Engenharia de Software

Capítulo 6

Projeto de Software

1. Introdução

A etapa de Projeto é o passo inicial de uma nova fase do ciclo de vida do software, a fase de Desenvolvimento. O Projeto consiste na aplicação de um conjunto de técnicas e princípios, de modo a definir um sistema num nível de detalhe suficiente à sua realização física. A tarefa do projetista nada mais é do que produzir um modelo de representação do software que será implementado. Nesta etapa, os requisitos definidos na etapa anterior devem servir de referência para a obtenção da representação do software.

Neste capítulo, serão apresentados os principais aspectos relativos ao projeto de software, como etapa fundamental para a obtenção de um software de qualidade. Serão apresentadas algumas das técnicas clássicas de projeto, assim como a aplicação de técnicas de projeto a aspectos mais inovadores do desenvolvimento de software como as interfaces homem-máquina e os sistemas de tempo-real.

2. O Processo de Projeto

2.1. Projetos Preliminar e Detalhado

A etapa de projeto é caracterizada pelo conjunto de atividades que vai permitir traduzir os requisitos definidos na etapa anterior em uma representação do software a ser construído. Na sua forma mais clássica, o primeiro resultado obtido no projeto é uma visão da estrutura do software em termos de componentes, sendo que, a partir de procedimentos de refinamento, chega-se a um nível de especificação bastante próxima da codificação do programa.

Do ponto de vista do gerenciamento do processo de desenvolvimento, a etapa de projeto é conduzida basicamente em dois principais estágios:

- o ***projeto preliminar***, o qual permite estabelecer, a partir dos requisitos, a arquitetura do software e da informação relacionada;
- o ***projeto detalhado***, o qual permite aperfeiçoar a estrutura do software e definir representações algorítmicas de seus componentes.

No contexto dos projetos preliminar e detalhado, uma conjunto de atividades técnicas de projeto são desenvolvidas. Num ponto de vista mais genérico, pode-se destacar os projetos *arquiteturais*, *procedimentais* e *de dados*. Em projetos mais recentes, e, particularmente, naqueles envolvendo interatividade com o usuário, o *projeto de interfaces* tem assumido um papel de fundamental importância, sendo considerada uma atividade do mesmo nível dos demais projetos.

2.2. Características desejáveis num projeto de software

A obtenção de um software de boa qualidade está relacionada ao sucesso de cada uma das etapas do seu desenvolvimento. No que diz respeito à etapa de projeto, é possível detectar algumas das características desejáveis:

- deve apresentar um organização hierárquica, definida de modo a utilizar racionalmente todos os elementos de software;
- deve basear-se em princípios de modularidade, ou seja, propor um particionamento do software em elementos que implementem funções ou subfunções do software;
- deve conduzir a uma definição em módulos com alto grau de independência, como forma de facilitar as tarefas de manutenção ao longo da vida do software;
- deve ser fiel à especificação dos requisitos definida na etapa anterior.

3. Aspectos fundamentais do projeto

Durante esta etapa, é importante que alguns conceitos sejam levados em conta para que se possa derivar um projeto contendo as características citadas acima. As seções que seguem discutirão alguns destes conceitos.

3.1. Abstração

O princípio de abstração está fortemente relacionado com as características de modularidade que um software pode apresentar. Quando se desenvolve um software que vai apresentar estas características, é comum que suas representações sejam feitas considerando vários níveis de abstração.

No que diz respeito à linguagem de representação, nos níveis mais altos de abstração, a linguagem utilizada é bastante orientada à aplicação e ao ambiente onde o software irá ser executado. À medida que se vai "descendo" nos níveis de abstração, a linguagem de representação vai se aproximando de questões de implementação, até que, no nível mais baixo, a solução é representada de modo a que possa ser derivada diretamente para uma linguagem de implementação.

Na realidade, o próprio processo de Engenharia de Software constitui-se num aprimoramento de níveis de abstração. Na Engenharia de Sistemas, por exemplo, parte das tarefas do sistema a desenvolver são atribuídas ao software, como elemento constituinte de um sistema computacional. Na Análise de Requisitos, a solução em termos de software é apresentada de forma conceitual. Durante o Projeto, partindo do Projeto Preliminar para o Projeto Detalhado, múltiplos níveis de abstração vão sendo definidos, aproximando-se cada vez mais da Implementação, que é o nível mais baixo de abstração.

3.2. Refinamento

O refinamento surgiu na forma de uma técnica de projeto (a técnica de Refinamentos Sucessivos, proposta por Wirth em 1971). Esta técnica sugere como ponto de partida a definição da arquitetura do software a ser desenvolvido, sendo que esta vai sendo refinada sucessivamente até atingir níveis de detalhes procedimentais. Este processo dá origem a uma hierarquia de representações, onde uma descrição macroscópica de cada função vai sendo decomposta passo-a-passo até se obter representações bastante próximas de uma linguagem de implementação. Este processo é bastante similar ao processo utilizado em diversas técnicas de Análise de Requisitos (como o DFD e o SADT), sendo que a diferença fundamental está nos níveis de detalhamento atingidos nesta etapa.

3.3. Modularidade

O conceito de modularidade tem sido utilizado já há bastante tempo, como forma de obtenção de um software que apresente algumas características interessantes como a facilidade de manutenção. Este conceito apareceu como uma solução aos antigos softwares "monolíticos", os quais representavam grandes dificuldades de entendimento e, conseqüentemente para qualquer atividade de manutenção. A utilização do conceito de modularidade oferece resultados a curto prazo, uma vez que, ao dividir-se um grande problema em problemas menores, as soluções são encontradas com esforço relativamente menor. Isto significa que, quanto maior o número de módulos definidos num software, menor será o esforço necessário para desenvolvê-lo, uma vez que o esforço de desenvolvimento de cada módulo será menor. Por outro lado, quanto maior o número de módulos, maior será o esforço no desenvolvimento das interfaces, o que permite concluir que esta regra deve ser utilizada com moderação.

