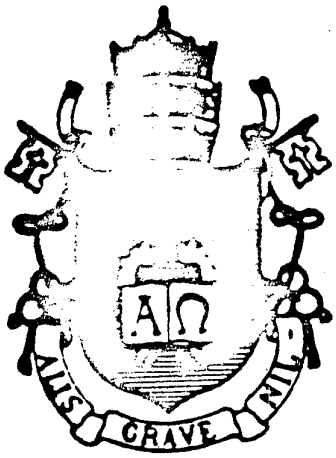

PUC

Série: Monografias em Ciência da Computação,
No. 18/91

A ANATOMIA DE UM AMBIENTE DE DESENVOLVIMENTO DE SOFTWARE

Carlos J. P. Lucena
Eduardo Tadao Takahashi

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP-22453
RIO DE JANEIRO - BRASIL

PUC RIO - DEPARTAMENTO DE INFORMÁTICA

Série: Monografias em Ciência da Computação, No. 18/91
Editor: Carlos J. P. Lucena Setembro, 1991

A ANATOMIA DE UM AMBIENTE DE DESENVOLVIMENTO DE SOFTWARE *

Carlos J. P. Lucena
Eduardo Tadao Takahashi **

* Trabalho agraciado com o V Prêmio Nacional de Informática - Categoria Software, de 1991.

Trabalho escrito com a colaboração de P. Savadovsky e o suporte e infraestrutura da SID Informática S.A., dentro do Projeto ADES e com o patrocinio parcial da Secretaria de Ciência e Tecnologia da Presidência da República.

** CNPq, Projeto ETHOS.

In charge of publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC Rio - Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453 - Rio de Janeiro, RJ
Brasil

Tel.:(021)529-9386 Telex:31078 Fax:(021)511-5645
E-mail:rosane@inf.puc-rio.br

Resumo:

Este trabalho procura justificar o papel dos ambientes de desenvolvimento de software como única alternativa para a solução do problema da produtividade em software. Procura-se também construir um ambiente-exemplo, que chamamos de ADES, cuja funcionalidade e arquitetura são descritos em um nível de detalhe que permite uma apresentação simulada da sua operação. Ambientes de desenvolvimento, que são sistemas de software tão ou mais complexos do que sistemas operacionais são, em geral, ainda bastante desconhecidos mesmo no âmbito da comunidade técnica de Informática. A simulação da operação do ambiente é apresentada para a produção de uma aplicação particular por uma equipe técnica que utiliza uma metodologia realista. O texto procura deixar claro que a gerenciabiblidade proporcionada por ambientes como o descrito é a única solução concebível para o problema da produção eficaz de sistemas de software.

Palavras-chave:

Engenharia de software, ambientes de desenvolvimento de software, base de dados, objetos, sistema de gerenciamento de janelas, desenvolvimento de software.

Abstract:

This paper attempts to justify the role of software engineering environments as the only alternative to thhe solution of the problem of software productivity. The paper also presents the design of a sample environment called ADES whose functionality and architecture are described at a level of detail that allows the simulation of its operation. Software Engineering Environments which are software systems at least as complex as operating systems are still not familiar to the computer science community. The environment is simulated while used to produce a particular application by a team of specialists that uses a realistic development methodology. The text tries to convey the idea that the manageability provided by environments such as the one described is the only conceivable solution to the problem of effective production of software systems.

Palavras-chave:

Software engineering, software development environments, data bases, objects, windows management system, processes, software development.

Conteúdo

1 Introdução

A situação que vem sendo descrita há vários anos na imprensa especializada e hoje na grande imprensa sob o título "a crise do software" pode ser resumida em três pontos essenciais:

- *o alto custo do software*: o componente software de um sistema computadorizado é responsável, ao longo do ciclo de vida deste sistema (da sua concepção até a sua desativação), por cerca de 90% dos custos;
- *a variabilidade da prática do desenvolvimento de software*: as técnicas de desenvolvimento de software praticadas na indústria (internacionalmente e no país) variam exageradamente e podem estar atrasadas em até 15 anos com relação ao estado-da-arte; em consequência, o desenvolvimento é difícil de administrar e os produtos software variam grandemente em custos, confiabilidade e manutenibilidade;
- *necessidade de uma maior produtividade*: as necessidades de software projetadas para a próxima década indicam que a demanda superará em muito a capacidade de oferta (ex.: deficit projetado de um milhão de especialistas nos EEUU e 600 mil no Japão em 1990); a única saída para a indústria de informática é um aumento substancial da produtividade na produção de software.

A solução de qualquer um dos três problemas acima depende da solução simultânea dos outros dois. Em resumo, é necessário atacar o processo de desenvolvimento de software como um todo.

Desde quase o início da historia do desenvolvimento de software, os esforços foram concentrados nas tarefas associadas a *programação*. Esta visão estreita cedeu lugar à percepção de que fazer um software compreende muito mais do que a produção de código. Inclui, entre outras coisas, a documentação e a gerência do processo de desenvolvimento.

A *produtividade* no desenvolvimento de software é uma função de duas variáveis: *eficiência* e *qualidade*. A eficiência se refere à velocidade com que se pode completar uma determinada tarefa. Gerar 100 linhas de especificação, código fonte ou documentação por dia, por pessoa, é mais eficiente do que produzir apenas 50 linhas. No entanto, isto pode não ser mais produtivo. A

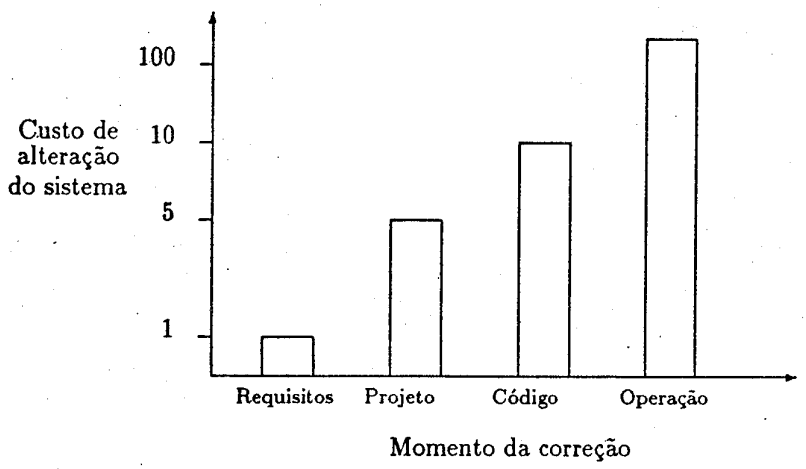

Figura 1: Custo de depuração

razão é que em condições normais, há uma probabilidade maior de que erros ocorram se o processo de desenvolvimento é apenas acelerado. Estes erros que terão de ser corrigidos através de manutenção corretiva, depois que o sistema estiver implementado, são responsáveis por 70% dos custos de toda atividade de desenvolvimento de software. A outra alternativa é corrigir o erro durante o processo de desenvolvimento. Ambas as alternativas requerem o dispêndio de recursos adicionais. Vários estudos mostram [1] que quanto maior o tempo entre a criação de um erro e sua correção, maior o custo da sua correção. O impacto da localização e correção de um erro está representado na figura 1.

A figura 1 mostra que o atraso na identificação e correção de um erro degrada significativamente a produtividade. Por este motivo a produtividade precisa ser medida em termos de eficiência (um fator corrente) e qualidade (uma medida de eficiência futura). Quando ambos os fatores são considerados, fica evidente que um dado nível de produtividade é uma curva e não um ponto [2]. A figura 2 ilustra que, como produtividade é uma curva, o impacto de aumentar-se apenas a eficiência ou apenas a qualidade pode levar à ilusão de que a produtividade está sendo melhorada.

Gerentes de desenvolvimento que buscam aumentar a eficiência podem procurar fazer com que programas, especificações e documentação sejam completados mais rápido. Se eles estão no ponto B da curva (figura 2) e tomam medidas para acelerar o processo e chegar no ponto A eles podem

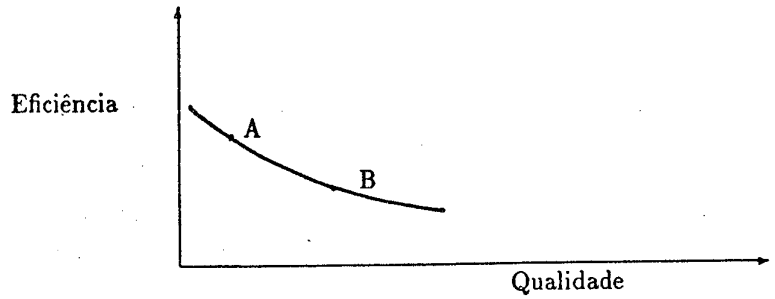

Figura 2: Relação entre eficiência e qualidade

ter a ilusão de maior produtividade (o trabalho gera resultados mais rápido). Freqüentemente, no entanto, a qualidade cai, gerando a necessidade de trabalho futuro, a um custo mais alto. Na verdade, a produtividade líquida não se altera. Por outro lado, a gerência do desenvolvimento pode estar no ponto A e pode tomar medidas para aumentar a qualidade do sistema antecipando o esforço futuro em manutenção. Isto, em geral, significa maior trabalho de revisão ou trabalho executado mais cuidadosamente. Naturalmente, isto implica um gasto maior de tempo no projeto, o que decresce a eficiência e leva para o ponto B. De novo, a produtividade não foi alterada. Negociou-se, apenas, um elemento de produtividade por outro.

Para, realmente, aumentar a produtividade a gerência do processo de desenvolvimento precisa deslocar *toda a curva* aumentando simultaneamente a eficiência e a qualidade conforme ilustra a figura 3.

