SBA: Controle & Automação, Vol. 1, Nº 3, pp. 194-206

AUTOMAÇÃO DO DESENVOLVIMENTO DE SOFTWARE

PARTE II: ABORDAGENS, AMBIENTES E ARQUITETURAS

Mario Jino
UNICAMP - FEE - DCA
Caixa Postal 6101 - Campinas, SP

Mario Bento de Carvalho
CTI - IA - DEI
Caixa Postal 6162 - Campinas, SP

Caetano Traina Júnior
ICMSC - USP
Caixa Postal 369 - São Carlos, SP

Resumo

Nesta segunda parte são discutidos e apresentados: abordagens emergentes de suporte à automação do desenvolvimento de software; exemplos de ambientes e ferramentas existentes ou em estudos; programas nacionais e internacionais de desenvolvimento de ambientes; e arquiteturas de sistemas automatizados de engenharia de software.

Automation of Software Development

Part II: Approaches, Environments and Architectures

Abstract

Emergent approaches for the support of the automation of software development are presented: examples of environments and tools under development or already implemented are briefly described; Brazilian and international programs for the development of software environments are presented; and architectures for software engineering automated systems are discussed.

1. INTRODUÇÃO

A primeira parte do artigo "Automação do Desenvolvimento de Software" discutiu conceitos fundamentais de engenharia de software, ferramentas e estratégias de desenvolvimento de sistemas automatizados para suporte à produção de software (Jino, Traina e Carvalho, 1987).

Dando continuidade ao tema, nesta segunda parte são discutidas e apresentadas abordagens emergentes de suporte à automação do desenvolvimento de software, dando-se destaque às técnicas de reutilização de software, protomodelagem e Inteligência Artificial. A seguir, são apresentados alguns sistemas que procuram atingir um elevado grau de automatização do processo de produção de software, tais como o sistema DRACO, CAIS, PCTE, etc. A apresentação não é exaustiva nem quanto à totalidade dos sistemas existentes, e nem quanto à cobertura mundial dos vários sistemas envolvidos; porém procura cobrir os principais enfoques que hoje se consideram importantes para a automação do processo de produção de software. Apresentam-se ainda considerações sobre características necessárias a arquiteturas de sistemas que visem automatizar o processo de desenvolvimento de software. Finalmente, são apresentadas considerações sobre a direção futura dos esforços de automação da produção de software.

2. ABORDAGENS EMERGENTES

Nesta seção são apresentadas sucintamente as seguintes abordagens de suporte à automação do desenvolvimento de software: **Protomodelagem** ou **Prototipação de Software** ("Software Prototyping"), **Reutilização de Software** ("Software Reuse") e **Inteligência Artificial**. Aplicação de Protomodelagem e de Reutilização no desenvolvimento de software já se faz há algum tempo; contudo, o reconhecimento de sua potencialidade e os esforços para a concretização de novas técnicas têm se ampliado muito nos últimos anos. No que diz respeito à Inteligência Artificial, a proliferação de sistemas especialistas em vários campos de aplicação estendem-se de uma maneira natural para a engenharia de software. Deve-se ressaltar, contudo, que técnicas destas abordagens e de outras que não são discutidas aqui não são mutuamente excludentes; elas coexistem em muito ambientes de engenharia de software.

Protomodelagem ou **Prototipação** ("Prototyping")

A idéia básica da protomodelagem no desenvolvimento de software vem de outras áreas de engenharia onde o emprego de protótipos para experimentação de conceitos e validação de requisitos do sistema definitivo já é consagrado. Um protótipo de software pode ser definido como um sistema inicial que inclui as características essenciais de um futuro sistema; é intencionalmente incompleto e está sujeito a alterações, expansões e, até mesmo, a ser eventualmente descartado (Naumann e Jenkins, 1982; Sroka e Rader, 1986). O protótipo normalmente não satisfaz todos os requisitos do usuário final; implementa somente os aspectos mais importantes do sistema a ser desenvolvido, mas permite um processo de realimentação do usuário para o projetista antes que o sistema final seja projetado. A protomodelagem, portanto, reforça as interações entre o usuário e o projetista; o usuário utiliza e avalia o protótipo e, nesse processo, identifica problemas e sugere soluções.

Várias vantagens têm sido apontadas, decorrentes da utilização da protomodelagem (Aranha, 1985; Jenkins, 1983; Naumann e Jenkins, 1982):

1) experimentação de idéias sem incorrer em grandes custos;
2) obtenção rápida de um sistema em funcionamento;
3) alto grau de participação do usuário no desenvolvimento do sistema;
4) tempo reduzido de desenvolvimento;
5) meio concreto para o usuário poder compreender e avaliar o sistema proposto;
6) identificação antecipada de problemas, antes da fase de implementação do sistema.

Entre as várias alternativas para a utilização de protótipos, uma é a de não utilizar o protótipo por este ter demonstrado que o sistema a ser desenvolvido é inadequado para os fins propostos. Um protótipo pode ser usado :

1) como sistema piloto para experimentação ou testes - para refinar e/ou ampliar o escopo do sistema, ou para permitir um número maior de usuários;
2) como protótipo inicial de um subsequente processo de protomodelagem - neste caso, critérios deverão ser adotados para estabelecimento do ponto de parada, para limitar o número de revisões das sucessivas versões do protótipo inicial;
3) como auxílio a fases específicas do ciclo de vida de desenvolvimento de software - na especificação dos requisitos do sistema (isto é, na definição do sistema a ser desenvolvido) ou na construção física do sistema (o próprio protótipo evolui até tornar-se o sistema definitivo).

Uma vantagem importante da protomodelagem é a de reduzir o tempo e o custo do processo de aprendizagem tanto do usuário como do analista/projetista em relação ao sistema a ser desenvolvido; normalmente, este processo ocorre ao longo de todo o desenvolvimento do projeto a um custo alto e demandando grande dispêndio de tempo de ambos (Mason e Carey, 1983). Outra grande vantagem é a possibilidade de permitir ao usuário a oportunidade de adquirir confiança com respeito ao sistema computacional, incrementando o seu uso, as interações com o projetista e a busca de novos requisitos; deste modo, cria-se um ambiente participativo e progressista. Um protótipo deve ser projetado para facilitar o processo de aprendizagem. O protótipo deve ser tão realista quanto possível, simples e relativamente fácil de ser criado, aprimorado ou reconstruído, isto é, deve ser econômico em termos de tempo e dinheiro. Um bom protótipo é aquele que tem o balanço certo de simplicidade e realidade dentro de um preço aceitável, e que habilita o projetista a obter informação e o usuário a obter maior confiança.

Um protótipo deve poder ser desenvolvido sem muito esforço, aproveitando ao máximo as facilidades que possam estar disponíveis em um computador. Características consideradas importantes em ferramentas de suporte ao desenvolvimento de protótipos estão listadas a seguir:

1) Interatividade - sistemas interativos estendem a potência aparente dos recursos de processamento da informação, tornando os recursos de computação mais facilmente acessíveis ao usuário.

2) Gestão de Bases de Dados - facilita o projeto e programação de facilidades de manipulação de dados a serem incluídas nos protótipos; dados podem ser reestruturados e a base de dados reorganizada a qualquer momento, permitindo maior agilidade no desenvolvimento de protótipos.

3) Interface Homem/Máquina - aqui incluem-se todas as facilidades amigáveis para a produção de saídas e a inserção de entradas tais como geradores de telas, geradores de relatórios, linguagens de consulta, capacidades gráficas, etc.

4) Linguagens de Altíssimo Nível - linguagens cujos comandos estejam mais próximos da aplicação, geralmente não procedurais, tornam muito mais ágil o desenvolvimento de protótipos.

5) Linguagens Procedurais - processamentos especiais ou aplicações complexas podem tornar necessária a utilização de linguagens procedurais no processo de protomodelagem.

6) Dicionário de Dados - atributos e relações entre as entidades envolvidas em um protótipo devem ser armazenadas para suportar a geração da documentação requerida no processo de protomodelagem.

7) Outras facilidades também importantes são: manipulação de arquivos, avaliação de produtividade, documentação, segurança de dados, recuperação de arquivos, etc.

