PROJETO
TOTEM

Meta-Ambiente de Desenvolvimento de Software

DEFINIÇÃO DO PROJETO

Arndt von Staa

Departamento de Informática
PUC-Rio

Outubro 1993

Projeto TOTEM
Meta-Ambiente de Desenvolvimento de Software
Definição do Projeto

Arndt von Staa
(e-mail: arndt@inf.puc-rio.br)

PUCRioInf - MCC 27/93

ABSTRACT

TOTEM's main objective is to contribute to the enhancement of the current practice of Software Engineering. In order to achieve this goal, a software engineering meta-environment will be built. With this meta-environment specific computer aided software engineering environment instances can be generated. Each of these instances provides support to a selected set of development, maintenace and/or management activities. The collection of instances supports the cooperative work of development teams. Each instance may be adapted to the specific needs of the object system to be developed and to the needs of the people using the set of instances. In adition to instantiating production environments, the TOTEM meta-environment will also provide support to experimental research in Software Engineering.

This document defines the requirements of the TOTEM meta-environment. It defines also the basic architecture of the meta-environment.

KEYWORDS: Cooperative development, Meta-environments, Quality assurance, Repositories, Representation langugages, Reuse, Software development processes, Software engineering environments, Version control

RESUMO

O projeto TOTEM tem por objetivo contribuir para uma melhoria substancial da prática atual de Engenharia de Software. Pretende-se alcançar este objetivo através da criação de um meta-ambiente de desenvolvimento de software. Com o meta-ambiente são criadas instâncias de ambientes de desenvolvimento de software assistidos por computador. Cada uma destas instâncias de ambiente apóia, de forma integrada, diversas etapas selecionadas do ciclo de vida dos sistemas objeto desenvolvidos com sua assistência. O conjunto de instâncias será capaz de apoiar o trabalho cooperativo das pessoas que integram uma equipe. Cada instância poderá ser adaptada às necessidades dos diversos sistemas objeto a serem desenvolvidos e mantidos e às equipe de desenvolvimento que utilizarão essa instância. Além de permitir a instanciação de ambientes de desenvolvimento, o meta-ambiente TOTEM estabelece um laboratório de software para a pesquisa, entre outros, em ambientes, em linguagens de representação, em técnicas de desenvolvimento e em processos de desenvolvimento de software.

Este documento define os objetivos e requisitos a serem alcançados pelo o meta-ambiente TOTEM. Define, ainda, a arquitetura básica do meta-ambiente TOTEM.

PALAVRAS CHAVE: Ambientes de engenharia de software; Controle de versão; Desenvolvimento cooperativo; Garantia de qualidade; Linguagens de representação; Meta ambientes; Processos de desenvolvimento de software; Repositórios; Reúso

1. Introdução

O projeto TOTEM[1] tem por objetivos:

1- contribuir para uma melhoria significativa da prática de Engenharia de Software encontrada em instituições que desenvolvem sistemas de programação.

2- estabelecer uma plataforma para a pesquisa aplicada em Engenharia de Software.

3- transferir tecnologia de Engenharia de Software para instituições que desenvolvam software.

Para alcançar estes objetivos optou-se por desenvolver um Meta-Ambiente de Desenvolvimento de Software assistido por computador. Com este meta-ambiente instanciam-se ambientes específicos. Estes serão utilizados em contextos de produção ou de pesquisa. Finalmente, para viabilizar o uso eficaz destas instâncias, é necessário estabelecer um cuidadoso processo de transferência de tecnologia.

Uma vez em uso, o meta-ambiente, bem com as instâncias de ambiente, serão avaliadas. Os resultados desta avaliação serão utilizados para direcionar a pesquisa e o desenvolvimento do projeto TOTEM.

Este texto especifica os objetivos e requisitos de longo prazo a serem satisfeito pelo meta-ambiente TOTEM. Uma parte substancial destes requisitos já foi atingida por versões já desenvolvidas do meta-ambiente. Outros documentos definirão projetos específicos, estabelecendo metas e planos para o seu alcance.

Para simplificar a redação e leitura deste texto, entenderemos o termo **desenvolvimento** de forma bastante abrangente, compreendendo:

1- o *desenvolvimento* própriamente dito, composto por especificação, arquitetura, projeto, implementação, integração e implantação.

2- a *manutenção*, composto por correção, perfecção, adaptação e evolução.

3- o proceso de *garantia de qualidade*, composto por definição e controle de qualidade.

4- os processos de *controle de versão* e de *gerência de configuração*.

5- a *gerência do processo de desenvolvimento*, composto por planejamento e acompanhamento do desenvolvimento.

1.1. Motivação

Recentemente surgiu um movimento em busca de uma maior *maturidade* no processo de desenvolvimento de software [Lai93, Humphrey 89]. Paralelamente a este movimento passou a existir uma preocupação maior com a *qualidade total* de sistemas, induzida pela divulgação das normas ISO9000 e ISO9001. Finalmente, há algum tempo vem sendo comentada a necessidade de *novas formas* de desenvolver software de modo a se romper os atuais limites de produtividade e qualidade [Brooks 86]. O foco central de todas essas preocupações é o *processo de desenvolvimento*.

Através de uma formalização do processo de desenvolvimento, do emprego de ferramentas, do emprego de geradores de programas e do emprego de dicionários corporativos é esperado que se consiga, a um só tempo, aumentar significativamente tanto a produtividade das equipes como a qualidade dos sistemas de programação desenvolvidos.

Para serem eficazes, ferramentas devem formar um todo íntegro, harmonioso e consistente. É pouco provável que se consiga resolver o problema do desenvolvimento de software através da composição de ferramentas isoladas e projetadas sem a preocupação com sua integração com outras ferramentas. Em adição, para serem eficazes, ferramentas

---

1 TOTEM - animal, vegetal ou qualquer objeto considerado como ancestral ou símbolo de uma coletividade (Aurélio). Significa aqui o gerador (ancestral) de famílias de instâncias de ambientes.

devem apoiar todas as atividades do desenvolvimento. Devem apoiar, ainda, o desenvolvimento de diferentes classes de software desde muito simples a muito complexo, desde operando em plataformas isoladas e pobres a operando em plataformas interligadas e ricas, desde pouco críticos a muito críticos com relação à qualidade, desde produzido e mantido por equipes de pouca formação a produzido e mantido por equipes com muita formação. Finalmente, as ferramentas devem ser capazes de gerar ou compor código e documentação para o usuário (manuais, "helps", etc.), uma vez que este é o fim último de qualquer projeto de desenvolvimento de software.

Transcrevendo de CASE '90 Report (© 1991 CASE Consulting Group) [CASE90 Rep]:

> Among the greatest challenges lying ahead is the need for tighter integration among tools in a manner that supports openess to a variety of methods, notations, processes, tools, and platforms. Understanding the software development process and getting developers to use software engineering techniques correctly and consistently will remain a problem, especially in the face of evolving technology. A more disciplined approach to software process engineering employing metrics, feedback and continuous improvements is needed. Management will have to take a long-range economic perspective, modify objectives and incentives, and incorporate principles of Total Quality Management to ensure the ultimate success of CASE.

Para resolver este desafio estão sendo perseguidas diversas linhas de ação [CASE90 Rep, pag 302]. Uma delas é a definição de uma estrutura *padrão de repositório* e de interfaces internas e externas a ser obedecida pelas diferentes ferramentas desenvolvidas pelos mais variados fornecedores. Esta linha tem o seu exemplo mais notório no projeto AD-CYCLE da IBM. Uma outra linha é a de definir *coletâneas integráveis de ferramentas*. Esta linha tem o seu exemplo mais notório em PCTE [Boudier89]. Nós estamos perseguindo a terceira linha, a de *meta-ambientes* de desenvolvimento de software.

Como explicitado na citação acima, um problema substancial é o de transferir para empresas a tecnologia de Engenharia de Software, em particular a tecnologia do uso de ambientes de desenvolvimento. Isto implica não só dar o necessário treinamento aos futuros usuários, como implica desenvolver as instâncias de ambiente para facilitar a sua introdução no meio produtivo, de modo que não se tornem "shelfware"[Forte92].

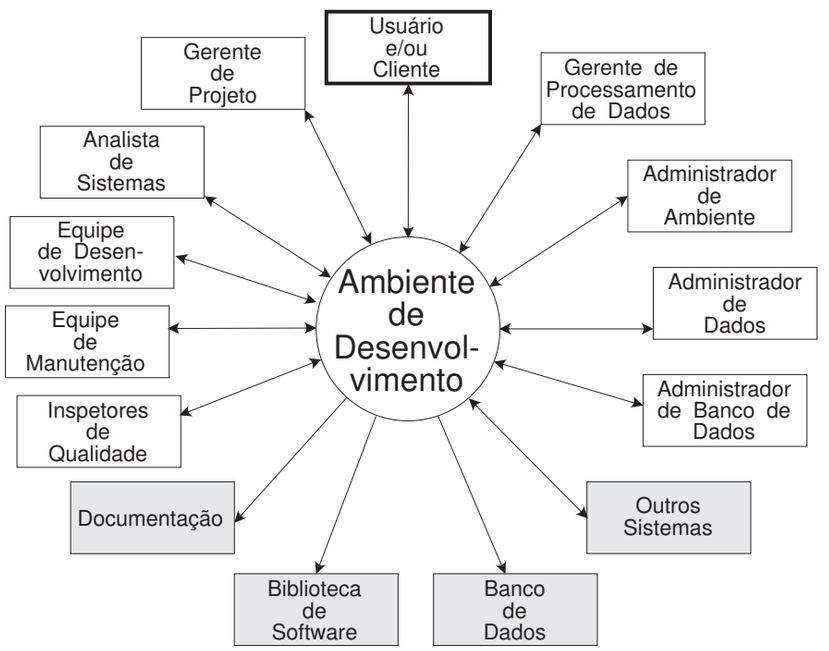

Figura 1. Usuários de um Ambiente de Desenvolvimento de Software

1.2. Meta-ambientes, conceituação

Um *ambiente de desenvolvimento de software*[2] é um conjunto de procedimentos (tarefas manuais), técnicas, métodos, heurísticas, ferramentas (programas), repositórios e modelos de desenvolvimento, cuidadosamente organizados visando ao desenvolvimento, à manutenção e à gerência do processo de desenvolvimento e manutenção de sistemas de software. Chamaremos de *sistemas objetivo* os sistemas desnvolvidos com o apoio de um ambiente de desenvolvimento.

Ambientes de desenvolvimento apóiam um grande número de pessoas e sistemas, cada qual requerendo ferramentas especializadas a sua disposição. Em virtude da necessidade de integração entre estas ferramentas, existirão interfaces complexas entre todas as ferramentas utilizadas no ambiente. Na figura 1, os blocos brancos correspondem a interfaces com pessoas, enquanto que os blocos acinzentados correspondem a interfaces com outros sistemas.

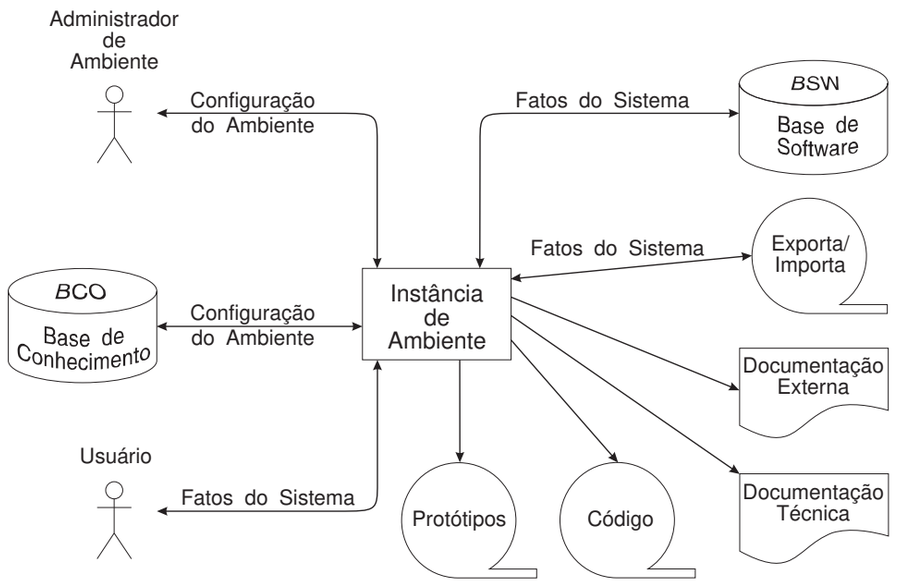

Figura 2. Interfaces externas típicas de uma instância de ambiente.