Finalmente, é importante distinguir o conceito de modularidade de projeto com o de modularidade de implementação. Nada impede que um software seja projetado sob a ótica da modularidade e que sua implementação seja monolítica. Em alguns casos, como forma de evitar desperdício de tempo de processamento e de memória em chamadas de procedimentos (salvamento e recuperação de contexto), impõe-se o desenvolvimento de um programa sem a utilização de funções e procedimentos. Ainda assim, uma vez que o projeto seguiu uma filosofia de modularidade, o software deverá apresentar alguns benefícios resultantes da adoção deste princípio.

3.4. A Arquitetura de Software

O conceito de arquitetura de software está ligado aos dois principais aspectos do funcionamento de um software: a estrutura hierárquica de seus componentes (ou módulos) e as estruturas de dados. A arquitetura de software resulta do desenvolvimento de atividades de particionamento de um problema, encaminhadas desde a etapa de Análise de Requisitos. Naquela etapa, é dado o pontapé inicial para a definição das estruturas de dados e dos componentes de software. A solução é encaminhada ao longo do projeto, através da definição de um ou mais elementos de software que solucionarão uma parte do problema global.

É importante lembrar que não existe técnica de projeto que garanta a unicidade de solução a um dado problema. Diferentes soluções em termos de arquitetura podem ser derivadas a partir de um mesmo conjunto de requisitos de software. A grande dificuldade concentra-se em definir qual a melhor opção em termos de solução.

3.5. Hierarquia de Controle

A hierarquia de controle nada mais é do que a representação, usualmente sob a forma hierarquizada, da estrutura do software no que diz respeito aos seus componentes. O objetivo não é apresentar detalhes procedimentais ou de sequenciamento entre processos, mas de estabelecer as relações entre os diferentes componentes do software, explicitando os níveis de abstração (refinamento) aos quais eles pertencem.

O modo mais usual de apresentar a hierarquia de controle utiliza uma linguagem gráfica, normalmente em forma de árvore, como mostra a figura 6.1. Com relação à estrutura de controle é importante apresentar algumas definições de "medição".

Utilizando a figura 6.1 como referência, é possível extrair alguns conceitos:

- a ***profundidade***, a qual está associada ao número de níveis de abstração definidos para a representação do software;
- a ***largura***, que permite concluir sobre a abrangência do controle global do software;

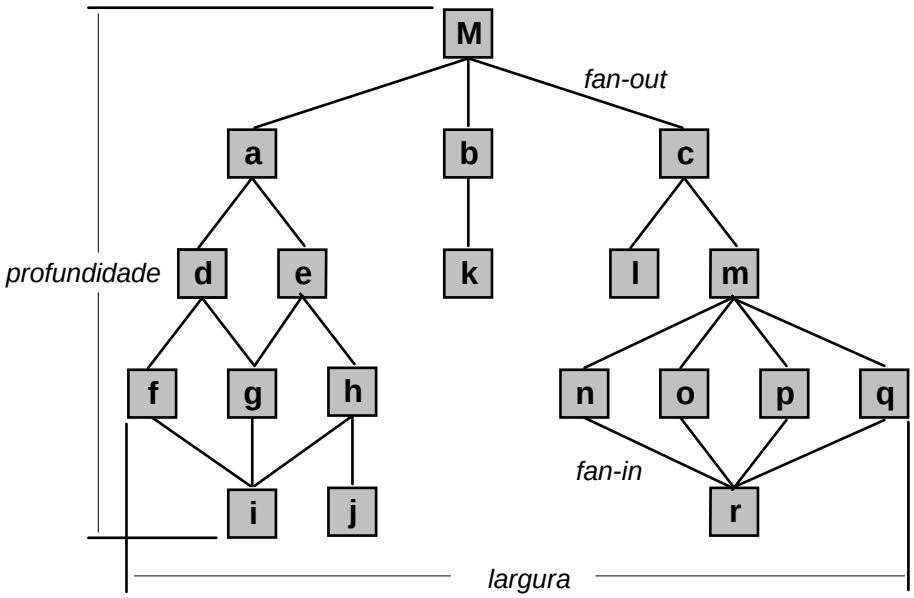

Figura 6.1 - Representação da hierarquia de controle.

- o ***fan-out***, que permite definir a quantidade de módulos controlados por um dado módulo;
- o ***fan-in***, que indica quantos módulos controlam um dado módulo.

Ainda, é possível estabelecer as relações de controle entre os módulos. Um módulo que exerce controle sobre outros módulos é denominado o módulo ***superordenado***. Um módulo que é controlado por outro é dito um módulo ***subordinado*** a ele.

3.6. A Estrutura dos Dados

A estrutura dos dados representa os relacionamentos lógicos entre os diferentes elementos de dados individuais. À medida que o projeto se aproxima da implementação, esta representação assume fundamental importância, já que a estrutura da informação vai exercer um impacto significativo no projeto procedimental final.

A estrutura dos dados define a forma como estão organizados, os métodos de acesso, o grau de associatividade e as alternativas de processamento das informações. Apesar de que a forma de organizar os elementos de dados e a complexidade de cada estrutura dependam do tipo de aplicação a desenvolver e da criatividade do projetista, existe um número limitado de componentes clássicos que funcionam como blocos de construção (building-blocks) de estruturas mais complexas. A figura 6.2 ilustra alguns destes construtores. Os construtores ilustrados podem ser combinados das mais diversas maneiras para obter estruturas de dados com grau de complexidade relativamente complexo.

3.7. Procedimentos de Software

Os procedimentos de software têm por objetivo expressar em detalhes a operação de cada componente de software (módulo) individualmente. Os aspectos a serem expressos nos procedimentos de software são o processamento da informação, o seqüenciamento de eventos, os pontos de decisão, as operações repetitivas e, eventualmente (se houver necessidade) algumas estruturas de dados.