A Protomodelagem pode ser usada em uma vasta gama de aplicações. Uma área bastante promissora é a de software para funções de gerenciamento, cujas tarefas consistem em planeja-

mento, controle, soluções de problemas e tomadas de decisão, que são exploratórias por natureza. Várias metodologias e o ferramental respectivo de suporte vêm sendo desenvolvidos para dar suporte à protomodelagem. Em geral, estas metodologias estão voltadas para áreas específicas de conhecimento. Como exemplos podemos citar: o ambiente AISPE (Bruno e Marchetto, 1986), voltado para desenvolvimento de software para sistemas de controle de processos, com ferramentas para edição e tradução de Redes PROT (extensão de Redes de Petri) e edição de programas associados a Redes PROT; e a metodologia USE (Wasserman e Shewmake, 1982; Warserman e outros, 1986), apropriados para o desenvolvimento de Sistemas de Informação Interativos, com ferramentas para construção de telas, RAPID, interpretação de diagramas de transição, TDI, e gestão de bases de dados, TROLL. As **Linguagens de Quarta Geração**, apesar de não terem sido desenvolvidas especificamente para desenvolvimento de protótipos, constituem um ferramental adequado à protomodelagem de sistemas de informação, pois permitem a criação relativamente rápida de protótipos; este tipo de linguagem, voltado para domínios ou aspectos específicos de aplicação, normalmente incorporam vários recursos como: sistema de gestão de base de dados relacional, recursos gráficos, geradores de aplicações, linguagem de consulta e linguagens de programação de altíssimo nível.

Reutilização de Software ("Software Reuse")

A reutilização de software parte do conceito de desenvolver sistemas a partir de módulos básicos que são conectados para formar módulos maiores, eliminando, desta forma, a necessidade de desenvolver todos os módulos desde o princípio; tenta-se usar a maior quantidade possível de software existente. O objeto da reutilização é qualquer informação que um analista/projetista possa necessitar no processo de criação de software (Freeman, 1984).

Várias técnicas de reutilização de software podem ser aplicadas: bibliotecas de rotinas, programação orientada por objetos, geradores de aplicações, etc. As várias técnicas de reutilização podem ser classificadas em dois grupos: **reutilização por composição** e **reutilização por geração** (Biggerstaff e Perlis, 1984).

Na **reutilização por composição** os elementos reutilizados são blocos elementares, organizados e combinados segundo regras bem definidas. Exemplos: "esqueletos" de códigos, sub rotinas, funções e objetos da linguagem Small Talk. Dois enfoques são dados neste grupo de técnicas:

1) **Reutilização por Biblioteca de Componentes**, no qual dá-se ênfase ao desenvolvimento e acumulação de componentes para constituir bibliotecas aplicativas. Inicialmente, o conceito de componentes reutilizáveis era o de funções simples que podiam ser inseridas em um programa sem necessidade de modificação como: funções trigonométricas, rotinas estatísticas, etc. Lanergan e Grasso (1984) mencionam uma biblioteca de 3.200 módulos escritos em COBOL para propósitos específicos em aplicações comerciais. O conceito de componentes reutilizáveis foi recentemente estendido para incluir funções que não são simplesmente adicionadas ao programa final desenvolvido pelo projetista: **estruturas lógicas** e **abstração de módulos**. **Estruturas lógicas** (Lanergan e Grasso, 1984) são "esqueletos" de programas com várias de suas partes constituintes indefinidas; representam esquemas gerais a partir dos quais trechos de códigos dedicados a aplicações específicas são inseridos para se obter o programa final. Quando os módulos reutilizáveis são **abstrações de módulos**, a biblioteca de componentes não é mais composta por "esqueletos" e módulos padronizados, mas sim por módulos representados em vários níveis de abstração (Matsumoto, 1984); a reutilização dos módulos dá-se em dois níveis: pela adoção direta do módulo já implementado ou pela modificação das abstrações, através de uma particularização da especificação abstrata às novas necessidades.

2) **Reutilização por Princípios de Organização e Composição**, onde procura-se definir os princípios de organização e composição pelos quais os módulos reutilizáveis são combinados para a obtenção do programa final. Três técnicas que se enquadram dentro deste enfoque são: **"pipelines" e filtros; programação orientada por objetos** e **programação parametrizada**. **"Pipelines"** é um mecanismo adotado no sistema operacional UNIX, bastante eficiente para compor programas complexos a partir de elementos simples (Kernighan, 1984); o mecanismo opera pelo redirecionamento da saída de um programa para a entrada de outro, sem a utilização de arquivos temporários; os componentes do "pipeline" são chamados **filtros**. Na **programação orientada por objetos**, todas as operações são executadas através de entidades computacionais chamadas **objetos** (Exemplos: documentos, arquivos, caracteres, etc) (Curry e Ayers, 1984). São associados a cada objeto um estado e um conjunto de operações que podem alterar este estado; um conjunto de objetos que compartilhem um mesmo conjunto de operações constitui uma classe e a ocorrência de um objeto da classe é chamada instância. A programação orientada por objetos permite a geração de subclasses de objetos pertencentes a uma classe mais ampla, através da adição de novas características; as subclasses "herdam" as características definidas para a classe à qual pertencem, ou seja, a definição de uma nova subclasse de objetos não afeta as definições já existentes, caracterizando uma reutilização de software. A estação de trabalho STAR da Xerox (Curry e Ayers, 1984) foi desenvolvida utilizando-se uma técnica para implementar geração de subclasses usando o conceito de "traits" e um conjunto de classes básicas; "traits" são operações primitivas que definem operações ou comportamentos correspondentes a uma classe inexistente, embora possam ser elementos constituintes de outras classes; a partir de um conjunto de opera-

ções ("traits") e poucas classes básicas, sistemas complexos de software podem ser obtidos através das propriedades de especialização e síntese. A idéia básica da **programação parametrizada** é maximizar a reutilizaçao de um programa pelo seu armazenamento na forma mais geral possível. Um novo módulo de programa pode ser construído a partir do antigo apenas pela geração de instâncias de um ou mais parâmetros ou pela substituição dos parâmetros formais do módulo antigo por parâmetros reais. OBJ (Goguen, 1984) é uma linguagem que enfatiza uma rigorosa especificação das interfaces dos módulos e extensa capacidade de parametrização. Três novos conceitos foram introduzidos em OBJ: **teorias**, que declaram propriedades globais de módulos de programas e interfaces; **visões**, que indicam como um dado módulo satisfaz uma determinada teoria; e **expressões** de módulos, uma transformação geral de programas que produz novos módulos pela combinação e modificação de módulos existentes. Com estas ferramentas, a programação parametrizada torna-se uma técnica poderosa para a reutilização segura de programas já existentes.

Na **reutilização por geração**, a reutilização é feita pela reativação do mecanismo de geração do software, e não pelo reaproveitamento de código ou abstrações. Três classes não muito distintas de sistemas e cujas interfaces se sobrepõem dão ênfase a esta linha: **Sistemas Geradores Baseados em Linguagens, Sistemas Geradores de Aplicações** e **Sistemas de Transformação**.

1) **Geradores Baseados em Linguagens** dão grande ênfase na notação, isto é, na linguagem utilizada para especificar o sistema em desenvolvimento. O mecanismo de reutilização baseia-se no fato de que módulos ou programas de alto nível de abstração possuem maior possibilidade de reutilização, com a vantagem de quanto mais abstrato o módulo maior a probabilidade de conter grande quantidade de código e, portanto, maiores níveis de reutilização serão obtidos. A idéia geral é criar uma linguagem de especificação e um gerador que recebe a descrição de um sistema escrita na linguagem de especificação e produz código executável para a solução do problema (Horowitz e Munson, 1984). Um exemplo expressivo desse tipo de sistema é a linguagem de especificação MODEL("Module DEscription Language") e o gerador associado (Cheng, Lock e Prywes, 1984).

2) **Geradores de Aplicações** têm a maior parte das informações específicas sobre um determinado domínio codificadas dentro do próprio programa gerador; isto é, o programa gerador possui conhecimento de várias características do ambiente para o qual foi projetado. Geralmente, possuem interface interativa ou entrada textual limitada. Vantagens deste esquema são a interface não procedural e o ocultamento de vários detalhes de implementação como operações com arquivos, sistemas geradores de bases de dados, etc. O sistema QBE/OBE, da IBM (Horowitz e Munson, 1984) possui capacidade de processamento de texto, correio eletrônico, gráficos e de dados em geral; os objetos da linguagem associada incluem cartas, gráficos, diagramas e documentos audíveis. Outro exemplo de gerador de aplicações sofisticado e complexo é o sistema DRACO (Neighbors, 1984) que é descrito na sequência deste artigo.