A figura 2 ilustra as interfaces externas típicas de uma instância de ambiente. O *Administrador de Ambientes* configura ou ajusta os serviços que a instância é capaz de prover. O *Usuário* (ver figura 1) utiliza os serviços da instância de ambiente para editar *representações.* Representações são os diferentes documentos técnicos e de gerência produzidos ao desenvolver o sistema objetivo. As representações são armazenadas na *base de software* (repositórios, enciclopédias). As definições das propriedades de representações, as regras de edição, as regras de controle de qualidade, etc. são armazenadas na *base de conhecimento*. Utilizando serviços especializados da instância de ambiente, pode-se importar e/ou exportar representações para outras instâncias de ambiente, ou mesmo para outras ferramentas. Pode-se, ainda, gerar código, gerar protótipos e gerar a documentação técnica, de gerência de desenvolvimento e de uso do sistema objetivo. Nas seções a seguir estes conceitos serão revistos em maior profundidade.

Meta-ambientes de desenvolvimento de software permitem a criação e a manutenção de ambientes de desenvolvimento específicos, as *instâncias de ambientes*. Cada

2 Em inglês, o termo que mais se aplica ao conceito de ambiente de desenvolvimento de software por nós utilizado é IPSE - Integrated Project Suport Environment. Um outro termo usado com conotação semelhante é SEE - Software Engineering Environment. O termo CASE - Computer Aided Software Engineering, não se aplica, uma vez que, não existe o compromisso explícito da ferramentas CASE apoiarem, de forma integrada, todas as atividades do desenvolvimento de software.

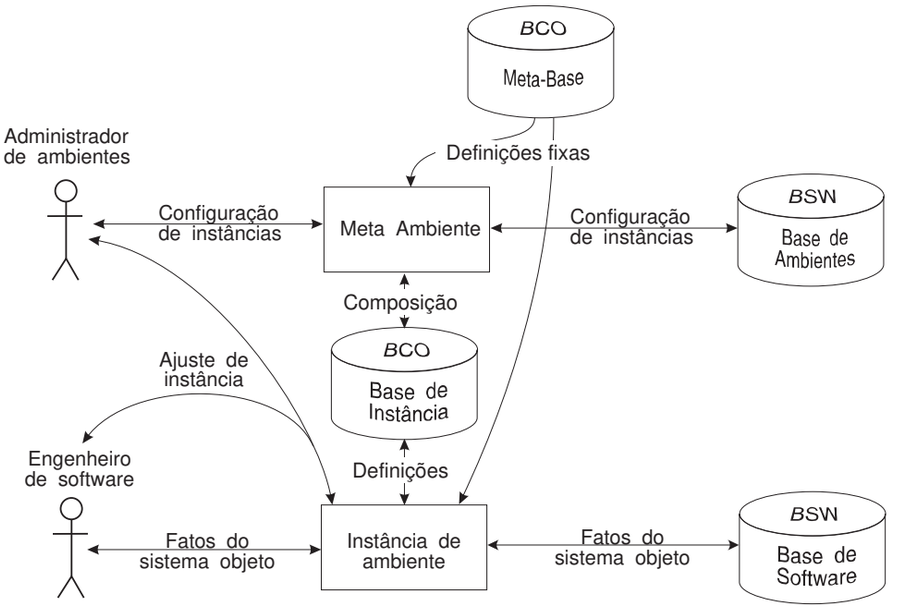

Figura 3. Interrelação entre o meta-ambiente e as instâncias de ambiente geradas

instância de ambiente é dirigida a uma classe de pessoas, à tecnologia utilizada, à natureza do sistema a ser desenvolvido e às etapas de desenvolvimento a cumprir. O conjunto de instâncias utilizadas ao desenvolver um determinado sistema objetivo deve ter o potencial de apoiar todas as atividades do desenvolvimento desse sistema.

O software que compõe as instâncias de ambiente é sempre o *mesmo* para todas as instâncias. No nosso caso, o software das instâncias é inclusive o mesmo que o do meta-ambiente. Isto tem como subproduto a uniformidade de interfaces e integrabilidade das ferramentas. Em contrapartida, os programas devem ser dirigidos por tabelas complexas para poderem prestar o serviço desejado. Estas tabelas são armazenadas na base de conhecimento.

A definição de uma instância de ambiente é realizada pelo Administrador de Ambientes, ver figura 3. Primeiro são definidas as linguagens de representação conhecidas, sendo especificadas as *propriedades* de cada uma das linguagens de representação suportadas pelo ambiente. Estas propriedades são armazenadas na base de ambientes. Ainda utilizando o meta-ambiente, o Administrador de Ambiente *seleciona* quais destas linguagens formam a instância de ambiente.

Uma vez a instância de ambiente definida, o Engenheiro de Software poderá criar e manter a base de software. Esta contém todas as informações relativas ao sistema objeto sendo desenvolvido. Durante o desenvolvimento do sistema objeto, pode-se querer alterar a composição dos formulários de especificação, dos relatórios produzidos, das regras de controle de qualidade, etc. Estes ajustes podem ser realizados diretamente pelo Engenheiro de Software, ou, então, pelo Administrador de Ambiente.

Em adição à capacidade de instanciar ambientes específicos, meta-ambientes estabelecem, ainda, uma plataforma para a pesquisa experimental em ambientes de desenvolvimento de software. Como exemplos de tais trabalhos de investigação podemos citar: estudos sobre propriedades de linguagens de representação, adequação de linguagens de representação a determinadas classes de problema, integração entre linguagens, controle de qualidade de representações, processos de desenvolvimento, prototipação, geração automática de código, controle de versões e de configurações, projeto e desenvolvimento visando o reúso, desenvolvimento em equipe, etc.

2. Objetivos do projeto TOTEM

2.1. Identificação

Como já havia sido mencionado, os objetivos do projeto TOTEM são:

1- contribuir para uma melhoria significativa da prática de Engenharia de Software encontrada em instituições que desenvolvem sistemas de programação.

2- estabelecer uma plataforma para a pesquisa aplicada em Engenharia de Software.

3- transferir tecnologia de Engenharia de Software para instituições que desenvolvam software.

Para alcançar estes objetivos optou-se por desenvolver um meta-ambiente de desenvolvimento de software assistido por computador. Com este meta-ambiente instanciam-se ambientes específicos. Estes serão utilizados em contextos de produção ou de pesquisa. Finalmente, para viabilizar o uso eficaz destas instâncias, é necessário estabelecer um cuidadoso processo de transferência de tecnologia de uso das instâncias e do meta-ambiente.

Entende-se por *melhoria significativa* um substancial (ordens de grandeza) aumento de produtividade no desenvolvimento de sistemas de programação com qualidade garantida. Para que esta contribuição seja efetiva, é imprescindível que as instâncias de ambiente geradas:

1- apóiem o trabalho cooperativo das *equipes* encontradas em empresas.

2- apóiem *métodos de trabalho* familiares a estas equipes.

3- operem em *plataformas comumente disponíveis*.

Criar protótipos de laboratório não basta, uma vez que já é fato conhecido que ambientes abrangentes e assitidos por computador são possíveis. Precisamos avaliar a eficácia das instâncias de ambientes e identificar as condições que ampliem esta eficácia.

Entende-se por *software de qualidade*, um software que atinja satisfatoriamente os objetivos de qualidade a seguir:

confiabilidade- operam corretamente sempre que solicitados.

utilidade- prestam serviços relevantes para os clientes (usuários).

utilizabilidade- possuem desempenho e interface humana satisfatória, permitindo que sejam utilizados confortavelmente em plataformas reais (hardware e software de suporte).

evolutibilidade- estão vinculados a um sistema de manutenção capaz de adicionar nova funcionalidade, adaptar a funcionalidade existente a novos contextos, aprimorar e corrigir deficiências.

monitorabilidade- possuem instrumentos que viabilizem a medição do desempenho do ambiente, bem como do uso do ambiente.

economicidade- produzem resultados compatíveis com os custos incorridos para obtê-los.

Faz parte do projeto adaptar os modelos de controle de qualidade encontrados na literatura (por exemplo [Conte86]) às necessidades deste projeto. A avaliação da conformidade das instâncias de ambiente com os objetivos de qualidade acima será realizada com base nestes modelos.

2.2. Estratégia utilizada para alcançar os objetivos

O projeto TOTEM é uma *diretriz* de pesquisa e de desenvolvimento de longa duração, aglutinando e coordenando diversos trabalhos de pesquisa e desenvolvimento

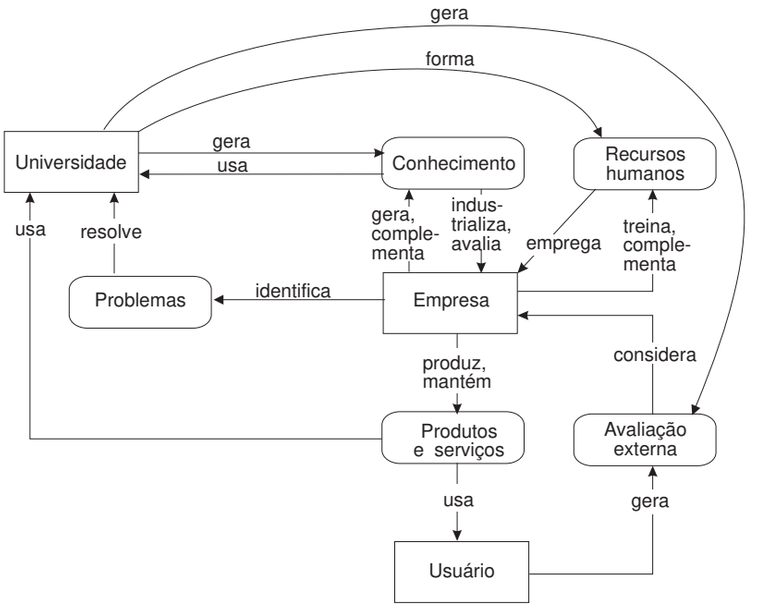

Figura 4. Modelo de interação Universidade Empresa

em ambientes de desenvolvimento de software. Como resultado da pesquisa e do desenvolvimento são tornados disponíveis:

1- meta-ambientes em níveis de abrangência cada vez maiores.

2- instâncias de ambientes com qualidade industrial, permitindo que sejam utilizadas eficaz e confiavelmente por empresas e pessoas que se dedicam ao desenvolvimento de software.

3- estudos e fundamentos vinculados ao processo de desenvolvimento e a meta-ambientes de desenvolvimento.

4- avaliações de instrumentos de Engenharia de Software.

2.2.1. Interação Universidade Empresa

Como visto na seção anterior, serão instanciados ambientes de desenvolvimento de software para o uso industrial. Estas instâncias de ambiente devem possuir qualidade satisfatória de modo que possam ser utilizadas em empresas. Como a instanciação pode, em princípio, ser realizada pelos próprios usuários, o meta-ambiente deve ser flexível e simples de utilizar. Outro objetivo é o de prover suporte para a pesquisa em desenvolvimento de software. Isto obriga o meta-ambiente a ser flexível e genérico, de modo que possa ser utilizado para a criação rápida e eficaz de um grande número de ferramentas experimentais. Requer, ainda, que tanto o meta-ambiente, como as instâncias de ambiente geradas, tenham elevada qualidade, para que os resultados da investigação não sejam comprometidos por eventuais deficiências contidas nestas instâncias. Finalmente é fundamental que o meta-ambiente e as suas instâncias de ambiente sejam mantidos e evoluídos.

