornece ao desenvolvedor e ao cliente os meios de avaliar a qualidade quando o software é construído.

A análise de requisitos de software pode ser dividida em cinco áreas de esforço: (1) reconhecimento do problema, (2) avaliação e síntese, (3) modelagem, (4) especificação e (5) revisão. Inicialmente, o analista estuda a *Especificação do Sistema* (se existe alguma) e o *Plano do Projeto de Software*. É importante entender o software no contexto do sistema e revisar o escopo do software que foi usado para gerar as estimativas de planejamento. A seguir, a comunicação para a análise deve ser estabelecida de modo que seja garantido o reconhecimento do problema. A meta é reconhecer os elementos básicos do problema tal como são percebidos pelos clientes/usuários.

A avaliação do problema e síntese da solução é a próxima área importante de esforço da análise. O analista deve definir todos os objetos de dados observáveis externamente, avaliar o fluxo e o conteúdo de informação, definir e refinar todas as funções do software, entender o comportamento do software no contexto dos eventos que afetam o sistema, estabelecer as características das interfaces do sistema e descobrir restrições adicionais de projeto. Cada uma dessas tarefas serve para descrever o problema de modo que uma abordagem ou solução global possa ser sintetizada.

Por exemplo, um sistema de controle de estoque é necessário para um fornecedor importante de autopeças. O analista descobre que os problemas com o atual sistema manual incluem (1) incapacidade de obter rapidamente o *status* de um componente, (2) dois ou três dias de rotatividade para atualizar um arquivo em cartões, (3) pedidos múltiplos de ressurgimento para o mesmo fornecedor, porque não há como associar fornecedores com componentes, e assim por diante. Uma vez identificados os problemas, o analista determina que informação deve ser produzida pelo novo sistema e quais dados vão ser fornecidos ao sistema. Por exemplo, o cliente deseja um relatório

Fig. 11.1 Análise como ponte entre a engenharia de sistemas e o projeto de software.



diário indicando que peças foram retiradas do estoque e quantas peças similares restaram. O cliente indica que os funcionários do almoxarifado vão registrar o número de identificação de cada peça quando ela sai da área do estoque.

Após avaliar os problemas atuais e a informação desejada (entrada e saída), o analista começa a sintetizar uma ou mais soluções. Para começar, os objetos de dados, funções de processamento e comportamento do sistema são definidos em detalhe. Uma vez estabelecida essa informação, arquiteturas básicas para implementação são consideradas. Uma abordagem cliente/servidor parece que seria adequada, mas o software para apoiar essa arquitetura se encaixa no escopo esboçado no *Plano do Software*? Parece que seria necessário um sistema de gestão de bases de dados, mas a necessidade de associatividade do usuário/cliente é justificada? O processo de avaliação e síntese continua até que o analista e o cliente se sintam seguros de que o software pode ser especificado adequadamente para os passos subsequentes de desenvolvimento.

Durante a avaliação e síntese de solução, o principal foco do analista é "que", não "como". Que dados o sistema produz e consome, que funções o sistema deve desempenhar, que comportamentos o sistema exhibe, que interfaces são definidas e que restrições se aplicam?

Durante a atividade de avaliação e síntese de solução, o analista cria modelos do sistema no esforço de entender melhor os dados e o fluxo de controle, o processamento funcional, o comportamento operacional e o conteúdo da informação. O modelo serve como fundamento para o projeto de software e como base para a criação de especificações para o software.

No Cap. 2, mencionamos que especificações detalhadas podem não ser possíveis neste estágio. O cliente pode estar inseguro do que é precisamente necessário. O

Citação

"Nós gastamos muito tempo — a maior parte do tempo total do projeto — não implementando ou testando, mas tentando decidir o que construir."
Brian Lawrence

SUGESTÃO

Espera fazer um pouco de projeto durante a análise de requisitos e um pouco de análise de requisitos durante o projeto.

Qual deveria ser o meu foco principal neste estágio?

Davis [DAV93] argumenta que os termos *que* e *como* são muito vagos. Para uma discussão interessante sobre esse assunto, o leitor deve se referir ao seu livro.

desenvolvedor pode estar inseguro de que uma abordagem específica vai conseguir função e desempenho adequados. Por essas e muitas outras razões, uma abordagem alternativa para a análise de requisitos, chamada *prototipagem*, pode ser conduzida. Discutiremos prototipagem mais adiante neste capítulo.

11.2 ELICITAÇÃO DE REQUISITOS PARA O SOFTWARE

Antes que os requisitos possam ser analisados, modelados ou especificados eles precisam ser reunidos usando-se um processo de elicitação. O cliente tem um problema que pode ser adequado para uma solução baseada em computador. Um desenvolvedor responde ao pedido de ajuda do cliente. A comunicação começou. Mas, como já mencionamos, o caminho da comunicação para o entendimento é frequentemente cheio de buracos.

11.2.1 Início do processo

A técnica mais comumente usada de elicitação de requisitos é conduzir uma reunião ou entrevista. A primeira reunião entre um engenheiro de software (o analista) e o cliente pode ser comparada com a falta de jeito de um primeiro encontro entre dois adolescentes. Nenhuma das pessoas sabe o que dizer ou perguntar; ambas estão preocupadas em serem mal-interpretadas; ambas estão pensando sobre aonde isso iria levar (provavelmente ambas tenham expectativas radicalmente diferentes quanto a isso); ambas desejam acabar logo com o encontro, mas ao mesmo tempo, desejam que ele seja um sucesso.

No entanto, a comunicação precisa ser iniciada. Gause e Weinberg [GAU89] sugerem que o analista comece formulando *questões livres de contexto*. Isto é, um conjunto de questões que levem a um entendimento básico do problema, do pessoal que deseja solução, da natureza da solução que é desejada e da efetividade do primeiro encontro propriamente dito. O primeiro conjunto de questões livres de contexto focaliza o cliente, os objetivos globais e os benefícios. Por exemplo, o analista poderia perguntar:

- Quem está por trás da solicitação deste trabalho?
- Quem vai usar a solução?
- Qual vai ser o benefício econômico de uma solução bem-sucedida?
- Há outra fonte para a solução que você necessita?

Essas questões ajudam a identificar todos os envolvidos que têm interesse no software a ser construído. Além disso, as perguntas identificam o benefício mensurável de uma implementação bem-sucedida e possíveis alternativas para o desenvolvimento de software sob encomenda.

O conjunto de questões a seguir possibilita ao analista obter um melhor entendimento do problema e ao cliente verbalizar sua percepção de uma solução:

- Como você caracterizaria "boas" saídas, que seriam geradas por uma solução bem-sucedida?
- Que problema(s) essa solução enfrentaria?
- Você pode me mostrar (ou descrever) o ambiente no qual a solução será usada?

Citação

"Aquele que faz uma pergunta é um tolo por cinco minutos; aquele que não faz permanece tolo para sempre."
Provérbio chinês

Citação

"Pergunta simples e resposta simples formam o menor caminho para a saída da maioria das perplexidades."
Mark Twain

- Tópicos especiais de desempenho ou restrições vão afetar o modo pelo qual a solução é abordada?

O conjunto final de questões focaliza a efetividade da reunião. Gause e Weinberg [GAU89] as chamam de *metaquestões* e propõem a seguinte lista (abreviada):

- Você é a pessoa certa para responder essas questões? Suas respostas são "oficiais"?
- Minhas questões são relevantes ao problema que você tem?
- Estou formulando muitas questões?
- Alguém mais pode fornecer informação adicional?
- Devo perguntar-lhe mais alguma coisa?

Essas questões (e outras) vão ajudar a "quebrar o gelo" e iniciar a comunicação, que é essencial para a análise bem-sucedida. No entanto, uma reunião na forma de perguntas e respostas não é uma abordagem que tem sido extremamente bem-sucedida. De fato, a seção Q&A deve ser usada apenas para o primeiro encontro e depois ser substituída por uma forma de reunião que combine elementos de solução de problemas, negociação e especificação. Uma abordagem para reuniões desse tipo é apresentada na seção seguinte.

11.2.2 Técnicas facilitadas de especificação de aplicações

Muito frequentemente, clientes e engenheiros de software têm inconscientemente um estado de espírito de "nós e eles". Em vez de trabalhar como equipe para identificar e refinar requisitos, cada parte define seu próprio "território" e se comunica por intermédio de uma série de memorandos, declarações formais de posicionamento, documentos e sessões de perguntas e respostas. A história tem mostrado que essa abordagem não funciona muito bem. Sobram mal-entendidos, informação importante é omitida e nunca é estabelecido um relacionamento de trabalho bem-sucedido.

É com esses problemas em mente que diversos pesquisadores independentes desenvolveram uma abordagem orientada a equipe para elicitação de requisitos que é aplicada durante os primeiros estágios da análise e especificação. Chamada de *técnicas facilitadas de especificação de aplicações* (*facilitated application specification techniques*, FAST), essa abordagem encoraja a criação de uma equipe conjunta de clientes e desenvolvedores, que trabalham juntos para identificar o problema, propor elementos da solução, negociar diferentes abordagens e especificar um conjunto preliminar de requisitos da solução [ZAH90]. A FAST pode ser usada predominantemente pela comunidade de sistemas de informação, mas a técnica oferece potencial para comunicação melhorada em aplicações de todos os tipos.

Muitas abordagens diferentes de FAST têm sido propostas.² Cada uma faz uso de um cenário ligeiramente diferente, mas todas aplicam alguma variante das seguintes diretrizes básicas:

- 2 Duas das abordagens mais populares de FAST são o projeto conjunto de aplicações (JAD), desenvolvido pela IBM, e o METHOD, desenvolvido por Performance Resources, Inc., Falls Church, Virgínia.

SUGESTÃO

Se um sistema ou produto vai servir a muitos usuários, certifique-se de que os requisitos sejam obtidos de uma amostra representativa dos usuários. Se apenas um usuário define todos os requisitos, o risco de aceitação é alto.

Na Web

Uma abordagem do FAST é chamada de "projeto conjunto de aplicações" (*joint application design*, JAD). Uma discussão detalhada do JAD pode ser encontrada em www.bee.net/bluebird/jadoc.htm

P O que torna uma reunião FAST diferente de uma reunião comum?

- Uma reunião é conduzida em um local neutro com a participação de engenheiros de software e de clientes.
- São estabelecidas regras para preparação e participação.
- É sugerida uma agenda que é suficientemente formal para cobrir todos os pontos importantes, porém suficientemente informal para encorajar o livre fluxo de idéias.
- Um "facilitador" (que pode ser um cliente, um desenvolvedor ou uma pessoa de fora) controla a reunião.
- É usado um "mecanismo de definição" (que podem ser folhas de rascunho, *flip charts*, ou papel auto-adesivo, ou um quadro de avisos eletrônico, salas de conversa ou fórum virtual).
- A meta é identificar o problema, propor elementos da solução, negociar diferentes abordagens e especificar um conjunto preliminar de requisitos da solução em uma atmosfera que propicie que a meta seja alcançada.

Citação

"Fatos não deixam de existir porque são ignorados."
Aldous Huxley

Ponto CHAVE

Antes da reunião FAST, faça uma lista de objetos, serviços, restrições e critérios de desempenho.

Para entender melhor o fluxo de eventos que ocorre numa reunião típica de FAST, apresentamos um cenário resumido que delineia a sequência de eventos que provocam a reunião, ocorrem durante e após a reunião.

Reuniões iniciais entre o cliente e o desenvolvedor (Seção 11.2.1) ocorrem e perguntas e respostas básicas ajudam a estabelecer o escopo do problema e a percepção geral de uma solução. Como resultado dessas reuniões iniciais, o desenvolvedor e o cliente redigem uma "solicitação de produto", de uma ou duas páginas. São selecionados um local de reunião, horário e data para FAST e é escolhido um facilitador. Participantes das organizações, tanto do desenvolvedor quanto do cliente/usuário, são convidados a comparecer. A solicitação de produto é distribuída para todos os participantes antes da data da reunião.

Enquanto revê a solicitação nos dias que precedem a reunião, é solicitado a cada participante que faça uma lista dos *objetos* que são parte do ambiente que cerca o sistema, outros objetos que devem ser produzidos pelo sistema e objetos que são usados pelo sistema para desempenhar suas funções. Além disso, é solicitado a cada participante que faça outra lista de *serviços* (processos ou funções) que manipulam ou interagem com os objetos. Finalmente, são também desenvolvidas listas de *restrições* (p. ex., custo, tamanho, regras de negócio) e *critérios de desempenho* (p. ex., velocidade, precisão). Os participantes são informados de que não se espera que as listas sejam exaustivas, mas espera-se que elas reflitam a percepção do sistema de cada um.

Como exemplo,³ considere que uma equipe FAST trabalhando para uma empresa de produtos para o consumidor obteve a seguinte descrição de produto:

Nossa pesquisa indica que o mercado para sistemas domésticos de segurança está crescendo a uma taxa de 40% ao ano. Gostariamos de entrar nesse mercado construindo um sistema de segurança doméstico baseado em microprocessador, que iria proteger contra e/ou reconhecer várias "situações" indesejáveis tais como entrada ilegal, fogo, inundação e outras. O produto, chamado experimentalmente de *SafeHome*, usará sensores adequados para detec-

³ Esse exemplo (com extensões e variações) será usado para ilustrar importantes métodos de engenharia de software em vários capítulos seguintes. Como exercício, valeria a pena conduzir sua própria reunião FAST e desenvolver um conjunto de listas para ele.

tar cada situação, poderá ser programado pelo proprietário e disará automaticamente para uma agência de monitoração quando uma situação for detectada.

Na verdade, muito mais informação seria fornecida nesse estágio. Mas mesmo com informação adicional, teríamos ambigüidade, e provavelmente omissões e erros poderiam ocorrer. Por enquanto, a "descrição de produto" precedente vai ser suficiente.

A equipe FAST é composta de representantes de comercialização, engenharia de software e hardware, e fabricação. Um facilitador externo será usado.

Cada pessoa da equipe FAST desenvolve as listas descritas anteriormente. Os objetos descritos para *SafeHome* poderiam incluir detectores de fumaça, sensores para janelas e portas, detectores de movimento, um alarme, um evento (um sensor foi ativado), um painel de controle, um mostrador, números de telefone, uma chamada telefônica e assim por diante. A lista de serviços poderia incluir ligar o alarme, monitorar os sensores, discar o telefone, programar o painel de controle, ler o mostrador (note que os serviços agem sobre os objetos). De modo semelhante, cada participante do FAST vai desenvolver listas de restrições (p. ex., o sistema deve ter um custo de fabricação de menos de \$80, deve ser amigável ao uso, deve fazer interface diretamente com uma linha telefônica padrão) e de critérios de desempenho (p. ex., um evento de sensoramento deve ser reconhecido dentro de um segundo, um esquema de prioridade de eventos deve ser implementado).

Quando a reunião FAST começa, o primeiro tópico de discussão é a necessidade e a justificativa do novo produto — todos devem concordar que o produto é justificável. Uma vez estabelecida a concordância, cada participante apresenta suas listas para discussão. As listas podem ser penduradas nas paredes da sala usando grandes folhas de papel, grudadas nas paredes usando folhas com adesivo na parte de trás ou escritas num quadro de parede. Alternativamente, as listas podem ter sido colocadas num quadro de avisos eletrônico ou postas num ambiente de bate-papo eletrônico para revisão anterior à reunião. O ideal seria que cada entrada em uma lista pudesse ser manipulada separadamente de modo que as listas possam ser combinadas, as entradas possam ser apagadas e possam ser feitas adições. Nesse estágio, críticas e debates são estritamente proibidos.

Depois que as listas individuais forem apresentadas sobre um assunto, uma lista combinada é criada pelo grupo. A lista combinada elimina entradas redundantes, adiciona qualquer idéia nova que surgiu durante a discussão, mas não apaga nada. Depois que listas combinadas para todos os assuntos tiverem sido criadas, a discussão — coordenada pelo facilitador — tem início. A lista combinada é reduzida, ampliada ou reescrita para refletir adequadamente o produto/sistema a ser desenvolvido. O objetivo é desenvolver uma *lista de consenso* em cada assunto (objetos, serviços, restrições e desempenho). As listas são então postas de lado para ação posterior.

Uma vez completadas as listas de consenso, a equipe é subdividida em equipes menores; cada uma trabalha para desenvolver *miniespecificações* para uma ou mais entradas de cada uma das listas.⁴ Cada miniespecificação é um detalhamento da

Ponto CHAVE

Objetos são manipulados por serviços e devem "conviver" com as restrições e desempenho definidos pelo equipe FAST.

SUGESTÃO

Evite o impulso de atacar uma idéia do cliente como "muito cara" ou "não-prática". A idéia aqui é negociar uma lista que seja aceitável por todos. Para fazer isso, você deve estar com a mente aberta.

⁴ Uma abordagem alternativa resulta na criação de casos de uso. Veja a Seção 11.2.4 para mais detalhes.

palavra ou frase que consta de uma lista. Por exemplo, a miniespecificação para o objeto painel de controle do *SafeHome* poderia ser

- montado em parede
- tamanho aproximado de 22 x 12 centímetros
- contém teclado padrão de 12 teclas e teclas especiais
- contém mostrador LCD com a forma mostrada no esboço (não apresentado aqui)
- toda a interação com o cliente ocorre através de teclas
- usado para ligar e desligar o sistema
- software fornece diretrizes para a interação, ecos etc.
- conectado a todos os sensores

Cada subequipe apresenta então cada uma das suas miniespecificações a todos os participantes do FAST para discussão. Adições, supressões e mais detalhes são feitos. Em alguns casos, o desenvolvimento de miniespecificações vai descobrir novos objetos, serviços, restrições ou requisitos de desempenho que serão adicionados às listas originais. Durante as discussões, a equipe pode levantar um assunto que não pode ser resolvido durante a reunião. Uma lista de assuntos é mantida de modo que as idéias possam ser trabalhadas depois.

Depois que as miniespecificações são completadas, cada participante FAST faz uma lista de *critérios de validação* para o produto/sistema e apresenta sua lista à equipe. Uma lista de consenso dos critérios de validação é então criada. Finalmente, atribui-se a um ou mais participantes (ou pessoa externa) a tarefa de redigir o rascunho completo da especificação, usando todas as entradas produzidas pela reunião FAST.

FAST não é uma panacéia para os problemas encontrados na elicitação inicial de requisitos. Mas a abordagem de equipe fornece os benefícios de muitos pontos de vista, discussão e refinamento instantâneos e é um passo concreto para o desenvolvimento de uma especificação.

11.2.3 Desdobramento da função de qualidade

Desdobramento da função de qualidade (*quality function deployment*, QFD) é uma técnica de gestão de qualidade que traduz as necessidades do cliente para requisitos técnicos do software. Originalmente desenvolvida no Japão e usada inicialmente no estaleiro de Kobe, da Mitsubishi Heavy Industries, Ltd., no início da década de 70, o QFD “concentra-se em maximizar a satisfação do cliente a partir do processo de engenharia de software [ZUL92]”. Para conseguir isso, o QFD enfatiza o entendimento do que tem valor para o cliente e depois desdobra esses valores através do processo de engenharia. O QFD identifica três tipos de requisitos [ZUL92]:

Requisitos normais. Os objetivos e metas que são estabelecidos para o produto ou sistema durante reuniões com o cliente. Se esses requisitos estiverem presentes, o cliente fica satisfeito. Exemplos de requisitos normais poderiam ser os tipos de mostradores gráficos requeridos, funções específicas do sistema e níveis de desempenho definidos.

Requisitos esperados. Esses requisitos estão implícitos no produto ou sistema e podem ser tão fundamentais que o cliente não se refere a eles explicitamente. Sua

SUGESTÃO

Todos querem implementar muitos requisitos excitantes, mas temia cuidado. É assim que a “febre de requisitos” se instala. Por outro lado, frequentemente os requisitos excitantes levam a um produto que se sabreza!

Na Web

O QFD Insitute é uma excelente fonte de informação:
www.qfdi.org

ausência seria causa de insatisfação significativa. Exemplos de requisitos esperados são: facilidade de interação homem/máquina, correção e confiabilidade operacional global e facilidade de instalação do software.

Requisitos excitantes. Essas características vão além das expectativas do cliente e mostram ser muito satisfatórias quando presentes. Por exemplo, um software de processamento de texto é solicitado com características padrão. O produto entregue dispõe de várias facilidades para layout de página, que são muito agradáveis e inesperadas.

Na verdade, o QFD cobre todo o processo de engenharia [AKA90]. No entanto, muitos conceitos QFD são aplicáveis à atividade de elicitação de requisitos. Apresentamos apenas um panorama desses requisitos (adaptados para software de computadores) nos parágrafos seguintes.

Em reuniões com o cliente, o *desdobramento (deployment) de funções* é usado para determinar o valor de cada função necessária ao sistema. O *desdobramento da informação* identifica tanto os objetos de dados quanto os eventos que o sistema deve consumir e produzir. Eles são ligados às funções. Finalmente, o *desdobramento de tarefas* examina o comportamento do sistema ou produto dentro do contexto do seu ambiente. A *análise de valor* é conduzida para determinar a prioridade relativa dos requisitos determinados durante cada um dos três desdobramentos.