Um programa gerencial para aumento da produtividade deve-se concentrar nos seguintes itens:

- *medidas*: a produtividade atual deve ser aferida para a avaliação de progressos;
- *qualidade versus eficiência*: o aumento da produtividade deve considerar essas duas variáveis;

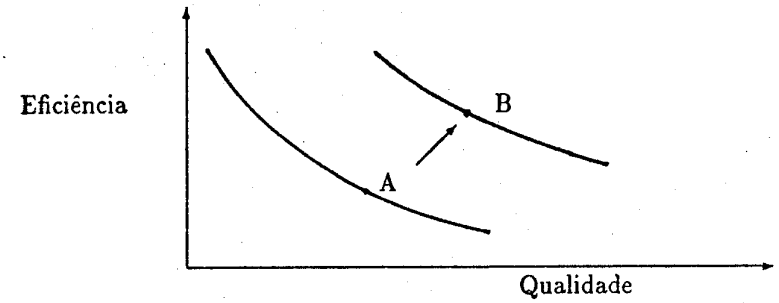

Figura 3: Aumento de produtividade real

- *escopo*: o escopo do aumento da produtividade (e sua aferição) deve incluir todas as fases do processo de desenvolvimento de software, dos requisitos à implementação final do sistema;
- *ferramentas*: as ferramentas de produtividade adotadas devem transcender a fase de programação incluindo, por exemplo, a documentação, especificação, revisão, etc; e devem ser suficientemente flexíveis para suportar os métodos de desenvolvimento adotados na organização.

Historicamente as ferramentas associadas à gerência do desenvolvimento de sistemas eram intituladas métodos para controle do projeto de sistemas. Esses métodos tipicamente previam:

- a especificação para o sistema das tarefas ou atividades a serem executadas durante o projeto
- a definição de dependências entre tarefas
- a definição de estimativas e atribuições relacionadas a cada tarefa
- cálculo do calendário para o projeto
- geração de listas de atribuições

- comparação entre o tempo efetivamente gasto e o projetado nas diversas tarefas

Embora não exista dúvida sobre a validade do uso de métodos para controle do projeto de sistemas, sabe-se hoje que eles não cobrem todas as necessidades da gerência do projeto. Os problemas mais freqüentes são os seguintes:

- a identificação e especificação de todas as tarefas, dependências e estimativas requeridas para um projeto particular consome muito tempo mesmo quando uma metodologia de desenvolvimento de sistemas está em uso;
- os dados reais coletados não correspondem necessariamente a uma porcentagem do trabalho efetivamente completado;
- mudanças no projeto inicial podem ser tão complexas quanto a preparação do projeto inicial.

A solução para os problemas enumerados vem sendo buscada através do uso de métodos automatizados para apoio à gerência, projeto e desenvolvimento de software. Referimo-nos aos chamados *ambientes para desenvolvimento de software* (software engineering environments) ou sistemas CASE (Computer-Aided Software Engineering). Esta tendência fica bem resumida através da sequência de gráficos na figura 4, publicada em Business Week.

Para que as necessidades de gerência de desenvolvimento a que nos referimos sejam satisfeitas um ambiente de desenvolvimento automatizado deve prever:

- a possibilidade de análise da metodologia de desenvolvimento para seleção das tarefas requeridas pelo projeto e o acréscimo de novas tarefas que podem ser únicas em um projeto particular;
- a possibilidade de estimar o esforço de projeto com base em algumas diretrizes e de modificar tais estimativas em função de fatores detetados pelo ambiente e que afetam o projeto;

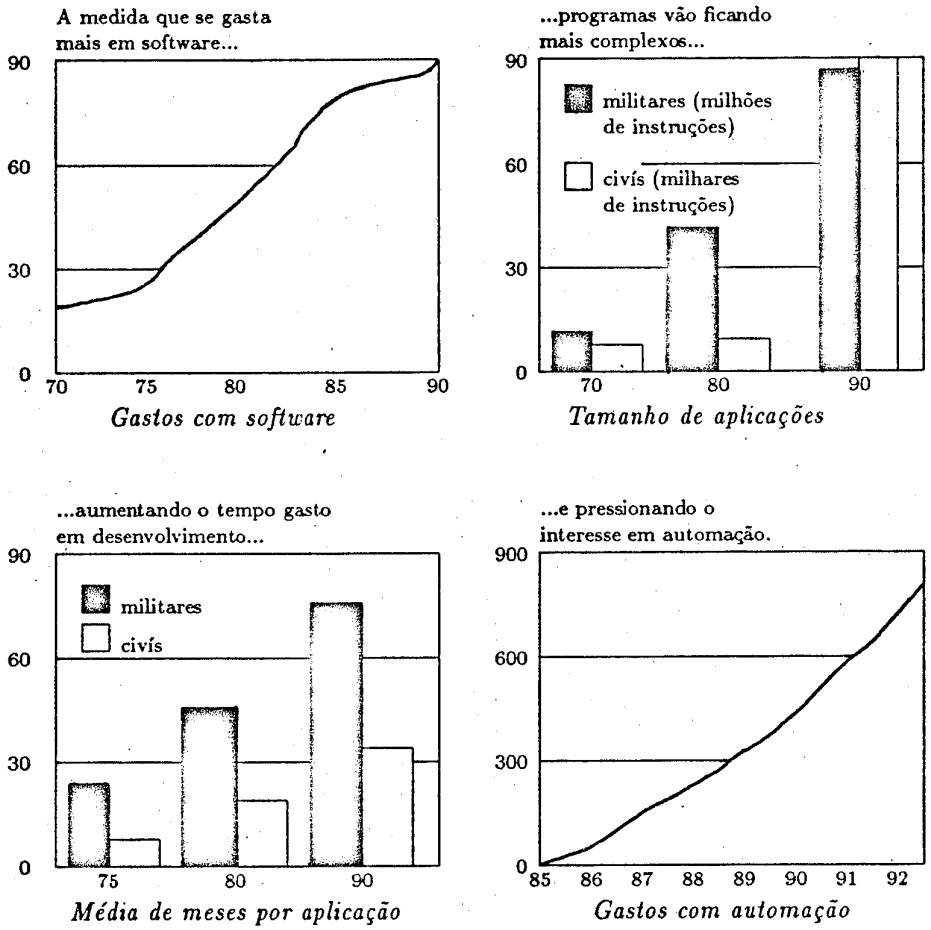

Figura 4: Tendência para a automação

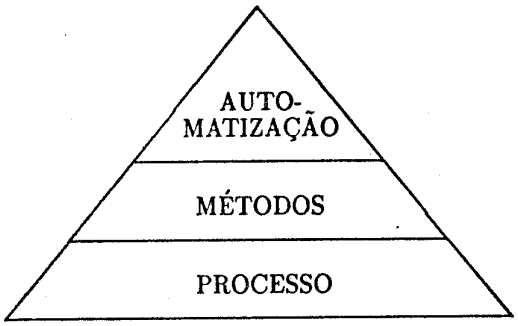

Figura 5: A pirâmide do desenvolvimento

- a possibilidade de controle do fluxo de trabalho para assegurar que tarefas são completadas segundo uma determinada sequência e são revistas por todas as partes necessárias;
- a possibilidade de rápida redefinição de um projeto com base na mudança de escopo, recursos ou atrasos.

O termo *ambiente* quando utilizado em conjunto com a expressão *desenvolvimento de software* tem diversas conotações. Usaremos neste trabalho a definição proposta em [3], segundo a qual um ambiente de desenvolvimento de software "são os *processos*, *métodos* e *automatização* requeridos para a produção de um sistema de software" (figura 5). O propósito de um ambiente é prover o contexto para a evolução racional, ordenada e administrável de um sistema de software face a todos as influências e fatores intangíveis que podem ocorrer. O *processo* de desenvolvimento de um software é o fundamento do ambiente (figura 5). Sua função básica é descrever a cadeia de eventos necessária para a criação de um produto software particular. Os *métodos* incluem tudo o que é necessário para a definição, descrição, abstração, modificação, refinamento e documentação do produto software e são definidos pelo processo de desenvolvimento sob eles (figura 5). O uso do computador para implementar os métodos necessários ao desenvolvimento de um produto software é a *automatização.*

Uma analogia bastante útil pode ser feita com a estruturação de um compilador típico. Na figura 6 [3] ilustra-se a subdivisão do muito conhecido ambiente de compilação nos componentes *processo*, *métodos* e *automatização*.

2 Funcionalidade do Ambiente

Ao escolher, na seção anterior, a definição de ambiente proposta em [3], escolhemos, na verdade, uma das quatro categorias de ambientes existentes, segundo a classificação proposta em [4]. As categorias são as seguintes:

- *ambientes centrados em linguagem*: construidos em torno de uma linguagem e fornecendo um conjunto de ferramentas adequados àquela linguagem;
- *ambientes orientados para estrutura*: que incorporam técnicas que permitem ao usuário manipular estruturas diretamente (independente de linguagem);
- *ambientes do tipo "toolkit"*: que fornecem uma coleção de ferramentas que inclui suporte independente de linguagem para programação em ponto grande (nível de módulos);
- *ambientes baseados em métodos*: que incorporam suporte para um amplo espectro de atividades no processo de desenvolvimento de software, incluindo tarefas como gerência de projeto (programming-in-the-many), especificação e projeto de software.

Desta forma os ambientes baseados em métodos correspondem à definição usada na seção anterior.

Passamos agora a definir a funcionalidade de um ambiente particular, que será estudado nas seções subseqüentes. A funcionalidade é expressa através da definição dos métodos particulares que o ambiente prevê para os três níveis do desenvolvimento de software e o inter-relacionamento entre eles. A figura 7 representa o relacionamento entre os métodos.