3) **Sistemas de Transformação** baseiam-se na obtenção de programas eficientes a partir de especificações operacionais de alto nível. Os programas originais são definidos pela utilização de procedimentos mutuamente recursivos. O sistema de transformação refina cada função sucessivamente, possivelmente introduzindo novas variáveis e estruturas de dados, até atingir a versão final. TAMPR (Boyle e Muralidharan, 1984) é um sistema que converte programas em LISP para FORTRAN, permitindo que o programa em LISP possa ser reutilizado em vários outros ambientes. Cheatham (1984) descreve uma metodologia e um ambiente de programação para a reutilização de programas abstratos escritos em uma linguagem que é uma extensão sintática de uma linguagem base. Transformações de programas são empregadas para refinar o programa abstrato e obter sua versão concreta.

Inteligência Artificial

O objetivo da Inteligência Aritificial é o de "conseguir que máquinas façam coisas que, segundo a concordância de seres humanos, requerem inteligência". Embora esta seja uma definição imprecisa, os trabalhos pioneiros nesta área levaram a uma importante conclusão: "desempenho inteligente requer conhecimento". O problema central da inteligência artificial passou a ser então o de como adquirir, representar, organizar e aplicar conhecimento. A aplicação de técnicas de inteligência artificial à solução de problemas em domínios específicos de conhecimento levou ao desenvolvimento dos chamados sistemas especialistas, incorporando bases de conhecimento sobre campos específicos de aplicação (Mostow, 1985).

Dois tipos de conhecimento distintos podem ser caracterizados no desenvolvimento de software: conhecimento das linguagens e metodologias de desenvolvimento de software, ou seja, **conhecimento do processo** de desenvolvimento de software; e **conhecimento do domínio de aplicação**. Ferramentas de suporte ao desenvolvimento de software, incorporando estes tipos de conhecimento, correspondem a uma aplicação efetiva da inteligência artificial à automação do desenvolvimento de software.

Um conceito central nos principais esforços de aplicação de inteligência artificial à engenharia de software é o de **assistentes de software**, automatizados, capazes de fornecer ao analista/projetista o conhecimento e as informações de que ele necessitar durante o processo de desenvolvimento (Balzer, 1985; Fickas, 1985; Smith, Kotik e Westfold, 1985; Waters, 1985; Barstow, 1985). Estes assistentes de software são programas com acesso a bases

de conhecimento do domínio da aplicação ou do domínio da metodologia que são usadas para direcionar e orientar o trabalho do projetista humano, liberando-o para atividades mais criativas.

3. AMBIENTES E FERRAMENTAS

Nesta seção serão apresentados alguns sistemas que procuram atingir um elevado grau de automatização do processo de produção de Software. A apresentação não é exaustiva nem quanto à totalidade dos sistemas existentes nem quanto à cobertura mundial dos vários sistemas envolvidos; porém procura cobrir os principais enfoques que hoje se consideram importantes para a automação do processo de produção de software. Assim, serão apresentados sistemas que aplicam as várias abordagens descritas na seção anterior e que cuidam de várias facetas do desenvolvimento de software.

Alguns desses sistemas já estão sendo estudados e desenvolvidos há algum tempo, e já existem versões comerciais para os mesmos. Tal é o caso do sistema DRACO (Teichroew e Hershey III, 1977). Outros estão em fase final de experimentação e, embora não disponíveis comercialmente, já existem protótipos em fase avançada de desenvolvimento e uso experimental. Esse é o caso do sistema SIPS.

Os demais são sistemas que se encontram em fases iniciais de desenvolvimento ou ainda em estudos de viabilidade e definição de metas. Esses sistemas correspondem ao resultado de esforços de grande envergadura, frequentemente envolvendo várias nações, para conseguir fazer frente à crescente complexidade de se produzir software. Esses esforços, a despeito da grande demanda de recursos que apresentam, têm sido em muitos casos descritos como vitais para permitir que organizações ou países mantenham a competividade de sua indústria de alta tecnologia.

DRACO

O sistema DRACO (Neighbors, 1984), desenvolvido por um grupo de pesquisadores do Departamento de Ciências da Informação e Computação da Universidade da California em Irvine, apoia-se nos paradigmas de Linguagens de Quarta Geração e Reutilização de Software para facilitar o desenvolvimento de vários sistemas semelhantes entre si. Vem sendo desenvolvido desde o final dos anos 70 e início dos anos 80, existindo atualmente vários sistemas que foram desenvolvidos com seu auxílio.

A filosofia de seu desenvolvimento (Neighbors, Arango e Leite, 1984) parte do pressuposto de que frequentemente é necessário que se produzam muitos sistemas de um mesmo "Domínio de Problemas", gerando então vários sistemas que possuem padrões de análise, projeto e código semelhantes. Sob esse aspecto pode- se perceber como os princípios de reusabilidade podem ser aplicados: o desenvolvimento de um sistema através da produção de módulos bem definidos permite a sua reutilização em outros sistemas semelhantes; o desenvolvimento de um padrão de análise para problemas semelhantes pode permitir sua aplicação em vários casos particulares; o uso de uma planilha de testes pode permitir seu uso em sistemas semelhantes.

Uma pessoa denominada "Analista de Domínio", com experiência no desenvolvimento de vários sistemas num mesmo domínio de problemas, poderia em um dado instante considerar que a classe de domínio desses sistemas esteja razoavelmente bem compreendida e definir então uma linguagem que possa descrever sistemas desse domínio de problemas. Essa linguagem facilita o entendimento e a construção de sistemas semelhantes, pois abstrai da descrição tudo o que é comum a todos os sistemas daquele domínio, e permite que um novo sistema seja descrito apenas através das peculiaridades que apresente.

DRACO oferece recursos para que o Analista de Domínio defina sua linguagem de uma maneira que pode ser manipulada pelo computador, e a seguir solicita que as operações e objetos envolvidos no particular domínio de seu interesse sejam definidos. Essa definição é feita usando a linguagem LISP e, dessa forma, em última análise o que se faz é suprir para o sistema um conjunto de módulos básicos que definem e implementam cada um dos objetos e operações que normalmente ocorrem em sistemas daquele domínio de problemas. A associação entre cada módulo escrito em LISP e a linguagem de descrição de problemas definida pelo Analista de Domínio é feita pela indicação de parâmetros de montagem do módulo que devem ser especificados pela linguagem. Assim, DRACO pode alterar os módulos cada vez que recebe uma nova definição de um sistema através do que é chamado de "transformações de programa fonte para programa fonte" e, dessa forma, gerar programas adaptados para o particular sistema que se pretende desenvolver.

Para que a linguagem, os objetos e as operações que caracterizam sistemas de um determinado domínio sejam obtidos, é que se efetua uma "análise do domínio de problemas", que pode ser vista como uma generalização da tradicional "Análise de Sistemas". Para se conseguir isso, é necessário que dois profissionais participem dessa tarefa: o Analista de Domínio, que é um especialista no desenvolvimento de sistemas daquele domínio, e o "Projetista de Domínio", que é uma pessoa com conhecimento em como traduzir os conceitos obtidos da análise de domínio em uma forma analisável por DRACO. Após isso ter sido feito, DRACO pode agora ser usado por um analista de sistemas para auxiliar a gerar sistemas daquele domínio. Para isso, esse analista deve apenas aprender a usar a linguagem definida (e, claro, ter algum conhecimento sobre o uso de DRACO). Assim, ele irá executar uma análise de sistema tradicional; porém, uma vez tendo completada essa análise, ele irá especificar seu sistema na linguagem construída anteriormente, a qual irá requerer muito menos trabalho do que o normalmente dispendido, uma vez que apenas a parametrização desse particular sistema será necessária.

DRACO irá então produzir o código do sistema descrito, o qual (possivelmente depois de algumas interações) poderá ser instalado no seu ambiente de execução.

CAIS

O projeto CAIS ("Common APSE Interface Set"), APSE = "ADA Programming Support Environments") (CAIS, 1985), iniciado em 1982 pelo Departamento de Defesa dos Estados Unidos, tem o objetivo de contribuir para uma maior inter-operabilidade de bases de dados de aplicação e para a portabilidade de ferramentas de desenvolvimento de software que suportam a programação com a Linguagem ADA.