O QFD usa entrevistas com o cliente e observação, levantamentos e exame de dados históricos (p. ex., relatórios de problemas) como dados em bruto para a atividade de elicitação de requisitos. Esses dados são então traduzidos para uma tabela de requisitos — chamada de *tabela da voz do cliente* — que é revisada com o cliente. Uma variedade de diagramas, matrizes e métodos de avaliação é então usada para elicitar os requisitos esperados e para tentar derivar os requisitos notáveis [BOS91].

11.2.4 Casos de uso

À medida que os requisitos são elicitados como parte de reuniões informais, FAST ou QFD, o engenheiro de software (analista) pode criar um conjunto de cenários que identifica uma linha de uso para o sistema a ser construído. Os cenários, frequentemente chamados de *casos de uso* [JAC92], fornecem uma descrição de como o sistema será usado.

Para criar um caso de uso, o analista deve primeiro identificar os diferentes tipos de pessoas (ou dispositivos) que usam o sistema ou produto. Esses *atores*, na verdade, representam papéis que as pessoas (ou dispositivos) desempenham quando o sistema opera. Definido de um modo um tanto mais formal, um ator é qualquer coisa que se comunica com o sistema ou produto e que não faz parte dele.

É importante notar que um ator e um usuário não são a mesma coisa. Um usuário típico pode desempenhar vários papéis diferentes quando usa o sistema, enquanto um ator representa uma classe de entidades externas (frequentemente, mas nem sempre, pessoas) que desempenha apenas um papel. Como exemplo, considere um operador de máquina (um usuário) que interage com o computador de controle de uma célula de fabricação, que contém um certo número de robôs e máquinas de controle numérico. Após cuidadosa revisão dos requisitos, o software para o computador de controle

Citação

“O começo é a parte mais importante do trabalho.”
Platão

Ponto CHAVE

QFD define requisitos de um modo que maximiza a satisfação do cliente.

Ponto CHAVE

Um caso de uso é um cenário que descreve como o software deve ser usado numa dada situação.



Casos de uso.

Ponto CHAVE

Casos de uso são definidos do ponto de vista de um ator. Um ator é um papel que pessoas (usuários) ou dispositivos desempenham quando eles interagem com o software.

Na Web

Uma discussão detalhada de casos de uso, incluindo exemplos, diretrizes e gabaritos é apresentada em members.aol.com/acackburn/

requer quatro diferentes modos (papéis) de interação: modo de programação, modo de teste, modo de monitoração e modo de reparação. Assim, quatro atores podem ser definidos: programador, testador, monitor e reparador. Em alguns casos, o operador de máquina pode desempenhar os quatro papéis. Em outros, diferentes pessoas podem desempenhar o papel de cada ator.

Como a elicitação de requisitos é uma atividade evolutiva, nem todos os atores são identificados durante a primeira iteração. É possível identificar atores principais, [JAC92] durante a primeira iteração, e atores secundários, quando se fica sabendo mais a respeito do sistema. Os atores principais interagem para conseguir a função desejada do sistema e derivam o benefício pretendido. Trabalham direta e frequentemente com o software. Os atores secundários dão suporte ao sistema, de modo que os atores principais possam fazer seu trabalho.

Uma vez identificados os atores, casos de uso podem ser desenvolvidos. O caso de uso descreve o modo pelo qual um ator interage com o sistema. Jacobson [JAC92] sugere algumas questões que devem ser respondidas pelo caso de uso:

- Que tarefas ou funções principais são desempenhadas pelo ator?
- Que informação do sistema o ator vai adquirir, produzir ou modificar?
- O ator vai ter que informar o sistema sobre as alterações do ambiente externo?
- Que informação o ator deseja do sistema?
- O ator deseja ser informado sobre modificações inesperadas?

Em geral, um caso de uso é simplesmente uma narrativa escrita que descreve o papel de um ator à medida que ocorre a interação com o sistema.

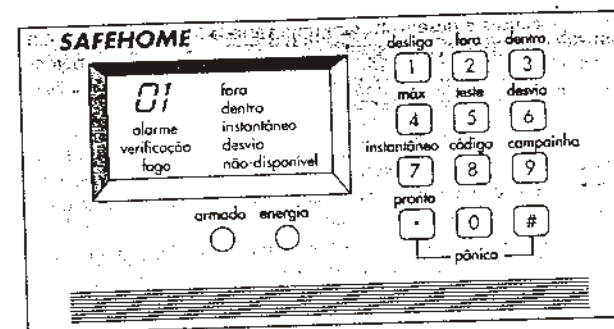
Voltando aos requisitos básicos do *SafeHome* (Seção 11.2.2), podemos definir três atores: o proprietário (usuário), sensores (dispositivos ligados ao sistema) e o subsistema de monitoração e resposta (a estação central que monitora *SafeHome*). Para a finalidade deste exemplo, consideramos apenas o ator proprietário. O proprietário interage com o produto de um certo número de modos diferentes:

- entra com uma senha para permitir todas as outras interações
- consulta o estado de uma zona de segurança
- consulta o estado de um sensor
- aperta o botão de pânico numa emergência
- ativa/desativa o sistema de segurança

Um caso de uso para *ativação do sistema* é o seguinte:

1. O proprietário observa um protótipo do painel de controle do *SafeHome* (Fig. 11.2) para determinar se o sistema está disponível para a entrada. Se o sistema não está pronto, o proprietário tem que fechar fisicamente as janelas/portas de modo que o indicador de pronto se apresente. (Um indicador de *não disponível* implica que um sensor está aberto; isto é, que uma porta ou janela está aberta.)
2. O proprietário usa o teclado para teclar uma senha de quatro dígitos. A senha é comparada com a senha válida armazenada no sistema. Se a senha não for correta, o painel de controle vai emitir um bip e preparar-se para uma entrada adicional. Se a senha estiver correta, o painel de controle espera a ação subsequente.

Fig. 11.2 Painel de controle do *SafeHome*.



3. O proprietário seleciona e tecla *dentro* ou *fora* (ver Fig. 11.2) para ativar o sistema. *Dentro* ativa somente os sensores periféricos (os sensores internos de detecção de movimento são desativados). *Fora* ativa todos os sensores.
4. Quando a ativação ocorre, uma luz de alarme vermelha pode ser observada pelo proprietário.

Casos de uso para outras interações do proprietário seriam desenvolvidos de modo semelhante. É importante notar que cada caso de uso deve ser revisado com cuidado. Se algum elemento da interação é ambíguo, é provável que uma revisão do caso de uso vá indicar um problema.

Cada caso de uso fornece um cenário não-ambíguo de interação entre um ator e o software. Ele pode também ser usado para especificar requisitos de tempo ou outras restrições do cenário. Por exemplo, no caso de uso em questão, requisitos indicam que a ativação ocorre 30 segundos depois que a tecla *dentro* ou *fora* é pressionada. Essa informação pode ser juntada ao caso de uso.

Casos de uso descrevem cenários que serão percebidos de modo diferente por diferentes atores. Wyder [WYD96] sugere que o desdobramento da função de qualidade pode ser usado para desenvolver um valor ponderado de prioridade para cada caso de uso. Para conseguir isso, os casos de uso são avaliados do ponto de vista de todos os atores definidos para o sistema. Um valor de prioridade é atribuído a cada caso de uso (p. ex., um valor de 1 a 10) por cada um dos atores.⁵ Uma prioridade média é então calculada, indicando a importância perceptível de cada um dos casos de uso. Quando é usado um modelo de processo iterativo para a engenharia de software, as prioridades podem influenciar qual funcionalidade do sistema é entregue primeiro.

11.3 PRINCÍPIOS DE ANÁLISE

Durante as últimas duas décadas, um grande número de métodos de modelagem da análise foi desenvolvido. Pesquisadores identificaram problemas de análise e sua

- 5 O ideal é que essa avaliação seja realizada por indivíduos da organização ou função e negócios representados por um ator.

causas e desenvolveram uma variedade de notações de modelagem e conjuntos de heurísticas correspondentes para resolvê-los. Cada método de análise tem um ponto de vista singular. No entanto, todos os métodos de análise estão relacionados por um conjunto de princípios operacionais:

1. O domínio de informação de um problema precisa ser representado e entendido.
2. As funções a serem desenvolvidas pelo software devem ser definidas.
3. O comportamento do software (como consequência de eventos externos) precisa ser representado.
4. Os modelos que mostram informação, função e comportamento devem ser partitionados de um modo que revele detalhes em forma de camadas (ou hierarquias).
5. O processo de análise deve ir da informação essencial até o detalhe de implementação.

Aplicando esses princípios, o analista aborda um problema sistematicamente. O domínio de informação é examinado de modo que a função possa ser entendida mais completamente. Modelos são usados de modo que características de função e comportamento possam ser relatadas de forma compacta. O partitionamento é aplicado para reduzir a complexidade. As visões essencial e de implementação do software são necessárias para acomodar as restrições lógicas impostas pelos requisitos de processamento e as restrições físicas impostas por outros elementos do sistema.

Além desses princípios operacionais de análise, Davis [DAV95a] sugere um conjunto de princípios diretivos para a engenharia de requisitos:

- *Entenda o problema antes que você comece a criar um modelo de análise.* Há uma tendência de correr para uma solução, mesmo antes do problema ser entendido. Isso frequentemente leva a um software elegante que resolve o problema errado!
- *Desenvolva protótipos que permitam ao usuário entender como a interação homem/máquina vai ocorrer.* Como a percepção da qualidade do software é frequentemente baseada na constatação da "amigabilidade" da interface, a prototipagem (e a iteração resultante) é altamente recomendável.
- *Registre a origem e a razão para cada requisito.* Esse é o primeiro passo para estabelecer a rastreabilidade até o cliente.
- *Use múltiplas visões dos requisitos.* Construir modelos de dados, funcional e comportamental dá ao engenheiro de software três diferentes visões. Isso reduz a probabilidade de que algo seja omitido e aumenta a probabilidade de que a inconsistência seja reconhecida.
- *Ordene os requisitos.* Prazos apertados podem impedir a implementação de cada requisito do software. Se um modelo de processo incremental (Cap. 2) é aplicado, os requisitos a serem satisfeitos no primeiro incremento devem ser identificados.

6. Apenas um pequeno subconjunto dos princípios de Davis para a engenharia de requisitos é apresentado aqui. Para mais informação, veja [DAV95a].

P Quais são os princípios subjacentes que guiam o trabalho de análise?

Ponto CHAVE

O domínio de informação de um problema abrange itens ou objetos de dados que contêm números, texto, imagens, áudio, vídeo ou qualquer combinação disso.

SUGESTÃO

Para começar a entender o domínio de informação, a primeira pergunta a ser feita é "Que informação este sistema produz como saída?"

- *Trabalhe para eliminar a ambiguidade.* Como a maioria dos requisitos são descritos em linguagem natural, a oportunidade de ambiguidade é grande. O uso de revisões técnicas formais é um modo de descobrir e eliminar a ambiguidade.

É mais provável que um engenheiro de software, que leva a sério esses princípios, desenvolva uma especificação de software que forneça excelente base para o projeto.

11.3.1 O domínio da informação

Todas as aplicações de software podem ser coletivamente chamadas de *processamento de dados*. É interessante que esse termo contém uma chave para o nosso entendimento de requisitos de software. O software é construído para processar dados, para transformar dados de uma forma para outra; isto é, para aceitar entrada, tratá-la de algum modo e produzir saída. Essa declaração de objetivo fundamental é verdadeira, quer estejamos construindo software com processamento por lotes para um sistema de folha de pagamento, ou software embutido de tempo real para controlar o fluxo de combustível para um motor de automóvel.

É importante notar, no entanto, que o software também processa eventos. Um evento representa algum aspecto do controle do sistema e realmente nada mais é do que dados booleanos — é ligado ou desligado, verdadeiro ou falso, presente ou ausente. Por exemplo, um sensor de pressão detecta que a pressão excedeu um valor seguro e manda um sinal de alarme para o software de monitoração. O sinal de alarme é um evento que controla o comportamento do sistema. Assim, dados (números, texto, imagens, sons, vídeo etc.) e controle (eventos) residem ambos no domínio de informação de um problema.

O primeiro princípio operacional de análise requer um exame do domínio de informação e a criação de um *modelo de dados*. O domínio de informação contém três diferentes visões de dados e controles à medida que cada um é processado por um programa de computador: (1) conteúdo da informação e relacionamentos (o modelo de dados), (2) fluxo da informação e (3) estrutura da informação. Para entender completamente o domínio de informação, cada uma dessas visões deve ser considerada.

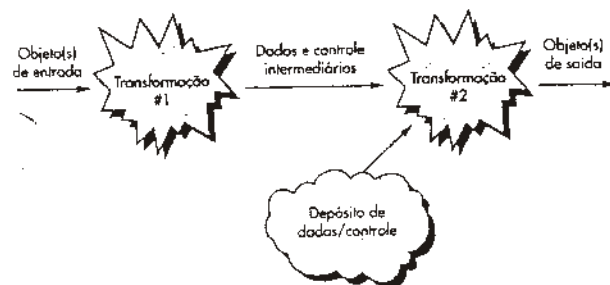
Conteúdo da informação representa os objetos individuais de dados e controle que constituem alguma grande coleção de informação transformada pelo software. Por exemplo, o objeto de dados *cheque de pagamento*, é uma composição de um certo número de peças importantes de dados: o nome de quem vai receber o pagamento, a quantia líquida a ser paga, a quantia bruta, as deduções e etc. Assim, o conteúdo de *cheque de pagamento* é definido pelos atributos que são necessários para criá-lo. Analogamente, o conteúdo de um objeto de controle chamado *estado do sistema* poderia ser definido como uma cadeia de bits. Cada bit representa um item de informação separado que indica se um certo dispositivo está ou não ligado ou desligado.

Objetos de dados e controle podem ser relacionados a outros objetos de dados e controle. Por exemplo, o objeto de dados *cheque de pagamento* tem uma ou mais relações com os objetos *cartão de ponto*, *empregado*, *banco* e outros. Essas relações devem ser definidas, durante a análise do domínio de informação.

Citação

"Um computador fará o que você disser a ele para fazer, mas isso pode ser muito diferente daquilo que você tinha em mente."
Joseph Weizenbaum

Fig. 11.3 Fluxo e transformação da informação.



O *fluxo da informação* representa o modo pelo qual dados e controle se modificam à medida que cada um se move através de um sistema. Com referência à Fig. 11.3, objetos de entrada são transformados em informação intermediária (dados e/ou controle), que são subsequentemente transformados em saída. Ao longo desse caminho (ou caminhos) de transformação, informação adicional pode ser introduzida a partir de um depósito de dados existentes (p. ex., arquivo de disco ou buffer de memória). As transformações aplicadas aos dados são funções ou subfunções que um programa deve realizar. Dados e controle que se movem entre duas transformações (funções) definem a interface de cada função.

Estrutura da informação representa a organização interna dos vários itens de dados e controle. Os itens de dados ou controle devem ser organizados em uma tabela *n*-dimensional ou como uma estrutura de árvore hierárquica? Dentro do contexto da estrutura, que informação é relacionada a outra informação? Toda informação está contida em uma única estrutura ou estruturas diferentes devem ser usadas? Como a informação de uma estrutura da informação se relaciona à informação de outra estrutura? Essas e outras questões são respondidas por uma avaliação da estrutura da informação. Deve-se notar que a estrutura de dados, conceito relacionado que é discutido posteriormente neste livro, refere-se ao projeto ou à implementação da estrutura da informação no software.

11.3.2 Modelagem

Criamos modelos funcionais para obter maior entendimento da entidade real a ser construída. Quando a entidade é uma coisa física (um edifício, um avião, uma máquina), podemos construir um modelo que é idêntico em forma e aspecto, mas menor em escala. No entanto, quando a entidade a ser construída é software, nosso modelo deve assumir uma forma diferente. Deve ser capaz de representar a informação, que o software transforma, as funções (e subfunções), que permitem que a transformação ocorra, e o comportamento do sistema, à medida que a transformação está sendo efetuada.

O segundo e o terceiro princípios operacionais de análise requerem que construamos modelos de função e comportamento.

? Que tipos de modelos criamos durante a análise de requisitos?

? Como usamos os modelos que criamos durante a análise de requisitos?

Modelos funcionais. Os software transforma a informação, e a fim de conseguir isso precisa realizar pelo menos três funções genéricas: entrada, processamento e saída. Quando modelos funcionais de uma aplicação são criados, o engenheiro de software focaliza funções específicas do problema. Um modelo funcional começa com um único modelo de contexto (i. e., o nome do software a ser construído). Durante uma série de iterações, mais e mais detalhes funcionais são fornecidos, até que um delineamento preciso de toda a funcionalidade do sistema é representado. **Modelos de comportamento.** A maioria do software responde a eventos do mundo exterior. Essa característica estímulo/resposta forma a base do modelo comportamental. Um programa de computador existe sempre em algum estado — um modo de comportamento externamente observável (p. ex., esperando, calculando, imprimindo, consultando), que é modificado somente quando ocorre algum evento. Por exemplo, o software vai permanecer no estado de espera até que (1) um relógio interno indica que algum intervalo de tempo passou, (2) um evento externo (p. ex., um movimento de mouse) causa uma interrupção, ou (3) um sistema externo sinaliza para o software agir de uma certa maneira. Um modelo de comportamento cria uma representação dos estados do software e dos eventos que causam mudança de estado no software.

Modelos criados durante a análise de requisitos servem a diversos papéis importantes:

- O modelo ajuda o analista no entendimento da informação, função e comportamento de um sistema, tornando conseqüentemente a tarefa de análise de requisitos mais fácil e mais sistemática.
- O modelo torna-se o focal para revisões e conseqüentemente a chave para determinação da completeza, da consistência e da precisão das especificações.
- O modelo torna-se base para o projeto, dando ao projetista uma representação essencial do software que pode ser "mapeada" para um contexto de implementação.

Os métodos de análise que são discutidos nos Caps. 12 e 21 são na verdade métodos de modelagem. Apesar de o método de modelagem usado ser freqüentemente assunto de preferência pessoal (ou organizacional), a atividade de modelagem é fundamental para um bom trabalho de análise.

11.3.3 Particionamento

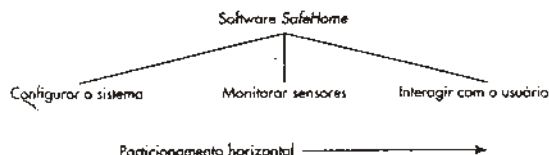
Os problemas são freqüentemente muito grandes e complexos para serem entendidos como um todo. Por isso tendemos a particionar (dividir) tais problemas em segmentos que podem ser facilmente entendidos e estabelecer interfaces entre os segmentos de modo que a função global possa ser compreendida. O quarto princípio operacional de análise sugere que os domínios de software informacional, funcional e comportamental podem ser particionados.

Em essência, o *particionamento* decompõe um problema nos seus segmentos constituintes. Conceitualmente estabelecemos uma representação hierárquica da função ou da informação e depois particionamos o elemento de cima (1) expondo cada vez mais detalhes, movendo-nos verticalmente na hierarquia ou (2) decompondo funcionalmente o problema, movendo-nos horizontalmente na hierarquia. Para ilustrar

Ponto CHAVE

Particionamento é um processo que resulta no detalhamento de dados, funções e comportamento. Pode ser realizada horizontal ou verticalmente.

Fig. 11.4 Particionamento horizontal da função do SafeHome.



essas abordagens de particionamento, vamos considerar novamente o sistema de segurança *SafeHome*, descrito na Seção 11.2.2. A atribuição de software para o *SafeHome* (derivada como consequência das atividades de engenharia de sistemas e FAST) pode ser enunciada nos seguintes parágrafos:

O software do *SafeHome* permite que o proprietário configure o sistema de segurança quando é instalado, monitorea todos os sensores conectados ao sistema de segurança e interage com o proprietário por intermédio de um teclado e teclas de função contidas no painel de controle do *SafeHome*, mostrado na Fig. 11.2.

Durante a instalação, o painel de controle do *SafeHome* é usado para "programar" e configurar o sistema. A cada sensor é atribuído um número e tipo, uma senha mestra é programada para armar e desarmar o sistema e números de telefone são introduzidos para serem discados quando um evento de sensor ocorrer.

Quando um evento de sensor é reconhecido, o software aciona um alarme sonoro conectado ao sistema. Após um tempo de espera, que é especificado pelo proprietário durante as atividades de configuração do sistema, o sistema discar o número do telefone de um serviço de monitoramento, fornece informação sobre o local e informa a natureza do evento que foi detectado. O número de telefone será rediscado a cada 20 segundos até que a conexão telefônica seja conseguida.

Toda a interação com o *SafeHome* é gerenciada por um subsistema de interação com o usuário, que lê a entrada fornecida pelo teclado e pelas teclas de função, exibe mensagens de interação no mostrador de LCD, e exibe informação sobre o estado do sistema no mostrador LCD. A interação pelo teclado torna a seguinte forma...