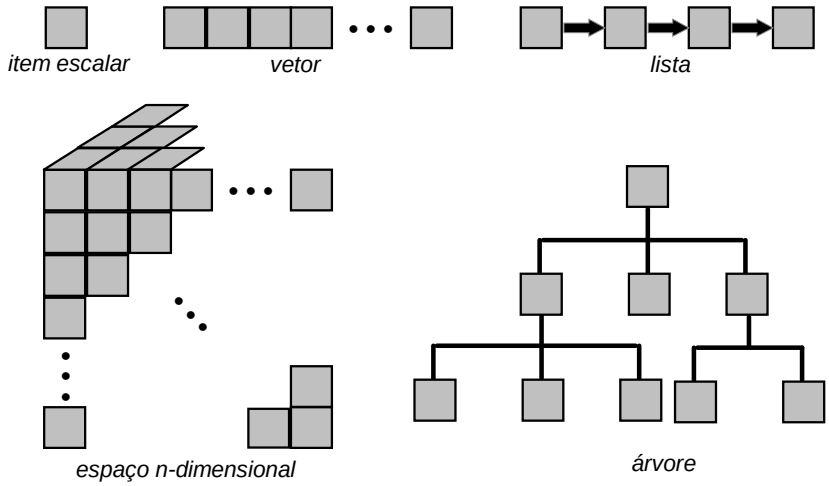

Figura 6.2 - Alguns construtores clássicos de estruturas de dados.

É evidente que a construção de procedimentos de software é uma atividade que deve ser feita tendo como referência a hierarquia de controle. Isto porque, na definição do procedimento de um dado módulo de software, deve ser feita referência a todos os módulos subordinados a ele.

Normalmente, esta referência será desenvolvida, num nível mais baixo de detalhamento, de modo a se obter o procedimento de software do módulo subordinado.

3.8. Ocultação de Informação

O princípio da ocultação de informações propõe um caminho para decompor sistematicamente um problema de modo a obter, de modo eficiente, os diferentes módulos do software a construir.

Segundo este princípio, os módulos devem ser decompostos de modo que as informações (os procedimentos e dados) contidas em cada módulo sejam inacessíveis aos módulos que não tenham necessidade destas informações. Ao realizar a decomposição segundo este princípio, o projetista proporciona um grau relativamente alto de independência entre os módulos, o que é altamente desejável num tal projeto.

Os benefícios da adoção deste princípio vão aparecer no momento em que modificações deverão ser encaminhadas a nível da implementação, por exemplo, por conseqüência de uma atividade de teste do software. Graças à ocultação de informação, erros introduzidos como resultado de alguma modificação num dado módulo não serão propagados a outros módulos do software.

4. CLASSES DE PROJETO

Olhando para os diversos aspectos apresentados na seção anterior, pode-se verificar que, de fato, a etapa de projeto de software é caracterizada por um conjunto de projetos, que desenrolam-se paralelamente. Nos itens abaixo, serão discutidos estes diferentes projetos.

4.1. O Projeto Modular

O princípio da modularidade é, sem dúvida, um dos mais difundidos e adotados em qualquer abordagem de desenvolvimento de software conhecida. A razão para isto reside nos benefícios da aplicação de tal princípio, tanto durante o desenvolvimento quanto ao longo da fase de manutenção do software.

Durante o projeto, os princípios de abstração e de ocultação de informação são aplicados como forma de obtenção dos módulos que constituem a arquitetura de um

programa. A aplicação destes dois princípios vai se refletir em características de operação do módulo, como:

- o ***tempo de incorporação***, que indica o momento da introdução do módulo no código-fonte do software (por exemplo, um macro de compilação, um subprograma, etc...);
- o ***mecanismo de ativação***, que indica a forma como o módulo será invocado durante a execução do programa (por exemplo, por meio de uma referência — instrução call — ou por interrupção);
- o ***padrão de controle***, o qual descreve como o módulo é executado internamente.

No que diz respeito aos tipos de módulos que podem ser definidos na arquitetura de um software, pode-se encontrar basicamente três categorias:

- os ***módulos seqüenciais***, que são ativados e sua execução ocorre sem qualquer interrupção;
- os ***módulos incrementais***, que podem ser interrompidos antes da conclusão do aplicativo e terem sua execução retomada a partir do ponto de interrupção;
- os ***módulos paralelos***, que executam simultaneamente a outros módulos em ambientes multiprocessadores concorrentes.

No que diz respeito ao projeto dos módulos, pode-se ainda apresentar algumas definições importantes:

- a ***independência funcional***, que surge como conseqüência da aplicação dos princípios de abstração e ocultação de informação; segundo alguns pesquisadores da área, a independência funcional pode ser obtida a partir da definição de módulos de "propósito único" e evitando-se excessivas interações com outros módulos;
- a ***coesão***, que está fortemente ligada ao princípio de ocultação e que sugere que um módulo pode realizar a sua função com um mínimo de interação com os demais módulos do sistema; é desejável que os módulos num software apresentem um alto grau de coesão;
- o ***acoplamento***, que permite exprimir o grau de conexão entre os módulos; os módulos de um software devem apresentar um baixo coeficiente de acoplamento.

4.2. O PROJETO DOS DADOS

À medida que a etapa de projeto evolui, as estruturas de dados vão assumindo um papel mais importante nesta atividade, uma vez que estas vão definir a complexidade procedimental do software (ou de seus componentes). O projeto dos dados nada mais é do que a seleção das representações lógicas dos objetos de dados identificados na etapa de Análise e Especificação dos Requisitos.

Como forma de obter resultados satisfatórios no que diz respeito ao projeto dos dados no contexto de um software, alguns princípios podem ser adotados:

- a realização de uma análise sistemática no que diz respeito aos dados, a exemplo do que é feito com os aspectos funcionais e comportamentais do software;
- identificação exaustiva das estruturas de dados e das operações a serem realizadas sobre elas;
- estabelecimento de um dicionário de dados (eventualmente, o mesmo definido na etapa de Análise de Requisitos, incluindo refinamentos);
- adiar decisões de baixa prioridade no que diz respeito ao projeto de dados (aplicação do princípio de refinamentos sucessivos);

- limitar a representação das estruturas de dados aos módulos que as utilizarão;
- estabelecimento de uma biblioteca de estruturas de dados úteis e das operações a serem aplicadas a elas (reusabilidade);
- adoção de uma linguagem de programação e projeto que suporte tipos abstratos de dados.