O projeto CAIS tem como objetivo colocar na linguagem ADA a funcionalidade necessária para implementar ferramentas, para auxiliar o desenvolvimento de software com ADA, e proporcionar o compartilhamento de ferramentas e bases de dados entre ambientes ADA. Para permitir que bases de dados sejam compartilhadas , ao mesmo tempo em que se garante segurança e controle de acesso, o modelo de dados foi definido, baseado no Modelo Entidade-Relacionamento, com a recomendação expressa de que a descrição (esquema) dos dados e as suas instâncias deverão estar separadas das ferramentas que operam sobre elas.

PCTE/ESPRIT

A Comunidade Econômica Européia é a responsável pelo Programa Estratégico Europeu de Pesquisa e Desenvolvimento em Tecnologia de Informação (ESPRIT, 1984). O programa ESPRIT pretende promover pesquisa e desenvolvimento genéricos e competitivos na área de tecnologia da informação através de projetos de cooperação dentro da comunidade européia,que deve financiá-lo parcialmente. O plano de trabalho estabelecido pelo projeto ESPRIT prevê cinco áreas ou subprogramas de importância estratégica: microeletrônica avançada, tecnologia de software, processamento avançado da informação, sistemas de apoio a escritórios e fabricação integrada por computadores ("Computer Integrated Manufacturing").

Dentro dos vários projetos que estão sendo suportados pelo Programa ESPRIT, o PCTE ("Portable Common Tool Environment") (PCTE , 1985) é considerado como uma das cinco principais áreas de trabalho para preservar a competitividade da indústria de tecnologia de informática na Europa. Os diferentes grupos de pesquisa ligados ao programa ESPRIT estabeleceram a necessidade de se dispor de um ambiente comum que pudesse suportar as ferramentas necessárias aos vários projetos, e que fossem disponíveis em diferentes categorias de equipamentos.

Quando pronto, esse ambiente deverá oferecer uma estrutura que suporte o desenvolvimento de um ferramental integrado que atenda variadas metodologias para a produção de software. Para tal, foi estabelecido um núcleo de conceitos baseados numa extensão do Modelo Entidade-Relacionamento (Chen, 1976), e ferramentas genéricas para o desenvolvimento e integração das ferramentas necessárias. A primeira versão desse núcleo do PCTE foi prevista para estar disponível no final de 1986.

Foram estabelecidos objetivos individuais dentro do contexto do projeto, visando aspectos de generalidade, flexibilidade, homogeneidade, portabilidade e compatibilidade.

A homogeneidade entre as ferramentas de um dado conjunto é conseguida em três diferentes níveis:

nível lógico - que corresponde às funções realizadas pelas várias ferramentas, as quais devem manter um objetivo unificado;

nível interno - que corresponde aos objetos e operações manipuladas em cada ferramenta. A representação interna dos dados manipulados deve ser consistente, e mantida pelo núcleo do PCTE;

nível externo - todas as ferramentas mantêm uma interface consistente e uniforme com o usuário.

A portabilidade do sistema deverá ser conseguida nessa primeira versão do PCTE através de seu desenvolvimento apoiado no Sistema Operacional UNIX. Espera-se que no futuro seja desenvolvida uma versão ADA para o sistema.

PROGRAMA ALEMÃO DE DESENVOLVIMENTO DE SOFTWARE

Além de sua participação no Programa ESPRIT, o Ministério para Pesquisa e Tecnologia da Alemanha Ocidental fomenta quatro projetos na área de Engenharia de Software, organizando e coordenando o trabalho conjunto de indústrias e instituições de pesquisa (Abbenhardt e outros, 1986). Os projetos atualmente em andamento contam com a participação de 30 empresas na área de Hardware e Software, e 12 Universidades e Instituições de Pesquisa. Praticamente todas as grandes empresas da indústria alemã de computadores, e várias pequenas empresas estão envolvidas nesse esforço conjunto.

Os projetos têm em vista uniformizar as ferramentas para Engenharia de Software, tanto aquelas ainda em desenvolvimento quanto as já existentes, dos parceiros conjuntos, com a finalidade de construir ambientes de produção de software que apresentem ao usuário uma interface consistente. Isso é necessário para permitir uma utilização mais ágil das ferramentas existentes, bem como facilitar a incorporação de novas ferramentas aos sistemas de produção de software.

Todos os quatro projetos estão sendo desenvolvidos usando o UNIX como sistema operacional. A principal diferença que existe entre eles está na finalidade de cada um:

O Projeto POINTE (Sistema Integrado e Portável de Desenvolvimento) tem uma ênfase no ambiente de produção de software técnico e comercial;

O Projeto PROSYT (Sistema de Engenharia de de Software para Sistemas Distribuídos em Tempo Real) é destinado para as ferramentas de apoio ao desenvolvimento de Hardware e Software de sistemas em tempo real na área técnica;

O Projeto RASOP (Produção Racional de Software) é orientado para sistemas de CAD para software de Micro-Computadores;

O Projeto UNIBASE (Produção de Software usando UNIX) é destinado ao desenvolvimento de software para aplicações comerciais.

O principal fator que motivou a criação desses quatro projetos é que o mercado de ferramentas para o desenvolvimento de software é dominado por ferramentas isoladas. Estas são frequentemente um meio de ajuda muito útil; porém constituem apenas ilhas de resolução de alguns problemas no processo de desenvolvimento de software.

Essa situação não resolve todos os problemas de quem quer produzir software com o auxílio dessas ferramentas, pois necessita-se de mais ferramentas isoladas para concluir e utilizar o que se tem, o que acarreta um maior esforço para a condução do processo como um todo. Existe pois a necessidade de integração entre as ferramentas, que devem poder se comunicar entre si, e apresentar uma forma padronizada de comunicação com o usuário. Além disso, existe o agravamento da situação com a existência de ambientes de produção que necessitam de uma determinada configuração de Hardware, os quais apresentam problemas quanto à portabilidade para os usuários que compram tais sistemas e ficam amarrados a essa configuração.

Os quatro projetos conjuntos adotam a mesma concepção técnica utilizando uma interface padronizada com o usuário, através da adoção de uma tela, utilização de menús, janelas e máscaras padronizados, e a definição da interface gráfica para as ferramentas que usam entrada e/ou saída gráfica. Os componentes dessa tela são básicos para a validação dos objetos de projetos que são manipulados pelas ferramentas, e para a comunicação entre as ferramentas e as ferramentas e a interface Homem / Máquina.

PROJETOS BRASILEIROS DE SISTEMAS DE APOIO AO DESENVOLVIMENTO DE SOFTWARE

No Brasil não existe ainda um esforço concreto de âmbito nacional no sentido de orientar o desenvolvimento de sistemas de apoio ao desenvolvimento de software que sejam criados pelas várias empresas e instituições de pesquisas que trabalham nessa área. No entanto, algumas iniciativas de construção de ambientes integrados têm partido de instituições de pesquisa e governamentais, as quais são descritas a seguir. Além dessas, várias empresas têm desenvolvido ferramentas isoladas de apoio a determinados aspectos do processo de produção de software, algumas já disponíveis para comercialização.

SIPS

O SIPS ("Sistema Integrado para Produção de Software") (Tsukumo e outros, 1985; Traina e outros, 1985; Traina e outros, 1986) vem sendo desenvolvido desde 1984 no CTI - Centro Tecnológico para Informática, um órgão federal para fomentar o desenvolvimento nacional de alta tecnologia na área de informática, com a colaboração de duas universidades. Originalmente concebido como um sistema de apoio ao desenvolvimento de software para sistemas em tempo real, seu escopo de abrangência foi estendido para suportar o desenvolvimento também de sistemas técnicos e comerciais.

O conceito de formulação do SIPS consiste na construção de um conjunto de ferramentas independentes entre si, porém com funções concatenadas, de maneira a atender as necessidades de desenvolvimento de um projeto. A integração entre as ferramentas é conseguida através da padronização das informações armazenadas e manipuladas pelas ferramentas, e pela existência de um núcleo de gerenciamento das informações; este se apóia em um Sistema de Gerenciamento de Bases de Dados especialmente construído, que adota um modelo de representação de dados orientado a objetos. Além disso, todas as ferramentas empregam uma mesma interface de comunicação com o usuário, mantendo, assim, uma coerência entre os comandos das várias ferramentas, o que facilita a interação com o usuário.