Os requisitos para o software do *SafeHome* podem ser analisados pelo particionamento dos domínios informacional, funcional e comportamental do produto. Para ilustrar, o domínio funcional do problema vai ser particionado. A Fig. 11.4 ilustra uma *decomposição horizontal do software do SafeHome*. O problema é particionado pela representação das funções de software que constituem o *SafeHome*, movendo-se horizontalmente na hierarquia funcional. Três funções importantes são notadas no primeiro nível da hierarquia.

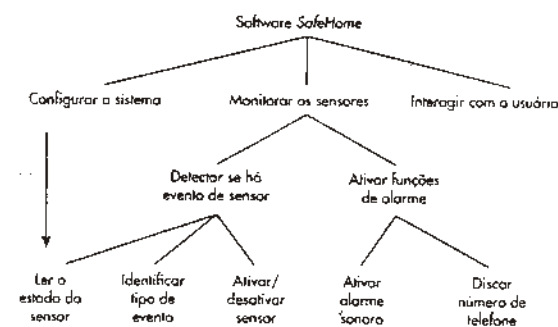
As subfunções associadas com cada função importante do *SafeHome* podem ser examinadas pela exposição dos detalhes verticalmente na hierarquia, como ilustrado na Fig. 11.5. Movendo-se para baixo ao longo do único caminho abaixo da função *monitorar sensores*, o particionamento ocorre verticalmente para mostrar níveis crescentes de detalhes funcionais.

A abordagem de particionamento aplicada às funções do *SafeHome* pode ser aplicada também ao domínio de informação e ao domínio de comportamento, de igual modo.

Citação

"Atividade frenética não é substituto para entendimento."
H.H. Williams

Fig. 11.5 Particionamento vertical da função *SafeHome*.



De fato, o particionamento do fluxo da informação e do comportamento do sistema (discutido no Cap. 12) proporcionará entendimento adicional dos requisitos do software. À medida que o problema é particionado, as interfaces entre as funções são derivadas. Itens de dados e controle, que se movem através de uma interface, devem se restringir às entradas necessárias para realizar a função declarada e às saídas que são necessárias a outras funções ou elementos do sistema.

11.3.4 Visões essenciais de implementação⁷

Uma *visão essencial* dos requisitos de software apresenta as funções a serem realizadas e a informação a ser processada, sem cuidar dos detalhes de implementação. Por exemplo, a visão essencial da função *ler o estado de sensor*, do *SafeHome* não se preocupa com a forma física dos dados ou com o tipo de sensor que é usado. De fato, pode-se argumentar que ler estado seria um nome mais adequado para essa função, de vez que ela não leva em conta detalhes sobre o mecanismo de entrada. Analogamente, um modelo essencial de dados do item de dados *número de telefone* (implícito na função *discar número de telefone*) pode ser representado nesse estágio sem considerar a estrutura de dados subjacente (se houver alguma) usada para implementar o item de dados. Concentrando a atenção na essência do problema, nos primeiros estágios da engenharia de requisitos, deixamos nossas opções abertas para especificar detalhes de implementação durante os últimos estágios da especificação de requisitos e do projeto de software.

A *visão de implementação* dos requisitos de software apresenta a manifestação do mundo real de funções de processamento e estruturas da informação. Em alguns casos, uma representação física é desenvolvida como primeiro passo do projeto do software. Entretanto, a maioria dos sistemas baseados em computador é especificada de um modo que determina a acomodação de certos detalhes de implementação. Um dispositivo de entrada do *SafeHome* é um sensor perimetral (não um cão de guarda, um guarda humano ou uma armadilha). O sensor detecta a entrada ilegal sentindo

⁷ Muitas pessoas usam os termos *visão lógica* e *física* para ressaltar o mesmo conceito.

uma interrupção no circuito eletrônico. As características gerais do sensor devem ser anotadas como parte da especificação de requisitos do software. O analista deve reconhecer as restrições impostas por elementos predefinidos do sistema (o sensor) e considerar a visão de implementação da função e da informação quando tal visão for apropriada.

Já mencionamos que a engenharia de requisitos de software deve focalizar o que o software deve realizar, ao invés de como o processamento vai ser implementado. No entanto, a visão de implementação não deve necessariamente ser interpretada como uma representação do como. Ao invés disso, um modelo de implementação representa o modo atual de operação; isto é, a alocação existente ou proposta para todos os elementos do sistema. O modelo essencial (de função ou dados) é genérico no sentido de que a realização da função não é indicada explicitamente.

11.4 PROTOTIPAGEM DE SOFTWARE

Citação

"Desenvolvedores podem construir e testar com base nas especificações, mas os usuários aceitam ou rejeitam com base nas realidades operacionais existentes e reais."
Bernard Boor

A análise deve ser conduzida independentemente do paradigma de engenharia de software que é aplicado. No entanto, a forma que a análise assume varia. Em alguns casos é possível aplicar princípios operacionais de análise e derivar um modelo do software a partir do qual o projeto pode ser desenvolvido. Em outras situações, a elicitação dos requisitos (usando FAST, QFD, casos de uso ou outras técnicas de *brainstorming* [JOR89]) é conduzida, os princípios de análise são aplicados e um modelo do software, chamado de *protótipo*, é construído para avaliação pelo cliente e pelo desenvolvedor. Finalmente, algumas circunstâncias exigem a construção de um protótipo no início da análise, tendo em vista que o modelo é o único modo pelo qual os requisitos podem ser efetivamente derivados. O modelo então evolui para software em produção.

11.4.1 Seleção da abordagem de prototipagem

O paradigma de prototipagem pode ser de finalidade aberta ou de finalidade fechada. A abordagem de finalidade fechada é freqüentemente chamada de *prototipagem para jogar fora*. Usando essa abordagem, um protótipo serve apenas como uma demonstração rústica dos requisitos. Ele é então descartado e o software é construído usando um paradigma diferente. Uma abordagem de finalidade aberta, chamada *prototipagem evolutiva*, usa o protótipo como primeira parte de uma atividade de análise que vai ter continuidade no projeto e construção. O protótipo do software é a primeira evolução do sistema acabado.

Antes que uma abordagem de finalidade fechada ou de finalidade aberta possa ser escolhida, é necessário determinar se o sistema a ser construído é adequado à prototipagem. Diversos fatores de candidatura a prototipagem [BOA84] podem ser definidos: área da aplicação, complexidade da aplicação, característica do cliente e características do projeto.⁹

⁹ Uma discussão útil de outros fatores de candidatura — "quando prototipar" — pode ser encontrada em [DAV95b].

Fig. 11.6 Seleção da abordagem adequada de prototipagem.

Questão	Protótipo descartável	Protótipo evolutivo	Necessário trabalho preliminar adicional
O domínio de aplicação é entendido?	Sim	Sim	Não
O problema pode ser modelado?	Sim	Sim	Não
O cliente está seguro dos requisitos básicos do sistema?	Sim/Não	Sim/Não	Não
Os requisitos estão estabelecidos e são estáveis?	Não	Sim	Sim
Há algum requisito ambíguo?	Sim	Não	Sim
Há contradições nos requisitos?	Sim	Não	Sim

Em geral, qualquer aplicação que cria mostradores visuais dinâmicos, interage pesadamente com o usuário ou exige algoritmos, ou processamento combinatório que deve ser desenvolvido de modo evolucionário, é candidata à prototipagem. No entanto, essas áreas de aplicação devem ser ponderadas em contraposição à complexidade da aplicação. Se uma aplicação-candidata (que tem as características mencionadas) vai exigir o desenvolvimento de dezenas de milhares de linhas de código antes que qualquer função demonstrável possa ser realizada, é provável que ela seja complexa demais para prototipagem.⁹ Se, no entanto, a complexidade puder ser particionada, pode ainda ser possível prototipar partes do software.

Como o cliente precisa interagir com o protótipo nos passos finais, é essencial que (1) os recursos do cliente sejam empenhados para a avaliação e refinamento do protótipo e (2) que o cliente seja capaz de tomar decisões sobre os requisitos no devido tempo. Finalmente, a natureza do projeto de desenvolvimento vai ter uma forte influência na eficácia da prototipagem. A gerência do projeto está querendo, e pode, trabalhar com o método de prototipagem? Há ferramentas de prototipagem disponíveis? Os desenvolvedores têm experiência com métodos de prototipagem? Andriole [AND92] sugere seis questões (Fig. 11.6) e indica conjuntos típicos de respostas, e a abordagem de prototipagem correspondente sugerida.

11.4.2 Métodos e ferramentas de prototipagem

Para que a prototipagem de software seja efetiva, um protótipo deve ser desenvolvido rapidamente de modo que o cliente possa avaliar os resultados e recomendar modificações. Para realizar a prototipagem rápida, três classes genéricas de métodos e ferramentas (p. ex. [AND92], [TAN89]) estão disponíveis:

⁹ Em alguns casos, protótipos extremamente complexos podem ser construídos rapidamente com o uso de técnicas de quarta geração ou de componentes reusáveis de software.

Técnicas de quarta geração. Técnicas de quarta geração (*fourth generation techniques*, 4GT) incluem um amplo conjunto de linguagens de relatório e de consulta a base de dados, geradores de programa e aplicações, e outras linguagens não-procedimentais de muito alto nível. Como 4GT permitem ao engenheiro de software gerar código executável rapidamente, elas são ideais para a prototipagem rápida. **Componentes de software reusáveis.** Outra abordagem para prototipagem rápida é montar, em vez de construir, o protótipo usando um conjunto de componentes de software existentes. A combinação de prototipagem e reuso de componentes de programa vai funcionar somente se um sistema de biblioteca for desenvolvido de modo que os componentes que de fato existem possam ser catalogados e depois recuperados. Deve-se notar que um produto de software existente pode ser usado como protótipo para um produto competitivo "novo e aperfeiçoado". De certo modo, essa é uma forma de reusabilidade para a prototipagem de software. **Ambientes de especificação formal e prototipagem.** Nas últimas décadas várias linguagens e ferramentas de especificação formais foram desenvolvidas para substituir técnicas de especificação em linguagem natural. Hoje, desenvolvedores dessas linguagens formais estão no processo de desenvolvimento de ambientes interativos que permitem (1) a um analista criar interativamente especificações baseadas na linguagem de um sistema ou software, (2) aplicar ferramentas automatizadas que traduzem as especificações baseadas na linguagem para código executável e (3) ao cliente usar o código executável do protótipo para refinar os requisitos formais.

11.5 ESPECIFICAÇÃO

Não há dúvida de que o modo de especificação tem muito a ver com a qualidade da solução. Engenheiros de software que tenham sido forçados a trabalhar com especificações incompletas, inconsistentes ou confusas experimentaram a frustração e a confusão, que invariavelmente resulta. A qualidade, pontualidade e completeza do software sofre como consequência.

11.5.1 Princípios de especificação

A especificação, independentemente do modo pelo qual nós a realizamos, pode ser vista como um processo de representação. Requisitos são representados de uma forma que, em última análise, leva à implementação bem-sucedida do software. Alguns princípios de especificação, adaptados do trabalho de Balzer e Goodman [BAL86], podem ser propostos:

1. Separe a funcionalidade da implementação.
2. Desenvolva um modelo do comportamento desejado do sistema que abranja os dados e as respostas funcionais a vários estímulos do ambiente.
3. Estabeleça o contexto no qual o software opera, especificando o modo pelo qual outros componentes do sistema interagem com o software.
4. Defina o ambiente no qual o sistema opera e indique como "uma coleção altamente entrelaçada de agentes reage a estímulos do ambiente (modificações em objetos) produzidas por esses agentes" [BAL86].



SUGESTÃO
No trabalho dos casos não é razoável esperar que a especificação vá "colocar os pingos nos is". Ela deve, na verdade, captar a essência do que o cliente exige.

5. Crie um modelo cognitivo ao invés de um modelo de projeto ou implementação. O modelo cognitivo descreve um sistema como ele é percebido pela sua comunidade de usuários.
6. Reconheça que "as especificações devem ser tolerantes à incompleteza e ampliáveis". Uma especificação é sempre um modelo — uma abstração — de alguma situação real (ou imaginada), que é normalmente bastante complexa. Consequentemente, ela vai ser incompleta e vai existir em vários níveis de detalhe.
7. Estabeleça o conteúdo e a estrutura de uma especificação de modo que ela possa ser facilmente modificada.

Essa lista de princípios básicos de especificação fornece a base para a representação dos requisitos de software. No entanto, os princípios devem ser traduzidos em realização. Na seção seguinte examinaremos um conjunto de diretrizes para criar uma especificação de requisitos.

11.5.2 Representação

Já vimos que requisitos de software podem ser especificados de diferentes modos. No entanto, se os requisitos serão postos no papel ou em meio eletrônico de apresentação (e eles quase sempre deveriam ser!), vale a pena seguir um conjunto simples de diretrizes:

O formato da representação e do conteúdo deve ser relevante para o problema.

Um esboço geral do conteúdo de uma *Especificação de Requisitos de Software* pode ser desenvolvido. No entanto, é provável que as formas de representação contidas na especificação variem com a área da aplicação. Por exemplo, a especificação para um sistema de automação de fabricação poderia usar simbologia e diagramas de linguagem diferentes da especificação, para um compilador de linguagem de programação.

A informação contida na especificação deve ser aninhada. As representações devem revelar camadas de informação de modo que um leitor possa ir para o nível de detalhes desejado. Os esquemas de numeração de parágrafos e diagramas devem indicar o nível de detalhes que está sendo apresentado. Algumas vezes vale a pena apresentar a mesma informação em diferentes níveis de abstração para ajudar no seu entendimento.

Diagramas e outras formas de notação devem ser restritos em quantidade e consistentes no uso. Notação confusa ou inconsistente, quer seja gráfica ou simbólica, degrada o entendimento e conduz a erros.

As representações devem poder ser revistas. O conteúdo de uma especificação vai sofrer modificações. O ideal é que ferramentas CASE estivessem disponíveis para atualizar todas as representações que são afetadas por modificação.

Pesquisadores conduziram vários estudos (p. ex., [HOL93], [CUR85]) sobre fatores humanos associados com a especificação. Parece haver pouca dúvida de que simbologia e disposição afetam o entendimento. No entanto, engenheiros de software parecem ter preferências individuais por formas simbólicas e diagramáticas específicas. A familiaridade frequentemente jaz na raiz da preferência de uma pessoa, mas outros



Quais são algumas das diretrizes básicas para representar requisitos?

fatores mais concretos como disposição espacial, padrões facilmente reconhecíveis e grau de formalidade freqüentemente determinam uma escolha individual.

11.5.3 A especificação de requisitos do software

A *especificação de requisitos de software* é produzida no auge da tarefa de análise. A função e o desempenho alocados ao software como parte da engenharia de sistemas são refinadas pelo estabelecimento de uma descrição completa da informação, de uma descrição funcional detalhada, de uma representação do comportamento do sistema, de uma indicação dos requisitos de desempenho e das restrições de projeto, de critérios de validação adequados e de outra informação pertinente aos requisitos. O National Bureau of Standards, o IEEE (Padrão Nº 830-1984) e o Departamento de Defesa Norte-Americano propuseram, cada um, candidatos a formatos para a especificação de requisitos de software (bem como para outra documentação de engenharia de software).

A *introdução* da especificação de requisitos de software declara as metas e objetivos do software, descrevendo-os no contexto do sistema baseado em computador. Na verdade, a introdução pode ser nada mais do que o escopo do software do documento de planejamento.

A *descrição da informação* fornece uma descrição detalhada do problema que o software deve resolver. O conteúdo, fluxo e estrutura da informação são documentados. O hardware, software e interfaces humanas são descritos para elementos externos do sistema e funções internas do software.

Uma descrição de cada função necessária para resolver o problema é apresentada na *descrição funcional*. Uma narrativa de processamento é fornecida para cada função, restrições de projetos são declaradas e justificadas, características de desempenho são declaradas e um ou mais diagramas são incluídos para representar graficamente a estrutura geral do software e a interação entre as funções do software e outros elementos do sistema. A seção da especificação *descrição comportamental* examina a operação do software como consequência de eventos externos e características de controle geradas internamente.

Critérios de validação é provavelmente a seção mais importante e ironicamente é com freqüência a mais negligenciada da *especificação de requisitos do software*. Como reconheceremos uma implementação bem-sucedida? Que classes de testes devem ser conduzidas para validar a função, o desempenho e as restrições? Nós negligenciamos essa seção porque completá-la demanda um profundo entendimento dos requisitos de software — algo que freqüentemente não temos nesse estágio. No entanto, a especificação dos critérios de validação age como uma revisão implícita de todos os outros requisitos. É essencial que tempo e atenção sejam dedicados a essa seção.

Finalmente, a especificação inclui uma *bibliografia e apêndice*. A bibliografia contém referências a todos os documentos que se relacionam ao software. Eles incluem outra documentação de engenharia de software, referências técnicas, literatura de fornecedores e padrões. O apêndice contém informação que suplementa as especificações. Tabelas de dados, descrição detalhada de algoritmos, figuras, gráficos e outros materiais são apresentados no apêndice.



Especificação de requisitos de software.

SUGESTÃO

Quando você desenvolver critérios de validação responda à seguinte questão: "Como eu reconheceria um sistema bem-sucedido se ele fosse deixado sobre a minha mesa amanhã?"

Em muitos casos, a *especificação de requisitos do software* pode ser acompanhada por um protótipo executável (que em alguns casos pode substituir a especificação), por um protótipo em papel ou por um *Manual do Usuário Preliminar*. O *Manual do Usuário Preliminar* apresenta o software como uma caixa-preta. Isto é, grande ênfase é colocada nas entradas do usuário e nas saídas resultantes. O manual pode servir como ferramenta valiosa para descobrir problemas na interface homem/máquina.

11.6 REVISÃO DA ESPECIFICAÇÃO



Revisão da especificação de requisitos do software.

Uma revisão da *especificação de requisitos do software* (e/ou protótipo) é conduzida tanto pelo desenvolvedor do software quanto pelo cliente. Como a especificação forma a base da fase de desenvolvimento, deve ser tomado extremo cuidado na condução da revisão.

A revisão é conduzida primeiro macroscopicamente; isto é, os revisores tentam garantir que a especificação esteja completa, consistente e precisa quando os domínios globais informacional, funcional e comportamental são considerados. No entanto, para explorar completamente cada um desses domínios, a revisão torna-se mais detalhada, examinando não apenas as descrições amplas mas o modo pelo qual os requisitos são redigidos. Por exemplo, quando especificações contêm "termos vagos" (p. ex., *alguns, algumas vezes, freqüentemente, usualmente, ordinariamente, a maior parte ou a maioria*), o revisor deve marcar os trechos para esclarecimento posterior.

Uma vez completada a revisão, a *especificação de requisitos do software* é "assinada", tanto pelo cliente quanto pelo desenvolvedor. A especificação torna-se um "contrato" para o desenvolvimento do software. Solicitações de modificação nos requisitos, depois que a especificação é completada, não serão eliminadas. Mas o cliente deve perceber que cada modificação posterior é uma extensão do escopo do software e conseqüentemente pode aumentar o custo e/ou comprometer o cronograma.

Mesmo com os melhores procedimentos de revisão em ação, vários problemas comuns de especificação persistem. A especificação é difícil de "testar" de qualquer modo significativo e, conseqüentemente, inconsistências ou omissões podem passar despercebidas. Durante a revisão, podem ser recomendadas modificações nas especificações. Pode ser extremamente difícil avaliar o impacto global de uma modificação; isto é, como uma modificação e uma função afeta os requisitos das outras funções. Ambientes modernos de engenharia de software (Cap. 31) incorporam ferramentas CASE que têm sido desenvolvidas para ajudar a resolver esses problemas.

11.7 RESUMO

A análise de requisitos é o primeiro passo técnico do processo de software. É nesse ponto que uma declaração geral do escopo do software é refinada para constituir uma especificação completa que se torna a base de todas as atividades de engenharia de software que se seguem.

A análise deve focalizar os domínios informacional, funcional e comportamental de um problema. Para entender melhor o que é necessário, modelos são criados, o

problema é particionado e representações que mostram a essência dos requisitos e depois os detalhes de implementação são desenvolvidas.

Em muitos casos, não é possível especificar completamente um problema num estágio inicial. A prototipagem oferece uma abordagem alternativa que resulta num modelo executável do software a partir do qual os requisitos podem ser refinados. Para conduzir a prototipagem adequadamente, são necessárias ferramentas e técnicas especiais.

A especificação de requisitos do software é desenvolvida como consequência da análise. A revisão é essencial para garantir que o desenvolvedor e o cliente tenham a mesma percepção do sistema. Infelizmente, mesmo com o melhor método, a questão é que o problema continua se modificando.