Os métodos assinalados com um asterisco podem ser acionados a níveis diferentes, ou seja, são usados desde o nível de gerência para planejamento e controle das atividades da equipe e podem também ser usados no nível de um

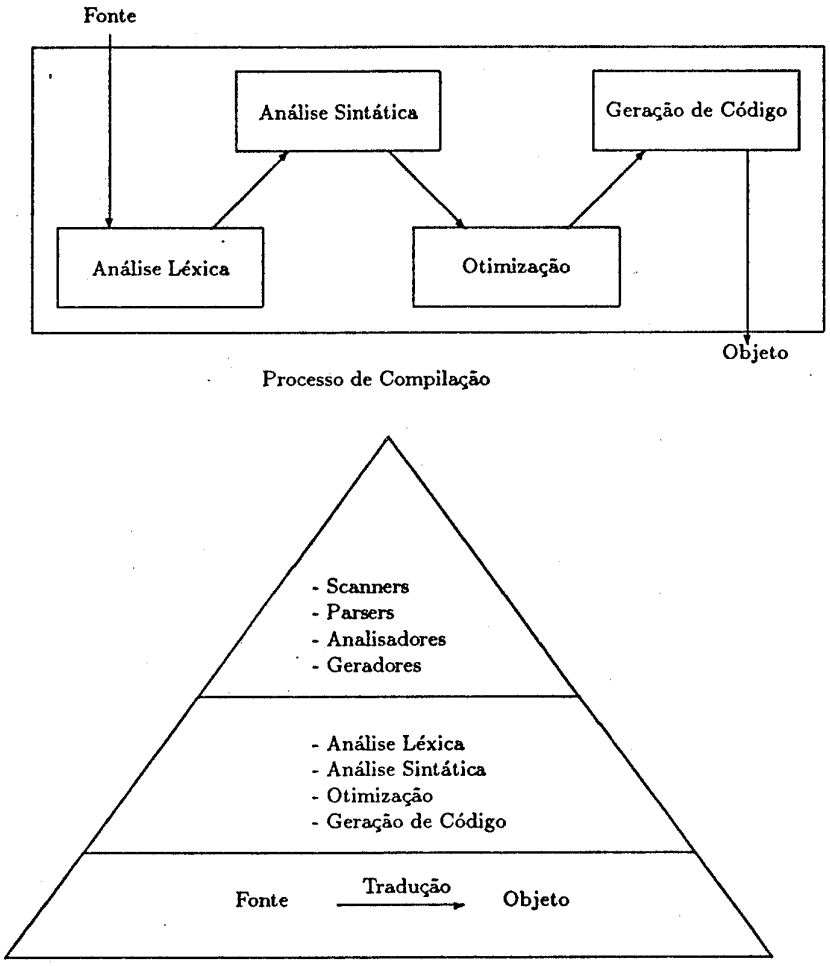

Figura 6: Exemplo da pirâmide de desenvolvimento

programador. Veremos mais adiante que o *processo* subjacente aos *métodos* (figura 5) fica definido quando o gerente do projeto define e aloca as tarefas a serem realizadas.

3 Arquitetura do Ambiente

Nesta seção estaremos preocupados com o nível de *automatização*, ou seja, a definição das ferramentas de software que permitem a aplicação dos métodos.

Antes de esboçar a arquitetura do ambiente-exemplo, faremos algumas considerações preliminares.

Ambientes de desenvolvimento são formados por um grande número de ferramentas que devem atuar cooperativamente. A cooperação depende da existência de alguns mecanismos básicos que a experiência no projeto de ambientes conseguiu identificar. Dois desses mecanismos são:

- um sistema de gerenciamento de janelas sofisticados para uso em um ambiente distribuído
- um sistema de gerenciamento de grandes quantidades de informação de forma consistente.

A arquitetura do conjunto de ferramentas que iremos descrever supõe que sobre um sistema operacional UNIX teremos disponíveis os sistemas X-WINDOWS (desenvolvido no MIT) e o sistema de suporte a bancos de dados não convencional DAMOKLES (desenvolvido na Universidade de Karlsruhe).

Ambos os sistemas foram desenvolvidos a partir de requisitos de apoio à aplicações complexas como são ambientes de desenvolvimento. Revemos a seguir os requisitos que guiaram o projeto de cada um desses sistemas.

X-WINDOWS seguiu o seguinte conjunto de requisitos [5]:

- O sistema deve ser implementável em uma variedade de displays; o sistema deve funcionar com, praticamente, qualquer tipo de "bitmap display" e uma grande variedade de mecanismos de entrada.
- As aplicações devem ser independentes de mecanismos (devices); não deve ser necessário reescrever, recompilar ou mesmo religar uma aplicação para cada novo hardware para "display".

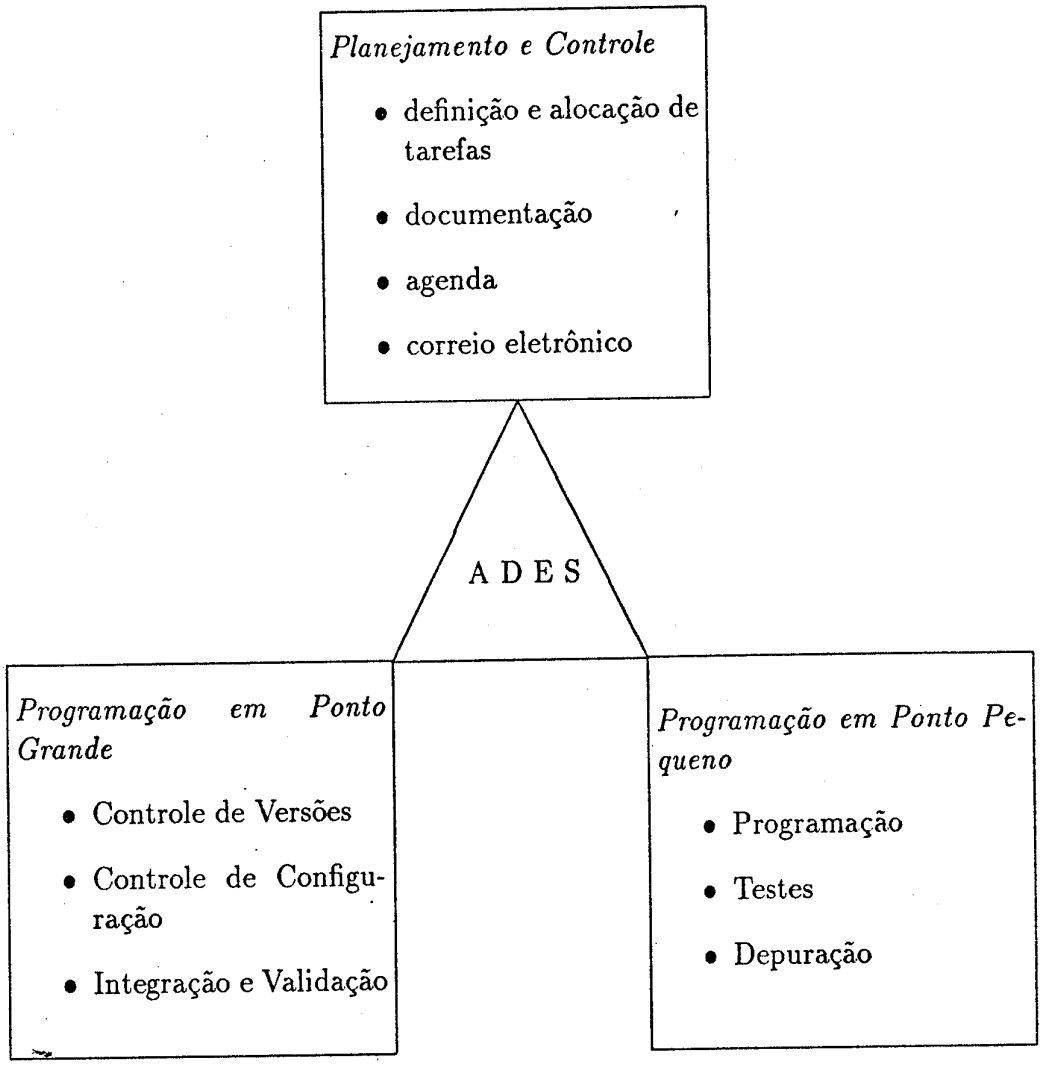

Figura 7: Funcionalidade do ADES

- O sistema deve ser transparente à rede; na aplicação (ferramenta) que esteja rodando em uma máquina deve ser capaz de utilizar o "display"de outra máquina e não deve ser necessário que as duas máquinas tenham a mesma arquitetura e sistema operacional.

- O sistema deve ser capaz de dar suporte a múltiplas aplicações que usam o "display" concorrentemente; por exemplo, deve ser possível mostrar um relógio em uma janela e simultaneamente editar um arquivo em outra.

- O sistema deve ser capaz de dar suporte a várias aplicações e métodos de gerência de interfaces; diferentes comunidades podem e devem utilizar diferentes interfaces com o usuário.

- O sistema deve dar suporte a superposição de janelas, inclusive a saida para janelas parcialmente obscurecidas.

- O sistema deve dar suporte a uma hierarquia de janelas dimensionáveis e uma aplicação deve ser capaz de usar muitas janelas ao mesmo tempo.

- O sistema deve fornecer alto desempenho, suporte de alta qualidade para texto, gráficos sintéticos 2-D e recursos para processamento de imagens.

- O sistema deve ser extensível.

DAMOKLES seguiu o seguinte conjunto de requisitos [6]:

- O modelo de dados para suporte as ferramentas de um ambiente de desenvolvimento deve permitir a declaração e manipulação de objetos complexos (levando em conta suas estruturas internas elaboradas) e relações arbitrárias entre objetos; versões de objetos devem ser apoiadas e um tipo especial *domínio* para representação de informação não estruturada deve complementar os tipos usuais.

- Restrições de consistência complexas que possam ser verificadas em momentos arbitrários devem ser suportadas; em adição, reações a violação de restrições devem poder ser definidas explicitamente.

- Longas transações para modelar unidades de trabalho significativas no processo de desenvolvimento de software devem usar técnicas de sincronização não suspensivas; mecanismos de recuperação especiais devem previnir a perda de resultados durante tais transações, embora a transação possa ainda não estar comprometida.

- Autorização e técnicas de controle de acesso devem ser feitas sob medida para os objetos suportados pelo modelo de dados.

- Bibliotecas de objetos representando projetos pré-definidos, produtos do sistema em desenvolvimento e dados privados dos membros da equipe de desenvolvimento devem ser acessados de maneira uniforme.

- Uma arquitetura de hardware compreendendo uma rede de estações de trabalho e, possivelmente, um servidor de banco de dados deve ser suportada.

- O desempenho de todo o sistema deve ser alto o suficiente para não prejudicar o trabalho da equipe de desenvolvimento.

Baseados no fato de que X-WINDOWS e DAMOKLES satisfazem os requisitos enumerados propõe-se para o ambiente-exemplo a arquitetura da figura 8, na qual as ferramentas implementam os métodos apresentados na seção 2.

No Anexo I apresentamos uma definição breve das diversas ferramentas que constituem a arquitetura.