Tendo em vista a complexidade e a abrangênciado problema proposto, o volume de pesquisa e inovação tecnológica requeridos para resolvê-lo, a forte interação de qualquer solução com o processo produtivo e, finalmente, os requisitos de acabamento, qualidade e manutenção, torna-se necessário um trabalho conjunto entre universidade e empresa, nos moldes ilustrados na figura 4. Assim a estratégia utilizada no projeto TOTEM baseia-se nos princípios:

1- pesquisa e desenvolvimento de protótipos realizados na Universidade.

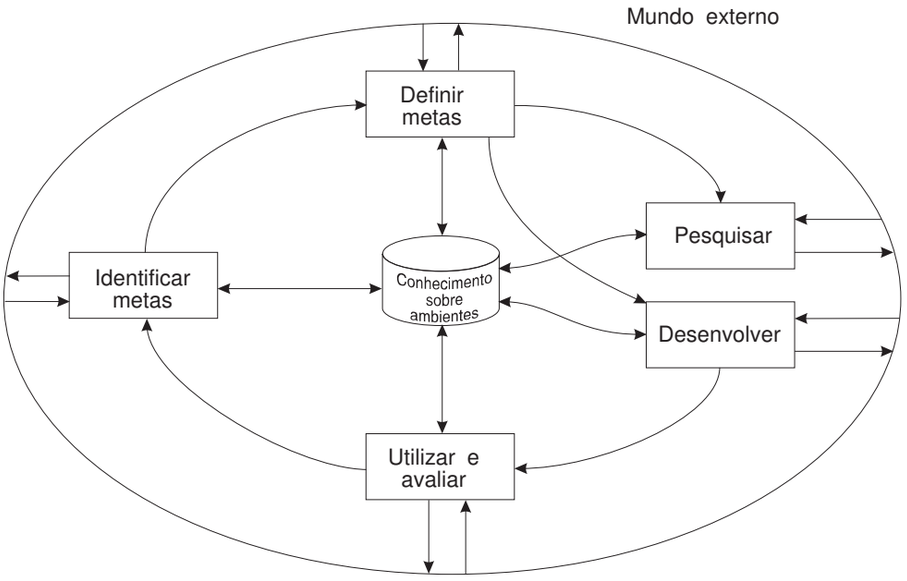

Figura 5. Modelo do processo de pesquisa e desenvolvimento

2- industrialização dos resultados de pesquisa, integração, acabamento, geração de documentação de usuário, manutenção, atendimento a usuários, etc. realizado por empresa.

3- desenvolvimento incremental e adaptativo, discutido a seguir, utilizando as avaliações e interações com os usuários como mecanismo de identificação de novas metas a alcançar.

O desenvolvimento conjunto com empresas foi trilhado com sucesso, resultando nas ferramentas CASE: MOSAICO-DEM (em colaboração com a IESA-TS) e TALISMAN [Staa92b] (em colaboração com a STAA Informática). A ferramenta CASE TALISMAN, na sua atual versão 4.2, encontra-se em uso em diversas empresas e em instituições de ensino e pesquisa. Em particular, TALISMAN é utilizado para desenvolver e manter TOTEM. TALISMAN é, por sua vez, uma instância de TOTEM. Isso ilustra a viabilidade de se gerar instâncias de ambiente utilizáveis em produção.

2.2.2. Objetivos e requisitos de longo prazo

Meta-ambientes são fronteira de tecnologia, assim é virtualmente impossível determinar um conjunto estável, abrangente e realista de requisitos. Para vencer esta dificuldade, optamos por definir objetivos e requisitos essenciais que qualquer meta-ambiente de desenvolvimento deverá satisfazer. Estes requisitos, bem como a arquitetura básica de TOTEM serão definidos neste artigo. Os requisitos estabelecem metas de longo prazo. Algumas destas metas já fora alcançadas. Outras serão o objeto de projetos de pesquisa e desenvolvimento específicos. No entanto, todas as metas específicas devem estar em conformidade com as metas gerais definidas neste documento.Os projetos podem ser dissertações de mestrado, teses de doutorado e, eventualmente, novos desenvolvimentos ou evoluções do software já disponível.

2.2.3. Avaliação e revisão sistemática de objetivos

Para assegurar convergência ao trabalho, optamos por um processo de pesquisa e desenvolvimento evolutivo, operando com as seguintes etapas, ver figura 5:

1- identificar metas gerais (de longo prazo) a serem satisfeitas pelo meta-ambiente e pelas instâncias de ambiente.

2- definir metas a curto prazo (horizonte de cerca de um ano).

3- pesquisar e/ou desenvolver, procurando atingir as metas de curto prazo.

4- utilizar, avaliar e manter a versão corrente e avaliar os resultados de pesquisa. A manutenção tem por objetivo, além de remover erros, efetuar pequenas evoluções que não impactem significativamente a funcionalidade disponível. A avaliação tem por objetivo identificar problemas a resolver e, desta forma, gerar "feedback" para os próximos passos de evolução.

5- em função da avaliações realizadas, identificar novos alvos de pesquisa, propor, justificando, novos objetivos e requisitos a serem alcançados pelo meta-ambiente, linguagens de representação ou instâncias de ambiente.

Este processo evolutivo é fortemente apoiado em "feedback" obtido a partir da avaliação crítica do resultado de cada etapa. Isto caracteriza um processo de aprendizado possuindo estabilidade e convergente para o objetivo inicial traçado: o de prover mecanismos para uma melhoria substancial do status quo em Engenharia de Software. É evidente que poderão ocorrer revisões profundas tanto de metas de curto prazo como de metas de longo prazo. Algumas destas revisões de metas podem até ter como resultado uma substancial re-engenharia do software já disponível.

3. Caracterização do processo de desenvolvimento de software

Nesta seção serão resumidamente descritos o processo e os produtos do desenvolvimento de software. A descrição assume que o desenvolvimento será apoiado por uma instância de ambiente gerada pelo meta-ambiente TOTEM.

3.1. Representações

O desenvolvimento de sistemas de programação leva à geração e à manutenção de diversos documentos, tais como especificações, arquiteturas, projetos, roteiros de teste, código fonte, layout de janelas, etc. Estes documentos são chamados de *representações.* O objetivo das representações é registrar e divulgar *fatos do sistema objetivo*. Estes constituem o conhecimento necessário para construir o sistema objetivo e seus componentes.

Representações podem ser:

1- documentos registrados em papel, como, por exemplo: especificações, projetos, listagens de programas fonte, manuais externos, etc.

2- registros armazenados em algum meio computacional, como por exemplo: bases de software, bibliotecas de programas, bases de dados, arquivos contendo auxílios "on-line", etc.

3- "documentos" voláteis tais como textos e diagramas apresentados em telas de vídeos.

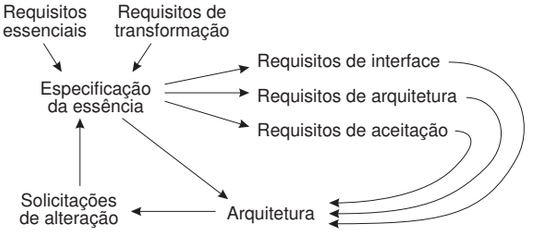

Figura 6. Ilustração de uma malha (parcial) de representações

As representações geradas por uma instância de ambiente formam uma *malha de representações* em que umas servem de entrada, *antecedentes*, para outras, *conseqüentes*, ver figura 6. Por exemplo, as representações de especificação são antecedentes das representações de arquitetura. Toda a informação contida em uma especificação antecedente está disponível, e pode ser referenciada, nas representações conseqüentes. Desta forma, o processo de desenvolvimento passa a operar principalmente com *transformações*

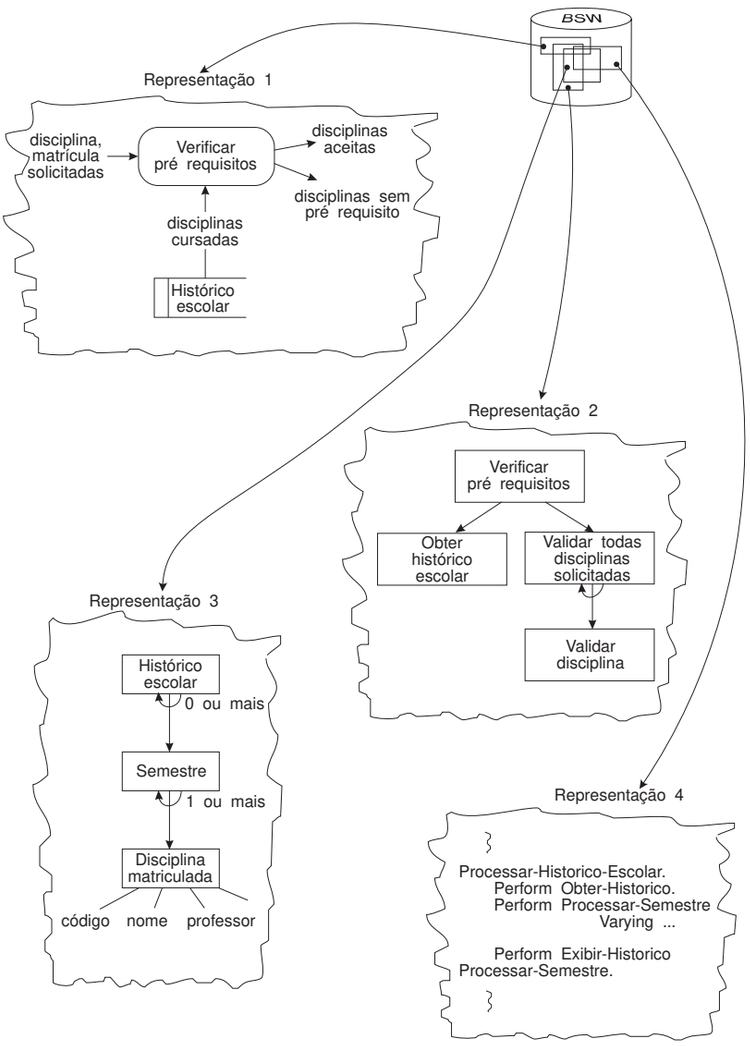

Figura 7. Representações são visões do conteúdo da base de software

de representações antecedentes para representações conseqüentes, minimizando o volume de atividade humana necesária. Isto contribui para o aumento da produtividade (menos esforço humano por volume de funcionalidade desenvolvido) e qualidade (menos chances de cometer erros ou de introduzir inconsistências). A derivação, ou, pelo menos, interligação, de representações conseqüentes com as respectivas representações antecedentes, assegura, entre outras:

1- a possibilidade de se validar as representações conseqüentes.

2- a possibililidade de rastrear como os requisitos vieram a ser atendidos no código (“traceability”).

3- a possibilidade de localizar, durante um passo de manutenção, os pontos a alterar e de determinar a abrangência desta alteração.