REFERÊNCIAS BIBLIOGRÁFICAS

- [AKA90] Akao, Y., ed., *Quality Function Deployment: Integrating Customer Requirements in Product Design* (translated by G. Mazur), Productivity Press, 1990.
- [AND92] Andriole, S., *Rapid Application Prototyping*, QED, 1992.
- [BAL86] Balzer, R. and N. Goodman, "Principles of Good Specification and Their Implications for Specification Languages," in *Software Specification Techniques* (Gehani, N. and A. McGetrick, eds.), Addison-Wesley, 1986, pp. 25-39.
- [BOA84] Boar, B., *Application Prototyping*, Wiley-Interscience, 1984.
- [BOS91] Bossert, J. L., *Quality Function Deployment: A Practitioner's Approach*, ASQC Press, 1991.
- [CUR85] Curtis, B., *Human Factors in Software Development*, IEEE Computer Society Press, 1985.
- [DAV93] Davis, A., *Software Requirements: Objects, Functions and States*, Prentice-Hall, 1993.
- [DAV95a] Davis, A., *201 Principles of Software Development*, McGraw-Hill, 1995.
- [DAV95b] Davis, A., "Software Prototyping," in *Advances in Computers*, volume 40, Academic Press, 1995.
- [GAU89] Gause, D.C. and G.M. Weinberg, *Exploring Requirements: Quality Before Design*, Dorset House, 1989.
- [HOL95] Holtzblatt, K. and E. Carmel (eds.), "Requirements Gathering: The Human Factor," special issue of *CACM*, vol. 38, no. 5, May 1995.
- [JAC92] Jacobson, I., *Object-Oriented Software Engineering*, Addison-Wesley, 1992.
- [JOR89] Jordan, P.W., et al., "Software Storming: Combining Rapid Prototyping and Knowledge Engineering," *IEEE Computer*, vol. 22, no. 5, May 1989, pp. 39-50.
- [REI94] Reifer, D.J., "Requirements Engineering," in *Encyclopedia of Software Engineering* (J.J. Marciniak, ed.), Wiley, 1994, pp. 1043-1054.
- [TAN89] Tanik, M.M. and R.T. Yeh (eds.), "Rapid Prototyping in Software Development," special issue of *IEEE Computer*, vol. 22, no. 5, May 1989.
- [WYD96] Wyder, T., "Capturing Requirements with Use-Cases," *Software Development*, February 1996, pp. 37-40.

- [ZAH90] Zahniser, R.A., "Building Software in Groups," *American Programmer*, vol. 3, nos. 7-8, July-August 1990.
- [ZUL92] Zultner, R., "Quality Function Deployment for Software: Satisfying Customers," *American Programmer*, February 1992, pp. 28-41.

PROBLEMAS E PONTOS A CONSIDERAR

- 11.1. A análise de requisitos do software é inquestionavelmente o passo de comunicação mais intenso do processo de software. Por que o caminho de comunicação frequentemente se rompe?
- 11.2. Há com frequência, repercussões políticas severas quando a análise dos requisitos do software (e/ou a análise do sistema) começa. Por exemplo, funcionários podem sentir que a estabilidade no emprego é ameaçada por um novo sistema automatizado. O que causa tais problemas? A tarefa de análise pode ser conduzida de modo que a política seja minimizada?
- 11.3. Discuta sua percepção sobre o treinamento e a formação ideais de um analista de sistemas.
- 11.4. Ao longo deste capítulo nós nos referimos ao "cliente". Descreva o "cliente" para os desenvolvedores de sistemas de informação, para os construtores de produtos baseados em computador e para os construtores de sistemas. Tenha cuidado, pode haver mais neste problema do que você antes imaginava!
- 11.5. Desenvolva um *kit* de técnicas facilitadas de especificação de aplicações. O *kit* deve incluir um conjunto de diretrizes para conduzir uma reunião FAST e materiais que podem ser usados para facilitar a criação de listas e de outros itens que poderiam ajudar na definição dos requisitos.
- 11.6. Seu instrutor vai dividir a classe em grupos de quatro ou seis alunos. Metade do grupo vai desempenhar o papel do departamento comercial e metade vai fazer o papel do engenheiro de software. Sua tarefa é definir os requisitos para o sistema de segurança *SafeHome* descrito neste capítulo. Conduza uma reunião FAST usando as diretrizes apresentadas neste capítulo.
- 11.7. É justo dizer que um *Manual do Usuário Preliminar* é uma forma de protótipo? Explique sua resposta.
- 11.8. Analise o domínio de informação do *SafeHome*. Represente (usando qualquer notação que pareça apropriada) o fluxo de informação do sistema, o conteúdo de informação e qualquer estrutura de informação que seja relevante.
- 11.9. Particione o domínio funcional do *SafeHome*. Primeiro faça o particionamento horizontal; depois faça o particionamento vertical.
- 11.10. Crie as representações essencial e de implementação para o sistema *SafeHome*.
- 11.11. Faça um protótipo em papel (ou um protótipo real) para *SafeHome*. Assegure-se de mostrar a interação com o proprietário e a função geral do sistema.

- 11.12. Tente identificar os componentes de software do *SafeHome* que poderiam ser "reusáveis" em outros produtos ou sistemas. Tente categorizar esses componentes.
- 11.13. Desenvolva uma especificação escrita para o *SafeHome* usando o esboço fornecido no site SEPWeb. (Nota: Seu instrutor vai sugerir que seções devem ser completadas nessa ocasião.) Certifique-se de aplicar as questões que são descritas para a revisão da especificação.
- 11.14. Como seus requisitos diferiram dos de outros que tentaram uma solução para o *SafeHome*? Quem fez um "Chevy" — quem fez um "Cadillac"?

LEITURAS E FONTES DE INFORMAÇÃO ADICIONAIS

Livros que tratam de engenharia de requisitos fornecem uma boa base para o estudo de conceitos e princípios básicos de análise. Thayer e Dorfman (*Software Requirements Engineering*, 2nd ed., IEEE Computer Society Press, 1997) apresentam uma antologia que vale a pena sobre o assunto. Graham e Graham (*Requirements Engineering and Rapid Development*, Addison-Wesley, 1998) enfatizam o desenvolvimento rápido e o uso de métodos orientados a objetos na sua discussão da engenharia de requisitos, enquanto Mac-Cauley (*Requirements Engineering*, Springer-Verlag, 1996) apresenta um tratamento acadêmico resumido do assunto.

Há alguns anos, a literatura enfatizava a modelagem de requisitos e os métodos de especificação, mas, hoje, igual ênfase tem sido dada a métodos efetivos para eliciação de requisitos. Wood e Silver (*Joint Application Development*, 2nd ed., Wiley, 1995) escreveram o tratado definitivo de JAD. Cohen e Cohen (*Quality Function Deployment*, Addison-Wesley, 1995), Terninko (*Step-by-Step QFD: Customer-Driven Product Design*, Saint Lucie Press, 1997), Gause e Weinberg [GAU89] e Zahniser [ZAH90] discutem os mecanismos de reuniões efetivas, métodos para *brainstorming* e abordagens de eliciação que podem ser usados para esclarecer resultados e diversos outros tópicos úteis. Casos de uso tornaram-se uma parte importante da análise de requisitos orientada a objetos, mas eles podem ser usados independentemente da tecnologia de implementação selecionada. Rosenberg e Scott (*Use-Case Driven Object Modeling with UML: A Practical Approach*, Addison-Wesley, 1999), Schneider *et al.* (*Applying Use-Cases: A Practical Guide*, Addison-Wesley, 1998) e Texel e Williams (*Use-Cases Combined With Booch/OMT/UML*, Prentice-Hall, 1997) fornecem diretrizes detalhadas e muitos exemplos úteis.

A análise do domínio da informação é um princípio fundamental da análise de requisitos. Os livros de Mattison (*The Object-Oriented Enterprise*, McGraw-Hill, 1994), Tillman (*A Practical Guide to Logical Data Modeling*, McGraw-Hill, 1993) e Modell (*Data Analysis, Data Modeling and Classification*, McGraw-Hill, 1992) cobrem vários aspectos desse importante assunto.

Um livro recente de Harrison (*Prototyping and Software Development*, Springer-Verlag, 1999) fornece uma perspectiva moderna de prototipagem de software. Dois livros de Connell e Shafer (*Structured Rapid Prototyping*, Prentice-Hall, 1989) e (*Object-Oriented Rapid Prototyping*, Yourdon Press, 1994) mostram como essa importante técnica de análise pode ser usada tanto em ambiente convencional quanto orientado a objetos.

Outros livros, de Pomberger *et al.* (*Object Orientation and Prototyping in Software Engineering*, Prentice-Hall, 1996) e Krief *et al.* (*Prototyping with Objects*, Prentice-Hall, 1996) examinam prototipagem sob a perspectiva orientada a objetos. O *IEEE Proceedings of the International Workshop on Rapid System Prototyping* (publicado anualmente) apresenta a pesquisa atual na área.

Uma grande variedade de fontes de informação sobre análise de requisitos e assuntos relacionados está disponível na Internet. Uma lista atualizada de referências World Wide Web, que são relevantes aos conceitos e métodos de análise pode ser encontrada no site do autor:

<http://www.rspa.com>

CONCEITOS
CHAVE

análise gramatical	313
CSPC	318
DFD	304
dicionário de dados	321
ERD	300
extensões para	
tempo real	306
mecanismo de análise	
estruturada	312
modelagem	
comportamental	310
de dados	296
funcional	302
modelo de fluxo	295
de controle	316
PSPC	320

PANORAMA

O que é? A palavra escrita é um veículo magnífico de comunicação, mas não é necessariamente o melhor modo de representar os requisitos de software para computador. A modelagem da análise usa uma combinação de formas textuais e diagramáticas para mostrar os requisitos de dados, função e comportamento de modo que seja relativamente fácil de entender e, mais importante, mais direto de revisar quanto a correção, completude e consistência.

Quem faz? Um engenheiro de software (algumas vezes chamado analista) constrói um modelo usando requisitos elicitados do cliente.

Por que é importante? Para validar requisitos de software, você precisa examiná-los de diferentes pontos de vista. A modelagem da análise representa os requisitos em três "dimensões", aumentando conseqüentemente a probabilidade de que sejam encontrados erros, apareçam inconsistências e que omissões sejam descobertas.

Quais são os passos? Os requisitos de dados, funcionais e comportamentais são modelados usando vá-

No nível técnico, a engenharia de software começa com uma série de tarefas de modelagem que levam a uma especificação completa dos requisitos e a uma representação abrangente do projeto para o software a ser construído.

O *modelo de análise*, na verdade um conjunto de modelos, é a primeira representação técnica do sistema. Ao longo do tempo muitos métodos foram propostos para modelagem da análise. No entanto, dois dominam atualmente. O primeiro, *análise estruturada*, é um método clássico de modelagem e é descrito neste capítulo. A outra abordagem, *análise orientada a objetos*, será considerada em detalhes no Cap. 21. Outros métodos de análise comumente usados são mencionados na Seção 12.8.

A análise estruturada é uma atividade de construção de modelo. Aplicando os princípios operacionais de análise discutidos no Cap. 11, criamos e particionamos os modelos de dados funcional e comportamental, que mostram a essência do que deve ser construído. A análise estruturada não é o único método aplicado consistentemente por todos que o usam. Ao invés disso, é uma amálgama que evoluiu durante mais de 30 anos.

Num livro pioneiro sobre o assunto, Tom DeMarco [DEM79] descreve a análise estruturada do seguinte modo:

Reverendo os problemas reconhecidos e as falhas da fase de análise, sugiro que façamos as seguintes adições ao nosso conjunto de metas para a fase de análise:

rios formatos diagramáticos diferentes. A modelagem de dados define objetos de dados, atributos e suas relações. A modelagem funcional indica como os dados são transformados num sistema. A modelagem comportamental mostra o impacto dos eventos. Uma vez criados os modelos preliminares, eles são refinados e analisados para avaliar sua clareza, completude e consistência. Uma especificação que incorpora o modelo é criada e depois validada tanto pelos engenheiros de software quanto pelos clientes/usuários.

Qual é o produto do trabalho? Descrições dos objetos de dados, diagramas entidade relacionamento, diagramas de fluxo de dados, diagramas de transição de estados, especificações de processo e especificações de controle são criados como parte da atividade de modelagem da análise.

Como garanto que fiz corretamente? Os produtos do trabalho da modelagem da análise devem ser revisados quanto à correção, completude e consistência.

- Os produtos da análise devem ser altamente manuteníveis. Isso se aplica particularmente ao documento-alvo [especificações de requisitos do software].
- Problemas de tamanho devem ser enfrentados usando um método efetivo de particionamento. A especificação da novela Victoriana está fora.
- Gráficos precisam ser usados sempre que possível.
- Precisamos diferenciar as considerações lógicas [essencial] e físicas [de implementação]...
No mínimo, precisamos de...
- Algo que nos ajude a particionar nossos requisitos e documentar esse particionamento antes da especificação...
- Algum modo de fazer o rastreo e avaliar as interfaces...
- Novas ferramentas para descrever lógica e política, algo melhor do que texto narrativo...

Não há provavelmente outro método de engenharia de software que tenha gerado tanto interesse, sido experimentalmente (e frequentemente rejeitado e depois experimentalmente novamente) por tantas pessoas, provocado tantas críticas e despertado tanta controvérsia. Mas o método prosperou e ganhou um número substancial de seguidores na comunidade de engenharia de software.

12.1 UM BREVE HISTÓRICO

Citação

"O problema não é haver problemas. O problema é querer o contrário e pensar que ter problemas é um problema."
Theodore Rubin

Como muitas contribuições importantes à engenharia de software, a análise estruturada não foi introduzida com um único trabalho marcante ou livro. O trabalho inicial na modelagem da análise começou no final dos anos 60 e no início dos anos 70, mas o primeiro aparecimento da abordagem de análise estruturada foi em conjunto com outro tópico importante — "projeto estruturado". Pesquisadores (p. ex., [STE74] e [YOU78]) precisavam de uma notação gráfica para representar dados e os processos que os transformavam. Esses processos iriam, em última análise, corresponder a uma arquitetura de projeto.

O termo *análise estruturada*, criado originalmente por Douglas Ross, foi popularizado por DeMarco [DEM79]. Em seu livro sobre o assunto, DeMarco introduziu e nomeou os símbolos gráficos-chave e os modelos que os incorporaram. Nos anos seguintes, variações da abordagem de análise estruturada foram sugeridas por Page-Jones [PAG80], Gane e Sarson [GAN82], e muitos outros. Em todas as instâncias, o método focalizou aplicações de sistemas de informação e não forneceu notação adequada para cuidar dos aspectos de controle e comportamento de problemas de engenharia de tempo real.

Na metade dos anos 80, "extensões" para tempo real foram introduzidas por Ward e Mellor [WAR85] e depois por Hatley e Pirbhai [HAT87]. Essas extensões resultaram num método de análise mais robusto que podia ser aplicado efetivamente a problemas de engenharia. Tentativas de desenvolver uma notação consistente foram sugeridas [BRU88] e tratamentos modernizados para acomodar o uso de ferramentas CASE foram publicados [YOU89].

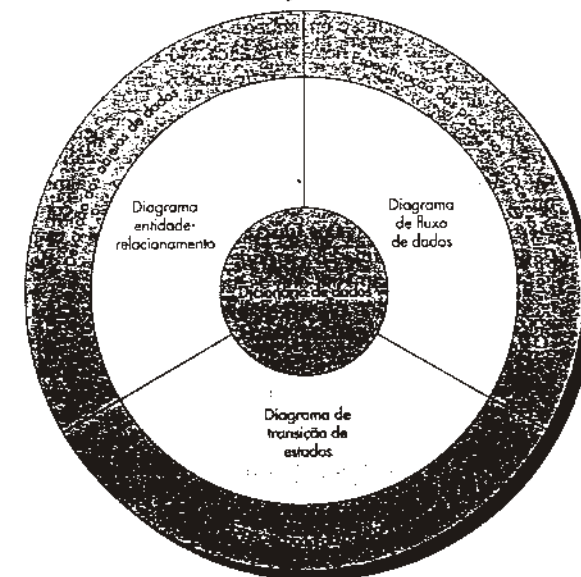
12.2 OS ELEMENTOS DO MODELO DE ANÁLISE

O modelo de análise deve atingir três objetivos principais: (1) descrever o que o cliente exige, (2) estabelecer a base para a criação de um projeto de software e (3) definir um conjunto de requisitos que possam ser validados quando o software for construído. Para conseguir esses objetivos, o modelo de análise derivado durante a análise estruturada torna a forma ilustrada na Fig. 12.1.

No núcleo do modelo fica o *dicionário de dados* — um repositório que contém descrições de todos os objetos de dados consumidos ou produzidos pelo software. Três diagramas diferentes envolvem o núcleo. O *diagrama entidade-relacionamento* (*entity relation diagram*, ERD) mostra as relações entre os objetos de dados. O ERD é a notação usada para conduzir a atividade de modelagem de dados. Os atributos de cada objeto de dados que figura no ERD podem ser descritos usando uma descrição de objeto de dados.

O *diagrama de fluxo de dados* (*data flow diagram*, DFD) serve a duas finalidades: (1) fornecer indicação de como os dados são transformados à medida que se movem através do sistema e (2) mostrar as funções (e subfunções) que transformam o fluxo de dados. O DFD fornece informação adicional, que é usada durante a análise de domínio de informação e serve como base para a modelagem de funções. Uma

Fig. 12.1 Estrutura do modelo de análise.



descrição de cada função apresentada no DFD é incluída numa *especificação de processos* (*process specification*, PSPEC).

O *diagrama de transição de estados* (*state transition diagram*, STD) indica como o sistema se comporta em consequência de eventos externos. Para conseguir isso, o STD representa os vários modos de comportamento (chamados *estados*) do sistema e a maneira pela qual transições de estado para estado são feitas. O STD serve de base para a modelagem comportamental. Informação adicional sobre os aspectos de controle do software é incluída na *especificação de controle* (*control specification*, CSPEC).

O modelo de análise abrange cada um dos diagramas, especificações, descrições e o dicionário mostrado na Fig. 12.1. Uma discussão mais detalhada desses elementos do modelo de análise é apresentada nas seções seguintes.

12.3 MODELAGEM DE DADOS

Que questões a modelagem de dados responde?

Citação

"A força da abordagem ER é a sua capacidade de descrever entidades do mundo real do negócio e as relações entre elas."
Martin Modell

A modelagem de dados responde a um conjunto de questões específicas que são relevantes para qualquer aplicação de processamento de dados. Quais são os principais objetos de dados a serem processados pelo sistema? Qual a composição de cada objeto de dados e que atributos descrevem o objeto? Onde os objetos costumam ficar? Quais são as relações entre cada objeto e os demais? Quais são as relações entre os objetos e os processos que os transformam?

Para responder a essas questões, os métodos de modelagem de dados fazem uso do diagrama entidade-relacionamento. O ERD, descrito em detalhes mais adiante nesta seção, permite ao engenheiro de software identificar objetos de dados e suas relações usando uma notação gráfica. No contexto da análise estruturada, o ERD define todos os dados que são introduzidos, armazenados, transformados e produzidos em uma aplicação.

O diagrama entidade-relacionamento focaliza apenas os dados (e satisfaz assim o primeiro princípio operacional da análise), representando uma "rede de dados" que existe para um determinado sistema. O ERD é especialmente útil para aplicações nas quais os dados e as relações que dirigem os dados são complexos. Diferentemente do diagrama de fluxo de dados (discutido na Seção 12.4 e usado para representar como os dados são transformados), a modelagem de dados considera os dados independentemente do processamento que os transforma.

12.3.1 Objetos de dados, atributos e relações

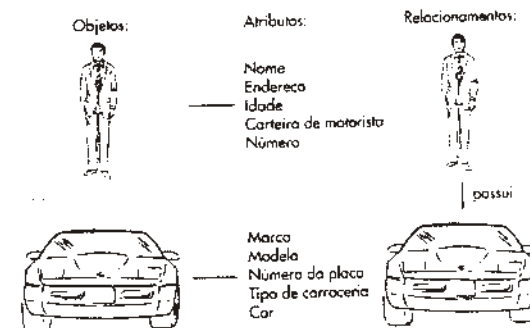
O modelo de dados consiste em três peças de informação inter-relacionadas: o objeto de dados, os atributos que descrevem o objeto de dados e as relações que conectam os objetos de dados uns aos outros.

Objetos de dados. Um *objeto de dados* é uma representação de quase toda a informação composta que deve ser compreendida pelo software. Por *informação composta*, queremos nos referir a algo que tem várias propriedades ou atributos diferentes. Assim, largura (um simples valor) não seria um objeto de dados válido, mas dimensões (incorporando altura, largura e profundidade) poderia ser definido como objeto.

Ponto
CHAVE

Um objeto de dados é uma representação de qualquer informação composta que é processada pelo software de computador.

Fig. 12.2 Objetos de dados, atributos e relacionamentos.



Um objeto de dados pode ser uma entidade externa (p. ex., qualquer coisa que produza ou consuma informação), uma coisa (p. ex., um relatório ou um mostrador), uma ocorrência (p. ex., uma chamada telefônica) ou um evento (p. ex., um alarme), um papel (p. ex., vendedor), uma unidade organizacional (p. ex., departamento de contabilidade), um lugar (p. ex., um depósito), ou uma estrutura (p. ex., um arquivo). Por exemplo, uma pessoa ou um automóvel (Fig. 12.2) pode ser visto como um objeto de dados no sentido de que ambos podem ser definidos em termos de um conjunto de atributos. A descrição do objeto de dados incorpora o objeto de dados e todos os seus atributos.