4 Aspectos Dinâmicos do Ambiente

A arquitetura interna de software do Ambiente é por si só bastante interessante, por seguir um modelo conhecido como "*blackboard*" que encontra utilidade em diversas áreas de aplicação.

Os principais requisitos que qualquer proposta de implementação de um ambiente como o aqui descrito deve satisfazer incluem:

- *extensibilidade*, para permitir a inclusão de novas ferramentas sem comprometer a integridade do ambiente,

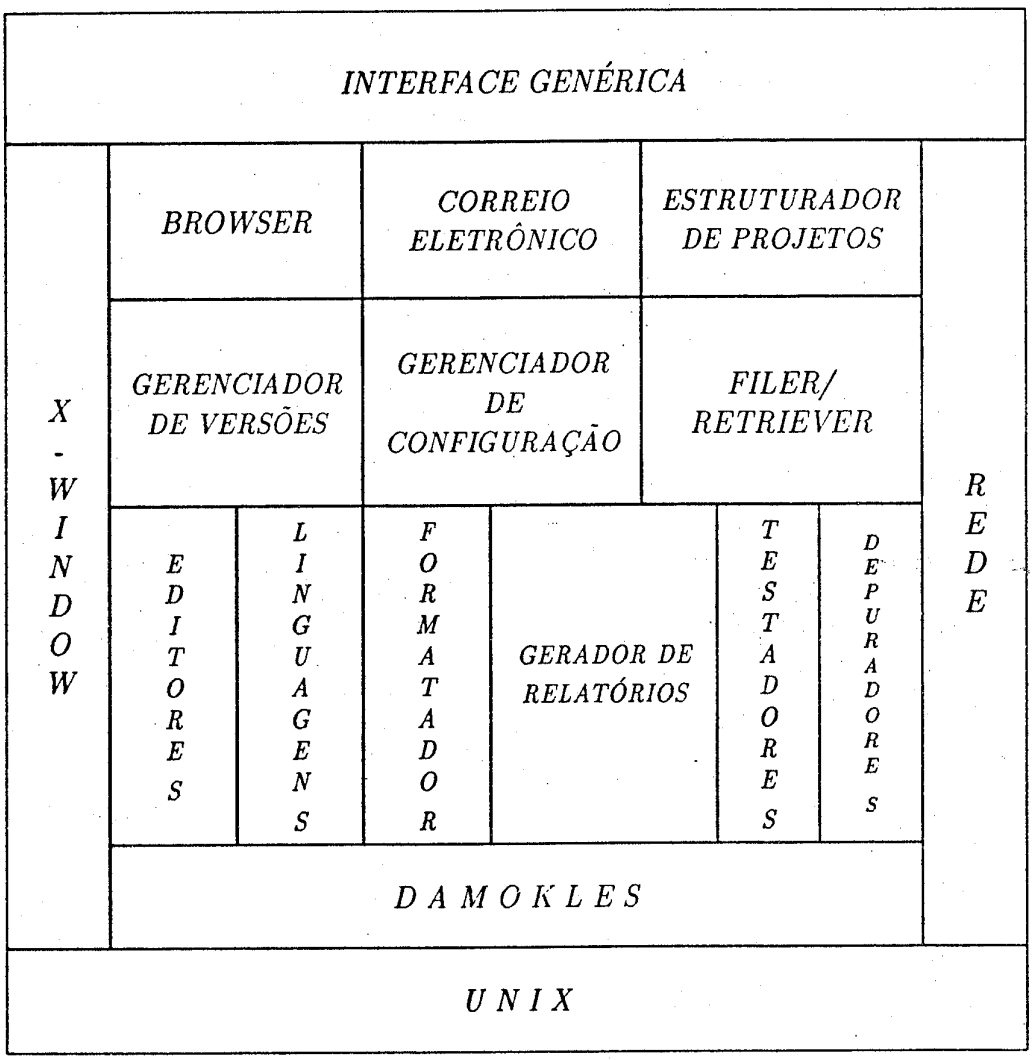

Figura 8: Arquitetura do ADES

- *modularidade*, para garantir que as diversas partes do ambiente tenham uma interface claramente definida entre si e independente de aspectos internos de implementação de cada parte, e
- *flexibilidade*, para permitir que o ambiente se reconfigure ao longo de sua utilização, ajustando-se melhor a idiossincrasias de operação da parte dos grupos de usuários.

O modelo de "blackboard", em sua versão mais simples, opera a grosso modo da seguinte forma:

- *"Blackboard"*: há uma *área comum*, visível a todos, denominada "blackboard", onde informações de interesse global podem ser lidas e escritas por todos. À semelhança de quadros-negros, a leitura de informações pode ser feita simultaneamente por todos, mas a escritura só é feita por um de cada vez.
- *"Knowledges Sources" (KSs)*: há um número indefinido de KSs (fontes de conhecimento), isto é, agentes capazes de processamento autônomo, com memória própria e com mecanismos de inferência/raciocínio próprios, que cooperam entre si na execução de algum trabalho complexo através do "blackboard".
- *Ciclo de Controle*: a execução de um sistema no modelo "blackboard" é constituída por sucessivos ciclos de controle, tais que um ciclo é composto dos seguintes quatro passos:
  - Um KS executa tarefas e registra informações no "blackboard", posicionando-o em um novo *estado*.
  - Os outros KSs examinam o novo estado do "blackboard", e alguns se *candidatam* a executar tarefas no próximo ciclo, em reação a esse novo estado.
  - O mecanismo de controle do "blackboard" arbitra quem executará tarefas no proximo ciclo e alterará o estado do "blackboard".
  - Retorna ao primeiro passo, até encerrar o processamento global.

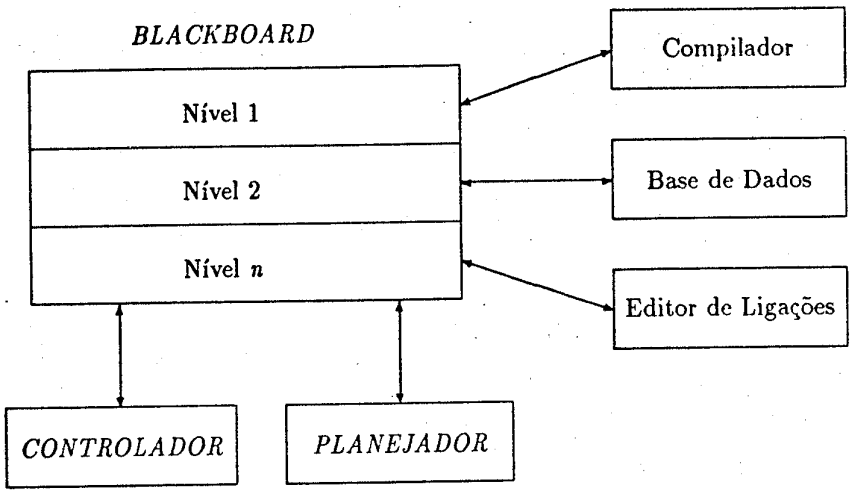

Figura 9: *Blackboards* no ADES

A figura 9 ilustra esse modelo, já aplicado a Ambientes de Desenvolvimento de Software.

Ferramentas típicas como Compiladores, Bases de Dados, Editores de Ligações, etc., são caracterizados de forma natural como "Knowledge Sources" ; o ponto a acrescentar que pode parecer inusitado é que, como KSs, tais ferramentas não ficarão *passivamente* à espera de ativação, mas se candidatarão ativamente a executar tarefas do Ambiente, sempre que o estado do "blackboard" as autorize.

Um outro KS a destacar é o PLANEJADOR, que ausculta o estado do "blackboard", registra os candidatos a execução dentre os KSs, avalia o andamento da operação do Ambiente e, a partir desses dados, propõe sequências de tarefas/alternativas a executar. Observe-se que o PLANEJADOR pode (e idealmente deve) estar baseado em conhecimento sobre o domínio de aplicação do "blackboard" (isto é, neste caso, metodologia de

desenvolvimento de software).

Finalmente, o CONTROLADOR, em sua versão mais simples, é o KS encarregado de decidir, a cada ciclo, quem executará tarefas e atualizará o estado do "blackboard".

A relevância do modelo de "blackboard" para a implementação de ambientes de desenvolvimento de software pode ser resumida nos seguintes pontos:

- atendimento aos requisitos acima mencionados de extensibilidade, modularidade e flexibilidade,
- uma estrutura de controle bastante mais sofisticada e flexível do que, por exemplo, *sistemas de produção* e/ou *sistemas especialistas* tradicionais,
- vasta documentação na literatura técnica [8], especialmente na área de IA Distribuída, propiciando articulação natural com temas de interesse atual como Sistemas Orientados a Objetos,
- variada aplicação em áreas tão diversas como:
  - sistema de compreensão de voz,
  - automação industrial,
  - automação de escritórios.

5 Definição dos Métodos Suportados pelo Ambiente

Nesta seção apresenta-se uma breve definição de cada método utilizado no ambiente-exemplo, ilustra-se a aplicação do método e indica-se que ferramentas promovem a automatização do método.

A apresentação utiliza a terminologia introduzida na figura 7.