3.2. Reúso de representações

Ao criar a malha de representações, é desejável poder vincular a malha sendo criada a representações já criadas anteriormente, mesmo que em outro projeto de desenvolvimento. Esta forma de *reúso* permite representações de alto nível de abstração já existentes a serem integradas em novos projetos. Ao fazer esta integração, é evidente que todas as

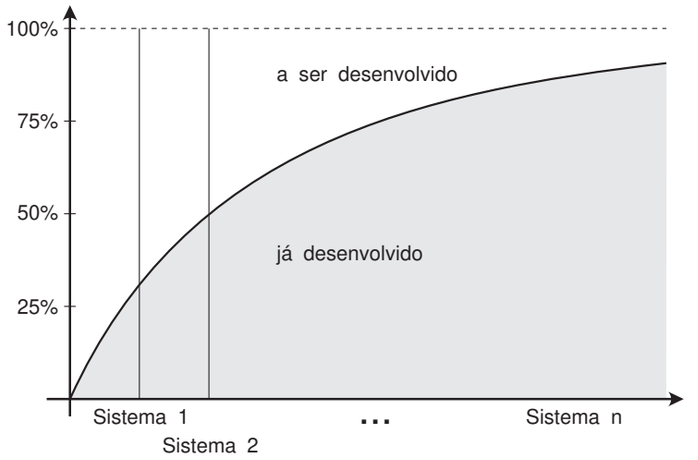

Figura 8. Crescimento do reúso

representações derivadas acompanham a representação reutilizada. Deve estar claro que isto reduz significativamente o volume de trabalho a ser realizado ao desenvolver novos sistemas. A figura 8 ilustra o crescimento do reúso a medida que sistemas vão sendo desenvolvidos ou incorporados (engenharia reversa) ao ambiente de desenvolvimento.

O reúso pode-se dar de, pelo menos, duas maneiras, uma é a cópia e a outra é o relacionamento da representação sendo desenvolvida a outra representação já existente. O reúso por interligação é mais eficaz, por assegurar a existência de uma única cópia a ser mantida. Ao evoluir uma representação interligada, todos os sistemas que referenciam esta representação serão evoluídos também. É evidente que isto reduz significativamente os custos de manutenção.

Ao reutilizar uma representação, é comum ter-se que adaptá-la para que se encaixe corretamente no novo contexto de uso. Isto pode ser realizado através ou de adaptadores de interface (chamados na gíria de *bacalhaus*), ou através de uma revisão da representação integrada. Esta última forma é mais cara, uma vez que provoca uma revisão em todas as representações derivadas bem como em todas as representações com que interfaceia. No entanto, é mais eficaz, uma vez que assegura uniformidade nas estruturas e nas interfaces dos componentes dos diferentes sistemas afetados. Existindo a interligação explícita entre as representações, o esforço para realizar a revisão, bem como o risco de falha, é minimizado, já que, através de simples navegações, se consegue explorar, avaliar e, se necessário, corrigir cada representação específica.

3.3. Operações sobre representações

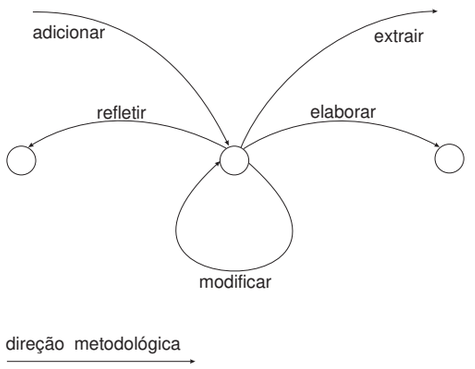

Figura 9. Operações sobre representações

Representações devem apoiar a transformação da informação nelas contida, seja através do acréscimo de detalhe (elaboração com adição), seja através da interligação de informação contida em várias representações, seja através da abstração ou outra operação qualquer. Ao transformar representações, são efetuadas diversas operações, ver figura 9. O processo de transformação deve poder ser realizado a partir da especificação de requisitos até dispor-se do código. Durante o processo de desenvolvimento serão criadas novas representações, a grande maioria delas em função da elabora-

ção a partir de outra representações já existentes. Freqüentemente, a transformação será realizada adicionando informação externa. Ou seja, a evolução do processo é criativa, tornando necessário registrar conhecimento adquirido e/ou decisões tomadas durante a evolução da etapa.

Representações são alteradas com freqüência. Após alterar uma representação, torna-se necessário propagar estas alterações para outras representações. Muitas vezes a propagação requer a *reflexão* da alteração para uma representação antecedente. Por exemplo, ao constatar uma deficiência do projeto durante a programação, o programa é implementado diferente do projeto original. Torna-se necessário, agora, refletir estas alterações para o projeto, assegurando, assim, a consistência entre ambos. Note que a reflexão corresponde a uma forma de engenharia reversa.

3.4. A base de software

As tarefas de desenvolvimento de um software objetivo são apoiadas por editores. Estes editores editam e mantêm os dados (fatos do sistema objetivo) contidos um banco de dados, o *repositório*, ou como nós preferimos chamar, a *base de software*. A base de software das instâncias geradas pelo meta-ambiente TOTEM, tem estrutura fina ("*fine grained structure*" [Harrison 92]), ou seja, ela é formada por uma rede de pequenos objetos fortemente interrelacionados. Documentos específicos, ou objetos de estrutura grossa ("*coarse grained structure*"), são reconstruídos percorrendo-se esta rede segundo as regras definidas pelas respectivas linguagens de representação.

A base de software é um repositório contendo todas as informações relativas a um dado sistema objetivo (ou de vários sistemas objetivo). Cada representação corresponde, essencialmente, a uma visão do conteúdo desta base, ver figura 7 na página 10. Estas visões possuem interseções não vazias. Para garantir a consistência do conjunto de representações, basta mostrar que cada representação é consistente, pois isto assegura que as interseções serão necessáriamente consistentes. Alterações em uma das representações pode tornar inconsistente a interseção que esta representação possui com outras. Isto obriga a revisão de todas essas outras representações. Esse mecanismo estabelece uma forma de *propagação de alterações*. Uma etapa de desenvolvimento (ou manutenção) terá sido concluída somente depois de estabilizado o conjunto de representações ("deliverables") a ser produzido.

A base de software pode ser implementada de diversas formas. Nós optamos por uma implementação *distribuída em múltiplas bases físicas* (arquivos) interligadas, ver

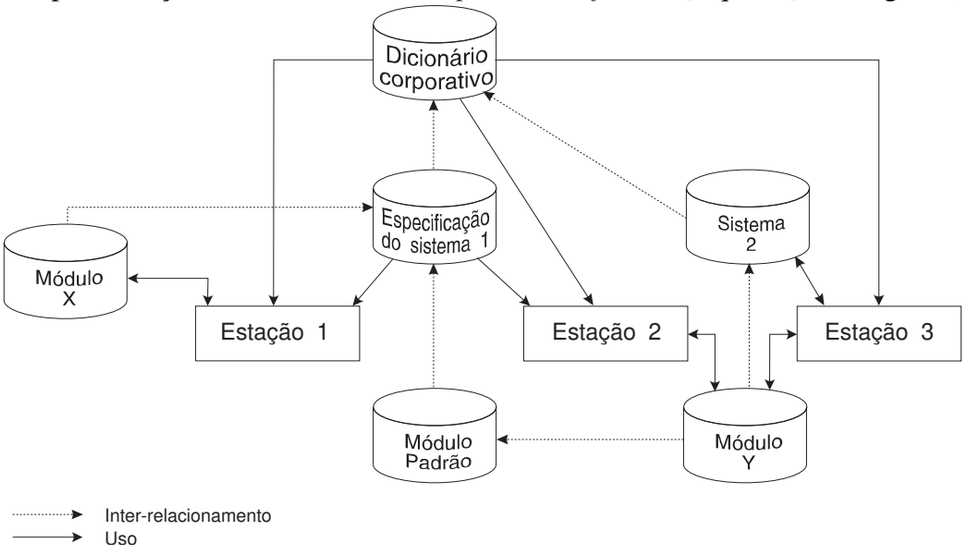

Figura 10. Ilustração de bases de software distribuídas

figura 10. Esta forma é mais condizente com sistemas distribuídos operando em redes locais. Bases físicas distribuídas interligadas permitem que se criem *padrões* (normas técnicas). Estes podem ser agregados segundo os mais variados critérios de organização, tais como, por abrangência, por assunto, etc. Por exemplo, na figura 10, o *dicionário de dados corporativo* estabelece um padrão para toda a empresa. Já a *especificação do sistema 1* complementa este padrão com vistas somente ao sistema 1. Pode-se, ainda, paticionar a base de software por assunto.Por exemplo, as definições de dados financeiros podem estar em uma base física, enquanto que as definições de dados de produção estarão em outra.

3.5. Representações como hipertextos

O conjunto de representações forma uma rede fortemente inter-relacionada. Para que seja possível criar, manter e explorar esta rede, é essencial que existam caminhos de navegação que permitam atingir qualquer representação a partir de qualquer outra representação. Esta forma de navegação é típica a hipertextos, ver figura 11. Ou seja, representações devem poder ser exploradas (navegadas), criadas e/ou editadas de modo semelhante a hipertextos [Cybulski92] formados por um misto de fragmentos de texto, de formulários contendo vários campos e de diagramas inter-relacionados.

Os caminhos de navegação fazem uso dos relacionamentos existentes entre os elementos contidos nas diferentes representações. Estes relacionamentos estabelecem

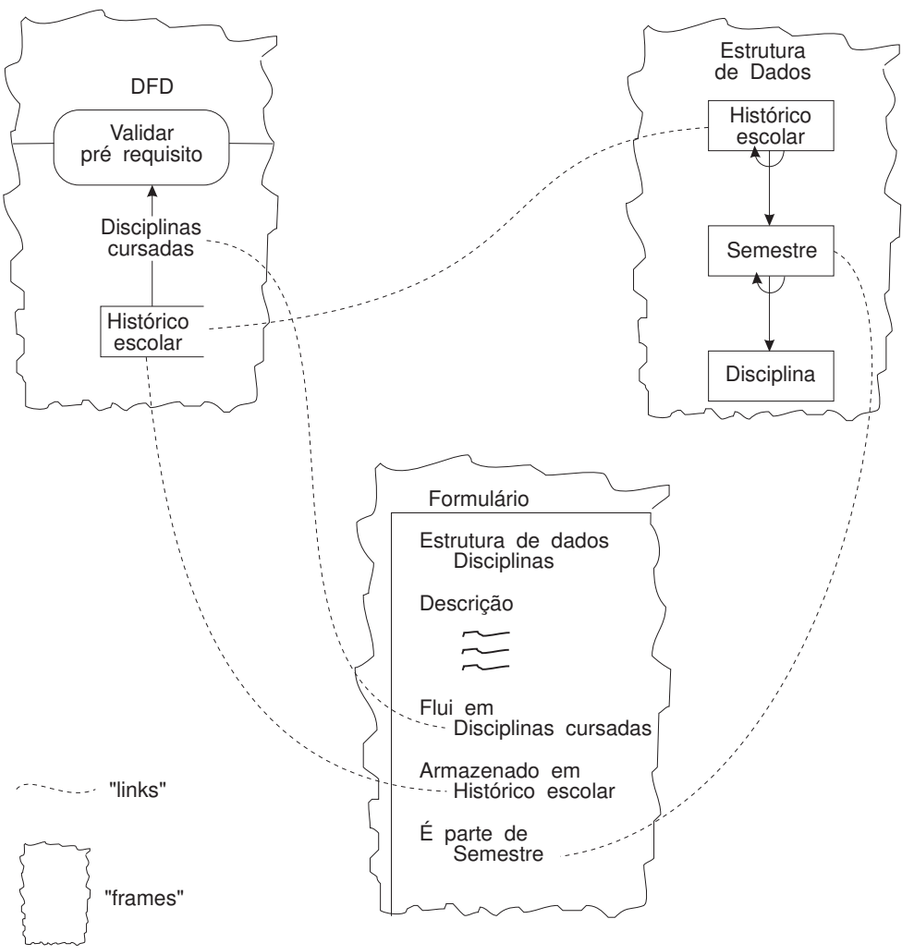

Figura 11. Estrutura de hipertexto formado por representações

vínculos (“links”) entre partes das representações (“frames”). Ao navegar para um elemento de uma determinada representação, toda a representação que contém este elemento é acessada. Para tal, o controle de navegação é sensível à visão com que se está examinando a base de software. Os relacionamentos entre representações possuem semântica própria, por exemplo: módulo *é composto* por pacotes (funções, “procedures”, etc.); blocos *se decompõem* em outros blocos formando uma árvore de código estruturado, etc. É fundamental que se possa navegar segundo estas referências cruzadas e, possivelmente, até compor textos complexos (linearizar) através desta navegação.