4.3. O Projeto Arquitetural

O projeto arquitetural visa a obtenção de uma estrutura modular de programa, dotada de uma representação dos relacionamentos de controle entre os módulos. Ainda, esta atividade permitem efetuar a fusão dos aspectos funcionais com os aspectos informacionais, a partir da definição de interfaces que irão definir o fluxo de dados através do programa.

Uma questão importante no que diz respeito ao projeto arquitetural é que ele deve ser encaminhado prioritariamente a outros projetos. É importante, antes que outras decisões possam ter tomadas, que se tenha uma visão global da arquitetura do software. O que estiver definido a nível do projeto da arquitetura do software vai ter, sem dúvida, grande impacto nas definições relativas aos demais projetos.

4.4. O Projeto Procedimental

O projeto procedimental é encaminhado a partir da definição da estrutura do software. Devido à riqueza de detalhes que pode caracterizar o projeto procedimental, a adoção de notações adequadas ao projeto é uma necessidade, como forma de evitar a indesejável ambigüidade que poderia ser resultante da utilização, por exemplo, da linguagem natural.

4.4.1. A Programação estruturada

A programação estruturada, introduzida no final dos anos 60, foi uma interessante forma de se encaminhar projetos e implementações de software oferecendo facilidade de programação e de entendimento. A idéia por trás da programação estruturada é a utilização de um conjunto limitado de construções lógicas simples a partir das quais qualquer programa poderia ser construído. Cada construção apresentava um comportamento lógico previsível, iniciando no topo e finalizando na base, o que propiciava ao leitor um melhor acompanhamento do fluxo procedimental.

Independente da linguagem utilizada para a implementação, ela era composta basicamente de três classes de construções: as seqüências, as condições e as repetições.

O uso de um número limitado de construções simples (as quais podem ser combinadas das mais variadas formas) permite obter, sem dúvida, programas mais simples. Um programador que domine efetivamente as construções básicas da programação estruturada, não encontra grandes dificuldades para combiná-las durante a atividade de síntese de um dado procedimento. A mesma observação pode ser feita para as atividades de análise.

4.4.2. O pseudocódigo

Esta notação pode ser aplicada tanto para o projeto arquitetural quanto para o projeto detalhado e a qualquer nível de abstração do projeto. O projetista representa os aspectos estruturais e de comportamento utilizando uma linguagem de síntese, com expressões de sua língua (Inglês, Português, etc...), e estruturada através de construções tais como: IF-THEN-ELSE, WHILE-DO, REPEAT-UNTIL, END, etc... Esta política permite uma representação e uma análise dos fluxos de controle que determinam o comportamento dos componentes do software bastante adequadas à posterior derivação do código de implementação.

O uso do pesudocódigo pode suportar o desenvolvimento do projeto segundo uma política "top-down" ou "bottom-up". No caso do desenvolvimento "top-down", uma frase definida num dado nível de projeto é substituída por seqüências de frases que representam o refinamento da frase original.

O pseudocódigo, apesar de não apresentar uma visão gráfica da estrutura do software ou de seu comportamento, é uma técnica bastante interessante pela sua facilidade de uso e pela sua similaridade com algumas das linguagens de implementação conhecidas, como, por exemplo, Pascal, C, etc... A seguir, é apresentada uma listagem exemplo de uso do pseudocódigo para o projeto detalhado de um componente de software.

```
INICIALIZA tabelas e contadores;
ABRE arquivos;
LÊ o primeiro registro de texto;
ENQUANTO houver registros de texto no arquivo FAZER
     ENQUANTO houver palavras no registro de texto FAZER
       LÊ a próxima palavra
       PROCURA na tabela de palavras a palavra lida
         SE a palavra lida é encontrada
           ENTÃO
             INCREMENTA o contador de ocorrências da palavra lida
           SENÃO
             INSERE a palavra lida na tabela de palavras
        INCREMENTA o contador de palavras processadas
     FIM_ENQUANTO
FIM_ENQUANTO
IMPRIME a tabela de palavras e o contador de palavras processadas
FECHA arquivos
FIM do programa
```

4.4.3. O uso de notações gráficas

Outra contribuição importante às atividades de projeto pode ser adoção de notações gráficas para representar os procedimentos a serem implementados. O fluxograma é uma conhecida técnica para a representação do fluxo de controle de programas, particularmente, os programas seqüenciais. Os fluxogramas estruturados são aqueles baseados na existência de um conjunto limitado de fluxogramas básicos que, combinados, compõem uma representação de comportamento de um elemento de software.

O resultado obtido é uma representação gráfica de comportamento que pode, eventualmente, ser representada por um pseudocódigo. Exemplos de fluxogramas básicos para a composição de fluxogramas mais complexos são apresentados na figura 6.3.

Uma técnica gráfica também definida para a representação de comportamento de componentes de um software é o diagrama de Nassi-Schneiderman. Esta técnica é baseada na representação através de caixas, das estruturas de controle clássicas.

De forma similar aos fluxogramas estruturados, os diagramas de Nassi-Schneiderman são baseados na existência de blocos básicos representando estruturas elementares de controle que serão combinados de modo a compor a representação total do software (ou do componente de software) projetado.

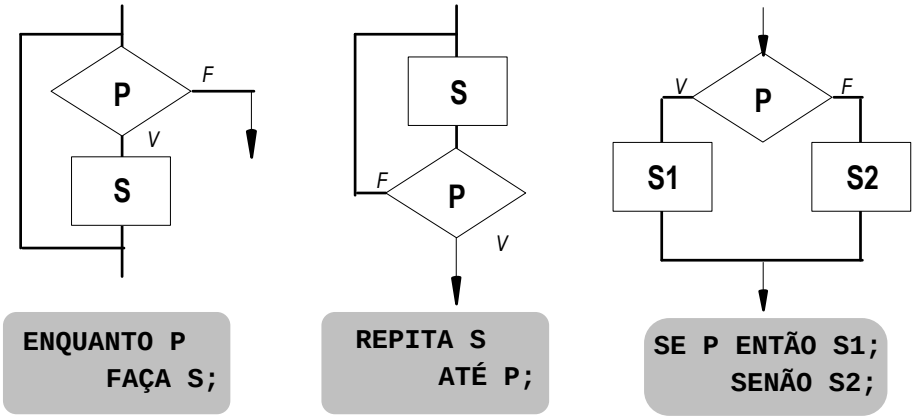

Figura 6.3 - Exemplos de blocos básicos para fluxogramas estruturados.