5.1 Planejamento e Controle

- *Definição e alocação de tarefas*

- Definição:
  Decomposição de uma atividade em um grafo de sub-atividades onde a cada sub-atividade podem-se associar atributos tais como:
  * relações de precedência parcial ou total
  * relações de sincronismo
  * tempo de duração
  * atribuição de competência
  * direito de acesso
  * restrições de uso do ADES
  * outras (p.ex. documentação pertinente)
- Exemplos de atividades:
  * o projeto completo
  * fase de especificação de requisitos
  * especificação de um módulo
  * codificação
  * teste de um módulo
  * organização de uma reunião de trabalho
- Ferramentas de suporte do ADES:
  * Estruturador de Projetos
  * "Browser"

- *Documentação*
  - Definição:
    Registro automático da história das atividades executadas no ambiente e apoio à definição e geração de documentos associados às atividades.
  - Exemplos de documentação
    * documentação automática
      · "log" de uso do ambiente
      · série de versões de um dado módulo

        - programas fonte e objeto
    * documentação por demanda
        - texto informal
        - programa formatado
        - diagrama de projeto
        - PERT de atividades do projeto
        - formulários e/ou diagramas de requisitos
  - Ferramentas de suporte do ADES
    * Estruturador de projetos
    * "Browser"
    * Editores
    * Formatadores
    * Gerador de Relatórios

- *Agenda*
  - Definição:
    Geração automática de uma lista de atividades pendentes em um dado momento e em diversos níveis, e apoio à definição e ativação de eventos considerados importantes para um dado usuário.
  - Exemplos de atividades da Agenda
    * Atividades automáticas
        - lista de compilações pendentes
        - atividades em atraso
        - tarefas pendentes para todos módulos de uma dada configuração
    * Atividades apoiadas pela Agenda por demanda
        - avisar quando um dado módulo está pronto
        - avisar alguém de um evento significativo para ele (automaticamente detetado ou não)
  - Ferramentas de suporte do ADES

    * Estruturador de projetos
    * "Browser"
    * Correio eletrônico

- *Correio Eletrônico*

  - Definição:
    Disseminar para os usuários toda informação recebida de usuários ou de outras funções (como a Agenda).
  - Exemplos de mensagens:
    * mensagem a quem for o responsável por determinada tarefa
    * mensagem geral (broadcasting)
  - Ferramentas de suporte do ADES
    * Correio Eletrônico
    * Estruturador de projetos

5.2 Programação em Ponto Grande

- *Controle de Versões*

  - Definição:
    Registro de histórico de evolução (ou de alternativas) de entidades[1] do sistema tais como:
    * programas
    * documentos
    * textos e gráficos do projeto
    * pseudo-código e módulos abstratos
  - Exemplos de controle de versões
    * histórico de edições sucessivas de um documento
    * histórico de um programa
  - Ferramentas de suporte do ADES

[1] O tipo de entidades controladas é definido na configuração do ambiente

    * Gerenciador de versões
    * Editores

- *Controle de Configuração*

  - Definição:
    Definição e controle de um conjunto de módulos executáveis que compõem um sistema.
  - Exemplos de controle de configuração
    * versão de aplicativo com área de trabalho maior que a usual
    * aplicativo configurado especialmente para uma dada agência bancária, considerando tamanho da agência, número de transações, etc.
    * aviso ao usuário de que uma configuração usa módulos especiais (não testados, etc)
  - Ferramentas de suporte do ADES
    * Gerenciador de configuração
    * Linguagens
    * Testadores
    * Gerenciador de versões
    * Filer/Retriever

- *Integração e Validação*

  - Definição:
    Teste de grupos de módulos ou do aplicativo completo.
  - Exemplos de integração
    * geração automática de testes para conjuntos de módulos
    * instrumentação de programas e análise dos resultados
    * teste de módulos implementados em conjunto com módulos somente especificados ("stubs")
  - Ferramentas de suporte do ADES

* Testadores
* Gerenciador de configuração
* Linguagens

5.3 Programação em Ponto Pequeno

Consiste no ciclo clássico 'edição-compilação-teste-depuração'. As funções oferecidas para os usuários são:

- *Programação*
- *Testes*
- *Depuração*

As ferramentas de suporte do ADES para programação em ponto pequeno são:

- Linguagens
- Editores
- Testadores
- Depuradores

6 O Ambiente em Operação

Nesta seção apresentamos uma simulação do ambiente-exemplo em operação. Para isto iniciamos supondo uma determinada metodologia de desenvolvimento, a aplicação a ser desenvolvida e o porte da equipe de desenvolvimento. A operação do ambiente é feita através da descrição de eventos que ocorrem enquanto o ambiente é utilizado, seguindo-se as fases previstas na metodologia. Associados aos eventos apresentaremos, na seção 7, telas-exemplos através das quais interagirão os membros da equipe de desenvolvimento. O conjunto de telas selecionado pretende dar uma sensação realística dos recursos fornecidos pelo ambiente-exemplo.

6.1 A Metodologia

Para dar realismo à simulação, escolhemos uma metodologia simples como as que são adotadas pelos principais grupos de desenvolvimento no país.

A metodologia se baseia no modelo de ciclo de vida em fases que é bastante difundido [1]. Observe-se, no entanto, que o ambiente-exemplo não está restrito a metodologias que adiram ao modelo em fases. Cabe aqui uma pequena digressão visando os que não conseguem visualizar um modelo diferente do modelo em fases. O modelo de desenvolvimento em espiral [7], por exemplo, seria também suportado pelo ambiente-exemplo. Nele, cada ciclo de uma espiral começa pela identificação de:

- objetivos da porção do produto que está sendo elaborada (desempenho, funcionalidade, capacidade de acomodar mudanças, etc)
- meios alternativos para implementar esta porção do produto (projeto A, projeto B, etc)
- as restrições impostas sobre a aplicação pelas alternativas (custo, calendário, interface, etc)

O próximo passo é a avaliação de alternativas com relação aos objetivos e restrições. Riscos são identificados e prossegue-se com a formulação de uma estratégia efetiva para equacionar as causas dos riscos. A dinâmica do modelo em espiral se desenvolve conforme está ilustrado na figura 10.

Retornando à metodologia simples que usaremos, fixamos a sua definição em termos de fases, marcos de controle e produtos. A metodologia é, assim, definida como a seguir:

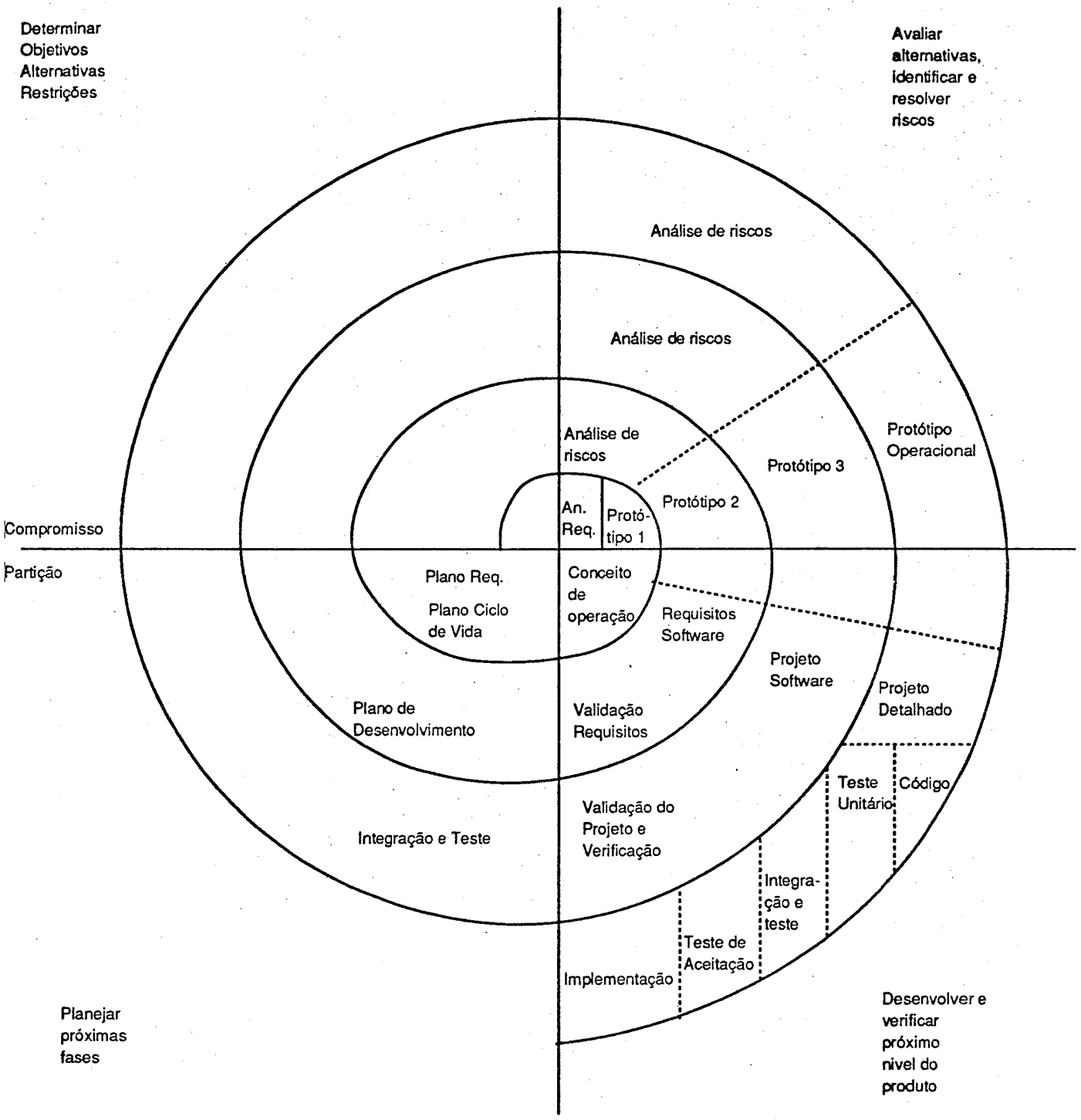

Figura 10: O modelo em espiral

Metodologia Escolhida	
Fases	**Produtos**
Requisitos	*Especificação de requisitos*
Design Global	*Documento geral (informal ou semi-formal: DFD[2])*
Design Detalhado	*Documentação do projeto (informal ou semi-formal: PDL[2])*
Codificação	
Teste	*Programas + doc. sistemas + documentação das falhas*
Configuração/Manutenção	

6.2 A Aplicação e a Equipe

A aplicação escolhida foi *automação bancária.* Aplicações desta área tem como características (1) a necessidade de se produzir um sistema configurável, segundo solicitação de diferentes clientes e (2) utilizar partes (módulos) já existentes (de software e de hardware).

O tamanho estimado do aplicativo é 50K linhas de código e as funções críticas para o seu desenvolvimento são 3, 4 e 5 na metodologia escolhida (eficiência e confiabilidade).

A equipe, no exemplo, será constituida por um coordenador do projeto e mais cinco analistas.

6.3 O ADES em operação

6.3.1 Configurando o ADES para a metodologia

A configuração do ambiente para um certo projeto é feita pelo coordenador do mesmo, baseado em sua experiência e conhecimento dos recursos do ADES. Ela consiste no ajuste do ambiente a uma dada metodologia de desenvolvimento de software e a uma dada equipe de trabalho.