3.6. Linguagens de representação

Cada representação é redigida em alguma *linguagem de representação*. Cada linguagem de representação possui formato, sintaxe e semântica própria, podendo ser formada por um misto de diagramas, tabelas, formulários e texto. As linguagens de representação podem ter vários graus de formalidade. O objetivo de linguagens de representação é estabelecer um mecanismo de comunicação fiel entre as pessoas (desenvolvedores, usuários, clientes, auditores, vendedores, etc.) que direta ou indiretamente participam do desenvolvimento e do uso do software. São exemplos de linguagens de representação: português, formulários a serem peenchidos em português restrito, diagramas de fluxo de dados, diagramas entidades e relacionamentos, redes de Petri, “state charts”, dicionários de dados, diagramas de estrutura organizacional, linguagem de programação, etc.

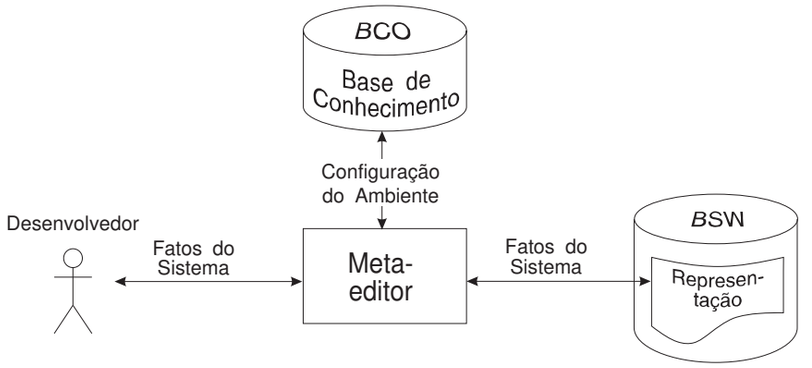

Figura 12. Componentes de um meta-editor

Para facilitar o uso das instâncias de ambientes, os editores de representações devem operar de acordo com as correspondentes linguagens de representação. Isso requer que as representações exibidas no vídeo sejam semelhantes às que seriam impressas em papel. Por outro lado, em um meta-ambiente não é conhecido a priori quais serão todas as linguagens de representação que poderão vir a ser utilizadas. Conseqüentemente, os editores utilizados devem ser meta-editores, e devem ser dirigidos pela definição das linguagens de representação. As definições das linguagens de representação estão contidas na base de conhecimento, ver figura 12.

Cada uma das etapas de desenvolvimento de sistemas de programação possui requisitos de comunicação próprios. Conseqüentemente, cada uma destas etapas requer uma linguagem de representação apropriada aos requisitos de comunicação específicos da etapa. Estes requisitos dependem do grau de abstração, do problema a ser resolvido, do redator e do leitor das representações geradas com esta linguagem. Ou seja, linguagens de repesentação possuem:

Nível de abstração- o grau de detalhe com que os fatos devem ser representados. Por exemplo: especificação essencial, especificação de arquitetura, código de programa, etc.

Ponto de vista- a natureza dos fatos a serem representados. Por exemplo: funcionalidade, organização dos dados, etc.

As conseqüências destas observações são:

1- é inviável imaginar uma única linguagem de representação para registrar todos os fatos descritores do sistema objetivo.

2- é necessário que existam regras de transformação de uma linguagem de representação para outra.

Representações registram somente fatos. Para transformar estes fatos em informação, é necessário que o leitor compreenda precisamente o significado destes fatos. A extração da informação contida numa representação é, portanto, fortemente dependente do leitor. Conseqüentemente, deve ser possível apresentar a mesma informação em diversos formatos e modos de agregação. Cada um destes formatos é dirigido a uma categoria de leitores. Desta forma facilita-se ao leitor obter um perfeito entendimento da informação contida na representação. É razoável, agora, esperar que o leitor passe a apontar erros, omissões, inadequações etc., contribuindo para uma melhoria substancial da qualidade do conteúdo da representação. Agora, a revisão da representação passa a contribuir efetivamente para uma melhoria substancial da qualidade do sistema de programação.

3.7. Atividades do desenvolvimento de software

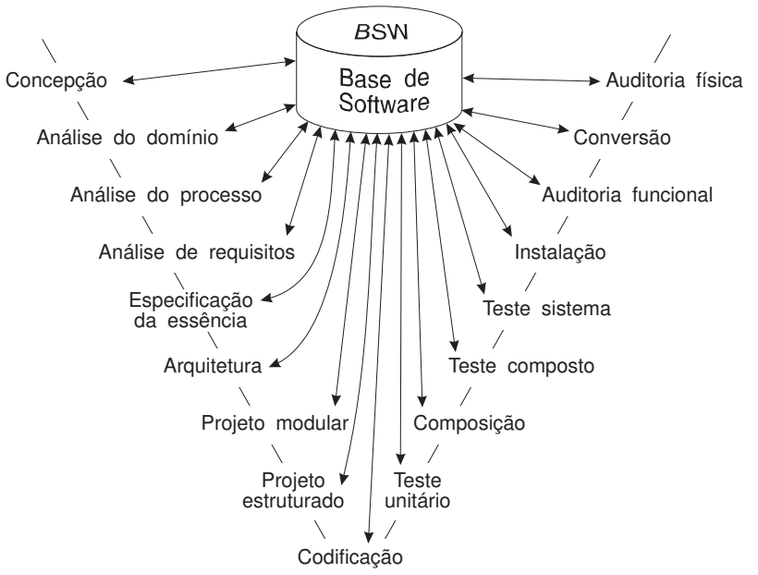

Figura 13. Etapas do desenvolvimento interagindo com uma base de software

O processo de desenvolvimento é uma rede de atividades interagindo com a base de software. O resultado de uma atividade é propagado para outras atividades através da base de software. Ou seja, o processo de desenvolvimento é, essencialmente, um processo de *povoamento e manutenção* do conteúdo da base de software. Uma versão do sistema objetivo estará alcançada, quando a correspondente base de software estiver totalmente povoada e consistente. Isto pode ocorrer ao final do desenvolvimento, como pode ocorrer, também, ao término de etapas de manutenção. Utilizando linearizadores ou geradores, pode-se gerar os programas fonte que constituem o software objetivo. Cabe salientar que esta forma *incremental* de produzir software é, em princípio, independente de processos de desenvolvimento específicos, tais como modelo cascata, modelo espiral, etc. Permite, então, que se defina processos adequados às necessidades pontuais da equipe e do sistema objetivo a ser desenvolvido.

Ao utilizar preceitos da Engenharia de Software, o desenvolvimento de sistemas de programação evolui do geral para o detalhe, ver figura 13. O desenvolvimento começa com uma idéia do software a desenvolver, ou da alteração a realizar. A seguir são definidos os objetivos e os requisitos a serem satisfeitos. Finalmente, são identificadas as interfaces com o universo, as funções do sistema e a organização dos dados. Na medida do possível utilizam-se especificações já padronizadas. Por exemplo, pode-se utilizar um dicionário de dados corporativo como instrumento de apoio à geração de especificações.

Durante a especificação podem ser conduzidos experimentos baseados em simulações e/ou protótipos. É importante, pois, que um ambiente de desenvolvimento apóie a construção de modelos e de protótipos. É importante, também, que este ambiente facilite a incorporação dos conhecimentos assim adquiridos nas especificações do produto a ser efetivamente desenvolvido.

Uma vez de posse da especificação, é arquitetada uma solução que satisfaça esta especificação. Ao produzir a arquitetura do sistema é produzido, também, o plano de desenvolvimento deste sistema. Este, em última análise, vem a ser a especificação do processo de desenvolvimento e ser efetuado ("enacted"). Após, é projetada e implementada esta solução. A implementação se dá passo a passo, desenvolvendo componentes, integrando estes componentes formando compostos, até que, eventualmente, o sistema esteja totalmente desenvolvido.

Independentemente do processo de desenvolvimento efetivamente utilizado para construir o sistema, as atividades apresentadas na figura 13 serão sempre cumpridas. Ao definir um processo de desenvolvimento específico, as atividades são particionadas em tarefas, cada uma resultando em produtos intermediários bem definidos. Neste momento pode-se organizar o processo de desenvolvimento de modo a aproveitar mais paralelismo, a obedecer a necessidades de técnicas de representação específicas, a utilizar simulação e/ou prototipação, etc. Como foi visto anteriormente, o processo de desenvolvimento é na realidade um processo de povoamento da base de software. A ordem deste povoamento é pouco relevante, desde que o resultado seja completo e consistente. Conseqüentemente, pode-se escolher a ordem de povoamento que melhor se adeqüe ao sistema objetivo e à equipe de desenvolvimento.

3.8. Produção com controle de qualidade

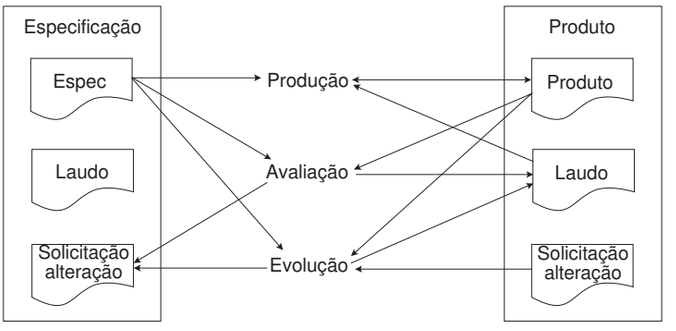

Figura 14. Produção com controle de qualidade passo a passo

Durante o desenvolvimento a qualidade das representações será constantemente controlada. O controle de qualidade visa assegurar que se está desenvolvendo *corretamente* o sistema *correto.* Pode-se visualizar o controle de qualidade como sendo a atividade conclusiva da produção de uma representação.

O controle de qualidade pode ser realizado por meios automatizados, tais como controladores de sintaxe, verificadores dirigidos por regras e heurísticas programadas, etc. O controle de qualidade pode ser realizado por pessoas através de avaliações e/ou testes. As avaliações podem ser realizadas com base em critérios de avaliação da qualidade. É

essencial que as linguagens de representação possuam mecanismos de apoio a todas essas formas de controle de qualidade. Em particular é necessário que representações possam ser inspecionadas por pessoas. Para tal é necessário que as representações sejam legíveis e inteligíveis pelos diversos avaliadores, leigos ou não em computação.

Inicialmente, o produto é gerado a partir da especificação. Cabe salientar que esta pode ser formada por um conjunto de representações. O produto resultado da produção pode ser qualquer coisa, por exemplo: uma especificação, um projeto, um módulo, fragmentos de código, documentação externa, arquivos de "helps", etc. Uma vez disponível, o produto é avaliado com relação a sua especificação ver figura 14. Evidentemente, a especificação identifica os requisitos de qualidade a serem satisfeitos. O resultado do controle de qualidade é:

1- um *laudo*, contendo todos os desvios relativos ao *produto* sendo avaliado. O laudo será utilizado para corrigir o produto.

2- uma *solicitação de alteração*, contendo todos os desvios relativos à *especificação* do produto sendo avaliado.

Solicitações de alteração podem ser geradas *externamente*, por solicitação do usuário, por mudanças de contexto operacional, etc. Elas podem ser geradas *internamente*, em virtude de uma avaliação ou de um controle de evolução. As solicitações de alteração de um determinado produto serão examinadas ao controlar a evolução desse produto. O resultado pode ser um laudo que efetivamente provocará a alteração do produto. Pode ser, ainda, uma solicitação de alteração na especificação desse produto. Conduzido dessa forma, o controle de evolução coordena o processo de alteração e propagação de alterações.

As atividades de controle de qualidade são:

Verificação: examinar a qualidade da representação em si. Ou seja, examinar se a representação está corretamente redigida, se satisfaz os critérios de avaliação da qualidade, se obedece às normas técnicas e legais em vigor, se é legível e inteligível, etc.