A existência do núcleo comum de gerenciamento das informações para todas as ferramentas e a interface única de comunicação com o usuário faz do SIPS um sistema integrado de produção de software, ao mesmo tempo em que a independência entre as ferramentas permite a incorporação gradativa de novas ferramentas, tornando-o expansível e capaz de atender a novas necessidades de produção de software.

Atualmente, o SIPS dispõe de um protótipo em estado avançado de desenvolvimento, com o lançamento de uma versão comercial prevista para o último trimestre de 1987, incorporando ferramentas de apoio à Análise Estruturada de Sistemas (Gane e Sarson, 1984). Várias outras ferramentas, apoiando a modelagem conceitual de dados usando o Modelo Entidade-Relacionamento, o projeto de sistemas para tempo real, etc, estão em desenvolvimento.

Tendo sido completado o desenvolvimento do núcleo de gerenciamento de informações e a interface Homem/Máquina do SIPS, a construção de ferramentas pode passar a ser desenvolvida de maneira independente das entidades originais que iniciaram o projeto. Dessa forma, está agora sendo criado um conglomerado de empresas e universidades, de aproximadamente 80 entidades, com o propósito de continuar o desenvolvimento de novas ferramentas e o desenvolvimento de tecnologia que pode ser empreendida com a utilização do SIPS.

ETHOS

No âmbito de um projeto bi-nacional envolvendo o Brasil e a Argentina, para cooperação científica e tecnológica em informática, está em andamento um projeto que visa o desenvolvimento de uma Estação de Trabalho Heurística Orientada a Software (ETHOS), isto é, uma estação de trabalho que suporte um ambiente que permita modelar e apoiar o desenvolvimento de software segundo diversas metodologias, incorporando e utilizando-se de bases de conhecimento sobre a aplicação de cada metodologia suportada para auxiliar o projetista na aplicação da metodologia em seu projeto.

A especificação detalhada da estação ETHOS e a escolha das entidades do Brasil e da Argentina que irão participar do projeto deverá ser completada até Dezembro de 1987; está prevista, em uma primeira fase do projeto, a construção de um protótipo de um ambiente de apoio a o desenvolvimento de software, operando como uma ferramenta isolada, até Fevereiro de 1988. A construção de um protótipo da estação ETHOS, operando como um sistema integrado de produção de software está previsto para o final de 1989.

FÁBRICA DE SOFTWARE

O Projeto Fábrica de Software (Fabrica , 1985) é um projeto de caráter nacional que deverá dotar o país de uma moderna tecnologia para produção industrial de software. Seu objetivo é aumentar significativamente a produtividade dos programadores e a qualidade dos programas produzidos pela implantação de metodologias e ferramentas baseadas: na utilização intensiva de técnicas formais de especificação e desenho; em aspectos relativos à gerência da configuração do processo e do produto; na inclusão de métodos quantitativos na formulação de métricas de controle de qualidade; e em técnicas de reutilização de programas.

A fábrica estará operacional dentro de cinco anos, quando empresas e software-houses brasileiras poderão utilizar os resultados obtidos, proporcionando a melhoria de qualidade e a redução dos custos no processo de produção de software.

As entidades envolvidas no projeto Fábrica de Software são: CTI - Centro Tecnológico para Informática, EMBRAPA - Empresa Brasileira de Pesquisa Agropecuária, e o Banco do Brasil.

4. ARQUITETURAS PARA AUTOMAÇÃO

A automação de um sistema de software apresenta alguns problemas comuns, que devem ser resolvidos qualquer que seja o tipo ou a extensão da automação que se deseja. Entre as principais características comuns a todos os sistemas que permitem automatizar o processo de desenvolvimento de software, as seguintes devem ser destacadas:

- As metodologias de desenvolvimento apoiadas devem sofrer adaptações, que permitam a sua automação, bem como devem poder ser integradas de maneira a oferecer ao projetista usuário do sistema um meio contínuo de desenvolvimento de seus sistemas;
- O sistema de apoio ao desenvolvimento usa , manipula e produz informações e deve, portanto, dispor de um Sistema de Gerenciamento de Bases de Dados, com características adequadas;
- A interface com o usuário deve seguir um padrão e ser consistente em todas as atividades que o sistema de desenvolvimento suporta.

Esse conjunto de características nem sempre é conseguido com facilidade, e muitas soluções diferentes têm sido adotadas. Atualmente, com a experiência de desenvolvimento de vários sistemas desse tipo, alguns conceitos podem ser identificados como vitais para a construção de sistemas integrados de apoio à produção de software, os quais serão considerados a seguir.

Automação de Metodologias

Toda ferramenta que automatiza determinado aspecto da produção de software está tecnicamente apoiada em um ou mais métodos de tratamento de atividade de produção de software, ou seja, é centrada em uma determinada metodologia de desenvolvimento. Existem ferramentas que automatizam metodologias originalmente criadas para uso manual, tal como as ferramentas que automatizam Análise Estruturada de Sistemas; e existem ferramentas apoiadas em metodologias desenvolvidas especialmente para o sistema computadorizado que as apóiam, tal como é o caso do sistema EPOS (Lauber, 1983).

Para que uma metodologia seja automatizada, ela deve ser formalizada. O processo de formalização de uma metodologia obriga que o escopo de abrangência dos casos tratados pela metodologia seja rigidamente definido, bem como a maneira como a metodologia é empregada . No caso de metodologias concebidas para uso automatizado, esse processo de formalização ocorre de maneira natural, uma vez que a própria concepção já leva em conta as restrições que a implementação pretendida terá. Além disso , como metodologias desse tipo somente se tornam utilizáveis quando a ferramenta correspondente também o estiver, os usuários somente terão aportunidade de usar a ferramenta quando esta já estiver razoavelmente bem depurada e, junto com a ferramenta que a apóia, constituir um sistema de apoio ao desenvolvimento conceitualmente consistente.

No entanto, muitas metodologias e conceitos de desenvolvimento de software originalmente criados para utilização manual devem poder contar (e os estão obtendo) com recursos automatizados. Nesse caso, o ambiente de ação da metodologia é modificado, e as restrições inevitavelmente impostas pela automatização podem afetar profundamente o uso da metodologia. Isso ocorre porque o ambiente muito mais informal da aplicação manual da metodologia permite que situações não previstas na definição original sejam suplementadas pelos próprios usuários da metodologia. Isso não é possível em um ambiente automatizado.

Além disso, se um ambiente integrado de produção de software tiver que suportar várias metodologias (e frequente é o caso, visto que uma mesma metodologia não é igualmente eficiente para atender a todos os aspectos da produção de software), devem ser criadas interfaces entre as várias ferramentas, que permitam a sua integração. Para que um sistema forneça um apoio integrado às várias fases do processo de produção, as metodologias suportadas pelo sistema devem estar adequadamente integradas.

Dessa forma, deve ser criado um modelo de formalização de metodologias que permita a formalização de todas as metodologias que deverão ser suportadas por um dado sistema. Atualmente está bem claro que a modelagem conceitual de metodologias é tão bem atendida pelo Modelo Entidade-Relacionamento quanto o é a modelagem conceitual de dados. Assim, o ME-R, que originalmente foi criado para a modelagem de dados, tem sido amplamente usado para a modelagem também de metodologias. As extensões que têm sido feitas ao modelo original são bastante adequadas à modelagem de metodologias, especialmente aquelas que procuram obter um modelo orientado a objetos.

A dualidade da capacidade de representação de dados e metodologias do ME-R é muito importante, pois ferramentas que automatizam metodologias usando uma formalização da metodologia pelo ME-R podem manipular os dados usados pela metodologia modelando-os também segundo o ME-R. Isso facilita a construção da própria ferramenta, ao mesmo tempo que mantém uma saudável consistência entre a representação da própria metodologia e a representação dos dados que manipula.

A Base de Dados de um Projeto

Para que um sistema baseado em computador possa apoiar o desenvolvimento de um projeto é necessário que as informações sobre esse projeto existam em uma maneira processável por computador. Em todos os sistemas de apoio ao desenvolvimento de software de conhecimento dos autores, a armazenagem e tratamento dos dados de projeto são feitos através de um Sistema de Gerenciamento de Bases de Dados (SGBD). Em geral, a implementação de um SGBD é feita de maneira genérica, ou seja, ela não é feita especialmente para o sistema de apoio ao projeto; o que ocorre é que ao ser feito o desenvolvimento de um tal sistema, como ele necessita empregar um SGBD, escolhe-se um que ofereça os recursos considerados necessários para ele. Essa tem sido a solução mais frequentemente adotada.