- suporte oferecido pelo ambiente

[2]DFD: Notação de Diagramas de Fluxo de Dados; PDL: pseudo-linguagem de programação (português estruturado).

- definição do formato da documentação (formulários/gráficos)
- catalogação da equipe de projeto (registro individual e definição da hierarquia do grupo)
- definição das etapas do projeto
- associação de funções/tarefas a cada etapa
- atribuição de direitos/responsabilidades aos analistas em cada etapa
- extensibilidade (agregação de novas funções ao ambiente) (p.ex. consistência de um DFD)

• papéis dos elementos da equipe

- coordenador: fazer a configuração do ADES
- analistas: nada

6.3.2 Dos requisitos ao "design" detalhado

Nesta fase os requisitos são definidos informalmente (em equipe), gerando depois uma descrição detalhada do aplicativo.

• suporte oferecido pelo ambiente

- agendar reuniões
- decompor e estruturar tarefas
- edição, formatação, disseminação e controle da documentação (controle de versões)
- visualização geral das tarefas (hierarquia de decomposição de tarefas e relações de dependência entre elas)

• papéis dos elementos da equipe

- coordenador do projeto:
  * atribuir responsabilidades
  * resolver conflitos (atribuições, calendário, recursos e interfaces)
  * "design" global do aplicativo

- analistas de software:
  * "design" detalhado do aplicativo
  * documentação
  * "troubleshooting" (qualquer um pode pedir a convocação de uma reunião)

6.3.3 Codificação e testes

Estas fases se compõem de quatro grandes tarefas relacionadas:

- *programação de módulos a partir de sua especificação em PDL (design detalhado)*
- *teste unitário de módulos*
- *integração de módulos e testes em configurações específicas*
- *documentação de implementação*
- suporte oferecido pelo ambiemte
  - edição
  - programação
  - depuração
  - testes
  - integração e validação
  - documentação
- papéis dos elementos da equipe
  - coordenador:
    * definição de configurações
    * definição e acompanhamento de testes
    * controle geral da execução do projeto (iniciativas autônomas e sob demanda)
  - analistas de software: preparação, documentação e teste de programas

6.3.4 O aplicativo no campo: configuração e manutenção

Nesta fase do projeto temos:

- *Geração de versões específicas do aplicativo desenvolvido para adequar-se aos requisitos de campo de um determinado usuário.*
- *Planejamento e execução de alterações em um aplicativo com versões já em operação no campo, para fins de correção ou modificação funcional.*
- suporte oferecido pelo ambiente:
  - recuperação da história de "design" e de implementação
  - explicitação de dependências (em relação à parte a ser modificada ou da parte a ser modificada)
  - reativação de mecanismos de teste e de integração
  - atualização da documentação
  - geração de configurações para atualização em campo

7 Visualização da Operação do Ambiente

Esta seção complementa a anterior apresentando exemplos da interação dos analistas da equipe com o ADES.

7.1 Estrutura de Menus

Há uma estrutura simples de menus que permite ativar as principais funções do Ambiente.

O primeiro menu, o mais geral, apresenta quatro opções, a saber:

- Administração
- PPG (Programação em Ponto Grande)
- PPP (Programação em Ponto Pequeno)
- Funções Especiais

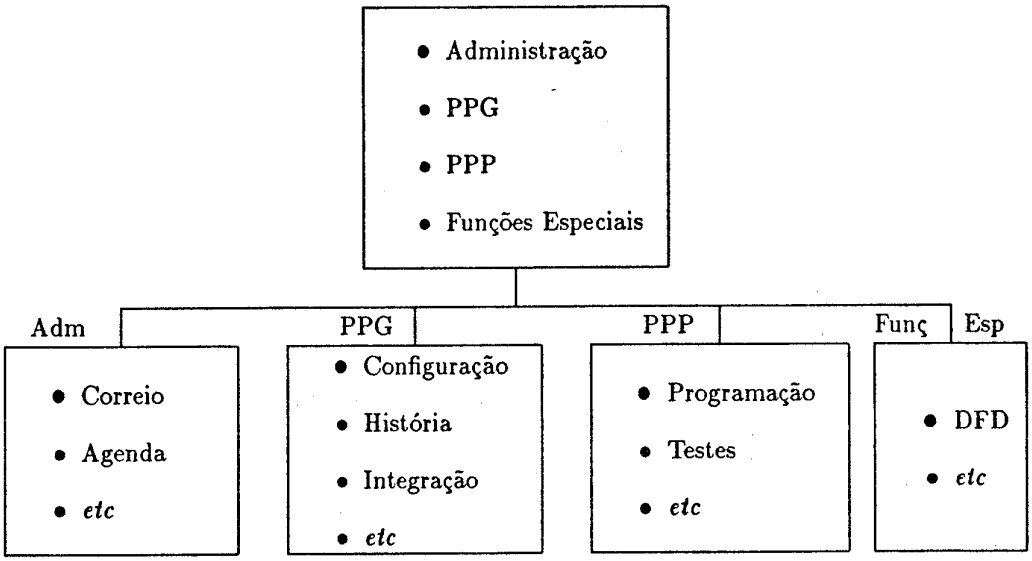

Figura 11: Estrutura de menus

onde a última opção (Funções Especiais) permite estender a funcionalidade do Ambiente segundo o mesmo paradigma de interação das outras funções.

Há no segundo nível quatro menus, que detalham as funções específicas oferecidas a partir de qualquer opção tomada pelo usuário com relação ao primeiro menu.

A estrutura de menus é resumida na figura 11.

7.2 Interação com o usuário

O mecanismo de interação com o usuário é bastante simples. O usuário aponta para determinada posição da tela e ativa janelas (que materializam na tela a ativação de funções do ambiente). Janelas se sobrepõem automaticamente, e podem ser deslocadas a critério do usuário pela tela.

Uma janela tem um *título*, seguido por diversos campos/opções. Uma opção será mostrada como *ativa* quando destacada por um *retângulo* em

negro.

Uma *flecha* mostrará, a cada instante, o ponto da tela/janela presentemente ativo para a próxima função.

7.2.1 Mecanismo de navegação

Consulte a figura 12.

A tarefa de NAVEGAÇÃO foi ativada dentre as possíveis funções de ADMINISTRAÇÃO. No instante mostrado, a usuária "Maria", em sessão do projeto ADES, está examinando o módulo XYZ, em uma versão específica em estado de revisão.

Duas telas mostram, respectivamente:

- a hierarquia de módulos em que XYZ está sendo utilizada, no projeto ADES, e
- um trecho da descrição interna de XYZ, que se revela ser um módulo-fonte em uma linguagem de programação.

7.2.2 Agenda

Consulte a figura 13.

A AGENDA, opção da função de ADMINISTRAÇÃO, foi ativada, para mostrar as atividades pendentes nesse dia para "Maria". (Obs.: Observe que a janela de ADMINISTRAÇÃO foi aberta ou, mais provavelmente, deslocada pela usuária para o canto direito para não congestionar o meio da tela ou para não desaparecer por debaixo da janela de agenda.)

A janela de agenda mostra os compromissos da usuária, basicamente compostos por:

- reuniões
- tarefas técnicas no projeto ADES, e
- mensagens recebidas.

Note que a usuária abriu uma janela de MEMO para lembrar-se de responder à primeira mensagem.