A figura 6.4 apresenta alguns exemplos de blocos básicos definidos nesta técnica. A representação de comportamento é obtida a partir do encaixe das estruturas básicas, formando uma caixa como mostra o exemplo da figura 6.5.

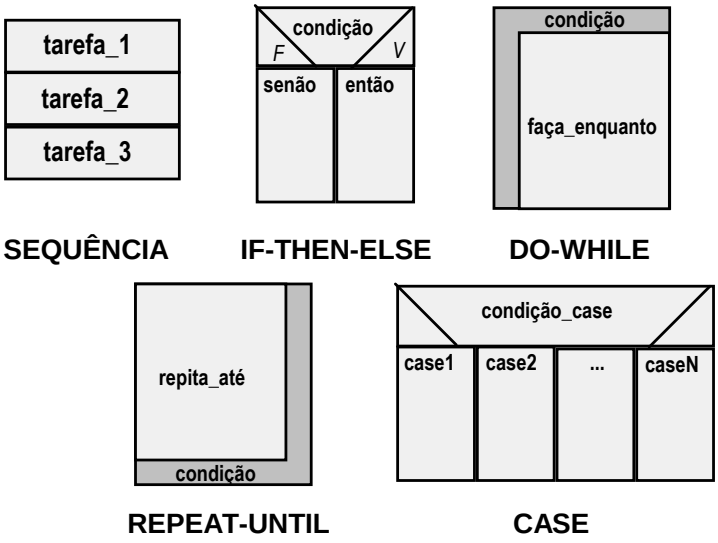

Figura 6.4 - Exemplos de blocos básicos dos diagramas de Nassi-Schneiderman.

5. DOCUMENTAÇÃO

A figura 6.6 apresenta uma proposta de estrutura para o documento de projeto de software. O objetivo deste documento é apresentar uma descrição completa do software, sendo que as seções vão sendo geradas à medida que o projetista avança no seu trabalho de refinamento da representação do software.

Na seção I, é apresentado o escopo global do software, sendo que grande parte do que vai conter esta seção é derivada diretamente do documento de Especificação de Requisitos obtido na etapa precedente, assim como de outros documentos gerados na fase de definição do software. A seção II apresenta as referências específicas à documentação de apoio utilizada.

A *Descrição de Projeto*, objeto da seção III, apresenta uma visão de projeto preliminar. Nesta parte do documento, é apresentada a estrutura do software, obtida a partir de diagramas de fluxos de dados ou outras técnicas de representação utilizadas durante a etapa de Análise de Requisitos. Juntamente com a estrutura do software, são apresentadas as diferentes interfaces de cada componente de software.

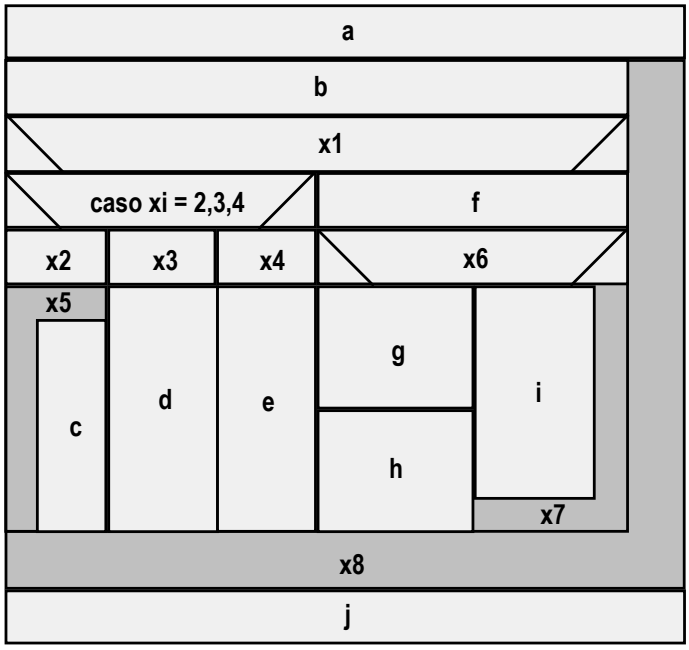

Figura 6.5 - Exemplo de representação de comportamento utilizando Nassi-Schneiderman.

Nas seções IV e V, são apresentados os resultados das atividades de refinamento, que conduzem aos níveis mais detalhados de projeto. Inicialmente, é apresentada uma narrativa (em linguagem natural) da operação de cada componente de software (módulos, procedimentos, funções, etc...). Esta narrativa deve concentrar-se na especificação da função a ser provida pelo componente, evitando detalhes algorítmicos.

A partir desta narrativa, com o auxílio de uma técnica de projeto procedimental, obtém-se uma descrição estruturada de cada componente. A seção V apresenta uma descrição da organização dos dados (arquivos mantidos em meios de armazenagem, dados globais, referência cruzada, etc..).

I - Escopo
II - Documentos de Referência
III - Descrição do Projeto
- **3.1 - Descrição dos Dados**
- **3.2 - Estrutura de Software**
- **3.3 - Interfaces dos Componentes**

IV - Descrição dos Módulos
- **4.1 - Narrativa de Processamento**
- **4.2 - Descrição da Interface**
- **4.3 - Descrição numa Técnica de Projeto**
- **4.4 - Outros**

V - Estrutura de Arquivos e Dados Globais
VI - Referência Cruzada dos Requisitos
VII - Procedimentos de Teste
VIII - Requisitos Especiais
IX - Observações
X - Apêndices

Figura 6.6 - Estrutura do documento de Projeto de Software.