Porém, não são todos os sistemas que têm usado SGBDs comerciais para a sua implementação. Alguns sistemas de apoio ao projeto e desenvolvimento de software têm se apoiado em SGBDs desenvolvidos especialmente para eles, tal como é o caso dos sistemas PSL/PSA (Teichroew e Hershey III, 1977) e PCTE.

Na realidade, muitas das características que um sistema de apoio ao projeto apresenta devem-se não ao sistema em si mas ao SGBD que ele emprega. No entanto, durante muito tempo isso passou desapercebido tanto por quem usa como por quem desenvolve esses sistemas; apenas recentemente começou a se reconhecer a necessidade de se desenvolver SGBDs apoiados em modelagens de dados voltadas especialmente para as necessidades de sistemas de apoio ao projeto e desenvolvimento de sistemas em geral e, em particular, de sistemas de software.

É interessante notar como muitas vezes as soluções para determinados problemas são dadas sem que se perceba o seu real significado, ou as implicações dessas soluções, sejam elas benéficas ou prejudiciais. Nesse caso em especial, a solução de se desenvolver modelos de dados especiais para os sistemas de apoio a projetos de software, que agora desponta como uma grande tendência nessa área, foi usada em um dos primeiros sistemas a usar computadores para apoiar o desenvolvimento de software, ou seja, o sistema PSL/PSA.

Esse sistema se apóia internamente em um sistema de gerenciamento de dados desenvolvido especialmente para ele e que fornecia condições para que todas as informações fornecidas pelo usuário através de descrições efetuadas na linguagem PSL (veja a primeira parte deste artigo). Olhando para esse sistema mais de uma década depois de seu desenvolvimento inicial, pode-se perceber que a sua grande força estava efetivamente no fato de ter sido feito apoiando-se em um modelo de dados que se ajustava perfeitamente ao que o PSL/PSA necessitava, e que para isso teve que ser especialmente desenvolvido.

Pode-se perceber hoje que, na época em que o PSL/PSA foi originalmente implementado, os SGBDs existentes não eram capazes de suprir os recursos que o PSL/PSA necessitava, e talvez devido a isso o seu próprio SGBD foi desenvolvido. Porém, uma vez tendo sido implementado apoiado no seu próprio modelo como o foi, o PSL/PSA oferecia características que atendiam a necessidades que efetivamente existiam, e era único na maneira como as atendia. O sistema PSL/PSA tornou-se então um sistema muito popular (a restrição ao seu uso foi efetivamente devida ao fato de requerer um sistema computacional muito grande, e demandar muitos recursos desse sistema, o que restringiu muito a classe de usuários que dele poderiam dispor), e no seu rastro surgiram incontáveis "descendentes".

Com o passar do tempo, avanços na área de sistemas de gerenciamento de bases de dados foram capazes de suprir de maneira natural os requisitos que um sistema tal como o PSL/PSA impunham sobre o seu SGBD. O avanço mais próximo nessa linha foi o surgimento do Modelo Entidade-Relacionamento (ME-R) (Chen, 1976), o qual passou a apoiar grande parte dos "descendentes" do sistema PSL/PSA. O próprio PSL/PSA foi posteriormente caracterizado como um sistema apoiado no ME-R (Teichroew e outros, 1980), apesar de ter sido desenvolvido muito antes que o próprio ME-R fosse formalizado ; com esse reconhecimento o sistema foi capaz de evoluir para um Meta-sistema, o "System Encyclopaedia Management System" - SEMS (Demetrovics, Knuth e Radó, 1982).

Hoje pode-se perceber que a força do PSL/PSA, apesar de não reconhecida na época, estava na modelagem de dados que adotou para o seu desenvolvimento. E isso continua sendo uma verdade hoje, muito embora os modelos de dados que estejam sendo estudados voltem-se para conceitos mais poderosos do que o ME-R.

Uma tendência forte atualmente é o emprego do conceito de modelagem de objetos para

a estruturação de sistemas de bases de dados. Esse conceito, tal como o ME-R, procura identificar os objetos que existem no mundo real, bem como as associações que existem entre eles e as suas propriedades, e representá-los na base de dados através de estruturas de dados que armazenem essas associações e propriedades.

A diferença principal que existe entre a modelagem usando o ME-R e a modelagem usando objetos é a seguinte: o ME-R representa as entidades (que correspondem às associações) sempre através dos valores dos atributos que os definem, e portanto permite o acesso aos dados armazenados através da identificação apenas dos valores dos atributos armazenados, sem se preocupar com o que cada entidade ou relacionamento recuperado realmente representa; já um modelo orientado a objetos tem todos os objetos e seus relacionamentos classificados segundo diferentes tipos, e a identificação de dados sempre está associada ao tipo de objeto que se quer recuperar. A modelagem orientada a objetos é então mais forte porque permite que se faça a recuperação dos dados de uma maneira mais "inteligente", pois não basta apenas reconhecer valores de atributos na base de dados e indicar a que entidade (ou relacionamento) está associado esse valor, mas é importante separar dos dados recuperados dessa forma aqueles que têm significado para o processamento em andamento. Modelos orientados a objetos permitem essa maior especificidade na seleção dos dados.

Tendo sido o primeiro modelo de dados a se preocupar com a representação e manipulação de informações semânticas (significado) sobre os dados armazenados em uma base de dados, o ME-R é frequentemente usado como a base sobre a qual novos modelos, muitos deles orientados a objetos, são desenvolvidos. Isso se deve também ao fato de que o conceito de objetos se desenvolveu paralelamente também como um paradigma de programação, do qual a linguagem de programação SmallTalk-80 (Goldberg,1981), desenvolvida pela Xerox Parc no início dos anos 80, é um dos exemplos mais completos. Como um paradigma de programação, o conceito de objeto apresenta características diferentes, e às vezes conflitantes, em relação às características que lhe são atribuídas como um modelo de dados para o desenvolvimento de bases se dados.

Um exemplo disso é o fato de que em Small Talk-80 (e em outras linguagens de programação orientadas a objetos, tal como Object-LISP) não existe uma representação precisa de associações entre objetos que possam ser estabelecidas através de programação. As associações são definidas rigidamente, intrinsecamente ao modelo, tal como o conceito de que um objeto pode ser composto por outros. Esse fato somente pode ser representado em SmallTalk porque se reconhece na linguagem que objetos podem ser "agregações" de outros. Usando o ME-R, poder-se-ia simplesmente deixar a cargo do "programador" de dados a definição de um conjunto de relacionamentos, por exemplo "Composto por" que indicaria de quais outras entidades uma entidade é composta.

A fixação de quais tipos de associações são contempladas em uma linguagem faz com que a definição da linguagem deva prever a priori todos os tipos de relacionamento que serão necessários às aplicações em que ela será usada, o que tem feito crescer muito a quantidade de linguagens e modelos de dados orientados a objetos, pois cada qual contempla uma determinada forma de se encarar a resolução do problema. Um exemplo que ilustra bem esse fato é o desenvolvimento atual de sistemas de gerenciamento de bases de dados para armazenar documentos de projeto, tal como o sistema MINOS (Christodoulakis, Ho e Theodoridou, 1987). Para efeito de comparação, é interessante notar como Woelk, Kim e Luther (1986) analisam e caracterizam os tipos de associações que um modelo para armazenagem de documentos em geral deve ter.

Um fato negativo que ocorre em decorrência da associação de modelos orientados a objetos com o ME-R é que, em decorrência de normalmente mapear-se fisicamente para o modelo relacional um problema conceitualmente definido usando o ME-R, tem-se com muita frequência procurado fazer o mesmo com modelos orientados a objetos, ou seja, é frequente encontrar-se modelagens orientadas a objetos que são implementadas mapeando-se os objetos para o modelo relacional. Um exemplo disso é o ambiente de desenvolvimento de software ALMA ("Atelier Logiciel sur Machine Abstraite") (Lamsweerde e outros, 1987), que desenvolveu uma modelagem orientada a objetos como uma extensão do ME-R, cuja implementação física, porém, usa o sistema INGRES, um SGBD puramente relacional. Tal como é bem caracterizado por Wiederhold (1986), e é também relatado pelo grupo de desenvolvimento do sistema ALMA, esse enfoque leva a um sistema com muito baixa eficiência, além de impor à sua realização restrições que uma modelagem orientada a objetos se propõe a eliminar, mas que acaba existindo em decorrência de estar apoiada no modelo relacional.