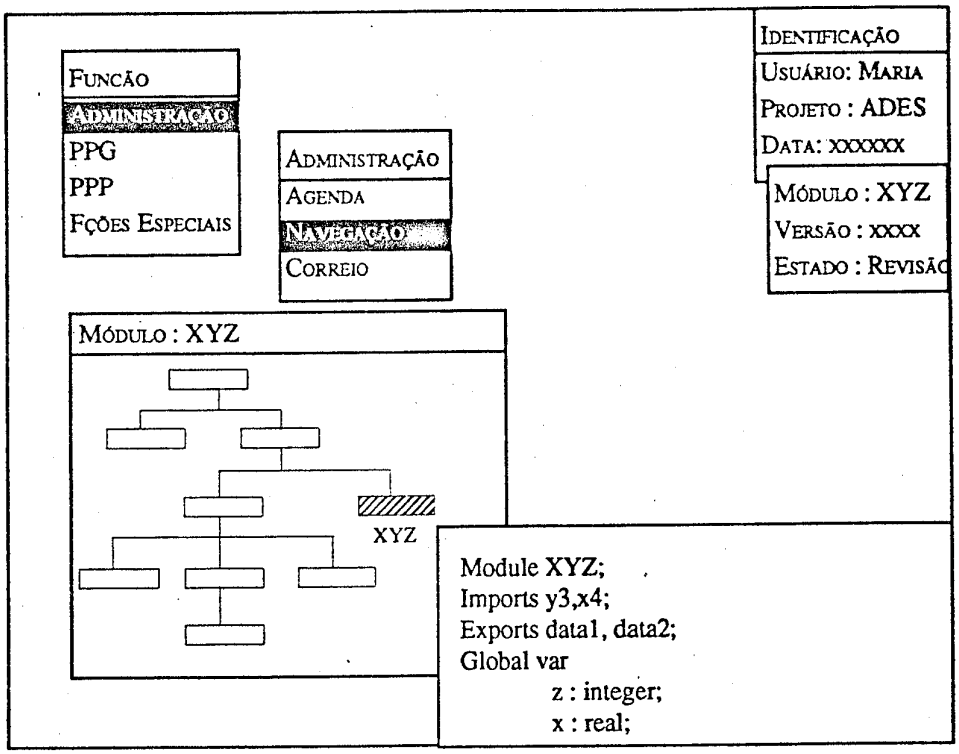

Figura 12: Navegação por uma estrutura de módulos

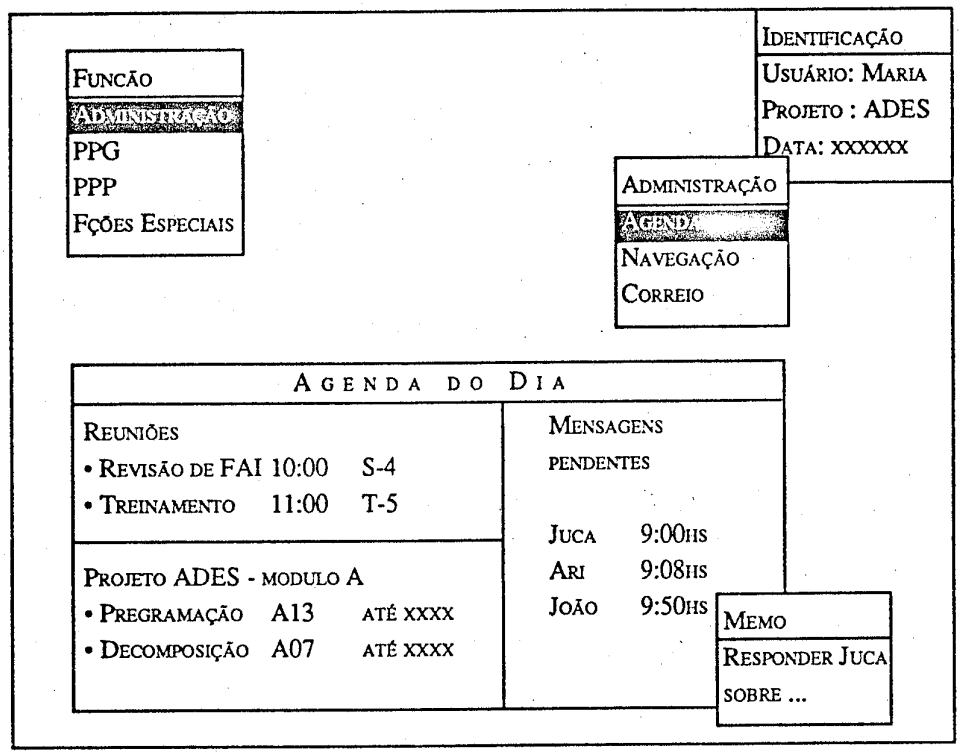

Figura 13: Agenda de um membro da equipe

7.2.3 Dos Requisitos ao "Design" Global

Consulte a figura 14.

Esta tela mostra um instante na utilização de um editor de DFDs, que é ativado como uma *função especial* de PPG (*programação em ponto grande*). Por razões de espaço, o usuário (cuja IDENTIFICAÇÃO não está sendo exibida) empilhou a chamada de funções no canto superior esquerdo, ocupando pouco espaço.

A janela maior, que não é complemente mostrada, mostra parte do DFD correspondente ao módulo ZZ1, dentro do qual um repositório de dados (denominado A) está sendo apontado.

7.2.4 "Design" Detalhado

Consulte a figura 15.

O projeto detalhado do módulo ZZ é mostrado, descrito em uma linguagem semi-livre, a PDL (Program Design Language), que constitui uma espécie de dialeto estruturado de português. Em particular, a linha que principia com *então* é apontada, possivelmente como candidata a alguma modificação.

Uma outra janela mostra a estrutura de módulos em que ZZ está inserido.

7.2.5 Depuração

Consulte a figura 16.

A tela mostra um instante na depuração de XY, um módulo escrito em uma linguagem *a la* Pascal. A janela à direita mostra as opções de depuração ativas nesse instante (a execução-passo-a-passo não está sendo utilizada; prescreve-se a parada da execução do módulo se uma de três condições se materializar). Uma das condições foi presumivelmente satisfeita antes da execução da linha 104, provocando a interrupção do andamento do módulo, e a solicitação de instruções ao usuário acerca do que fazer em seguida.

7.2.6 Configuração

Consulte a figura 17.