Na seção VI, é elaborada uma referência cruzada dos requisitos, cujo objetivo é determinar que aspectos do projeto desenvolvido estão satisfazendo os requisitos especificados.

De uma forma mais concreta, deve ser feita a indicação de que conjunto de módulos é determinante para a implementação dos requisitos especificados.

Na seção VII é apresentada a especificação dos procedimentos de teste, o que é possível efetuar graças à definição já estabelecida da estrutura do software e das interfaces.

Na seção VIII, são apresentadas as restrições e requisitos especiais para o desenvolvimento do software (necessidade de overlay, gerenciamento de memória virtual, processamento de alta velocidade, interface gráfica específica, etc...).

As seções IX e X apresentam dados complementares, como a descrição de algoritmos, procedimentos alternativos, dados tabulares. Finalmente, pode ser encaminhado o desenvolvimento de um Manual de Instalação/Operações Preliminares, a ser incluído como apêndice ao documento.

6. PROJETO DE INTERFACES HOMEM-MÁQUINA

À medida que os sistemas computacionais foram evoluindo e conquistando um número cada vez maior de usuários, os aspectos de interface com o usuário passaram a assumir um papel de fundamental importância na construção de softwares.

Os computadores pessoais, que apareceram no final dos anos 70, são um exemplo típico desta evolução. Basta olhar a forma como evoluiu a interface de seu sistema operacional, do DOS, em sua versão baseada em linguagens de comando, passando pela construção de uma interface gráfica executando sobre o DOS (o Windows) até chegar, nos dias atuais, ao Windows 95, um sistema inerentemente gráfico que permite ao usuário manipular todos os objetos e eventos de seu sistema através de ícones e movimentos e ações com o mouse. O uso do teclado, num tal sistema, restringe-se à digitação de nomes de arquivos e poucos parâmetros de configuração.

Esta evolução tem sido uma conseqüência, por um lado, da evolução do hardware dos computadores e, por outro lado, da necessidade em eliminar o "terror tecnológico" criado graças à existência de interfaces de difícil aprendizado, complexas no uso e totalmente frustantes em boa parte dos casos.

6.1. Os fatores humanos

Isto faz com que o projeto das interfaces homem-máquina deixe de ser visto como mais uma componente do projeto do software como um todo, mas que passe a ser considerada uma atividade onde uma série de fatores técnicos e, principalmente, humanos sejam levados em conta de modo a que o usuário possa explorar completamente, e da forma mais amigável possível, as potencialidades do software.

O projetista de interfaces homem-máquina deve conhecer alguns conceitos que serão fundamentais para o encaminhamento de suas atividades:

- ***o mecanismo da percepção humana***, particularmente com relação aos sentidos de visão, audição e de tato, exprimem a capacidade do ser humano em adquirir, armazenar e utilizar as informações; na maior parte das operações realizadas num computador, o usuário faz uso dos olhos e do cérebro para o processamento das informações, sendo capaz de diferenciar os diferentes tipos de informação segundo diversos parâmetros visuais (a cor, a forma, as dimensões, etc...); a leitura é um outro processo importante na comunicação usuário-software, apesar da grande tendência em se utilizar recursos gráficos nas interfaces homem-máquina;
- ***os diferentes níveis de habilidades*** das pessoas são um outro aspecto de importância no projeto de uma interface de software; o projetista deve ter a

preocupação de dirigir a interface para o usuário típico do software; uma interface que seja orientada e adequada ao uso por um engenheiro pode representar grandes dificuldades de utilização por parte de um trabalhador inexperiente; no caso de softwares de aplicação vertical (destinados a áreas de aplicação específicas), é importante que a linguagem utilizada suporte termos e conceitos próprios da área;

- ***as tarefas a realizar*** com a introdução do software são, em grande parte dos casos, as mesmas que eram realizadas no sistema não-automatizado; a introdução do software para automatizar um sistema raramente viabiliza a realização de novas tarefas, mas permite que as tarefas antes realizadas manualmente possam vir a ser realizadas de forma mais eficiente e menos dispendiosa; em muitos casos, o software assume tarefas que eram realizadas manualmente; sendo assim, as tarefas essenciais do sistema deve ser um outro aspecto a ser considerado no projeto da interface.

6.2. Aspectos dos projetos de interface

6.2.1. Os diferentes modelos utilizados no projeto

Dado que as interfaces homem-máquina vão envolver diferentes elementos do sistema computacional (pelo menos o software e pessoas), o projetista deve manipular diferentes modelos para obter um resultado em termos de interface que seja compatível com os fatores técnicos e humanos desejados.

Um primeiro modelo a ser definido e utilizado pelo projetista é o **modelo de projeto**, que está relacionado à representação de todos os aspectos do software (procedimentos, arquitetura, dados, etc...).

Um outro modelo a ser desenvolvido pelo projetista é um **modelo de usuário**, o qual permite descrever o perfil típico do usuário do software. Parâmetros como idade, sexo, formação, motivação, capacidades físicas, personalidade, etc..., são levados em conta na construção deste modelo. Além destas características, o grau de experiência e de utilização de um software pode intervir, o que permite classificar os usuários em *principiantes*, *instruídos e intermitentes* e *instruídos e freqüentes*. A localização nesta classificação corresponde ao grau de conhecimento semântico (o conhecimento das funções básicas relativas à aplicação) e conhecimento sintático (a forma de interação com o software) apresentado pelo usuário.

A **percepção do sistema** corresponde a um modelo estabelecido pelo usuário que representa a sua visão do que será o software. Neste modelo, o usuário descreve como ele pensa que deverá ser a sua interação com o software: os objetos da aplicação que ele considera importantes (e que devem, portanto, ser ressaltados), as operações que ele desejará realizar sobre os objetos do sistema, como ele pretende visualizar estes objetos na interface do software, etc...

A **imagem do sistema** é uma representação do implementador do que o software vai oferecer como interface e todo o material de apoio à utilização (manuais, videotapes, livros, etc...). A coincidência entre a percepção do sistema e a imagem do sistema não é um acontecimento óbvio, mas quanto mais próximos estiverem estes modelos, mais chances se terá de projetar uma interface que venha satisfazer aos anseios de seus usuários e que viabilize a utilização efetiva do software.