Validação: examinar a qualidade do conjunto de representações geradas até o momento. Ou seja, examinar se a representação corresponde a uma correta transformação ou reflexão de uma outra representação, verificar se o conjunto de representações é consistente, etc.

Aprovação: examinar a qualidade do serviço do sistema ou componente a que se refere a representação. Ou seja, examinar se o sistema ou componente corresponde às necessidades e aos desejos do usuário. A aprovação poderá ser preditiva, por exemplo, ao examinar uma especificação com o intuito de assegurar que o sistema se comportará conforme esperado pelo usuário. A aprovação poderá ser efetiva, por exemplo, ao testar um componente do sistema, ou ao conduzir experimentos utilizando um protótipo.

3.9. Desenvolvimento utilizando geradores

Geradores podem ser utilizados para gerar protótipos, ou para gerar produtos diretamente utilizáveis, ver figura 15. Estes produtos podem ser programas, textos ou outros produtos intermediários do processo de desenvolvimento [Balzer81].

Protótipos têm sido utilizados para aprovar especificações. A aprovação de especificações pode ser difícil em virtude da abstração, ou mesmo do despreparo dos que devem participar do controle de qualidade. Através de protótipos gerados diretamente a partir das especificações, pode-se tornar menos penosa essa aprovação.

Desenvolvendo geradores capazes de produzir código (ou produtos intermediários) de qualidade razoável e possuindo interfaces humanas também razoáveis, pode-se gerar programas utilizáveis em produção diretamente a partir de especificações. Dispensa-se,

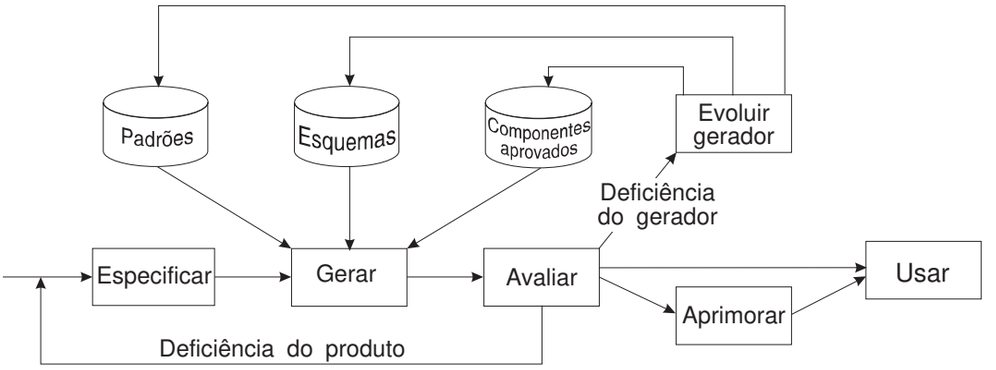

Figura 15. Processo de geração, ou transformação, de produtos

assim, o processo caro e demorado de desenvolver manualmente o programa de produção. É evidente que isto contribui para uma substancial redução do esforço de desenvolvimento, ao mesmo tempo que assegura um elevado nível de qualidade. No entanto, geradores não devem requerer um elevado número de parâmetros, tampouco devem requerer uma linguagem de especificação complexa, uma vez que isto jogaria por terra as vantagens da geração, pois o esforço a ser despendido se deslocaria meramente das etapas de projeto e construção para a etapa de especificação, sem sofrer uma redução significativa.

A geração pode ser realizada combinando esquemas de solução com as especificações em um processo semelhante ao de geração de macros. Neste caso, os esquemas de solução são esqueletos de especificações, arquiteturas, planos, programas junto com regras de instanciação e preenchimento. Os esquemas e as correspondentes regras de instanciação levam a soluções comprovadas. Em outras palavras, a instanciação dos esquemas tende a produzir soluções corretas por construção, sendo a causa de defeitos quase exclusivamente a falta de entendimento do problema a resolver (erros de especificação). Os esquemas são instanciados durante o processo de geração, selecionando-se e adicionando-se informações provenientes da especificação. Note que esta mesma técnica pode ser adotada para realizar transformações de uma representação para outra. Ou seja, o mecanismo de transformação pode, em princípio, ser visto como um mecanismo de instanciação de esqueletos.

3.10. Engenharia reversa e reengenharia

Existe hoje um enorme volume de software já desenvolvido. Este software vem sendo usado e mantido, em geral de forma satisfatória, apesar dos elevados custos de manutenção e evolução. Existem autores que afirmam que os custos da manutenção aumentam em virtude da gradativa deterioração da estrutura do software provocada pela própria manutenção [Glass81]. Por outro lado, é economicamente inviável querer reconstruir uma parcela significativa dos sistemas deteriorados que se encontram em uso. Para "salvar" tais sistemas, dando-lhes uma expectativa de vida mais longa, tem sido proposta a engenharia reversa seguida da re-engenharia.

Em sua forma típica, a engenharia reversa procura criar a documentação de engenharia do sistema a partir do código fonte, ver figura 16. Em geral, este é o único documento técnico confiável que resta de um sistema mais antigo. Em outros casos, pode-se querer criar ou alterar uma representação antecedente em função de uma representação conseqüente gerada ou modificada. De posse das representações antecedentes devidamente "revertidas", reorganizam-se, ou melhor, "reengenheiram-se", as diversas representações, agora na direção normal: de antecedente para conseqüente. É claro que este processo, se realizado sem ferramentas, é caro e demorado e, conseqüentemente, pouco trilhado.

A engenharia reversa e a re-engenharia podem ocorrer, entre outras, nas seguintes situações:

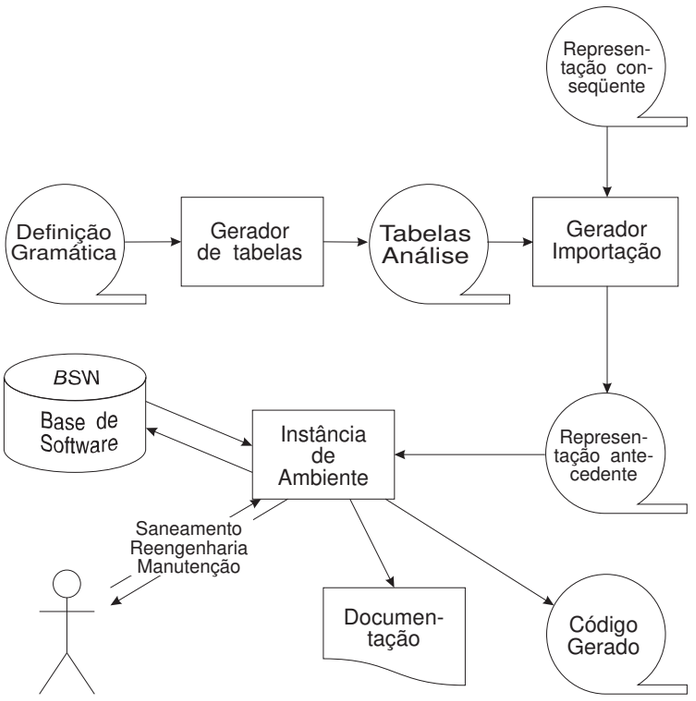

Figura 16. Processo de engenharia reversa

1- recuperar a documentação de engenharia de um sistema em uso.

2- corrigir erros de projeto e/ou especificação durante o desenvolvimento.

3- generalizar partes de um sistema de modo que possa vir a ser reultilizado.

4- reorganizar a especificação e o projeto de um sistema ao migrar de uma linguagem para outra com características diferentes. Por exemplo ao migrar de C para C++, ou de DOS para Windows.

Ao utilizar uma instância de ambiente, a engenharia reversa e a reengenharia podem ser realizadas nas seguintes formas:

1- se a representação origem da reengenharia já se encontra na base de software, utilizam-se as próprias ferramentas da instância.

2- se a representação origem não se encontra na base de software, mas existe em algum arquivo ou mesmo em outra ferramenta, gera-se um arquivo de importação e importa-se esta representação. Em alguns casos geração do arquivo de importação pode ser realizada com auxílio de ferramentas [Guedes93].

3- se a representação existe somente sob a forma de documento deve ser criada utilizando os editores disponíveis. Neste caso pode ser interessante criar uma linguagem de representação igual ou próxima à linguagem utilizada no documento papel.

3.11. Desenvolvimento cooperativo em equipe

O desenvolvimento de sistemas raramente é um trabalho realizado por uma única pessoa. Normalmente várias pessoas cooperam formando uma equipe e realizando diversas atividades. Cada membro da equipe utiliza uma estação de trabalho, operando uma instância de ambiente própria, ver figura 17. O ambiente de desenvolvimento é, agora, formado por esta coletânea de instâncias de ambiente. Cada uma destas instâncias poderá ser diferente, porém todas terão sido geradas a partir do meta-ambiente. Um ambiente precisa então ser capaz de prover apoio a várias instâncias de ambiente operando simultaneamente.

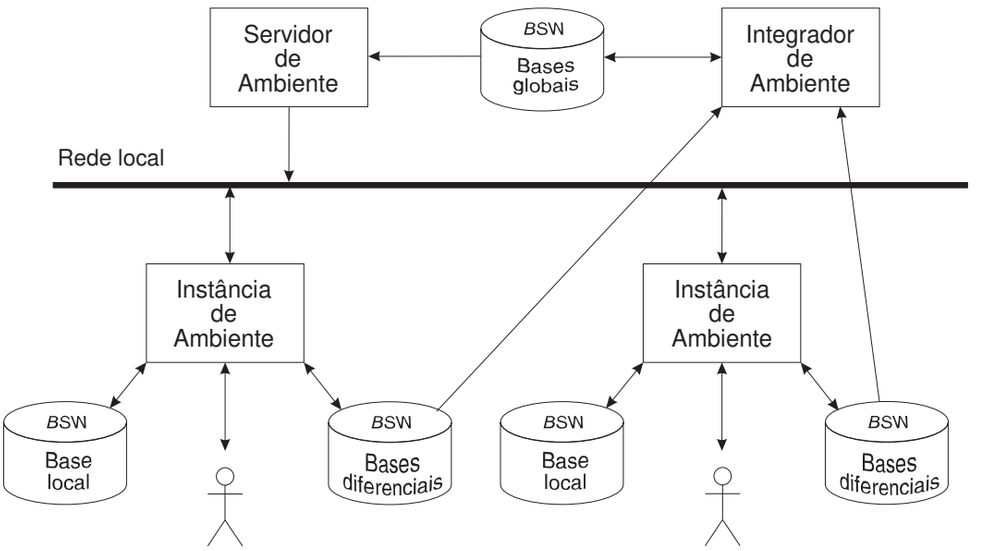

Figura 17. Ambiente de desenvolvimento cooperativo

A criação ou alteração de representações é, em geral, uma atividade demorada e que, e virtude do forte interrelacionamento entre representações, opera com um volume grande de dados. Nesta condições, mecanismos de sincronização de acesso convencionais têm apresentado dificuldades. Optamos, assim, por utilizar a seguinte organização:

1- internamente a cada base, os atributos estão sujeitos a controle de versão e gerência de configuração. Várias *versões* de um mesmo atributo poderão existir em uma determinada base de software. Através da definição de *configurações*, são selecionadas as versões do atributo que formam uma determinada composição (*construto*, "build"). A gerência de configuração procura assegurar a integridade de cada construto, levando em conta a evolução dos sistemas e de seus componentes.

2- bases locais contêm os atributos que formam as representações sendo alteradas. Estas bases locais podem ser cópias de bases públicas e estão sujeitas a controles do gênero *retirar* / *devolver* ("check-in" / "check-out"). Somente cópias retiradas podem ser alteradas diretamente. A cada momento poderá existir, no máximo, uma cópia retirada.