Objetos de dados (representados em negrito) são relacionados uns aos outros. Por exemplo, **pessoa** pode *possuir* **automóvel**, em que o relacionamento *possuir* conota uma "conexão" específica entre **pessoa** e **automóvel**. Os relacionamentos são sempre definidos pelo contexto do problema que está sendo analisado.

Um objeto de dados encapsula apenas dados — não há referência dentro de um objeto de dados a operação que age sobre os dados.¹ Assim, o objeto de dados pode ser representado por uma tabela, como mostrado na Fig. 12.3. Os títulos das colunas da tabela refletem os atributos do objeto. Nesse caso, um automóvel é definido em termos de marca, modelo, número da placa, tipo de carroceria, cor e proprietário. O corpo da tabela representa instâncias específicas do objeto de dados. Por exemplo, Chevy Corvette é um exemplo do objeto de dados automóvel.

Atributos. Os atributos definem as propriedades de um objeto de dados e assumem uma de três diferentes características. Podem ser usados para (1) nomear um exemplo do objeto de dados, (2) descrever o exemplo ou (3) fazer referência a outro exemplo em outra tabela. Além disso, um ou mais dos atributos podem ser definidos como **identificador** — isto é, o atributo identificador toma-se uma "chave" quando desejamos encontrar um exemplo do objeto de dados. Em alguns casos, valores do identificador(s) são únicos, apesar disso não ser exigido. Com referência ao objeto de dados **automóvel**, um identificador razoável poderia ser o número de identificação.

1 Essa distinção separa o objeto de dados da *classe* ou *objeto* definido como parte do paradigma orientado a objetos, discutido na Parte Quatro deste livro.

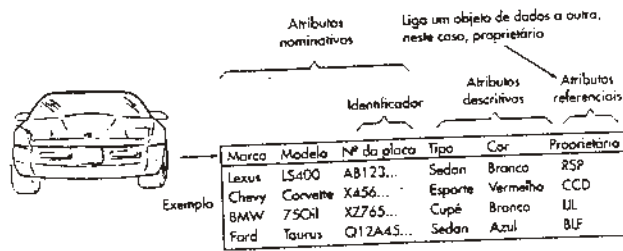
Na Web

Informação útil sobre modelagem de dados pode ser encontrada em www.datamodel.org

Ponto
CHAVE

Os atributos nomeiam um objeto de dados, descrevem suas características e, em alguns casos, fazem referência a outro objeto.

Fig. 12.3
Representação tabular
de objetos de dados.



O conjunto de atributos que é adequado para um determinado objeto de dados é determinado entendendo-se o contexto do problema. Os atributos de automóvel poderiam servir bem para uma aplicação que seria usada pelo departamento de veículos automotores, mas esses atributos seriam inúteis para uma empresa de automóveis que necessita de software para controle de fabricação. Nesse último caso, os atributos para automóvel poderiam também incluir número de identificação, tipo de carroceria e cor, mas atributos adicionais (p. ex., código do interior, tipo do conjunto de direção, designador do pacote de opções do acabamento interno e tipo de transmissão) teriam que ser adicionados para tornar automóvel um objeto significativo no contexto do controle de fabricação.

Relacionamentos. Objetos de dados são conectados uns aos outros de diferentes modos. Considere dois objetos de dados, *livro* e *livraria*. Esses objetos podem ser representados usando a notação simples ilustrada na Fig. 12.4a. Uma conexão é estabelecida entre *livro* e *livraria* porque os dois objetos são relacionados. Mas quais são os relacionamentos? Para determinar a resposta, precisamos entender o papel de livros e livrarias no contexto do software a ser construído. Podemos definir um conjunto de pares objeto/relacionamento que define os relacionamentos relevantes. Por exemplo,

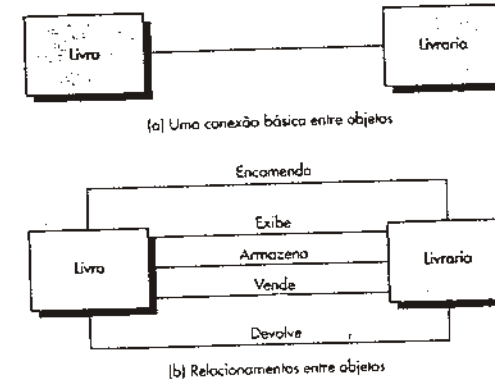
- Uma livraria encomenda livros.
- Uma livraria exibe livros.
- Uma livraria armazena livros.
- Uma livraria vende livros.
- Uma livraria devolve livros.

Os relacionamentos *encomenda*, *exibe*, *armazena*, *vende* e *devolve* definem as conexões relevantes entre *livro* e *livraria*. A Fig. 12.4b ilustra esses pares objeto/relacionamento graficamente.

É importante notar que os pares objeto/relacionamento são bidirecionais. Isto é, podem ser lidos em ambas as direções. Uma livraria encomenda livros ou livros são encomendados por uma livraria.²

2 Para evitar a ambigüidade, deve ser considerado o modo pelo qual uma relação é rotulada. Por exemplo, se o contexto não é considerado para uma relação bidirecional, a Fig. 12.4b poderia ser mal-interpretada, significando que livros encomendam livrarias. Em tais casos, é necessário reformular os termos.

Fig. 12.4
Relacionamentos.



12.3.2 Cardinalidade e modalidade

Os elementos da modelagem de dados — objetos de dados, atributos e relacionamentos — fornecem a base para o entendimento do domínio de informação de um problema. No entanto, informação adicional relacionada a esses elementos básicos deve também ser entendida.

Definimos um conjunto de objetos e representamos os pares objeto/relacionamento que os ligam. Mas um simples par que declara: *objeto X está relacionado a objeto Y* não fornece informação suficiente para os fins da engenharia de software. Devemos entender quantas ocorrências do *objeto X* são relacionadas a quantas ocorrências do *objeto Y*. Isso leva a um conceito de modelagem de dados chamado *cardinalidade*.

Cardinalidade. O modelo de dados deve ser capaz de representar o número de ocorrências dos objetos numa dada relação. Tillmann [TIL93] define *cardinalidade* de um par objeto/relacionamento do seguinte modo:

Cardinalidade é a especificação do número de ocorrências de um [objeto] que pode ser relacionado ao número de ocorrências de outro [objeto]. Cardinalidade é usualmente expressa como simplesmente 'um' ou 'muitos'. Por exemplo, um marido pode ter apenas uma esposa (na maioria das culturas), enquanto um progenitor pode ter muitos filhos. Levando em consideração todas as combinações de 'um' e 'muitos', dois [objetos] podem ser relacionados como

- Um para um (1:1) — Uma ocorrência de [objeto] 'A' pode ser relacionada a uma e somente uma ocorrência de [objeto] 'B', e uma ocorrência de 'B' pode se relacionar a apenas uma ocorrência de 'A'.
- Um para muitos (1:N) — Uma ocorrência de [objeto] 'A' pode se relacionar a uma ou mais ocorrências de [objeto] 'B', mas uma ocorrência de 'B' pode se relacionar a apenas uma ocorrência de 'A'. Por exemplo, uma mãe pode ter muitos filhos, mas um filho pode ter apenas uma mãe.

- Muitos para muitos (M:N) — Uma ocorrência de [objeto] 'A' pode se relacionar a uma ou mais ocorrências de 'B', enquanto uma ocorrência de 'B' pode se relacionar a uma ou mais ocorrências de 'A'. Por exemplo, um tio pode ter muitos sobrinhos, enquanto um sobrinho pode ter muitos tios.

Cardinalidade define "o número máximo de objetos que podem participar de um relacionamento" [TIL93]. Todavia, não indica se um objeto de dados particular deve ou não participar de um relacionamento. Para especificar essa informação, o modelo de dados acrescenta modalidade ao par objeto/relação.

Modalidade. A *modalidade* de uma relação é 0 se não há necessidade explícita de um relacionamento ocorrer ou se o relacionamento é opcional. A modalidade é 1 se a ocorrência do relacionamento for obrigatória. Para ilustrar, considere o software que é usado por uma empresa telefônica local para processar pedidos de serviço no campo. Um cliente indica que há um problema. Se o problema é diagnosticado como relativamente simples, uma única ação de reparo ocorre. No entanto, se o problema é complexo, múltiplas ações de reparo podem ser necessárias. A Fig. 12.5 ilustra o relacionamento, cardinalidade e modalidade entre os objetos de dados cliente e ação de reparo.

Observando a figura, uma relação de cardinalidade de um para muitos é estabelecida. Isto é, um único cliente pode ser atendido com zero ou muitas ações de reparo. Os símbolos na conexão de relação, mais próximos dos retângulos dos objetos de dados, indicam cardinalidade. A barra vertical indica um e o garfo de três dentes indica muitos. Modalidade é indicada pelos símbolos que estão mais longe dos retângulos dos objetos de dados. A segunda barra vertical à esquerda indica que precisa haver um cliente para que uma ação de reparo ocorra. O círculo à direita indica que pode não haver necessidade de ação de reparo para o tipo de problema relatado pelo cliente.

12.3.3 Diagramas entidade/relacionamento

O par objeto/relacionamento (discutido na Seção 12.3.1) é a pedra fundamental do modelo de dados. Esses pares podem ser representados graficamente usando o *diagrama entidade/relacionamento*. O ERD foi originalmente proposto por Peter Chen [CHE77] para o projeto de sistemas de base de dados relacional e foi estendido por

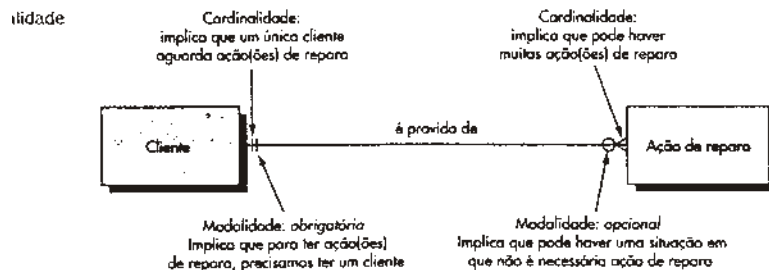
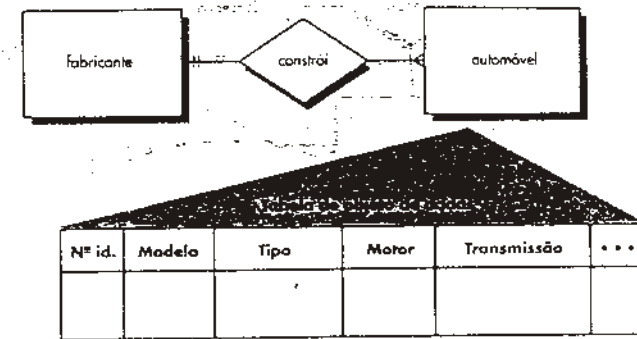


Fig. 12.6 Um ERD simples e tabela de objetos de dados (Nota: neste ERD a relação *constrói* é indicada por um losango).



Ponto CHAVE

A finalidade principal do ERD é representar entidades (objetos de dados) e seus relacionamentos.

outros. Um conjunto de componentes primordiais é identificado para o ERD: objetos de dados, atributos, relacionamentos e indicadores de vários tipos. A finalidade principal do ERD é representar objetos de dados e seus relacionamentos.

Uma notação rudimentar do ERD já foi introduzida na Seção 12.3.1. Objetos de dados são representados por um retângulo rotulado. Relacionamentos são indicados por uma linha rotulada conectando objetos. Em algumas variantes do ERD, a linha de conexão contém um losango, que é rotulado com um relacionamento. Conexões entre objetos de dados e relacionamentos são estabelecidas usando diversos símbolos especiais que indicam cardinalidade e modalidade (Seção 12.3.2).

O relacionamento entre os objetos de dados **automóvel** e **fabricante** seria representado como mostrado na Fig. 12.6. Um fabricante fabrica um ou vários automóveis. Considerando o contexto implícito no ERD, a especificação do objeto de dados **automóvel** (tabela de objeto de dados na Fig. 12.6) seria radicalmente diferente da especificação anterior (Fig. 12.3). Examinando os símbolos nas extremidades da linha de conexão entre os objetos, pode ser visto que a modalidade de ambas as ocorrências é obrigatória (as linhas verticais).

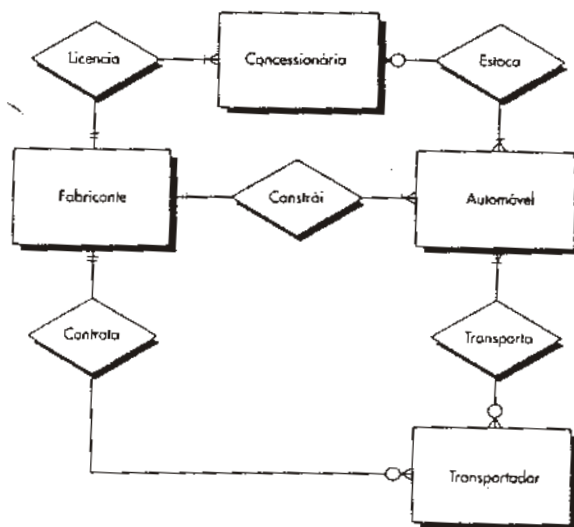
Expandindo o modelo, representamos um ERD grosseiramente supersimplificado (Fig. 12.7) do elemento distribuição do negócio de automóveis. Novos objetos de dados, **transportador** e **concessionária**, são introduzidos. Além disso, novos relacionamentos — **transporta**, **contrata**, **licencia** e **estoca** — indicam como os objetos de dados mostrados na figura se associam uns aos outros. Tabelas para cada um dos objetos de dados contidos no ERD teriam que ser desenvolvidas de acordo com as regras introduzidas anteriormente neste capítulo.

Além da notação básica de ERD introduzida nas Figs. 12.6 e 12.7, o analista pode representar *hierarquias de tipos de objetos de dados*. Em muitos exemplos, um objeto de dados pode na verdade representar uma classe ou categoria de informação. Por exemplo, o objeto de dados **automóvel** pode ser categorizado como nacional, europeu

SUGESTÃO

Desenvolva o ERD iterativamente refinando tanto os objetos de dados quanto as relações que os conectam.

Fig. 12.7 Um ERD expandido.



ou asiático. A notação ERD mostrada na Fig. 12.8 representa essa categorização na forma de uma hierarquia [ROS85].

A notação ERD também fornece um mecanismo que representa a associatividade entre objetos. Um *objeto de dados associativo* é representado como mostrado na Fig. 12.9. Na figura, cada um dos objetos de dados que modelam os subsistemas individuais é associado com o objeto de dados *automóvel*.

Modelagem de dados e o diagrama entidade-relacionamento fornecem ao analista uma notação concisa para examinar os dados no contexto de uma aplicação de software. Na maioria dos casos, a abordagem de modelagem de dados é usada para criar uma peça do modelo de análise, mas ela também pode ser usada para projeto da base de dados e para apoiar quaisquer outros métodos de análise de requisitos.

12.4 MODELAGEM FUNCIONAL E FLUXO DA INFORMAÇÃO

A informação é transformada à medida que flui através de um sistema baseado em computador. O sistema aceita entrada de diversas formas; aplica hardware, software e elementos humanos para transformá-la e produz saída de diversas formas. A entrada pode ser um sinal de controle transmitido por um transdutor, uma série de números digitados por um operador humano, um pacote de informação transmitido em uma ligação de rede ou um arquivo de dados volumoso recuperado de armazenamento secundário. A(s) transformação(ões) pode(m) compreender uma única comparação lógica, um algoritmo numérico complexo ou uma abordagem de regra de inferência

Fig. 12.8 Hierarquias de tipos de objetos de dados.

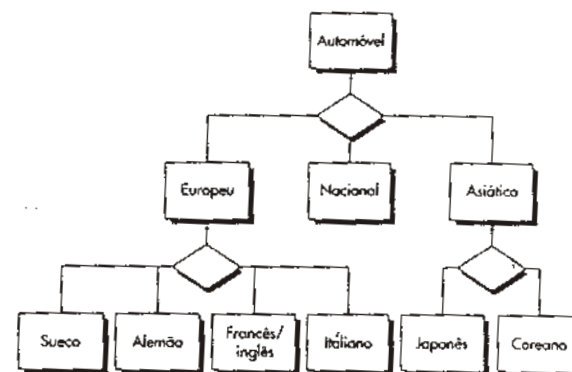
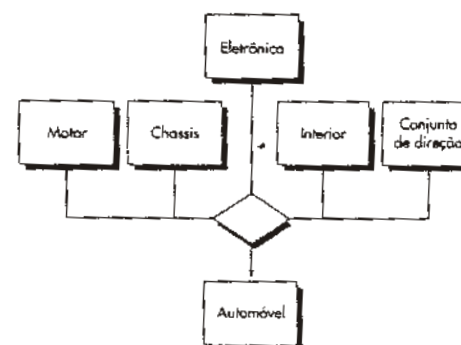


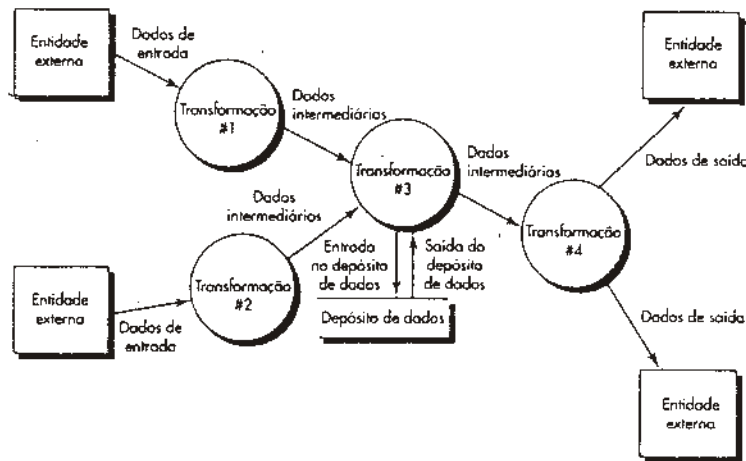
Fig. 12.9 Objetos de dados associativos.



de um sistema especialista. A saída pode acender um único LED ou produzir um relatório de 200 páginas. Com efeito, nós podemos criar um *modelo de fluxo* para qualquer sistema baseado em computador, independentemente do tamanho e da complexidade.

A análise estruturada começou como uma técnica de modelagem do fluxo da informação. Um sistema baseado em computador é representado como uma transformação de informação como mostrado na Fig. 12.10. Um retângulo é usado para representar uma *entidade externa*; isto é, um elemento do sistema (p. ex., hardware, uma pessoa ou outro programa) ou outro sistema que produz informação para transformação pelo software ou recebe informação produzida pelo software. Um círculo (algumas vezes chamado de *bolha*) representa um *processo* ou *transformação* que é aplicado aos dados (ou controle) e os modifica de algum modo. Uma seta representa um ou mais *itens de dados* (objetos de dados). Todas as setas de um diagrama de fluxo de dados devem ser rotuladas. A linha dupla representa um depósito de dados — info

Fig. 12.10 Modelo de fluxo da informação.

**SUGESTÃO**

O DFD não é procedimental. Isto é, não tenta representar processamento condicional ou "loops" com essa forma diagramática. Simplesmente, mostre o fluxo de dados.

**Ponto
CHAVE**

O DFD fornece um mecanismo para a modelagem do fluxo de informação e a modelagem funcional.

mação armazenada que é usada pelo software. A simplicidade da notação DFD é uma razão pela qual as técnicas de análise estruturada são amplamente usadas.

É importante notar que nenhuma indicação explícita da sequência de processamento ou da lógica condicional é fornecida pelo diagrama. Procedimento ou sequência pode estar implícito no diagrama, mas detalhes lógicos explícitos são geralmente adiados até o projeto de software. É importante não confundir DFD com fluxograma.

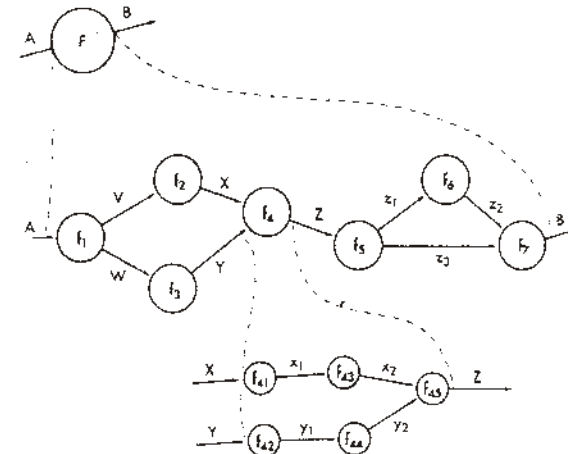
12.4.1 Diagramas de fluxo de dados

À medida que a informação se move através do software, ela é modificada por uma série de transformações. Um diagrama de fluxo de dados é uma representação gráfica que mostra o fluxo da informação e as transformações que são aplicadas à medida que os dados se movem da entrada para a saída. A forma básica de um diagrama de fluxo de dados, também conhecido como gráfico de fluxo de dados ou gráfico de bolhas, está ilustrada na Fig. 12.10.