O trabalho do projetista é o de definir os aspectos de interface à luz dos modelos produzidos, procurando harmonizar os desejos dos usuários com as limitações impostas pelas técnicas e ferramentas disponíveis e pelos critérios de projeto do software.

6.2.2. Análise e Modelagem de Tarefa

Como se pôde verificar nas discussões relativas ao projeto num contexto mais geral, as atividades de síntese (no caso, as atividades de projeto) são antecedidas por atividades de análise (no caso, a análise de requisitos).

Ainda, como já foi citado, os softwares são introduzidos nos sistemas como forma de automatizar um conjunto de tarefas que vêm sendo realizadas de forma inadequada ou ineficiente (ou utilizando equipamentos obsoletos ou realizados artesanalmente). Sendo assim, uma análise do conjunto destas tarefas é primordial ao projeto da interface.

O objetivo desta análise é viabilizar um mapeamento da interface não necessariamente idêntico, mas próximo do conjunto de tarefas que vem sendo realizado pelos profissionais envolvidos na operação do sistema (e que serão os usuários finais do software).

O processo de análise das tarefas é normalmente conduzido segundo uma abordagem por refinamentos sucessivos. O projetista identifica as principais tarefas executadas no contexto da aplicação, sendo que cada tarefa pode ser, eventualmente, refinada em duas ou mais subtarefas.

A partir desta identificação e refinamento das tarefas, o projeto da interface pode ser encaminhado, observando-se os seguintes passos iniciais:

- estabelecimento dos objetivos de cada tarefa;
- definição da seqüência de ações necessárias para cada tarefa;
- especificar como as ações de cada tarefa estarão acessíveis a nível de interface;
- indicar o aspecto visual da interface para cada acesso a uma ação da seqüência especificada (estado do sistema);
- definir os mecanismos disponíveis para que o usuário possa alterar o estado do sistema;
- especificar o efeito de cada mecanismo de controle no estado do sistema;
- indicar o significado de cada estado do sistema (que informações deverão ser oferecidas pela interface em cada estado).

6.2.3. Parâmetros de Projeto

Durante o desenvolvimento de uma interface, existem alguns parâmetros visíveis pelo usuário que devem fazer parte das preocupações do projetista:

- o ***tempo de resposta*** é um parâmetro que, se mal definido (ou omitido num projeto) pode conduzir a resultados frustantes a despeito do cuidado com o qual tenham sido tratados outros fatores; o tempo de resposta a comandos do usuário não tem de ser necessariamente o mais curto possível; em muitas aplicações, pode ser interessante se ter um tempo de resposta mai elevado (sem que o usuário venha a se aborrecer de esperar), de forma a permitir que o usuário reflita sobre as operações que está realizando; o tempo de resposta deve ser projetado observando dois aspectos: a *extensão* (o espaço de tempo entre um comando do usuário e a resposta do software) e a *variabilidade* (que se refere ao desvio de tempo de resposta do tempo de resposta médio de todas as funções do sistema);
- a ***ajuda ao usuário*** é um outro parâmetro importante, uma vez que é muito comum a ocorrência de dúvidas durante a utilização de um software; na maioria dos softwares, dois tipos de facilidades de ajuda podem ser definidas: a ajuda integrada, que é projetada juntamente com o software e que permite que o usuário obtenha respostas rapidamente sobre tópicos relativos à ação que ele está realizando, e a ajuda "add-on", a qual é incorporada após a instalação do software e que exige que o usuário percorra uma lista relativamente longa de tópicos para encontrar a resposta à sua dúvida; é óbvio que o primeiro tipo de ajuda pode tornar a interface mais amigável;

- o **tratamento de mensagens de erro** é um outro aspecto com o qual o projetista deve preocupar-se; a forma como as mensagens são apresentadas para o usuário deve ser cuidadosamente estudada; em muitos casos, as mensagens de erro indicam situações (anormalidades) às quais os usuários não têm nenhum controle (por exemplo, memória insuficiente para executar uma dada função); em outros casos, elas sinalizam uma manipulação incorreta por parte do usuário, a qual pode ser corrigida (por exemplo, indicação de um parâmetro fora dos limites válidos); neste caso, a mensagem deve apresentar informações suficientemente completas e que permitam que o usuário recupere a origem do erro; no caso em que o erro provoque conseqüências danosas à execução de uma dada tarefa (por exemplo, adulteração de conteúdo de um arquivo), estas deverão ser explicitadas na mensagem; resumindo, quanto melhor elaboradas forem as mansagens de erro, menor poderá ser a frustação do usuário quando elas ocorrerem;
- a ***rotulação de comandos*** (ou de menus e opções) corresponde a um outro aspecto que deve ser tratado com especial atenção pelo projetista; os softwares onde a interação é feita através de uma linguagem de comandos correspondem a uma abordagem que tende a desaparecer dando lugar ao uso de técnicas de "point-and-pick", baseadas no uso de janelas e tendo o mouse como ferramenta essencial de entrada de dados e comandos; de todo modo, mesmo softwares com interfaces gráficas de última geração podem oferecer ao usuário uma opção de uso do teclado para todas (ou pelo menos para as mais importantes) funções disponíveis no software (os atalhos de teclado); um outro aspecto presente em aplicações mais recentes é o conceito de macros, que podem ser definidos pelo usuário para armazenar as seqüências mais comuns de uso do software; segundo este conceito, ao invés de digitar todos os comandos necessários à realização de uma dada tarefa, o usuário digita simplesmente o nome do macro; um último aspecto relacionado aos comandos é o uso de padronizações em relação aos rótulos ou atalhos de teclado; um exemplo claro deste tipo de padronização está na maior parte das aplicações construídas para os ambientes gráficos, que utilizam o mesmo rótulo para ações compatíveis como as ações de cópia e reprodução de blocos de texto ou gráfico (copy/paste), as ações de salvamento de arquivos (save/save as...), o abandono de um programa (exit).