3- bases diferenciais contêm os atributos das bases globais, não retiradas, alterados ao editar uma representação. De tempos em tempos, usualmente após aprovar uma representação alterada, as bases diferenciais são integradas às bases globais.

4. Requisitos essenciais

Nesta seção identificaremos os requisitos que, no nosso julgamento atual, devem ser satisfeitos por qualquer ambiente que tenha por objetivo ser integrado e abrangente, de modo que efetivamente apóie todas as atividades do processo de desenvolvimento.

4.1. Requisitos do meta-ambiente

Os requisitos primários do meta-ambiente TOTEM são:

Requisito 1: o conjunto de linguagens de representação a ser suportado no contexto do meta-ambiente é variável. Ou seja, dependendo da evolução do conhecimento, *deve-se poder adicionar novas linguagens de representação ao conjunto de linguagens suportadas.*

Figura 18. Modelo entidade/relacionamento estendido da base de software.

Requisito 2: o conjunto de linguagens de representação utilizadas em uma determinada instância de ambiente é variável. Ou seja, dependendo da natureza do sistema a desenvolver e da equipe, *deve-se poder selecionar o conjunto de linguagens de representação a serem utilizadas*.

Requisito 3: a composição das linguagens de representação é variável. Ou seja, dependendo das necessidades instantâneas do desenvolvimento, *deve-se poder ajustar as classes de objetos que compõem cada uma das linguagens de representação utilizadas.*

Requisito 4: o conjunto de atributos de cada classe de objetos é variável. Ou seja, para um dado projeto e uma dada etapa de desenvolvimento dentro do projeto, *deve-se poder definir e ajustar (dinamicamente) os atributos de cada classes de objetos manipulados por estas linguagens de representação.*

Requisito 5: os critérios de controle de qualidade são variáveis, evoluindo junto com o desenvolvimento e/ou manutenção. Ou seja, *deve ser possível definir dinamicamente o conjunto de regras de controle de qualidade a serem aplicadas.*

Requisito 6: As especificações, projetos e implementações são inerentemente variáveis no tempo, devido ao aprendizado relativo ao sistema e à sua implementação, devido a transformações sócio-técnicas introduzidas pelo sistema posto em uso, e devido a condições externas impostas aos sistemas (legislação, práticas empresariais, evolução tecnológica etc.). Conseqüentemente, *o ambiente deve suportar e prover mecanismos de propagação para alterações realizadas em qualquer lugar e a qualquer momento.*

Requisito 7: a seqüência ideal de criação e de manutenção (processo de desenvolvimento) das representações é variável. Tampouco é fixa a seqüência de detalhamento a ser obedecida em determinado projeto. No entanto, sabe-se que existem formas de apresentação do resultado que facilitarão a manutenção dos sistemas de programação. Entende-se por manutenção o conjunto de atividades de leitura, acesso e edição, realizadas por pessoas (usualmente diferentes das que desenvolveram o sistema ou componente), visando a evolução e/ou adaptação do sistema a novos requisitos. A manutenção deve *aprimorar*, ou *pelo menos manter*, a qualidade original do sistema, inclusive de sua documentação técnica. Ou seja, *a organização ideal das representações deve ser assegurada, mesmo que o processo de desenvolvimento real efetuado não tenha seguido explicitamente uma ordenação de tarefas com esta organização em mente.*

Requisito 8: As linguagens de representação e seus respectivos complementos dicionarizados devem poder possuir *nível de abstração* (omissão deliberada de detalhes de especificação, projeto e/ou de implementação) e *ponto de vista* (visão deliberadamente parcial focalizando em um dado aspecto do sistema). Ou seja, *as representações correspondem a visões parciais de todos os fatos que constituem o conhecimento sobre o sistema objeto sendo desenvolvido e/ou mantido.*

Requisito 9: As linguagens de representação e seus respectivos complementos dicionarizados devem admitir a decomposição hierárquica generalizada, assegurando que se possa vencer a complexidade utilizando níveis de abstração uniformes. Ou seja, *em cada nível de abstração, deve-se poder descrever uma parte do sistema utilizando diagramas e especificações em um mesmo nível de abstração.*

Requisito 10: Ao reduzir o nível de abstração, deve ser possível elaborar todos os componentes participantes desta nova visão, mantendo a uniformidade de abstração também neste novo nível. Ou seja, *a decomposição sucessiva deve permitir a elaboração uniforme do nível de abstração.*

Requisito 11: As informações registradas em uma representação devem poder ser utilizadas em outra representação sem necessitar a redigitação, a reescrita, ou o redesenho destas informações. Ou seja, *deve ser possível interrelacionar objetos mesmo que façam parte de diferentes representações. Em adição, deve ser*

possível observar a evolução dos requisitos ao longo da documentação técnica do sistema.

Requisito 12: Os fatos registrados em uma estação de trabalho devem poder ser utilizados em outras estações, mesmo que estas operem com outros ambientes. Ou seja, *deve ser possível estabelecer vínculos de comunicação e cooperação entre as diferentes estações de trabalho utilizadas pela equipe de desenvolvimento, mesmo que o conjunto de ambientes seja heterogêneo.*

4.2. Requisitos das instâncias de ambiente

Da mesma forma como se deve definir requisitos para o meta-ambiente, devem ser definidos requisitos a serem satisfeitos por qualquer uma das instâncias de ambiente geradas. Cada instância gerada pelo meta-ambiente TOTEM deve satisfazer os seguintes requisitos:

Requisito 1: *deve ter qualidade satisfatória* de modo que possa ser efetivamente utilizada como instrumento de desenvolvimento de sistemas objeto (confiabilidade, utilidade, utilizabilidade, desempenho).

Requisito 2: *deve permitir a rápida adaptação às necessidades específicas do projeto e da equipe de desenvolvimento*.

Requisito 3: *deve permitir o compartilhamento seletivo de todas as informações de uma representação em outra representação.*

Requisito 4: *deve apoiar todas as atividades do processo de desenvolvimento*, incluindo aí especificação, arquitetura, projeto, codificação, geração de programas compiláveis e/ou executáveis, controle de qualidade, manutenção e gerência do processo de desenvolvimento.

Requisito 5: *deve operar eficientemente em plataformas generalizadamente disponíveis*.

Requisito 6: *deve apoiar e coordenar o trabalho em equipe*.

Requisito 7: *deve enfatizar o reúso de fatos registrados em épocas anteriores*, quer seja neste projeto, quer seja em outros projetos, até mesmo em projetos realizados sem o emprego do ambiente assistido por computador específico.

4.3. Requisitos da base de software

A base de software é um banco de dados que mantém os fatos relativos a um determinado sistema, o *sistema objeto*. O sistema gerente da base de software deve satisfazer os seguintes requisitos:

Requisito 1: *deve suportar operações de edição (transações) longas*.

Requisito 2: *deve apoiar operações de navegação típicas a hipertextos*.

Requisito 3: *deve poder ser incrementalmente povoada*, mesmo que semanticamente os dados estejam incorretos e/ou incompletos durante longos períodos de tempo. Isto obriga o emprego de dois níveis de representação, um de *armazenamento* (estrutura) garantidamente correto a cada instante, e outro *semântico*, cuja validade deverá ser controlada de tempos em tempos através de validadores.

Requisito 4: *deve assegurar eficiência de armazenamento e acesso de valores de tamanho variável*, (fragmentos de texto virtualmente ilimitados e de diagramas gráficos também de tamanho virtualmente ilimitado).

Requisito 5: *deve viabilizar a operação com objetos*.

Requisito 6: *deve viabilizar a adição dinâmica de novas linguagens de representação.*

Requisito 7: *deve viabilizar a adição e/ou alteração dinâmica de classes de objetos, relacionamentos e atributos em representações já armazenadas.*

Requisito 8: *deve viabilizar a interligação de diferentes bases físicas.*

Requisito 9: *deve prover mecanismos de apoio ao armazenamento de múltiplas versões*, de apoio ao controle de versão e de apoio à gerência de configuração.

4.4. Composição da base de software

A base de software armazena objetos, diagramas e dicionários. Cada um destes elementos é um plexo possuindo diversos atributos. Em adição podem existir diversas relações entre os elementos.

O modelo de dados da base de software não procura refletir propriedades específicas das linguagens de representação, ver figura 18 na página 21. Ou seja, o modelo de dados é um meta-modelo e trata somente das categorias lingüísticas (objetos, atributos, dicionários, diagramas etc.) utilizadas pelas linguagens de representação. A instanciação deste modelo se dá através da base de conhecimento.

Cada classe de objeto corresponde a um elemento lingüístico. São exemplos: depósito de dados, processo, entidade, módulo, bloco, tarefa, fluxo de dados, diagramas de fluxo de dados, diagramas de entidades e relacionamentos etc. Para cada classe de objeto é definido um dicionário. Dicionários agregam objetos de uma mesma classe, ordenando-os alfabeticamente por nome.

Cada objeto pode ter diversos atributos. Os atributos de objetos são particionados em categorias. As categorias de atributos são:

Nome: é a identificação única do objeto. Cada objeto possui exatamente um nome, este estará sempre no correspondente dicionário.

Aliases: são nomes ou identificações alternativas do objeto. São usados tipicamente como nomes abreviados. Aliases não são dicionarizados. Cada objeto pode possuir diversos aliases. A semântica de cada aliás é determinada pelo uso que se faz dele.

Fragmentos de texto: são seqüências de zero ou mais (ilimitado) linhas de texto. São usados tipicamente para registrar especificações formais, descrições informais, texto de código, texto de documentação, etc. Cada classe de objeto pode possuir diversos fragmentos de texto. A semântica de cada fragmento de texto é determinada pelo uso que se faz dele.

Valores estruturados: são registros compostos por diversos campos e possuindo um formato fixo. Os campos podem ser inteiros curtos e longos, data, hora, e seqüências de caracteres de tamanho definido. Intuitivamente, um valor formatado é uma linha de uma tabela em um banco de dados relacional. Este valor desprovido de chave, já que a chave é a do atributo do objeto. Os campos de valores formatados não podem ser chaves. Cada classe de objeto pode possuir diversos diferentes valores formatados. A semântica de cada fragmento de cada valor formatado é determinada pelo uso que faz dele.

Relacionamentos: estabelecem vínculos entre dois objetos, usualmente, mas não necessariamente, distintos. Os relacionamentos são agrupados em relações. Relações definem permissões de se estabelecer relacionamentos entre objetos de duas classes. As classes inter-relacionadas podem ser iguais ou diferentes, podendo, até, pertencer a diferentes linguagens de representação. Cada classe de objeto pode ter diversas relações com outras classes e/ou consigo mesmo. Para cada uma das relações pode exsistir um número ilimitado de relacionamentos.

Diagramas: são representações gráficas. Cada diagrama é desenhado sobre um única superfície imaginária, a *folha eletrônica*. Diagramas são, por sua vez, objetos, conseqüentemente todas as regras que valem para objetos valem também para

Figura 19. Modelo entidade/relacionamento estendido da meta-base de conhecimento

diagramas. Objetos podem estar relacionados a diagramas através da relação "especificado por", indicando que o diagrama especifica um determinado aspecto deste objeto. Ou seja, do ponto de vista semântico, diagramas são plexos que descrevem graficamente determinado aspecto de um objeto.

Diagramas sozinhos são instrumentos de especificação incompletos. Para completá-los, é necessário adicionar especificações mais detalhadas a cada uma das instâncias de elemento que figuram no diagrama. Estas especificações correspondem aos atributos do objeto instanciado no diagrama. Para facilitar o manuseio das especificações junto com diagramas, pode-se acessar diretamente as especificações das instâncias de objeto contidas no diagrama, bastando, para isto, posicionar-se em algum lugar ocupado por esta instância de objeto.

Diagramas são armazenados de forma otimizada, assegurando a busca e o redesenho eficiente a cada vez que venham a ser exibidos. O redesenho a partir da base de software assegura que o diagrama reflita exatamente o conteúdo da base, independentemente de edições realizadas pelos diferentes editores, ou possivelmente até por ações externas.