O diagrama de fluxo de dados pode ser usado para representar um sistema ou software em qualquer nível de abstração. De fato, DFD podem ser particionados em níveis que representam cada vez mais detalhes de fluxo da informação e funcionais. Assim, o DFD fornece um mecanismo para modelagem funcional bem como para modelagem do fluxo da informação. Fazendo isso ele satisfaz o segundo princípio operacional de análise (i. e., criando um modelo funcional) discutido no Cap. 11.

Um DFD 0, também chamado de *modelo fundamental do sistema* ou *modelo de contexto*, representa todo o elemento software como uma única bolha, com dados de entrada e saída indicados respectivamente por setas que chegam e setas que saem. Processos (bolhas) e caminhos de fluxo da informação adicionais são representados à medida que o DFD 0 é particionado para revelar mais detalhes. Por exemplo, um DFD

Fig. 12.11 Refinamento do fluxo da informação.

**SUGESTÃO**

O refinamento de um nível de DFD para o seguinte deve seguir uma relação de aproximadamente 1:5, sendo reduzida à medida que o refinamento prossegue.

SUGESTÃO

Apesar da continuidade do fluxo da informação, precisa ser mantida, reconheça que o item de dados representado em um nível pode ser refinado nas suas partes constituintes no nível seguinte.

nível 1 poderia conter cinco ou seis bolhas com as setas de interconexão. Cada um dos processos representados no nível 1 é uma subfunção do sistema global mostrado no modelo de contexto.

Como mencionamos anteriormente, cada uma das bolhas pode ser refinada ou decomposta em camadas para mostrar mais detalhes. A Fig. 12.11 ilustra esse conceito. Um modelo fundamental para o sistema F indica que a entrada principal é A e a saída principal é B . Nós refinamos o modelo F nas transformações de f_1 a f_7 . Note que a *continuidade do fluxo de informação* deve ser mantida; isto é, a entrada e a saída para cada refinamento deve permanecer a mesma. Esse conceito, algumas vezes chamado de *balanceamento*, é essencial para o desenvolvimento de modelos consistentes. Maior refinamento de f_4 mostra detalhes na forma das transformações f_{41} a f_{45} . Novamente, a entrada (X , Y) e a saída (Z) permanecem inalteradas.

A notação básica usada para desenvolver um DFD não é, em si, suficiente para descrever os requisitos do software. Por exemplo, uma seta mostrada num DFD representa um objeto de dados que é entrada ou saída de um processo. Um depósito de dados representa alguma coleção organizada de dados. Mas qual é o conteúdo dos dados implícito na seta ou representado pelo depósito? Se a seta (ou depósito) representa uma coleção de objetos, quais são eles? Essas questões são respondidas pela aplicação de outro componente da notação básica da análise estruturada — o *dicionário de dados*. O uso do dicionário de dados será discutido posteriormente neste capítulo.

A notação gráfica do DFD pode ser ampliada por um texto descritivo. Uma *especificação de processo* (*process specification*, PSPEC) pode ser usada para especificar os detalhes de processamento implícitos numa bolha de um DFD. A especificação de processo descreve a entrada para uma função, o algoritmo que é aplicado para trans-

formar a entrada e a saída que é produzida. Além disso a PSPEC indica as restrições e limitações impostas ao processo (função), as características de desempenho que são relevantes ao processo e as restrições de projeto que podem influenciar o modo pelo qual o processo será implementado.

12.4.2 Extensões para sistemas de tempo real

Muitas aplicações de software são dependentes de tempo e processam tanto ou mais informação orientada a controle quanto dados. Um sistema de tempo real deve interagir com o mundo real numa base de tempo ditada pelo mundo real. Avionica, controle de processo de fabricação, produtos de consumo e instrumentação industrial são apenas algumas das centenas de aplicações de software de tempo real.

Para acomodar a análise de software de tempo real, algumas extensões da notação básica de análise estruturada foram definidas. Essas extensões, desenvolvidas por Ward e Mellor [WAR85], e Hatley e Pirbhai [HAT87] e ilustradas nas seções seguintes, permitem ao analista representar fluxo de controle e processamento de controle, bem como fluxo e processamento de dados.

12.4.3 Extensões de Ward e Mellor

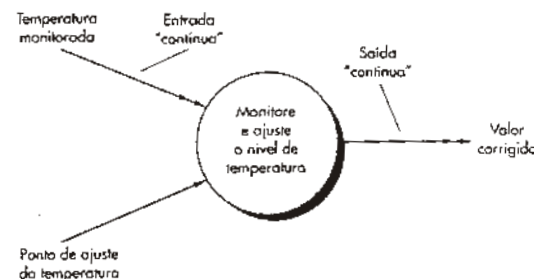
Ward e Mellor [WAR85] estendem a notação básica da análise estruturada para acomodar as seguintes demandas impostas por um sistema de tempo real:

- O fluxo de informação é coletado ou produzido em uma base contínua no tempo.
- Informação de controle é passada através do sistema e do processamento de controle associado.
- Múltiplos exemplos da mesma transformação são algumas vezes encontrados em situações multitarefas.
- Os sistemas possuem estados, e um mecanismo provoca a transição entre estados.

Numa porcentagem significativa de aplicações de tempo real, o sistema precisa monitorar informação contínua no tempo gerada por algum processo do mundo real. Por exemplo, um sistema de tempo real, de monitoração de teste para motores de turbina a gás, pode ser necessário para monitorar a velocidade da turbina, a temperatura de combustão e diversas amostras de pressão em base contínua. A notação convencional de fluxo de dados não faz distinção entre dados discretos e dados contínuos no tempo. Uma extensão da notação básica de análise estruturada, mostrada na Fig. 12.12, fornece um mecanismo para representar fluxo de dados contínuos no tempo. A seta com dupla ponta é usada para representar fluxo de dados contínuos no tempo, enquanto uma seta com uma única ponta é usada para indicar fluxo de dados discreto. Na figura, *temperatura monitorada* é medida continuamente, enquanto um único valor do *ponto de ajuste da temperatura* também é fornecido. O processo mostrado na figura produz uma saída contínua em relação ao tempo, *valor corrigido*.

A distinção entre fluxo de dados discreto e contínuo no tempo tem implicações importantes tanto para o engenheiro de sistemas quanto para o projetista de software. Durante a criação do modelo do sistema, um engenheiro de sistemas estará mais apto

Fig. 12.12 Fluxo de dados contínuos no tempo.



a isolar os processos que podem ser críticos em relação ao desempenho (é frequentemente provável que a entrada e a saída de dados contínuos no tempo sejam sensíveis ao desempenho). Quando o modelo físico ou de implementação é criado, o projetista precisa estabelecer um mecanismo para coleta de dados contínuos no tempo. Obviamente, o sistema digital coleta dados de um modo quase-contínuo usando técnicas tais como consulta de alta velocidade. A notação indica onde será necessário hardware analógico-para-digital e que transformações vão provavelmente exigir software de alto desempenho.

Em diagramas de fluxo de dados convencionais, fluxos de controle ou eventos não são explicitamente representados. De fato, o engenheiro de software é prevenido para excluir especificamente a representação de fluxo de controle do diagrama de fluxo de dados. Essa exclusão é demasiada restritiva quando são consideradas aplicações de tempo real e, por essa razão, uma notação especializada para representar fluxo de eventos e processamento de controle foi desenvolvida. Dando continuidade à convenção estabelecida para os diagramas de fluxo de dados, o fluxo de dados é representado usando uma seta sólida. O *fluxo de controle*, no entanto, é representado usando uma seta tracejada ou sombreada. Um processo que trata apenas fluxos de controle, chamado de *processo de controle*, é analogamente representado usando uma bolha tracejada.

Fluxo de controle pode ser direcionado diretamente para um processo convencional ou para um processo de controle. A Fig. 12.13 ilustra fluxo de controle e processamento como seriam representados usando a notação de Ward e Mellor. A figura ilustra uma visão de alto nível do fluxo de dados e de controle para uma célula de fabricação. À medida que os componentes a serem montados por um robô são colocados em dispositivos, um bit de *status* é ajustado num *buffer de status das peças* (depósito de controle) que indica a presença ou a ausência de cada componente. A informação de evento contida no *buffer de status das peças* é passada como uma cadeia de bits para um processo, *monitorar interface com dispositivos e operador*. O processo vai ler comandos do operador apenas quando a informação de controle, cadeia de bits, indicar que todos os dispositivos contêm componentes. Um sinal de evento, *sinal de partida/parada*, é enviado a *controlar iniciação do robô*, um processo de controle que habilita o processamento de comandos posterior. Outros fluxos de dados ocorrem como consequência do evento *ativação do processo* que é enviado a *processar comandos do robô*.

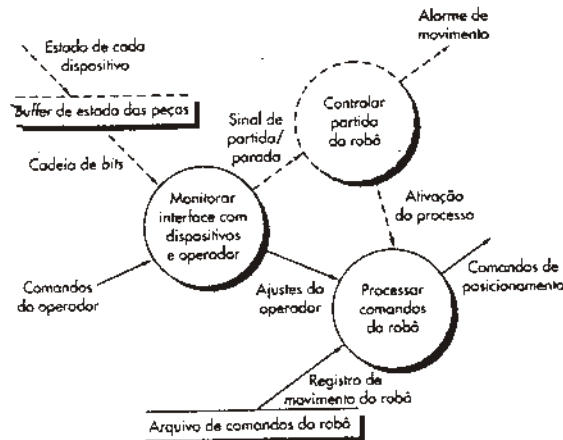
Ponto CHAVE

Para modelar adequadamente um sistema de tempo real, a notação de análise estruturada precisa estar disponível para o processamento de dados e eventos contínuos no tempo.

Citação

"O ambiente de um sistema de tempo real frequentemente contém dispositivos que agem como os sentidos do sistema."
Paul Ward e Stephen Mellor

Fig. 12.13 Fluxos de dados e controle usando a notação de Ward e Mellor [WAR85].



Em algumas situações, múltiplos exemplos do mesmo processo de transformação de controle ou dados podem ocorrer num sistema de tempo real. Isso pode acontecer num ambiente multitarefa, quando tarefas são criadas como resultado de processamento interno ou de eventos externos. Por exemplo, alguns *buffers* de estado de peças podem ser monitorados de modo que diferentes robôs possam ser sinalizados no devido tempo. Além disso, cada robô pode ter seu próprio sistema de controle de robô. A notação de Ward e Mellor costumava representar *múltiplos exemplos equivalentes* simplesmente sobrepondo bolhas de processo (ou processos de controle) para indicar multiplicidade.

12.4.4 Extensões de Hatley e Pirbhai

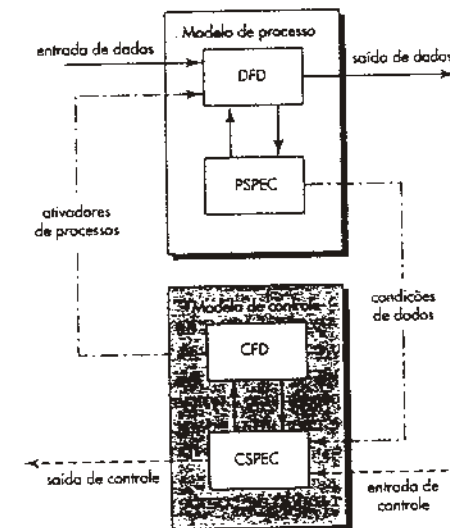
As extensões de Hatley e Pirbhai [HAT87] para a notação básica da análise estruturada focalizam menos a criação de símbolos gráficos adicionais e mais a representação e especificação dos aspectos orientados a controle do software. A seta tracejada é mais uma vez usada para representar fluxo de controle ou de evento. Diferente de Ward e Mellor, Hatley e Pirbhai sugerem que a notação tracejada e a sólida sejam representadas separadamente. Assim, um *diagrama de fluxo de controle* (*control flow diagram*, CFD) é definido. O CFD contém os mesmos processos que o DFD, mas mostra fluxo de controle, ao invés de fluxo de dados. Em vez de representar processos de controle diretamente no modelo de fluxo, uma referência notacional (uma barra sólida) para uma *especificação de controle* (*control specification*, CSPEC) é usada. Em essência, a barra sólida pode ser vista como uma "janela" para um "executivo" (a CSPEC), que controla os processos (funções) representados no DFD, com base no evento que é passado através da janela. A CSPEC, descrita em detalhe na Seção 12.6.4, é usada para indicar (1) como o software se comporta quando um evento ou sinal de controle é percebido e (2) quais processos são invocados como consequência da

ocorrência do evento. Uma especificação de processo é usada para descrever o trabalho interno de um processo representado num diagrama de fluxo.

Usando a notação descrita nas Figs. 12.12 e 12.13, em conjunto com a informação adicional contida em PSPEC e CSPEC, Hatley e Pirbhai criam um modelo de sistemas de tempo real. Diagramas de fluxo de dados são usados para representar dados e os processos que os tratam. Diagramas de fluxo de controle mostram como os eventos fluem entre processos e ilustram os eventos externos que causam a ativação de vários processos. A inter-relação entre os modelos de processo e de controle é mostrada esquematicamente na Fig. 12.14. O modelo de processo é "conectado" ao modelo de controle por condições de dados. O modelo de controle é "conectado" ao modelo de processo através de informação de ativação de processo contida na CSPEC.

Uma *condição de dados* ocorre sempre que entrada de dados em um processo resulta em saída de controle. Essa situação é ilustrada na Fig. 12.15, parte de um modelo de fluxo para um sistema automatizado de monitoração e controle para tanques de pressão numa refinaria de petróleo. O processo *checar e converter pressão absoluta* implementa o algoritmo descrito no pseudocódigo da PSPEC mostrada. Quando a *pressão absoluta do tanque* é maior que um máximo permitido, um evento *acima da pressão* é gerado. Note que quando a notação de Hatley e Pirbhai é usada, o fluxo de dados é mostrado como parte de um DFD, enquanto o fluxo de controle é mostrado separadamente como parte de um diagrama de fluxo de controle. Como mencionamos anteriormente, a barra vertical sólida para a qual o evento *acima da pressão* flui é um ponteiro para a CSPEC. Assim, para determinar o que acontece quando esse evento ocorre, precisamos verificar a CSPEC.

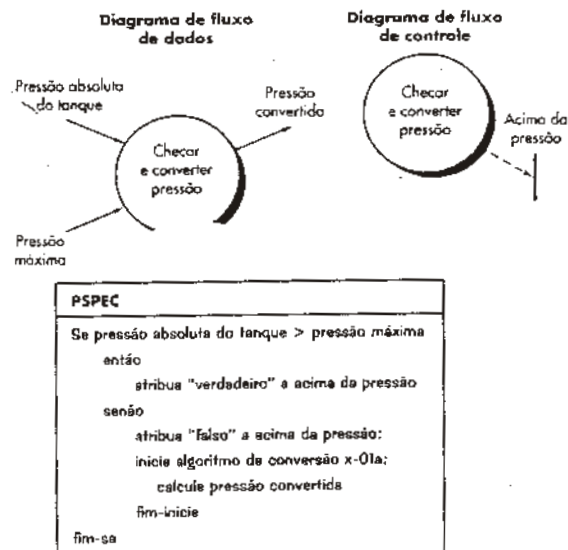
Fig. 12.14 A relação entre os modelos de dados e de controle [HAT87].



Ponto CHAVE

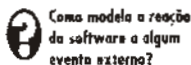
O CFD mostra como os eventos se movem através do sistema. O CSPEC indica como o software se comporta como consequência de eventos e quais processos entram em jogo para gerir eventos.

Fig. 12.15 Condições de dados.



A especificação de controle (CSPEC) contém algumas ferramentas de modelagem importantes. Uma *tabela de ativação de processos* (descrita na Seção 12.6.4) é usada para indicar que processos são ativados por um determinado evento. Por exemplo, uma tabela de ativação de processos (*process activation table*, PAT) para a Fig. 12.15 poderia indicar que o evento **acima da pressão** iria fazer com que o processo *reduzir a pressão do tanque* (não-mostrado) fosse invocado. Além da PAT, a CSPEC pode conter um diagrama de transição de estados. O STD é um modelo de comportamento que se apoia na definição de um conjunto de estados do sistema e será descrito na seção seguinte.

12.5 MODELAGEM COMPORTAMENTAL

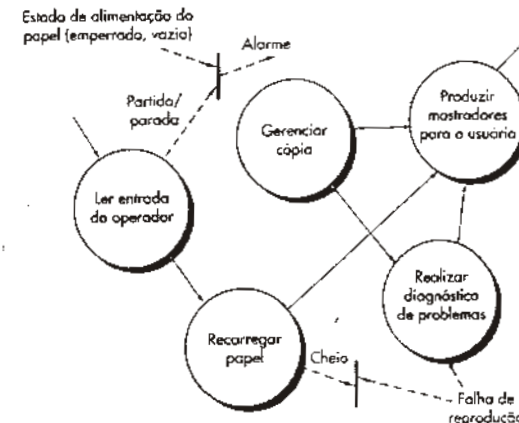


Como modela a reação do software a algum evento externo?

Modelagem comportamental é um princípio operacional para todos os métodos de análise de requisitos. No entanto, apenas versões estendidas da análise estruturada ([WAR85], [HAT87]) fornecem notação para esse tipo de modelagem. O diagrama de transição de estados representa o comportamento de um sistema mostrando seus estados e os eventos que causam mudança de estado. Além disso, o STD indica que ações (p. ex., ativação de processo) são tomadas em consequência de um determinado evento.

Um estado é qualquer modo de comportamento observável. Por exemplo, os estados de um sistema de monitoração e controle de tanques de pressão descrito na Seção

Fig. 12.16 CFD nível 1 para software de fotocopiadora.



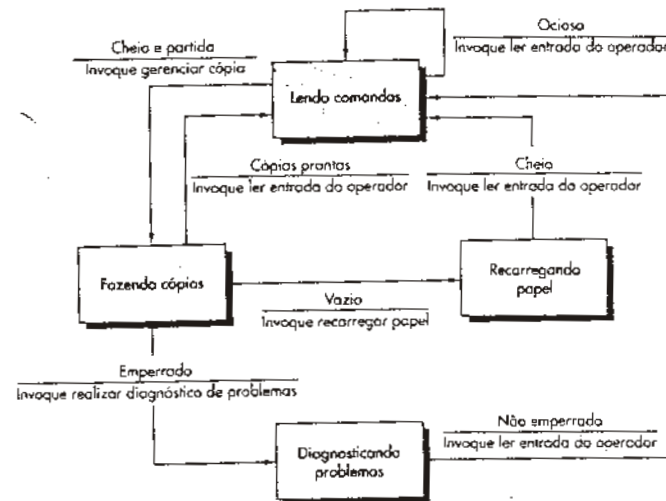
12.4.4 poderiam ser *estado de monitoração*, *estado de alarme*, *estado de alívio de pressão*, e assim por diante. Cada um desses estados representa um modo de comportamento do sistema. Um diagrama de transição de estados indica como o sistema vai de estado a estado.

Para ilustrar o uso das extensões de controle e comportamento de Hatley e Pirbhai, considere o software embutido em uma máquina de fotocópia de escritório. Uma representação simplificada do fluxo de controle para o software da fotocopiadora é mostrado na Fig. 12.16. As setas de fluxo de dados foram sombreadas ligeiramente para fins ilustrativos, mas na realidade elas não são mostradas como parte de um diagrama de fluxo de controle.

Fluxos de controle são mostrados entrando e saindo de processos individuais, e da barra vertical representando a "janela" da CSPEC. Por exemplo, os eventos **estado de alimentação do papel** e **partida/parada** fluem para a barra CSPEC. Isso implica que cada um desses eventos vai causar a ativação de algum processo representado no CFD. Se nós examinássemos o interior da CSPEC, o evento **parada/partida** seria mostrado como ativador/desativador do processo *gerenciar cópia*. Analogamente o evento **emperrado** (parte do estado de alimentação do papel) iria ativar *realizar diagnóstico de problema*. Deve-se notar que todas as barras verticais de um CFD referem-se à mesma CSPEC. Um fluxo de evento pode entrar diretamente num processo, como mostrado com **falha de reprodução**. No entanto, esse fluxo não ativa o processo, mas em vez disso fornece informação de controle para o algoritmo do processo.

Um diagrama de transição de estado simplificado para o software da fotocopiadora é mostrado na Fig. 12.17. Os retângulos representam estados do sistema e as setas representam transições entre estados. Cada seta é rotulada com uma expressão regada. O valor superior indica o(s) evento(s) que causa(m) a ocorrência da transição. O

Fig. 12.17 Diagrama de transição de estados para software de fotocopiadora.