6.2.4. A Implementação de Interfaces

O projeto de uma interface homem-máquina consiste invariavelmente num processo iterativo, onde um modelo de projeto é criado, implementado na forma de protótipo e vai sendo refinado e aproximando-se da versão definitiva à medida que recebe realimentações por parte de usuários típicos do software.

Um aspecto importante neste processo é a disponibilidade de ferramentas de suporte à abordagem que permitam obter, com certa agilidade, os modelos de projeto e realizar as modificações necessárias de modo a obter as versões finais.

Atualmente, existem diversos conjuntos de ferramentas de suporte ao desenvolvimento de interfaces, as quais oferecem bibliotecas para a construção dos elementos básicos como janelas, menus, caixas de diálogos, mensagens de erros, etc...Alguns ambientes de programação mais recentes incluem a possibilidade do programador "desenhar" o aspecto visual das telas da interface e obter automaticamente o código dos módulos de interface a partir dos desenhos.

Uma vantagem do uso de tais ferramentas, além da redução do esforço de desenvolvimento, é a obtenção de uma interface que respeite a determinados padrões de aparência e acesso a comandos, já adotados por outras aplicações de uma mesma plataforma.

6.3. Princípios do Projeto de Interfaces

De modo a cumprir o objetivo de gerar uma interface que permita ao usuário explorar completa e eficientemente as potencialidades do software, um conjunto de princípios deve ser observado. Não existe uma fórmula ótima para o desenvolvimento das interfaces, mas, em linhas gerais, pode-se classificar os princípios sob três diferentes pontos de vista: a *interação*, a *exibição de informações* e a *entrada de dados*.

6.3.1. A Interação

No que diz respeito à interação, pode-se relacionar as seguintes diretrizes:

- a ***coerência***, na definição das escolhas do menu, entradas de comandos e outras funções importantes;
- utilização de ***recursos visuais e auditivos*** nas respostas a comandos do usuário;
- exigir ***confirmação*** de qualquer ação destrutiva não trivial (eliminação de arquivos, abandono do software, alterações substanciais de configuração, etc...), proteger eventualmente com *senha* ações que possam acarretar em conseqüências catastróficas para o sistema;
- possibilitar a ***reversão*** (*"undo"*) da maior parte das ações (por exemplo, um usuário que selecione um bloco de texto a ser copiado e que digite acidentalmente uma tecla qualquer antes do comando de cópia pode perder parte do seu trabalho se não houver a possibilidade de anulação da digitação);
- ***redução da quantidade de informações*** a ser memorizada no intervalo entre ações;
- ***otimização de diálogo***, definindo cuidadosamente o layout das telas e dos atalhos de teclado;
- admissão de erros do usuário (atalhos de teclado incorretos, comandos e dados inadequados, etc...), evitando a ocorrência de funcionamento anormal (relacionado à robustez);
- ***categorização das atividades***, organizando a tela segundo a classificação escolhida (operações de arquivos, operações de edição, ajuda, formatação, configurações, etc...);
- oferecimento de ***facilidades de ajuda sensíveis ao contexto***;
- ***nomeação de comandos*** através de verbos ou expressões verbais simples.

6.3.2. A Exibição de Informações

Com relação à exibição de informações, os seguintes aspectos devem ser observados:

- a ***concisão***, ou seja, mostrar apenas informações relevantes ao contexto do sistema;
- priorizar a utilização de ***símbolos gráficos*** para a expressão de informações, quando houver possibilidade;
- ***precisão com relação às mensagens de erro***, permitindo que o usuário possa tomar alguma ação que possa recuperar o erro ou que ele possa evitar uma situação semelhante;
- em ***informações textuais***, o uso de letras maiúsculas e minúsculas pode facilitar o entendimento;
- uso de ***janelas*** para separar diferentes tipos de informação;
- utilizar ***displays análogos*** para representar valores de grandezas reais de um dado sistema (por exemplo, a utilização da figura de um termômetro para indicar o valor da temperatura de uma determinada substância é mais significativa para um

usuário do que um valor numérico desfilando na tela; eventualmente, uma combinação dos dois pode ser uma boa solução — redundância);
- ***uso eficiente do espaço da tela***, particularmente em interfaces baseadas no uso de múltiplas janelas; a disposição destas deve ser tal que pelo menos uma parte de cada janela continue sendo visualizada; caso contrário, definir um mecanismo que permita fazer rapidamente o chaveamento entre janelas (menu Janelas, ou comandos de teclado — ALT+TAB, por exemplo);

6.3.3. A Entrada de Dados

Durante a utilização de um software com alto nível de interatividade, o usuário passa boa parte do seu tempo fazendo a escolha de comandos e inserindo dados de processamento. De forma a melhorar as condições nas quais estas interações ocorrem, os seguintes princípios devem ser considerados:

- ***minimização do número de ações necessárias à entrada de dados***, reduzindo, principalmente a quantidade de texto a ser digitada; um recurso interessante para esta redução é o uso de recursos gráficos sensíveis ao mouse, como por exemplo, escalas variáveis;
- a ***coerência visual da interface em relação às entradas***, mantendo cores, tamanhos e formas compatíveis com o restante da interface;
- ***configuração da entrada por parte do usuário***, permitindo minimizar, em função das habilidades do usuário, etapas na entrada dos dados (por exemplo, eliminar quadros de informação sobre como o usuário deve proceder para executar determinado comando ou definir um dado);
- ***desativação de comandos*** inadequados ao contexto das ações realizadas num dado momento, o que pode evitar que o usuário gere erros de execução pela introdução de um comando inválido (por exemplo, invalidar o comando Reduzir Zoom, quando a redução chegou ao seu limite);
- ***fornecimento de poder de interação ao usuário***, permitindo que ele controle a navagação aolongo do software, eliminando ações de comando e recuperando-se de erros sem a necessidade de abandonar o software;
- ***minimizar o esforço do usuário para a entrada de dados*** (evitar que o usuário tenha de digitar unidades na entrada de grandezas físicas, utilizando uma unidade default modificável; eliminar a necessidade de digitação de ",00" quando os valores não contiverem componentes decimais, etc...).