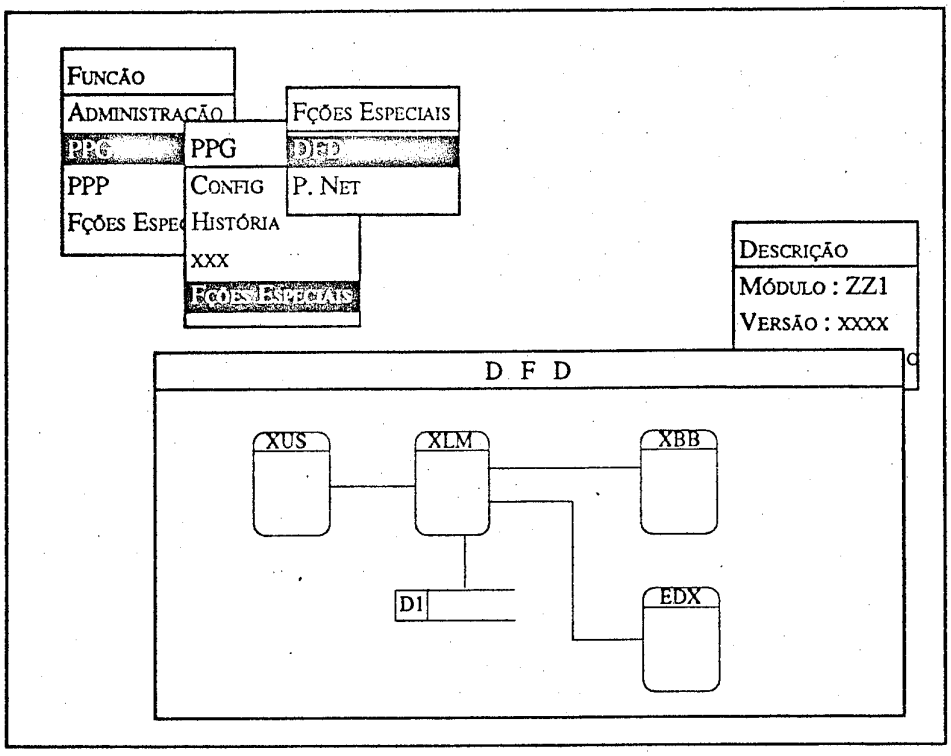

Figura 14: Editando DFDs

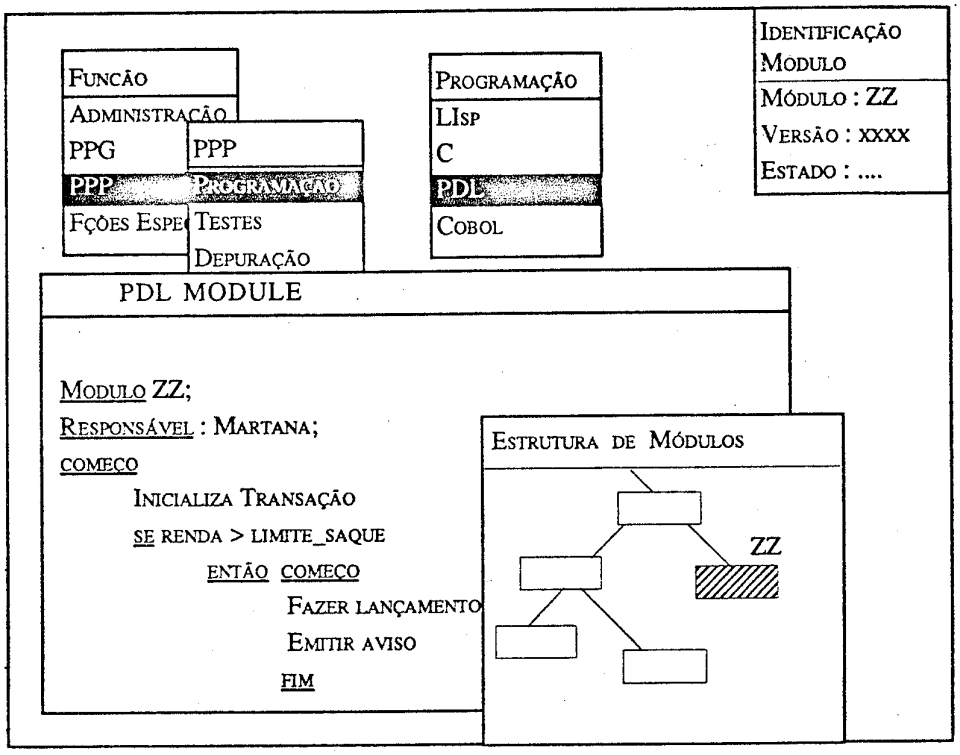

Figura 15: Fazendo o "design" detalhado

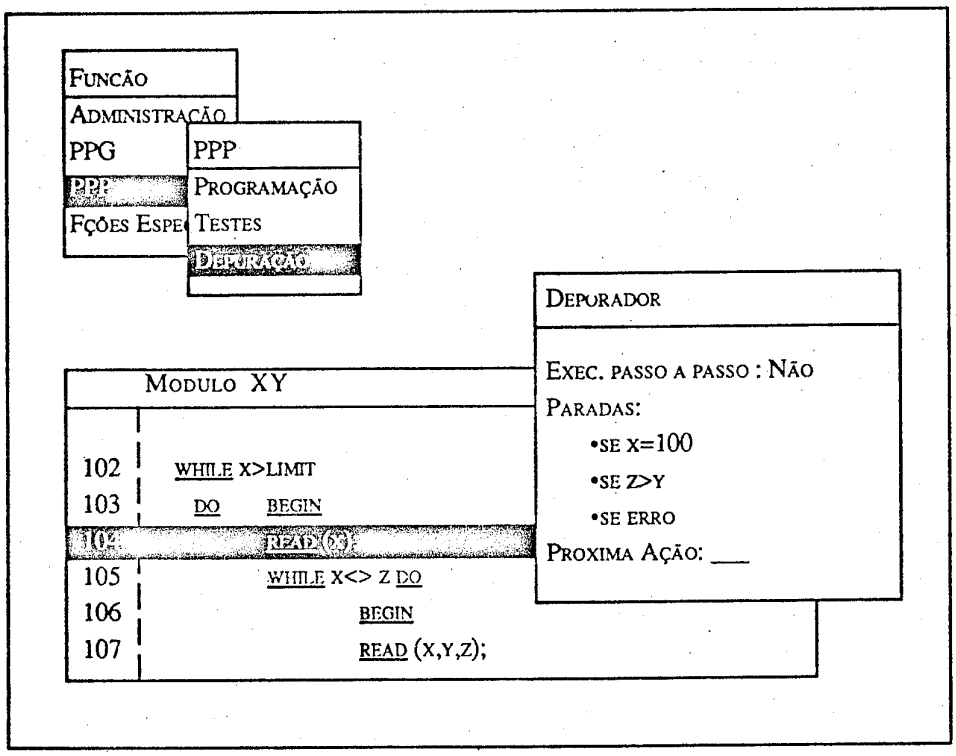

Figura 16: Depuração de um módulo

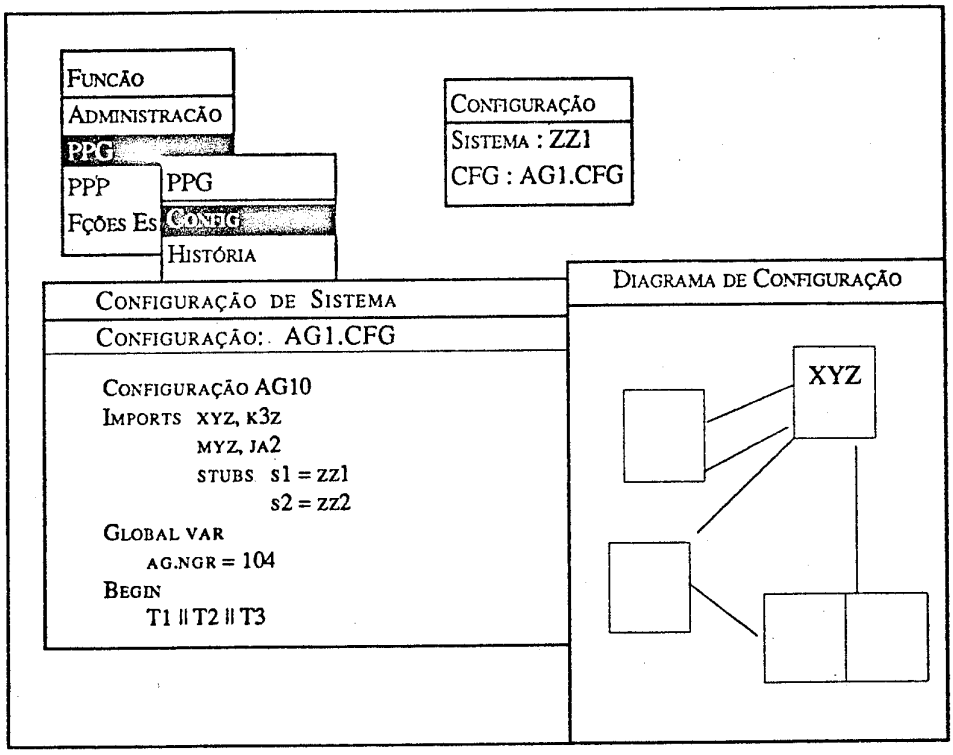

Figura 17: Configurando um sistema

A configuração de um sistema é mostrada, conforme descrição em AG1.CFG, codificada em uma linguagem de interconexão de módulos que permite, entre outras facilidades:

- explicitar relações de importação/exportação entre módulos
- incluir "stubs" no lugar de módulos, e
- explicitar composições de módulos em forma (notacionalmente) similar a "pipes".

Um *diagrama de configuração*, em uma janela independente, ilustra a macro-arquitetura de composição de módulos que vai resultando da montagem da configuração AG1.CFG.

8 Conclusões

Na introdução deste trabalho apresentamos uma argumentação justificando a utilização de ambientes automatizados para o desenvolvimento de software (sistemas de CASE) como solução para a questão de produtividade em sistemas de software. Da seção 2 até a seção 7 apresentamos e simulamos a utilização de um ambiente-exemplo que cobre todas as fases do processo de desenvolvimento (ambiente baseado em métodos).

Neste ponto retomamos a discussão sobre como sistemas CASE asseguram um aumento de produtividade. Em resumo, para que se possa assegurar que o aumento de produtividade é, de fato, alcançado é necessário que o processo de desenvolvimento seja controlado. A gerência deve ser capaz de examinar o processo e efetuar mudanças ou no trabalho realizado ou no plano original para assegurar que um determinado nível de produtividade seja atingido.

Sistemas CASE fornecem ferramentas (como os editores de texto e gráficos) que podem reduzir substancialmente o tempo requerido para se completar todo o universo de produtos em que se constitui um software. Isto é consideravelmente ampliado pela possibilidade de localização de especificações, projetos e código reutilizável através do uso de um dicionário de projetos. Um dicionário de projetos pode viabilizar também a deteção de componentes

de projeto redundantes (como processos e programas que tratam entradas comuns e produzem saidas iguais). Esses componentes podem ser consolidados em um único módulo.

Os erros de projeto que ocorrem no início do processo podem ser detetados quando é mais barato corrigi-los. Se expressos de forma gráfica segundo uma metodologia baseada em regras claramente explicitáveis, é possível estabelecer a consistência de projetos automaticamente.

Com a automatização do ciclo de vida da produção de software a gerência tem, ao mesmo tempo, a possibilidade de influenciar diretamente as tarefas que vão sendo completadas e a visibilidade total sobre o projeto como um todo que seria quase impossível de obter sem a existência do ambiente.

Referências

[1] B. Boehm. *Software Engineering Economics.* Prentice-Hall, 1981

[2] A. F. Case. *Information Systems Development: Principles of Software Engineering & CASE.* Prentice-Hall, 1986

[3] R. N. Charette. *Software Engineering Environments.* McGraw Hill, 1986

[4] S. A. Dart et all. *Software Development Environments.* IEEE Computer, Nov. 1987

[5] R. W. Scheifler et all. *The X-WINDOW System.* ACM Transactions on Graphics, abril 1986

[6] K. R. Dittrich et all. *DAMOKLES - A Database System for Software Engineering Environments.* Relatório Técnico, Universitat Karlsruhe, 1987

[7] B. Boehm. *A Spiral Model of Software Development and Enhancement.* ACM Software Engineering Notes, agosto 1986

[8] R. Engelmore, ed. *Blackboard Systems.* Addison-Wesley, 1988

9 Apêndice: Descrição das Ferramentas

- GERADOR DE RELATÓRIOS
  Ferramenta que permite compor documentos a partir de partes de diferentes origens e imprimi-los em dispositivos diversos. Aceita descrições de formatos de documentos (textos mais gráficos), a partir dos quais permite a configuração de editores e formatadores e o ajuste adequado de "drivers" de saida.

- FORMATADORES
  Ferramentas que processam arquivos de entrada contendo textos/gráficos e geram arquivos de saida ajustados para formatos específicos definidos pelo

- INTERFACE GENÉRICA
  Gerenciador de comunicação usuário/sistema, utilizando ícones e texto estruturado, para acesso às diversas funções do Ambiente de forma uniforme independente das ferramentas acionadas.

- BROWSER (NAVEGADOR)
  Ferramenta de consulta a todas as funções do Ambiente e de navegação pelos objetos existentes no mesmo, com nível ajustável de detalhe e granularidade. Comunica-se diretamente com o Gerador de Relatórios, para fins de impressão seletiva de relatórios.

- CORREIO ELETRÔNICO
  Ferramenta para controle de envio/recebimento de mensagens entre usuários do Ambiente, permitindo "store-and-forward", "broadcasting", retorno de informação sobre recebimento de mensagem, etc.

- ESTRUTURADOR DE PROJETOS
  Permite registrar estruturas arbitrariamente complexas de atividades interrelacionadas, usuários e objetos das atividades, incluindo informações sobre tempos de execução, marcos, resultados esperados, etc. A partir dessas informações, permite ao usuário, executar e/ou reescalonar atividades, servindo como agenda e registrando possíveis cursos de ação a cada instante.

- GERENCIADOR DE VERSÕES
  Ferramenta configurada para cada projeto/metodologia, que permite registrar na Base de Dados (DAMOKLES) versões de objetos nela armazenados, conciliando *segurança* (isto é, garantia de se registrar versões historicas de objetos essenciais sempre que necessário) e *eficiência* (isto é, garantia de seletividade no registro de versões, evitando desperdício de tempo registrando informações inúteis).

- GERENCIADOR DE CONFIGURAÇÃO
  Ferramenta que aceita a descrição de uma configuração específica de sistema (incluindo explicitação de versões, quando pertinentes; substituição de módulos por outros, incluindo "stubs"; medidas específicas para depuração/ monitoração, etc.) e gera um módulo executável que atende aos requisitos impostos pelo usuário acerca da configuração desejada.

- FILER/RETRIEVER
  Ferramentas complementares que permitem, respectivamente, cadastrar um objeto/item no Ambiente (agregando informações acerca do conteúdo desse objeto/item), e recuperar esse objeto/item (através de referências ao seu conteúdo e não meramente ao seu nome). Para tal, o esquema de descrição de conteúdo requer que o usuário agregue ao objeto algumas palavras-chave representativas, compondo ou não um "thesaurus".

- EDITORES
  Ferramentas para edição de textos/gráficos, configuráveis para facilitar a verificação/satisfação de restrições sintáticas impostas por notações/ metodologias específicas.

- LINGUAGENS
  Ferramentas para processamento (compilação, interpretação, etc) de linguagens de programação, especificação, etc. Em particular, ferramentas para processamento de C e C++.

- TESTADORES
  Ferramentas diversas destinadas a testar módulos e sistemas, com vis-

tas a evidenciar sua execução satisfatória, com relação aos requisitos estabelecidos/examinados pelo usuário. Podem incluir:

  - geradores de casos de testes
  - gerenciadores de aplicação de testes
  - executores simbólicos

- DEPURADORES
  Ferramentas para a instrumentação controlada (a nível de linguagem-fonte) de módulos e sistemas escritos em linguagens de especificação e programação de alto nível.

- X-WINDOW
  Ferramenta gerenciadora de funções de distribuição e integração de processamento controlado por janelas de interação com os usuários.

- REDE
  Ferramentas de conexão de estações em rede local ou não. As funções básicas oferecidas devem incluir:

  - esquema de "naming global",
  - transferência de arquivos
  - "log-in" remoto.

- DAMOKLES
  Base de dados orientada a objetos, isto é, voltada para a manipulação eficiente de objetos estruturalmente complexos e suporte a funções críticas como a de versões de objetos.