Que passos são necessários para construir um ERD?

atributos que definem as propriedades desses objetos e suas relações. Como a maioria dos elementos do modelo de análise, o ERD é construído de modo iterativo. A seguinte abordagem é adotada:

1. Durante a elicitação dos requisitos, os clientes são solicitados a listar as "coisas" com que a aplicação ou processo de negócio lida. Essas "coisas" evoluem para uma lista de objetos de dados de entrada e saída bem como de entidades externas que produzem ou consomem informação.
2. Tomando os objetos, um de cada vez, o analista e o cliente definem se existe ou não uma conexão (sem nome nesse estágio) entre o objeto de dados e outros objetos.
3. Se existe uma conexão, o analista e o cliente criam um ou mais pares objeto/relacionamento.
4. Para cada par objeto/relacionamento, são exploradas cardinalidade e modalidade.
5. Os passos de 2 a 4 são continuados iterativamente até que todos os objetos/relacionamento tenham sido definidos. É comum descobrir omissões à medida que esse processo continua. Novos objetos e relacionamentos serão invariavelmente adicionados à medida que o número de iterações cresce.
6. Os atributos de cada entidade são definidos.
7. Um diagrama entidade/relacionamento é formalizado e revisado.
8. Os passos de 1 a 7 são repetidos até que a modelagem de dados seja completada.

Para ilustrar o uso dessas diretrizes básicas, o sistema exemplo de segurança, *SafeHome*, discutido no Cap. 11, será usado. Voltando à narrativa do processamento para o *SafeHome* (Seção 11.3.3), a seguinte lista (parcial) de "coisas" é relevante para o problema:

- proprietário
- painel de controle
- sensores
- sistema de segurança
- serviço de monitoração

Tomando essas "coisas" uma de cada vez, as conexões são exploradas. Para conseguir isso, cada objeto é desenhado e linhas ligando os objetos são traçadas. Por exemplo, observando a Fig. 12.18, existe uma conexão direta entre **proprietário** e **painel de controle**, **sistema de segurança** e **serviço de monitoração**. Existe uma conexão simples entre sensor e sistema de segurança, e assim por diante.

Uma vez definidas todas as conexões, um ou mais pares objeto/relacionamento são identificados para cada conexão. Por exemplo, a conexão entre **sensor** e **sistema de segurança** é determinada como tendo os seguintes pares objeto/relacionamento:

- sistema de segurança *monitora* sensor
- sistema de segurança *habilita/desabilita* sensor
- sistema de segurança *testa* sensor
- sistema de segurança *programa* sensor

Cada um desses pares objeto/relacionamento é analisado para determinar cardinalidade e modalidade. Por exemplo, considerando o par objeto/relacionamento **sistema**

Citação

"A única coisa que falta é um estado de confusão."
Um revisor diante do esboço por um STD extremamente complexo.

12.6 O MECANISMO DA ANÁLISE ESTRUTURADA

SUGESTÃO

O modelo de análise permite a um revisor examinar os requisitos de software sob três diferentes pontos de vista. Assim, certifique-se de usar ERD, DFD e STD quando construir a modelo.

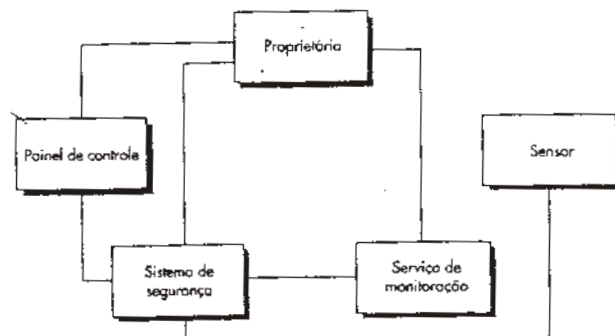
Na seção anterior discutimos a notação básica e estendida para a análise estruturada. Para ser usada efetivamente na análise de requisitos de software, essa notação deve ser combinada com um conjunto de heurísticas que permitem ao engenheiro de software derivar um bom modelo de análise. Para ilustrar o uso dessas heurísticas, uma versão adaptada das extensões de Hatley e Pirbhai [HAT87] à notação básica de análise estruturada será usada ao longo do restante deste capítulo.

Nas seções seguintes, examinamos cada um dos passos que devem ser aplicados para desenvolver modelos completos e precisos usando a análise estruturada. Durante essa discussão, a notação introduzida na Seção 12.4 será usada e outras formas de notação mencionadas anteriormente serão apresentadas com algum detalhe.

12.6.1 Criação de um diagrama entidade/relacionamento

O diagrama entidade/relacionamento permite ao engenheiro de software especificar completamente os objetos de dados que são entrada e saída de um sistema, os

Fig. 12.18
Estabelecimento de
conexões.



de segurança *monitora* sensor, a cardinalidade entre sistema de segurança e sensor é um para muitos. A modalidade é uma ocorrência de sistema de segurança (obrigatória) e pelo menos uma ocorrência de sensor (obrigatória). Usando a notação ERD, introduzida na Seção 12.3, a linha de conexão entre sistema de segurança e sensor seria modificada como mostrado na Fig. 12.19. Análise semelhante seria aplicada a todos os outros objetos de dados.

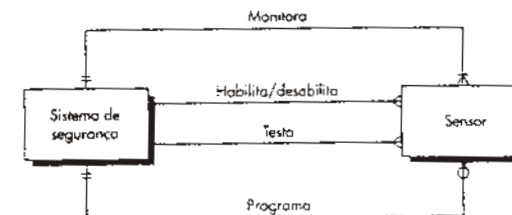
Cada objeto é estudado para determinar seus atributos. Como estamos considerando o software que deve dar apoio ao *SafeHome*, os atributos devem focalizar os dados que precisam ser armazenados para permitir a operação no sistema. Por exemplo, o objeto sensor poderia ter os seguintes atributos: tipo de sensor, número de identificação interno, zona de localização e nível de alarme.

12.6.2 Criação de um modelo de fluxo de dados

O diagrama de fluxo de dados permite ao engenheiro de software desenvolver modelos do domínio informacional e do domínio funcional ao mesmo tempo. À medida que o DFD é refinado em maior nível de detalhe, o analista realiza uma decomposição funcional implícita do sistema, cumprindo assim o quarto princípio operacional de análise para função. Ao mesmo tempo, o refinamento do DFD resulta num refinamento correspondente dos dados à medida em que se movem através dos processos que constituem a aplicação.

Algumas diretrizes simples podem ajudar imensamente durante a derivação do diagrama de fluxo de dados: (1) o diagrama de fluxo de dados de nível 0 deve mostrar o software/sistema como uma única bolha; (2) a entrada e a saída principal devem ser cuidadosamente registradas; (3) o refinamento deve começar pelo isolamento dos processos, objetos de dados e depósitos de dados candidatos a serem representados no nível seguinte; (4) todas as setas e bolhas devem ser rotuladas com nomes significativos; (5) a continuidade do fluxo de informação deve ser mantida de nível para nível e (6) uma bolha de cada vez deve ser refinada. Há uma tendência natural de supercomplicar o diagrama de fluxo de dados. Isso ocorre quando o analista tenta mostrar excesso de detalhes prematuramente ou representar aspectos procedimentais do software em vez do fluxo da informação.

Fig. 12.19
Desenvolvimento de
relacionamentos e
cardinalidade/
modalidade.



Novamente considerando o produto *SafeHome*, um DFD de nível 0 para o sistema é mostrado na Fig. 12.20. As entidades externas principais (caixas) produzem informação para uso do sistema e consomem informação gerada pelo sistema. As setas rotuladas representam objetos de dados ou hierarquias de tipos de objetos de dados. Por exemplo, **comandos e dados do usuário** inclui todos os comandos de configuração, todos os comandos de ativação/desativação, todas as variedades de interações e todos os dados que dão entrada para qualificar ou expandir um comando.

O DFD de nível 0 é agora expandido para um modelo de nível 1. Mas como devemos proceder? Uma abordagem simples, mas efetiva, é realizar uma "análise gramatical" da narrativa de processamento que descreve a bolha no nível de contexto. Isto é, isolamos todos os substantivos (e frases substantivas) e verbos (e frases verbais) na narrativa de *SafeHome* apresentada originalmente no Cap. 11. Para ilustrar, reproduzimos novamente a narrativa de processamento sublinhando a primeira ocorrência de todos os substantivos e colocando em itálico a primeira ocorrência de todos os verbos.³

O software do *SafeHome* permite que o proprietário configure o sistema de segurança quando é *instalado*, *monitora* todos os sensores *conectados* ao sistema de segurança e *interage* com o proprietário por intermédio de um teclado e teclas de função *contidas* no painel de controle do *SafeHome*, mostrado na Fig. 11.2.

Durante a instalação, o painel de controle do *SafeHome* é usado para "programar" e configurar o sistema. A cada sensor é *atribuído* um número e tipo, uma senha mestra é programada para *armar* e *desarmar* o sistema e números de telefone são *introduzidos* para serem *discados* quando um evento de sensor ocorrer.

Quando um evento de sensor é *reconhecido*, o software *aciona* um alarme sonoro conectado ao sistema. Após um tempo de espera, que é *especificado* pelo proprietário durante as atividades de configuração do sistema, o sistema *disca* o número do telefone de um serviço de monitoramento, *fornece* informação sobre o local e *informa* a natureza do evento que foi detectado. O número de telefone será *rediscado* a cada 20 segundos até que a conexão telefônica seja *conseguida*.

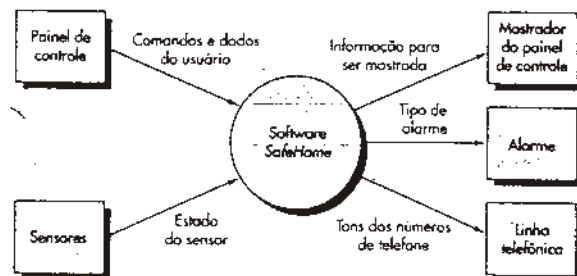
SUGESTÃO

A análise gramatical não é a toda prova, mas pode proporcionar-lhe um excelente ponto inicial se você está lutando para definir objetos e transformações de dados.

3 Há alguma diretriz útil para a criação de DFD?

3 Deve-se notar que todos os substantivos e verbos que são sinônimos ou que não têm relevância direta na modelagem do processo são omitidos.

Fig. 12.20 DFD em nível de contexto para SafeHome.



Toda a interação com o SafeHome é gerenciada por um subsistema de interação com o usuário, que lê a entrada fornecida pelo teclado e pelas teclas de função, exibe mensagens de interação no mostrador de LCD, e exibe informação sobre o estado do sistema no mostrador LCD. A interação pelo teclado toma a seguinte forma...

SUGESTÃO

Certifique-se de que a narrativa de processamento que você pretende analisar esteja toda escrita no mesmo nível de abstração.

Observando a "análise gramatical", um padrão começa a emergir. Todos os verbos são processos *SafeHome*; isto é, eles podem em última análise ser representados como bolhas em um DFD subsequente. Todos os substantivos ou são entidades externas (caixas), ou objetos de dados ou de controle (setas), ou depósitos de dados (linhas duplas). Note ainda que substantivos e verbos podem ser acoplados uns aos outros (p. ex., sensor tem um número e um tipo a ele atribuídos). Assim, pela realização de uma análise gramatical na narrativa de processamento de uma bolha, em qualquer nível de DFD, podemos gerar muita informação útil sobre como prosseguir com o refinamento para o nível seguinte. Usando essa informação, um DFD de nível 1 é mostrado na Fig. 12.21. O processo no nível de contexto, mostrado na Fig. 12.20, foi expandido em seis processos derivados do exame da análise gramatical. Analogamente, o fluxo de informação entre processos no nível 1 foi derivado da análise.

Deve-se notar que a continuidade do fluxo de informação é mantida entre os níveis 0 e 1. A elaboração do conteúdo das entradas e saídas nos DFD de nível 0 e 1 é postergada até a Seção 12.7.

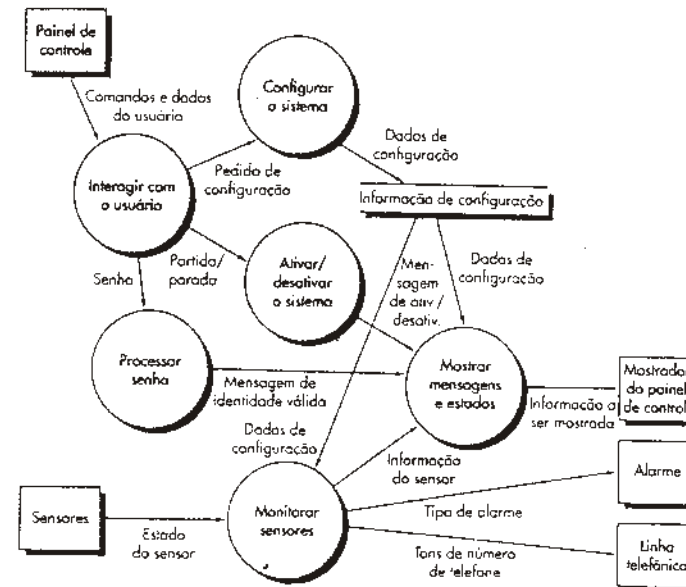
Os processos representados no DFD de nível 1 podem ser refinados em níveis inferiores. Por exemplo, o processo *monitorar sensores* pode ser refinado em um DFD de nível 2, como mostrado na Fig. 12.22. Note novamente que a continuidade do fluxo de informação foi mantida entre os níveis.

O refinamento dos DFD continua até que cada bolha realize uma simples função. Isto é, até que o processo representado pela bolha desempenhe uma função que seria facilmente implementada como um componente de programa. No Cap. 13, discutimos um conceito chamado *coesão*, que pode ser usado para avaliar a simplicidade de uma dada função. Por enquanto, nós tentamos refinar os DFD até que cada bolha tenha um "único objetivo".

12.6.3 Criação de um modelo de fluxo de controle

Para muitos tipos de aplicação de processamento de dados, o modelo de dados e o diagrama de fluxo de dados são tudo o que é necessário para obter significativo

Fig. 12.21 DFD de nível 1 para SafeHome.



conhecimento dos requisitos do software. Como já mencionamos anteriormente, no entanto, uma grande classe de aplicações é "conduzida" por eventos ao invés de dados; produzem informação de controle, em vez de relatórios ou imagens, e processam informação com grande preocupação de tempo e desempenho. Tais aplicações requerem o uso de modelagem de fluxo de controle, além da modelagem de fluxo de dados.

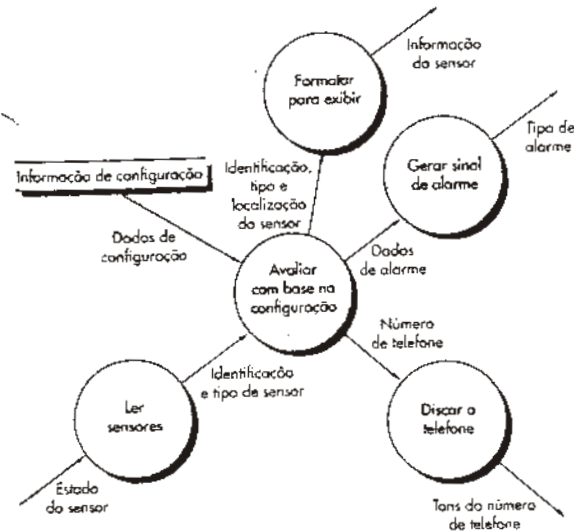
A notação gráfica necessária para criar um diagrama de fluxo de controle foi apresentada na Seção 12.4.4. A fim de rever a abordagem para criar um CFD, um modelo de fluxo de dados é "despido" de todas as setas de fluxo de dados. Eventos e itens de controle (setas tracejadas) são então adicionados ao diagrama e uma "janela" (barra vertical) para a especificação do controle é mostrada. Mas como são selecionados os eventos?

Já mencionamos anteriormente que um evento ou item de controle é implementado como um valor booleano (p. ex., verdadeiro ou falso, ligado ou desligado, 1 ou 0) ou uma lista discreta de condições (vazio, emperrado, cheio). Para selecionar potenciais candidatos a eventos, as seguintes diretrizes são sugeridas:

Como seleciono eventos potenciais para um CFD, STD e CSPEC?

- Liste todos os sensores que são "lidos" pelo software.
- Liste todas as condições de interrupção.
- Liste todas as "chaves" que são ajustadas por um operador.
- Liste todas as condições de dados.
- Voltando à análise substantivo/verbo que foi aplicada à narrativa de processamento, reveja todos os "itens de controle" como possíveis entradas/saídas da CSPEC.

Fig. 12.22 DFD de nível 2 que refina o processo monitorar sensores.



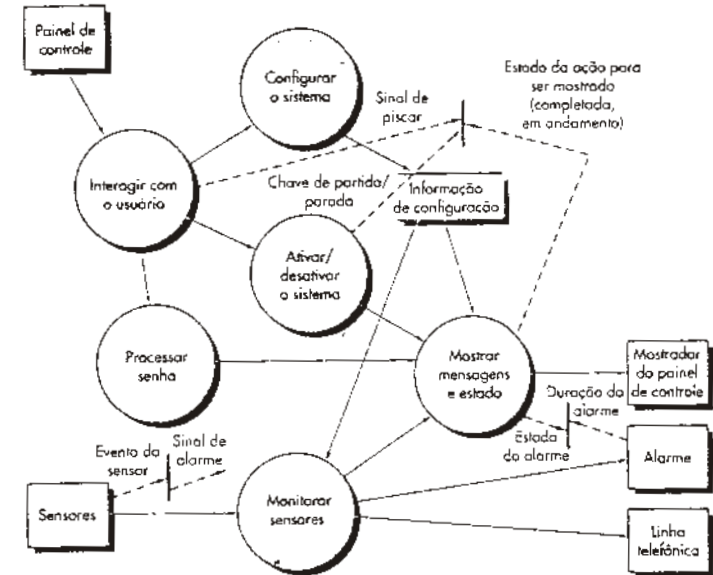
- Descreva o comportamento de um sistema pela identificação de seus estados; identifique como cada estado é atingido; e defina as transições entre estados.
- Enfoque possíveis omissões — um erro muito comum na especificação de controle; por exemplo, pergunte: "Há qualquer outro modo de chegar a esse estado ou sair dele?"

Um CFD de nível 1 para o software *SafeHome* é ilustrado na Fig. 12.23. Entre os eventos e itens de controle mostrados estão **evento de sensor** (i. e., um sensor foi sensibilizado), **sinal de piscar** (sinal para piscar o mostrador LCD) e **chave de partida/parada** (sinal para ligar ou desligar o sistema). Quando o evento flui para a janela CSPEC vindo do mundo exterior, implica que a CSPEC vai ativar um ou mais dos processos mostrados no CFD. Quando um item de controle vem de um processo e flui para a janela CSPEC, está implícito o controle e ativação de algum outro processo ou entidade externa.

12.6.4 A especificação de controle

A especificação de controle (CSPEC) representa o comportamento do sistema (no nível em que ela foi referenciada) de dois modos diferentes. A CSPEC contém um diagrama de transição de estados, que é uma especificação de comportamento sequencial. Ela pode também conter uma tabela de ativação de programa — uma especificação de comportamento combinatória. Os atributos subjacentes da CSPEC foram introduzidos na Seção 12.4.4. Agora é hora de considerar um exemplo dessa importante notação de modelagem para a análise estruturada.

Fig. 12.23 CFD de nível 1 para *SafeHome*.

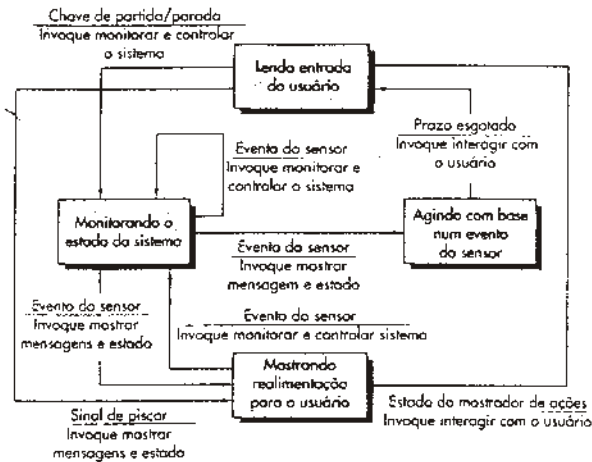


A Fig. 12.24 mostra um diagrama de transição de estados para o modelo de fluxo de controle de nível 1 de *SafeHome*. As setas de transição rotuladas indicam como o sistema responde a eventos à medida que passa pelos quatro estados definidos nesse nível. Estudando o STD, um engenheiro de software pode determinar o comportamento do sistema e, mais importante, pode verificar se há "furos" no comportamento especificado. Por exemplo, o STD (Fig. 12.24) indica que a única transição que se origina no estado *lendo entrada do usuário* ocorre quando **chave de partida/parada** é encontrado e ocorre uma transição para o estado *monitorando o status do sistema*. No entanto, parece não haver outro modo, além da ocorrência de um **evento do sensor**, que permita ao sistema voltar a *lendo entrada do usuário*. Esse é um erro de especificação e, esperamos, seria descoberto durante a revisão e corrigido. Examine o STD para determinar se há quaisquer outras anomalias.

Um modo de representação de comportamento um tanto diferente é a tabela de ativação de processos. A PAT representa a informação contida no STD, no contexto de processos, não de estados. Isto é, a tabela indica que processos (bolhas) no modelo de fluxo serão invocados quando um evento ocorrer. A PAT pode ser usada como diretriz por um projetista que precisa construir um executivo que controla os processos representados nesse nível. Uma PAT para o modelo de fluxo de nível 1 do software de *SafeHome* é mostrada na Fig. 12.25.