No entanto, o desenvolvimento de um SGBD é uma tarefa bastante complexa, e isso tem limitado o desenvolvimento de SGBDs específicos para muitos sistemas de desenvolvimento de software, que acabam tendo que usar os sistemas disponíveis comercialmente. Não existem disponíveis comercialmente SGBDs apoiados em modelagem orientadas a objetos e, assim, apenas projetos de desenvolvimento muito grandes podem se dar ao luxo de desenvolver seus próprios SGBDs. É o caso no entanto dos sistemas ESPRIT e SIPS.

Para dar suporte aos vários projetos em andamento no programa europeu ESPRIT, foi iniciado o projeto PCTE, o qual deve criar um núcleo sobre o qual as ferramentas desenvolvidas pelos outros projetos irão se apoiar. Para isso foram efetuados estudos para a especificação de um SGBD que pudesse atender às necessidades desse projeto. O resultado foi a especificação de um modelo derivado do ME-R, porém

orientado a objetos, com diversas extensões , principalmente com o objetivo de atender à manipulação de versões, textos longos não estruturados e tratamento de diversas visões de dados pelos vários usuários. A sua implementação está sendo realizada e irá se constituir no sistema Emeraude (Gallo, Minot e Thomas , 1987).

O mesmo ocorre com o projeto brasileiro SIPS, que dispõe de um núcleo de gerenciamento de informações que centraliza todas as atividades de armazenagem e manipulação de informações de todas as suas ferramentas. O modelo de dados usado é o Modelo de Representação de Objetos (Traina, 1986), sobre o qual o Sistema de Gerenciamento de Bases de Dados foi totalmente desenvolvido.

A Interface do Sistema com o Usuário

Um aspecto de vital importância para um Sistema de Apoio Computadorizado ao Projeto e Desenvolvimento de Software é a maneira como esse sistema interage com o usuário. Atualmente têm aumentado muito os recursos de comunicação com o usuário, com o advento de novos dispositivos de Entrada, tais como "mouses" , "tablets", mesas digitalizadoras, etc, e dispositivos de Saída, tais como plotters, unidades de múltiplos vídeos, etc. Além disso, novos conceitos de interação, tais como janelas, menus, edição orientada por sintaxe, etc, têm também contribuído bastante para aumentar o arsenal de recursos que um programador de sistemas dispõe para efetivar a comunicação de um sistema com seus usuários.

Quando um sistema é composto por inúmeras ferramentas, como é o caso dos modernos ambientes integrados de apoio à produção de software, uma atenção especial deve ser dada ao planejamento da comunicação com o usuário, a qual deve ser definida desde as primeiras fases de especificação desses sistemas. Isso porque a variedade de situações em que o sistema será empregado fornece margem a muitas maneiras diferentes de se comunicar com o usuário. No entanto, tanto para facilitar o aprendizado do uso desse sistema, quanto para evitar erros de operação devido a diferentes maneiras de se solicitar uma mesma operação em diferentes ferramentas. Assim, deve existir uma consistência entre todas as ferramentas, no sentido de que a representação de uma mesma informação seja sempre a mesma em todas as ferramentas, os comandos que solicitam determinada ação tenham sempre a mesma estrutura, representação e opções, a forma de comunicação empregada seja sempre a mesma, etc.

5. CONSIDERAÇÕES FINAIS

Neste artigo, dividido em duas partes, são discutidos aspectos considerados relevantes à automação do desenvolvimento de software dentro do escopo abrangido pela especificação de requisitos, especificação funcional, projeto lógico e implementação. Aspectos de desenvolvimento de software, embora de grande importância e passíveis de automação, não discutidos aqui, incluem: manutenção, teste, métrica, gerenciamento de configuração e de projeto, etc.

Dentre as abordagens e sistemas discutidos destacam-se as seguintes observações:

1) Sistemas integrados permitem um controle efetivo da produção de software ao longo do ciclo de vida, contribuindo para uma melhor qualidade do software produzido;

2) Sistemas flexíveis (meta) permitem a coexistência de metodologias e ferramentas diversas; sistemas flexíveis tornam-se integrados quando as metodologias suportadas abrangem as fases do ciclo de vida que o sistema tem por objetivo apoiar;

3) Protomodelagem (prototipação) enfatiza a participação do usuário/cliente na definição do sistema, contribuindo para o desenvolvimento de software cujas características são mais próximas às suas necessidades;

4) Assistentes de software (inteligência artificial) auxiliam o projetista/analista através do conhecimento disponível (domínio da aplicação e domínio do processo de software) para a solução de problemas no desenvolvimento de software;

5) Linguagens de altíssimo nível permitem ao programador/projetista programar/projetar sistemas de software em poucos comandos de uma notação bem próxima ao domínio de sua aplicação;

6) Reutilização de software enfatiza o aproveitamento de componentes prontos ou de processos definidos de geração de componentes, aumentando consideravelmente a produtividade dos programadores/projetistas; a padronização de componentes e a sua organização em bibliotecas é um dos pontos cruciais para a aplicação efetiva da reutilização de software (Prieto-Diaz e Freeman, 1987).

O estado da prática no desenvolvimento de software mesmo nos países mais desenvolvidos, como o Japão e os EUA, mostra uma utilização relativamente pequena de ferramentas, até mesmo das ferramentas manuais cuja utilização comprovadamente contribui para melhorar o processo de produção de software (Zelkowitz e outros, 1984). Ferramentas automatizadas e sistemas integrados são um passo decisivo para mudar esta situação, provendo facilidades mais amigáveis e menos laboriosas que permitirão quebrar a barreira que hoje separa o analista/programador tradicional dos métodos e abordagens em desenvolvimento nas universidades e centros de pesquisa.

6. REFERÊNCIAS BIBLIOGRÁFICAS

Abbenhardt,H.; Abendroth,D.; Niederau,G.; Schneider,M.; Steusloff,H.; Timm,M. & Tontsch, F. (1986). "Ein längeres Leben für Software-Produkte". Computer Magazin, outubro de 1986: 67-78.

Aranha,M.C.L.F.M.(1985). Uma Análise da Aplicabilidade da Abordagem de Construçao de Protótipos ao Desenvolvimento de Sistemas

de Informação. Tese de Mestrado, ICMSC-USP, Sao Carlos.

Balzer,R. (1985). "A 15 Year Perspective on Automatic Programming". IEEE Transactions on Software Engineering, vol.SE-11,nº 11: 1257-1268.

Barstow,D.R. (1985). "Domain-Specific Automatic Programming". IEEE Trans.on Software Engineering, vol.SE-11, nº 11:1321-1336.

Biggerstaff,T.J. & Perlis,A.J. (1984). Special Issue on Software Reusability - Foreword. IEEE Trans.on Software Engineering, vol. SE-10, nº 5: 474-477.

Boyle,J.M. & Muralidharan,M.N. (1984). "Program Reusability Through Program Transformation". IEEE Trans. on Software Engineering, vol. SE-10, nº 5: 574-588.

Bruno,G. & Marchetto,G. (1986). "Process-Translatable Petri Nets for the Rapid Prototyping of Process Control Systems". IEEE Trans. on Software Engineering, vol.SE-12, nº 3: 346-357.

CAIS (1985). DoD Requirements and Design Criteria for the APSE Interface Set (CAIS). Relatório do Ada Joint Program Office, Washington, D.C.

Cheathan Jr.,T.E. (1984). "Reusability Through Program Transformation". IEEE Trans.on Software Engineering, vol.SE-10, nº 5: 589-594.

Chen,P.P. (1976). "The Entity-Relationship Model - Toward a Unified View of Data". ACM Trans.on Database Systems, vol.1, nº 1 : 9-36.

Cheng,T.T.; Lock,E.D. & Prywes,N.S. (1984). "Use of Very High Level Languages and Program Generation by Management Professionals". IEEE Trans. on Software Engineering, vol.SE-10, nº 5: 552-563.

Christodoulakis,S.; Ho,F. & Theodoridou,M. (1986). "The Multimedia Object Presentation Manager of MINOS: A Symmetric Approach". Proc.SIGMOD-85 Conference, também ACM SIGPLAN Notices, vol.21, nº12:295-310.

Curry,G.A. & Ayers,R.M. (1984). "Experience with Traits in the Xerox Star Workstation". IEEE Trans. on Software Engineering, vol. SE-10, nº 5: 519-527.

Demetrovics,J.; Knuth,E. & Radó,P. (1982). "Specification Meta Systems". IEEE Computer, vol. 15, nº 5: 29-35.

ESPRIT (1984). ESPRIT 1985 WORKPLAN. Plano de Trabalho, Commission of the European Communities, submetido a The Council of the European Communities, adotado através de Draft Council Decision, Bruxelas, 1984.

Fábrica (1985). Projeto Fábrica de Software-Documento de Divulgaçao, CTI- Centro Tecnológico para Informática.

Fickas,S.F. (1985). "Automating the Transformational Development of Software". IEEE Trans. on Software Engineering, vol.SE-11, nº 11: 1268-1277.

Freeman,P. (1983). "Reusable Software Engineering: Concepts and Research Directions". Tutorial on Software Design Techniques, Freeman,P. & Wasserman,A.I. (Eds.), IEEE Computer Society, 4th Edition: 63-76.

Gallo,F.; Minot,R. & Thomas,I. (1987). "The Object Management System of PCTE as a Software Engineering Database Management System". Proc. ACM SIGSOFT/SIGPLAN Software Development Environments, Palo Alto,California, em ACM SIGPLAN Notices, vol. 22, nº 1: 12-15.

Gane,C. & Sarson, T.(1984). Análise Estrutura da de Sistemas. Rio de Janeiro, Livros Técnicos e Científicos Editora S.A.

Goguen,J.A. (1984). "Parameterized Programming" IEEE Trans. on Software Engineering, vol. SE-10, nº 5: 528-543.

Goldberg,A. (1981). "Introducing the SmallTalk -80 System". Byte, vol. 6, nº 8: 14-26.

Horowitz,E. & Munson,J.B. (1984). "An Expansive View of Reusable Software". IEEE Trans. on Software Engineering, Vol. SE-10,nº 5: 477-487.

Jenkins,A.M. (1983). "Prototyping: A Methodology for the Design and Development of Applications Systems". Indiana University, Discussion Paper, April 1983.

Jino,M.; Traina Jr.,C. & Carvalho, M.Bento de (1987). "Automação do Desenvolvimento de Software - Parte I: Conceitos Fundamentais, Ferramentas e Automação". Revista SBA: Controle & Automação, Vol. 1, nº 2 : 99-109.

Kernighan,B.W. (1984). "The UNIX System and Software Reusability". IEEE Trans.on Software Engineering,vol.SE-10, nº5: 513-518.

Lamsweerde,A. van; Buyse,M.; Delcourt,B.; Delor, E.; Ervier,M.; Schayes,M.C.; Bouquelle,J. P.; Champagne,R.; Nisole,P. & Seldeslachts,J. (1987). "The Kernel of a Generic Software Development Environment". Proc. ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Palo Alto, California, em ACM SIGPLAN Notices, vol.22,nº1:16-26.

Lanergan,R.G. & Grasso,C.A. (1984). "Software Engineering with Reusable Design and Code". IEEE Trans. on Software Engineering, vol. SE-10, nº 5: 498-501.

Lauber,R. (1983). EPOS PRIMER - A Short introduction to the EPOS system (Engineering and Project-management Oriented Support system). Institute for Control Engineering and Process Automation of the University of Stuttgart e GPP.

Mason,R.E.A. & Carey,T.T. (1983). "Prototyping Interactive Information Systems". Comm.of ACM, vol. 26, nº 5: 347-354.

Matsumoto,Y. (1984). "Some Experience in Promoting Reusable Software: Presentation in Higher Abstract Levels". IEEE Trans. on Software Engineering, vol.SE-10,nº5:502-513.

Mostow,J. (1985). Special Issue on Artificial Intelligence and Software Engineering - Foreword: What is AI ? And What Does It Have to Do with Software Engineering ?. IEEE Trans. on Software Engineering, vol. SE-11, nº 11: 1253-1256.

Naumann,J.D. & Jenkis,A.M. (1982). "Prototyping: The New Paradigm for Systems Development". MIS Quarterly, Sept.1982: 29 - 44.

Neighbors,J.M. (1984). "The DRACO Approach to Constructing Software from Reusable Components". IEEE Trans. on Software Engineering, vol. SE-10, nº 5: 564-574.

Neighbors,J.M.; Arango,G. & Leite,J.C. (1984). DRACO 1.3 Users Manual, Dept.Inform.Comp. Sci., Un. California, Irvine, Tech.Rep. TR 230.

PCTE (1985). A Basis for a Portable Common Tool Environment. Relatório Preliminar , BULL, GEC, ICL, NIXDORF COMPUTERS AG, OLIVETTI SA, SIEMENS AG.

Prieto-Diaz,R. & Freeman,P. (1987) "Classifying Software for Reusability". IEEE Software, vol. 4, nº 1: 6-16.

Smith,D.R.; Kotik,G.B. & Westfold,S.J. (1985). "Research on Knowledge-Based Software Environments at Kestrel Institute".IEEE Trans. on Software Engineering, vol.SE-11, nº 11: 1278-1295.

Sroka,J.M. & Rader, M.H. (1986). "Prototyping Increases Chance of Systems Acceptance". Data Management, Mar.1986: 12-19.

Teichroew,D. & Hershey III,E.A. (1977). "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems". IEEE Trans. on Software Engineering, vol. SE-3, nº 1: 41-48.

Teichroew,D.; Macasovic,P.; Hershey III,E.A. & Yamamoto,Y. (1980). "Application of the Entity-Relationship Approach to Information Processing Systems Modeling". in Entity-Relationship Approach to System Analysis and Design, P.P. Chen (Ed.) North-Holland Publishing Co.: 15-38.

Traina Jr.,C. (1986). Máquina e Modelo de Dados Dedicados para Aplicaçoes de Engenharia, Tese apresentada ao IFQSC-USP para obtenção do Título de Doutor, São Carlos, Dezembro 1986.

Traina Jr.,C.; Capretz,L.F.; Carvalho,M.B.; Jino,M. & Capretz,M.A.M. (1985). "Gerenciamento de Configuração de Software Usando o Modelo Entidade-Relacionamento". Anais do II CONAI, SUCESU, São Paulo: 387-395.

Traina Jr.,C.; Akhras,F.N.; Capretz,L.F.; Carvalho,M.B.; Jino,M. & Capretz,M.A.M. (1986). "SIPS - Estado Atual de Desenvolvimento". Anais do XIX Congresso Nacional de Informatica, SUCESU, Rio de Janeiro: 297-306.

Tsukumo,A.N.; Traina Jr.,C.; Sampaio,C.B. Capretz,L.F.; Carvalho,M.B.; Jino,M. & Capretz,M.A.M. (1985). "Um Sistema Expansível para Produção de Software". Anais do II CONAI, SUCESU, São Paulo: 379-386.

Wasserman,A.I. & Shewmake,D.T. (1982). "Rapid Prototyping of Interactive Information Systems". ACM SIGSOFT, vol. 7, nº 5, Dec. 1982.

Wasserman,A.I.; Pircher,P.A.; Shewmake,D.T. & Kersten,M.L. (1986). "Developing Interactive Information Systems with the User Software Environment Methodology". IEEE Trans. on Software Engineering, vol.SE-12, nº 2: 326-345.

Waters,R.C. (1985). "The Programmer's Apprentice: A Session with KBEmacs". IEEE Trans. on Software Engineering, vol.SE-11,nº 11: 1296-1320.

Wiederhold,G. (1986). "Views, Objects, and Databases". IEEE Computer, vol.19, nº 12: 37-44.

Woelk,D.; Kim,W. & Luther,W. (1986). "An Object-Oriented Approach to Multimedia Databases". in Proc. SIGMOD-86 Conference, também ACM SIGPLAN Notices, vol. 21, nº12: 311-325.

Zelkowitz,M.V.; Yeh,R.T.; Hamlet,R.G.; Gannon, J.D. & Basili,V.R. (1984). "Software Engineering Practices in the U.S. and Japan". IEEE Computer, vol. 17, nº 6: 57-66.