A CSPEC descreve o comportamento do sistema, mas não nos dá informação sobre o trabalho interno dos processos que são ativados como resultado desse comporta-

Fig. 12.24 Diagrama de transição de estado para SafeHome.



mento. A notação de modelagem que fornece essa informação é discutida na seção seguinte.

12.6.5 A especificação de processo

A especificação de processo (PSPEC) é usada para descrever todos os processos do modelo de fluxo que aparecem no nível de refinamento final. O conteúdo da especificação de processo pode incluir texto narrativo, uma descrição em *linguagem de projeto de programa* (*program design language*, PDL) do algoritmo do processo, equações matemáticas, tabelas, diagramas ou gráficos. Fornecendo uma PSPEC para acompanhar cada bolha do modelo de fluxo, o engenheiro de software cria uma "mini-especificação" que pode servir como primeiro passo para criação da *especificação de requisitos do software* e como diretriz para o projeto do componente de software que vai implementar o processo.

Para ilustrar o uso da PSPEC, considere a transformação *processar senha* representada no modelo de fluxo de *SafeHome* (Fig. 12.21). A PSPEC para essa função poderia tomar a forma:

PSPEC: processar senha

A transformação *processar senha* realiza todas as validações de senha para o sistema *SafeHome*. *Processar senha* recebe uma senha de quatro dígitos da função *interagir como usuário*. A senha é comparada com a senha mestra armazenada no sistema. Se a senha mestra coincide, < mensagem de id válida = verdadeiro > é passada para a função *mostrar mensagem e status*. Se a senha mestra não coincide, os quatro dígitos são comparados com uma tabela de senhas secundárias (que podem ser distribuídas a hóspedes e/ou empregados da casa que precisam ter acesso à casa quando o proprietário não está presente). Se a

Fig. 12.25 Tabela de ativação de processos para SafeHome.

Eventos de entrada						
Evento do sensor	0	0	0	0	1	0
Sinal de piscar	0	0	1	1	0	0
Chave de partida/parada	0	1	0	0	0	0
Estado do mostrador de ação						
Completa	0	0	0	1	0	0
Em andamento	0	0	1	0	0	0
Prazo esgotado	0	0	0	0	0	1
Saída						
Sinal de alarme	0	0	0	0	0	0
Ativação de processo						
Monitorar e controlar o sistema	0	1	0	0	1	1
Ativar/desativar o sistema	0	1	0	0	1	0
Mostrar mensagens e estado	1	0	1	1	0	1
Interagir com o usuário	1	0	0	1	0	1

senha coincide com uma entrada da tabela, < mensagem id válida = verdadeiro > é passada para a função *mostrar mensagem e status*. Se não há coincidência, < mensagem id válida = falso > é passada para a função *mostrar mensagem e status*.

Se detalhes algorítmicos adicionais são desejados nesse estágio, uma representação em linguagem de projeto de programa pode ser incluída como parte da PSPEC. No entanto, muitos acreditam que a versão PDL deve ser adiada até o início do projeto de componentes.

12.7 O DICIONÁRIO DE DADOS

O modelo de análise abrange representações de objetos de dados, funções e controle. Em cada representação, objetos de dados e/ou itens de controle desempenham um papel. Assim, é necessário fornecer uma abordagem organizada para representar as características de cada objeto de dados e itens de controle. Isso é conseguido com o dicionário de dados.

O dicionário de dados tem sido proposto com uma gramática quase formal para descrever o conteúdo dos objetos definidos durante a análise estruturada. Essa notação de modelagem importante foi definida da seguinte maneira [YOT89]:

O *dicionário de dados* é uma listagem organizada de todos os elementos de dados que são pertinentes ao sistema, com definições precisas e rigorosas, de modo que tanto o usuário quanto o analista de sistemas tenham um entendimento comum das entradas, saídas, componentes de depósitos e [até] cálculos intermediários.

? Como representa o conteúdo dos objetos de dados que identifiquei?

Atualmente, o dicionário de dados é sempre implementado como parte de uma "ferramenta de análise e projeto estruturado" CASE. Apesar do formato dos dicionários variar de ferramenta para ferramenta, a maioria contém a seguinte informação:

- **Nome** — o nome principal do item de dados ou controle, depósito de dados ou entidade externa.
- **Sinônimo** — outros nomes usados alternativamente.
- **Onde usado/como usado** — uma listagem dos processos que usam o item de dados ou controle e como ele é usado (p. ex., entrada de processo, saída do processo, como depósito, como entidade externa).
- **Descrição do conteúdo** — notação para representar conteúdo.
- **Informação suplementar** — outra informação sobre tipos de dados, valores preestabelecidos (se conhecidos), restrições ou limitações, etc

Uma vez introduzido um nome e seus sinônimos, de um objeto de dados ou item de controle no dicionário de dados, a consistência na denominação pode ser imposta. Isto é, se um membro da equipe de análise decidir nomear um item de dados recém-derivados xyz, mas xyz já está no dicionário, a ferramenta CASE que apoia o dicionário exibe um alerta para indicar nomes duplicados. Isso aperfeiçoa a consistência do modelo de análises e ajuda a reduzir erros.

Informação "onde usado/como usado" é registrada automaticamente com base nos modelos de fluxo. Quando uma entrada no dicionário é criada, a ferramenta CASE percorre os DFD e CFD para determinar quais processos usam a informação de dados ou controle e como ela é usada. Apesar disso parecer sem importância, é na verdade um dos benefícios mais importantes do dicionário. Durante a análise há um fluxo de modificações quase contínuo. Para projetos grandes, é freqüentemente muito difícil determinar o impacto de uma modificação. Muitos engenheiros de software têm perguntado, "Onde esse objeto de dados é usado? O que mais tem que ser modificado se o modificarmos? Qual vai ser o impacto global da modificação?" Como o dicionário de dados pode ser tratado como uma base de dados, o analista pode formular questões "Onde usado/como usado" e obter resposta a essas consultas.

A notação usada para desenvolver uma descrição de conteúdo é denotada na seguinte tabela:

Construção de Dados	Notação	Significado
	=	é composto de
	+	e
Sequência	{ }	uma ou outra
Seleção	{ }	uma ou outra
Repetição	{ } ⁿ	n repetições de
	()	dados opcionais
	...	delimita comentários

A notação permite a um engenheiro de software representar dados compostos em um dos três modos fundamentais em que eles podem ser construídos:

1. Como uma sequência de itens de dados.
2. Como uma seleção entre elementos de um conjunto de itens de dados.
3. Como um agrupamento repetido de itens de dados. Cada entrada de item de dados, que é representada como parte de uma sequência, seleção ou repetição, pode, ela própria, ser um item de dados composto que precisa de posterior refinamento dentro do dicionário.



Ferramentas CASE de análise estruturada.

Para ilustrar o uso do dicionário de dados, voltamos ao DFD de nível 2 para o processo *monitorar sensores* de *SafeHome*, mostrado na Fig. 12.22. Observando a figura, o item de dados **número de telefone** é especificado como entrada. Mas o que exatamente é um número de telefone? Pode ser um número local de sete dígitos, um ramal de quatro dígitos ou uma sequência de interurbano de 25 dígitos. O dicionário de dados nos fornece uma definição precisa de **número de telefone** para o DFD em questão. Além disso, indica onde e como esse item de dados é usado e qualquer informação suplementar que é relevante a ele. A entrada no dicionário de dados começa da seguinte forma:

```

nome:                número de telefone
sinônimos:           nenhum
onde usado/como usado: avaliar com base na configuração (saída)
                      , discar o telefone (entrada)

descrição:
número de telefone = [número local | número interurbano]
número local = prefixo + número de acesso
número interurbano = 1 + código de área + número local
código de área = [800 | 888 | 561]
prefixo = * número de três dígitos que nunca começa com 0 ou 1*
número de acesso = * qualquer cadeia de quatro números *
  
```

A descrição de conteúdo é expandida até que todos os itens de dados compostos tenham sido representados como itens elementares (itens que não requerem mais expansão) ou até que todos os itens compostos estejam representados em termos que seriam bem-entendidos e não ambíguos para todos os leitores. É também importante notar que uma especificação de dados elementares freqüentemente restringe o sistema. Por exemplo, a definição de código de área indica que apenas três códigos de área (dois de tarifa zero e um no sul da Flórida) são válidos para esse sistema.

O dicionário de dados define itens de informação de maneira não-ambígua. Apesar de podermos considerar que o número de telefone representado no DFD da Fig. 12.22 poderia acomodar um número de acesso ao operador de interurbano com 25 dígitos, a descrição de conteúdo do dicionário de dados nos diz que tais números não são parte dos dados que podem ser usados.

Para sistemas baseados em computador de grande porte, o dicionário de dados cresce rapidamente em tamanho e complexidade. De fato, é extremamente difícil manter um dicionário manualmente. Por essa razão, ferramentas CASE devem ser usadas.

12.8 OUTROS MÉTODOS DE ANÁLISE CLÁSSICOS



DSSD, JSD e SADT.

Durante anos, muitos outros métodos de análise de requisitos de software que valem a pena têm sido usados pela indústria. Enquanto todos seguem os princípios operacionais de análise discutidos no Cap. 11, cada um usa uma notação diferente e um conjunto de heurísticas próprio, para derivar o modelo de análise. Um resumo de três importantes métodos de análise:

- *Desenvolvimento de Sistemas Estruturados pelos Dados (Data Structured Systems Development, DSSD)* [WAR81], [ORR81]
- *Jackson System Development (JSD)* [JAC83]
- *Técnica de Análise e Projeto Estruturados (Structured Analysis and Design Technique, SADT)* [ROS77], [ROS85]

é apresentado no site SEPAWeb para os leitores interessados numa visão mais ampla da modelagem da análise.

12.9 RESUMO

A análise estruturada, um método de modelagem de requisitos amplamente usado, apóia-se na modelagem de dados e na modelagem de fluxo para criar a base de um modelo abrangente de análise. Usando diagramas entidade-relacionamento, o engenheiro de software cria uma representação de todos os objetos e dados que são importantes para o sistema. Diagramas de fluxo de dados e controle são usados como base para representação da transformação de dados e controle. Ao mesmo tempo, esses modelos são usados para criar um modelo funcional do software e para fornecer um mecanismo para o particionamento de função. Um modelo comportamental é criado usando o diagrama de transição de estados e o conteúdo dos dados é desenvolvido com um dicionário de dados. Especificações de processo e controle fornecem capacidade adicional de detalhe.

A notação original da análise estruturada foi desenvolvida para aplicações convencionais de processamento de dados, mas extensões tornaram o método aplicável a sistemas de tempo real. A análise estruturada é apoiada por um conjunto de ferramentas CASE que assistem na criação de cada elemento do modelo e também ajudam a garantir consistência e correção.

REFERÊNCIAS BIBLIOGRÁFICAS

- [BRU88] Bruyn, W. et al., "ESML: An Extended Systems Modeling Language Based on the Data Flow Diagram," *ACM Software Engineering Notes*, vol. 13, no. 1, January 1988, pp. 58-67.
- [CHE77] Chen, P., *The Entity-Relationship Approach to Logical Database Design*, QED Information Systems, 1977.
- [DEM79] DeMarco, T., *Structured Analysis and System Specification*, Prentice-Hall, 1979.
- [GAN82] Gane, T. and C. Sarson, *Structured Systems Analysis*, McGraw-Hill, 1982.
- [HAT87] Hatley, D.J. and I.A. Pirbhai, *Strategies for Real-Time System Specification*, Dorset House, 1987.
- [JAC83] Jackson, M.A., *System Development*, Prentice-Hall, 1983.
- [ORR81] Orr, K.T., *Structured Requirements Definition*, Ken Orr & Associates, Inc., 1981.
- [PAC80] Page-Jones, M., *The Practical Guide to Structured Systems Design*, Yourdon Press, 1980.
- [ROS77] Ross, D. and K. Schoman, "Structured Analysis for Requirements Definition," *IEEE Trans. Software Engineering*, vol. SE-3, no. 1, January 1977, pp. 6-15.

- [ROS85] Ross, D., "Applications and Extensions of SADT," *IEEE Computer*, vol. 18, no. 4, April 1984, pp. 25-35.
- [STE74] Stevens, W.R., G.J. Myers, and L.L. Constantine, "Structured Design," *IBM Systems Journal*, vol. 13, no. 2, 1974, pp. 115-139.
- [TIL93] Tillmann, G., *A Practical Guide to Logical Data Modeling*, McGraw-Hill, 1993.
- [WAR81] Warnier, J.D., *Logical Construction of Systems*, Van Nostrand-Reinhold, 1981.
- [WAR85] Ward, P.T. and S.J. Mellor, *Structured Development for Real-Time Systems* (three volumes), Yourdon Press, 1985.
- [YOU78] Yourdon, E.N. and Constantine, L.L., *Structured Design*, Yourdon Press, 1978.
- [YOU89] Yourdon, E.N., *Modern Structured Analysis*, Prentice-Hall, 1990.

PROBLEMAS E PONTOS A CONSIDERAR

12.1. Consiga pelo menos três das referências discutidas na Seção 12.1 e redija um trabalho curto que dá uma idéia de como a percepção da análise estruturada foi modificada ao longo do tempo. Como seção de conclusões, sugira maneiras pelas quais você pensa que o método vai se modificar no futuro.

→ 12.2. Você foi solicitado a construir um dos seguintes sistemas:

- Um sistema de matrícula em curso baseado em rede para sua universidade.
- Um sistema de processamento de pedidos baseado na Web para uma loja de computadores.
- Um sistema simples de faturamento para um pequeno negócio.
- Software que substitui um Rolodex e é embutido num telefone sem fio.
- Um livro de receitas automatizado que é embutido num forno elétrico ou de microondas.

Selecione o sistema que é de seu interesse e desenvolva um diagrama entidade/relacionamento que descreve objetos de dados, relações e atributos.

12.3. Qual é a diferença entre cardinalidade e modalidade?

→ 12.4. Desenhe um modelo no nível de contexto (DFD de nível 0) para um dos cinco sistemas que são listados no Problema 12.2. Redija uma narrativa de processamento no nível de contexto para o sistema.

→ 12.5. Usando o DFD no nível de contexto desenvolvido no Problema 12.4, desenvolva diagramas de fluxo de dados de nível 1 e 2. Use uma "análise gramatical" da narrativa de processamento no nível de contexto para dar início ao seu trabalho. Lembre-se de especificar todo o fluxo de informação rotulando todas as setas entre bolhas. Use nomes significativos para cada transformação.

→ 12.6. Desenvolva CFD, CSPEC, PSPEC e um dicionário de dados para o sistema que você selecionou no Problema 12.2. Tente fazer seu modelo tão completo quanto possível.

12.7. O conceito de continuidade do fluxo de informação significa que, se uma seta de fluxo aparece como entrada no nível 0, então uma seta de fluxo precisa aparecer como entrada nos níveis subsequentes? Discuta sua resposta.

12.8. Usando as extensões de Ward e Mellor, refaça o modelo de fluxo contido na Fig. 12.16. Como você vai acomodar a CSPEC que está implícita na Fig. 12.16? Ward e Mellor não usam essa notação.

12.9. Usando as extensões de Hatley e Pirbhai, refaça o modelo de fluxo contido na Fig. 12.13. Como você acomodaria o processo de controle (bolha tracejada) que está implícito na Fig. 12.13? Hatley e Pirbhai não usam essa notação.

12.10. Descreva um fluxo de evento com suas próprias palavras.

12.11. Desenvolva um modelo de fluxo completo para o software de fotocopadora discutido na Seção 12.5. Você pode usar ou o método de Ward e Mellor ou de Hatley e Pirbhai. Certifique-se de desenvolver um diagrama de transição de estados detalhado para o sistema.

12.12. Complete as narrativas de processamento para o modelo de análise do software SafeHome mostrado na Fig. 12.21. Descreva a mecânica de interação do usuário do sistema. Sua informação adicional vai mudar os modelos de fluxo para SafeHome apresentados neste capítulo? Em caso afirmativo, como?

12.13. O departamento de obras públicas de uma cidade grande decidiu desenvolver um sistema de acompanhamento e reparo de buracos (SARP) baseado na Web. Segue uma descrição:

Os cidadãos podem obter acesso a um site da Web e relatar a localização e gravidade dos buracos. À medida que os buracos são relatados eles são registrados num "sistema de reparo do departamento de obras públicas" e lhes é atribuído um número de identificação, armazenado por endereço da rua, tamanho (numa escala de 1 a 10), localização (no meio da rua, na calçada etc.), distrito (determinado pelo endereço da rua) e prioridade de reparo (determinada pelo tamanho do buraco). Dados da ordem de serviço são associados com cada buraco e incluem a localização e tamanho do buraco, número de identificação da equipe de reparo, número de pessoas na equipe, equipamento atribuído, horas aplicadas no reparo, estado do buraco (trabalho em andamento, reparado, reparo temporário, não reparado), quantidade de material de enchimento usado e custo do reparo (calculado a partir de horas aplicadas, quantidade de pessoas, material e equipamento usados). Finalmente, um arquivo de danos é criado para conter informação sobre danos relatados devido ao buraco e incluem nome do cidadão, endereço, número do telefone, tipo de dano, quantia em reais de prejuízo causado pelo dano. O SARP é um sistema on-line; todas as consultas devem ser feitas interativamente.

Usando a notação de análise estruturada, desenvolva um modelo de análises completo para o SARP.

12.14. Um software de próxima geração para um sistema de processamento de textos deve ser desenvolvido. Faça algumas horas de pesquisa na área da aplicação e realize uma reunião FAST (Cap. 11) com seus colegas para desenvolver requisitos (seu instrutor vai ajudá-lo a coordenar isso). Construa um modelo de requisitos do sistema usando análise estruturada.

12.15. Software para um videogame deve ser desenvolvido. Proceda como no Problema 12.14.

12.16. Entre em contato com quatro ou cinco fornecedores que vendem ferramentas CASE para análise estruturada. Reveja a literatura deles e redija um trabalho resumido que dê uma ideia das características genéricas que parecem distinguir uma ferramenta da outra.

LEITURAS E FONTES DE INFORMAÇÃO ADICIONAIS

Dúzias de livros têm sido publicados sobre análise estruturada. Todos cobrem o assunto adequadamente, mas apenas alguns fazem um trabalho realmente excelente. O livro de DeMarco [DEM79] continua sendo uma boa introdução para a notação básica. Os livros de Hoffer *et al.* (*Modern Systems Analysis and Design*, Addison-Wesley, 2nd ed., 1998), Kendall e Kendall (*Systems Analysis and Design*, 2nd ed., Prentice-Hall, 1998), Davis e Yen (*The Information Systems Consultant's Handbook: Systems Analysis and Design*, CRC Press, 1998), Modell (*A Professional's Guide to Systems Analysis*, 2nd ed., McGraw-Hill, 1996), Robertson e Robertson (*Complete Systems Analysis*, 2 volumes, Dorset House, 1994) e Page-Jones (*O Guia Prático para o Projeto Estruturado de Sistemas*, 2nd ed., Prentice-Hall, 1988) são referências que valem a pena. O livro de Yourdon sobre o assunto [YOU89] continua entre as coberturas mais abrangentes publicadas até hoje.

Para ênfase em engenharia [WAR85] e [HAT87] são os livros de preferência. No entanto, Edwards (*Real-Time Structured Methods: Systems Analysis*, Wiley, 1993) também cobre a análise de sistemas de tempo real em considerável nível de detalhe, apresentando alguns exemplos úteis extraídos de aplicações reais.

Muitas variações de análise estruturada evoluíram na última década. Cutts (*Structured Systems Analysis and Design Methodology*, Van Nostrand-Reinhold, 1990) e Hares (*SSADM for the Advanced Practitioner*, Wiley, 1990) descrevem SSADM, uma variação da análise estruturada que é amplamente usada na Inglaterra e na Europa.

Flynn *et al.* (*Information Modeling: An International Perspective*, Prentice-Hall, 1996), Reingruber e Gregory (*Data Modeling Handbook*, Wiley, 1995) e Tillman [TIL93] apresentam tutoriais detalhados para criar modelos de dados de qualidade industrial. Kim e Salvatore ("Comparing Data Modeling Formalisms", *Communications of the ACM*, Junho 1995) escreveram uma excelente comparação de métodos de modelagem de dados. Um livro interessante, de Hay, (*Data Modeling Patterns*, Dorset House, 1995) apresenta "padrões" típicos de modelos de dados que são encontrados em muitos negócios diferentes. Um tratamento detalhado da modelagem comportamental pode ser encontrado em Kowal (*Behavior Models: Specifying User's Expectations*, Prentice-Hall, 1992).

Uma grande variedade de fontes de informação sobre análise estruturada e assuntos relacionados está disponível na Internet. Uma lista atualizada de referências da World Wide Web que são relevantes para os conceitos e métodos de análise pode ser encontrada no site do autor:

<http://www.rspa.com>