De uma forma simplificada, diagramas são formados pelos seguintes elementos:

Instâncias de objetos: são ocorrências, nos diagramas, de referências a objetos. As instâncias de objeto definem ainda a posição geográfica do objeto no diagrama. Um objeto pode possuir diversas instâncias, possivelmente em diferentes diagramas. O editor gráfico tem acesso ao objeto diretamente através de qualquer uma de suas instâncias. Isto permite, por exemplo, navegar-se diretamente da instância gráfica para a sua especificação. Instâncias de objeto podem ser hiper-nós, no sentido que armazenam dentro delas outras instâncias e, possivelmente, operadores sobre estas instâncias [Harel].

Ligações: estabelecem a existência de uma interfaces entre duas instâncias de objeto. Indiretamente estabelecem a existência de relações entre os objetos correspondentes a estas instâncias. No entanto, do ponto de vista estrito da base de software, é desconhecido o fato de ligações poderem definir indiretamente uma relação. Ligações podem ou não ter rótulos associados a elas.

Rótulos: especificam a natureza das interfaces estabelecidas pelas ligacões que rotulam. Na realidade, rótulos são instâncias de objetos, utilizados como atributos do relacionamento definido pela correspondente ligação.

Terminações: detalham a semântica das relações estabelecidas através de ligações. Por exemplo em diagramas de fluxo de dados definem a direção do fluxo; em diagramas entidades e relacionamentos definem a cardinalidade do relacionamento.

4.5. Composição da base de conhecimento

A base de conhecimento define as propriedades de uma instância de ambiente. Como mencionado anteriormente, a instanciação se dá através da criação de diferentes bases de conhecimento. Desta forma pode-se manter constantes os programas, independentemente da configuração específica a cada instância de ambiente.

A base de conhecimeto é particionada em duas bases fisicas. Na primeira, a *meta-base*, encontram-se todas as definições não acessíveis ao usuário. Esta base é ler somente. São exemplos de definições contidos na meta-base: as tabelas de analise sintática, as tabelas de símbolos, as listas de mensagens e a meta-linguagem de representação. A meta-linguagem de representação é utilizada para definir instâncias de ambientes. Na segunda, a *base de instância*, encontram-se todos os parâmetros variáveis de controle da estação de trabalho, bem como as definições das diferentes linguagens de representação disponíveis na instância de ambiente.

A figura 19 na página 25 apresenta o modelo da meta-base de conhecimento. Intuitivamente, este modelo corresponde à *linguagem de definição de dados* (DDL) da base de software. O modelo define os componentes que formam linguagens de representação. O conteúdo da base de instância é dirigido por este modelo. Por sua vez, o conteúdo da base de instância determina como os editores devem interagir com a base de software de modo a produzir representações segundo a linguagem definida.

As principais definições contidas na base de instância são:

1- a relação das linguagens de representação disponíveis na instância de ambiente.

2- conjunto de classes de objeto e respectivos atributos para cada uma das linguagens de representação disponíveis na instância de ambiente.

3- definições das linguagens gráficas. Estas especificam a forma de *representação externa* (como o usuário vê), a *representação interna* (como é armazenada na base de software) e a *representação de edição* (como se interage para criar e/ou alterar as representações gráficas).

4- programas de formulários de especificação, definem os atributos a serem extraídos, a diagramação da tela, os títulos etc. que compõem uma determinada busca (“query”, texto de especificação) de um objeto de determinada classe, visando a edição ou a geração de relatórios a partir de um objeto desta classe.

5- programas de formulários de validação, definem as regras de controle sintático e semântico do conteúdo de um determinado objeto ou diagrama. A aplicação de um formulário de validação a um dado objeto gera o laudo de validação deste objeto. Este laudo conterá a lista de defeitos identificados ao efetuar a validação. Por intermédio dos formulários de validação pode-se assegurar a validade semântica da base de software.

6- programas de formulários de linearização, descrevem o processo de busca e diagramação do conteúdo da base de software visando a geração de arquivos contendo programas compiláveis, textos formatáveis, declarações de dados, disposições de

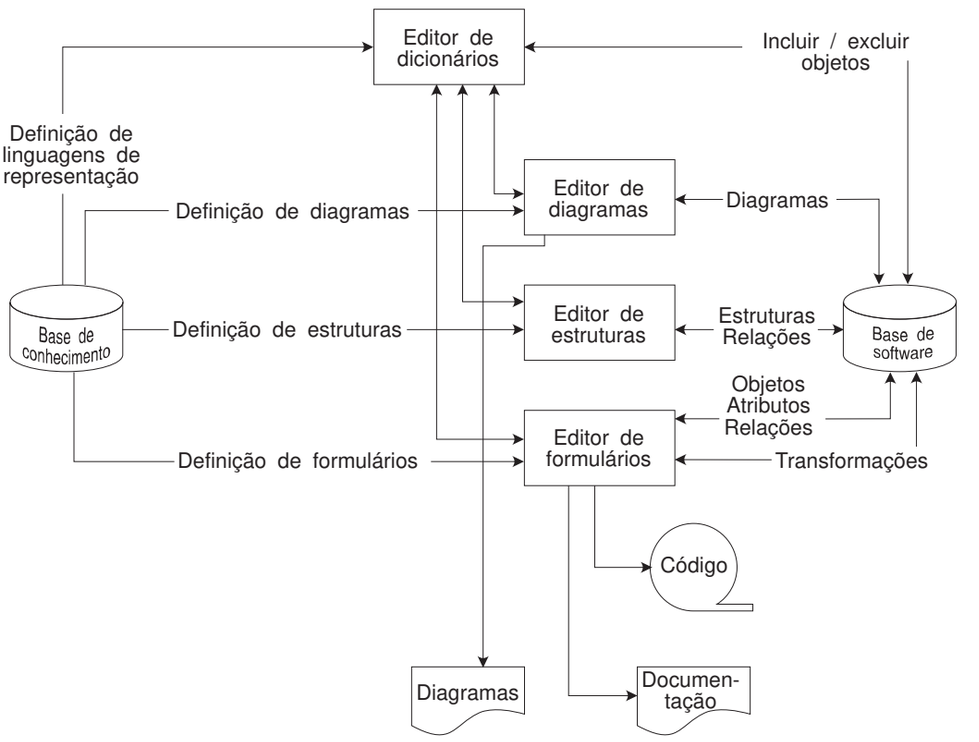

Figura 20. Arquitetura de meta-editores

campos em telas e/ou relatórios, etc. Os programas de formulários de linearização podem ser utilizados, ainda, para a geração de protótipos de programas. Por exemplo é possível gerar protótipos de programas de manutenção de bancos de dados a partir dos respectivos modelos de dados definidos na linguagem entidade e relacionamentos estendida.

7- programas de formulários de transformação, descrevem o processo de transformação de uma linguagem de representação em outra.

4.6. Arquitetura de meta-editores

O código de programa de todas as instâncias de ambiente, inclusive do meta-ambiente é igual. Este código contém, além do sistema de gerência de banco de dados, diversos editores integrados entre si. Todos estes editores são *desinterpretados*, isto é, efetuam operações de edição desconhecendo a semântica do que está sendo editado. A interpretação, ou seja, a semântica, é definida pelos programas de formulário e demais regras e tabelas contidos na base de conhecimento, ver figura 20.

O *editor de diagramas* apóia a criação e a alteração de diagramas quaisquer. A base de conhecimento instancia o editor de diagramas para cada uma das linguagens de representação gráficas disponíveis em uma instância de ambiente. Cada instância de objeto exibida possui uma especificação no correspondente dicionário. Pode-se ter acesso a esta especificação diretamente, posicionando-se na instância e solicitando a edição da especificação. Ou seja, o editor de diagramas permite a manipulação direta dos elementos exibidos.

O *editor de formulários* edita os atributos de um ou mais objetos buscados e exibidos segundo um determinado programa de formulário. O editor de formulários é dirigido por programas de formulários. Estes programas são redigidos em uma linguagem própria e disponível ao usuário da instância de ambiente. Desta forma o usuário pode realizar adaptações locais, sem afetar as propriedades básicas da instância de ambiente, tais como definidas pelo Administrador de Ambiente. O programa de formulário, ver figura 21, definem os atributos a serem buscados, as expressões a serem avaliadas, sub-formulários a serem aplicados, condicionalmente ou não, quantificadores sobre conjuntos de objetos, diagramação de apresentação, emissão de laudos etc. Através da programação do editor de formulários pode-se especificar editores de especificações, validadores de especificações e/ou diagramas, linearizadores de especificações e/ou diagramas, geradores de relatórios etc. Intuitivamente, o editor de formulário pode ser entendido como uma *linguagem de manipulação dos dados* (DML) contidos na base de software que, além de buscar os dados, os formata, exibe e permite editar.

```
Prototype Public void "Edit process"( Void ) ;
Prototype        void "Flows"( Integer In  ) ;

DefineForm void "Edit process"( Void )

    Indent 3 ;
    Title "Process name: " ;
    NoNewLine ;
    Name ;
    Title "Informal description" ;
    Text Description ;

    Form "Flows"( IN  ) ;
    Form "Flows"( OUT ) ;

    Title "Relation of implementing code blocks" ;
    Relation BrogramBlocks ;

EndForm
```

Figure 21. Ilustração de programa de formulário

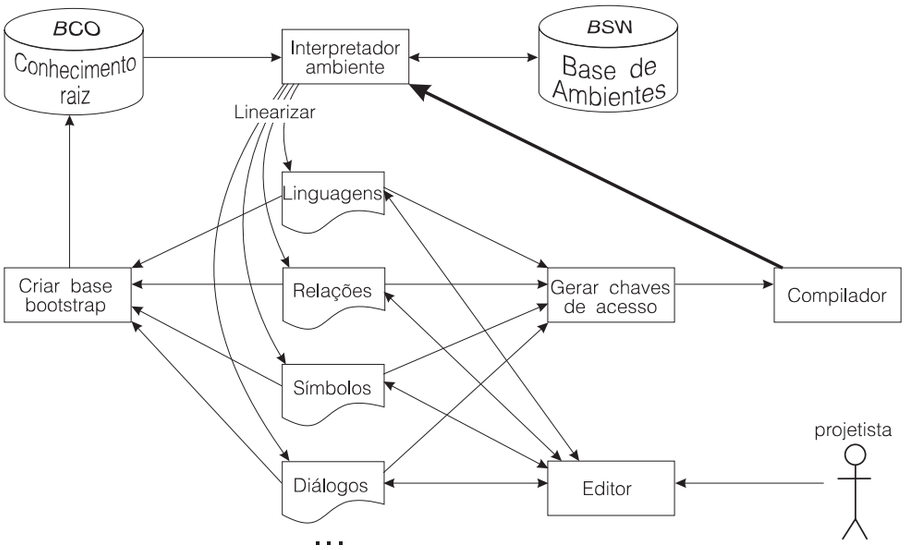

Figura 22. Geração “bootstrap” do meta-ambiente.

Em adição à capacidade de editar formulários, o editor de formulários pode navegar entre diferentes especificações (formulários) em uma modalidade semelhante à da navegação em hipertextos. Basta para isto, posicionar-se no nome do objeto (usualmente contido em uma relação) e solicitar a edição da especificação deste objeto.

O *editor de estrutura* edita estruturas formadas por objetos de uma mesma classe. O editor de relações pode ser configurado, por exemplo, para editar estruturas de projeto de programas. Através de linearizadores estas estruturas podem ser convertidas para arquivos seqüenciais a serem posteriormente compilados. Editores de estruturas podem ser utilizados para produzir programas, para produzir documentos, para projetar hierarquias de interface humana, para definir hierarquias de tarefas (“work break down structures”), etc.

O *editor de dicionários* é um mecanismo de acesso aos objetos contidos na base de software. Em adição a isto, o editor de dicionários permite a inserção, exclusão e alteração (via outro editor) dos objetos contidos na base de software.

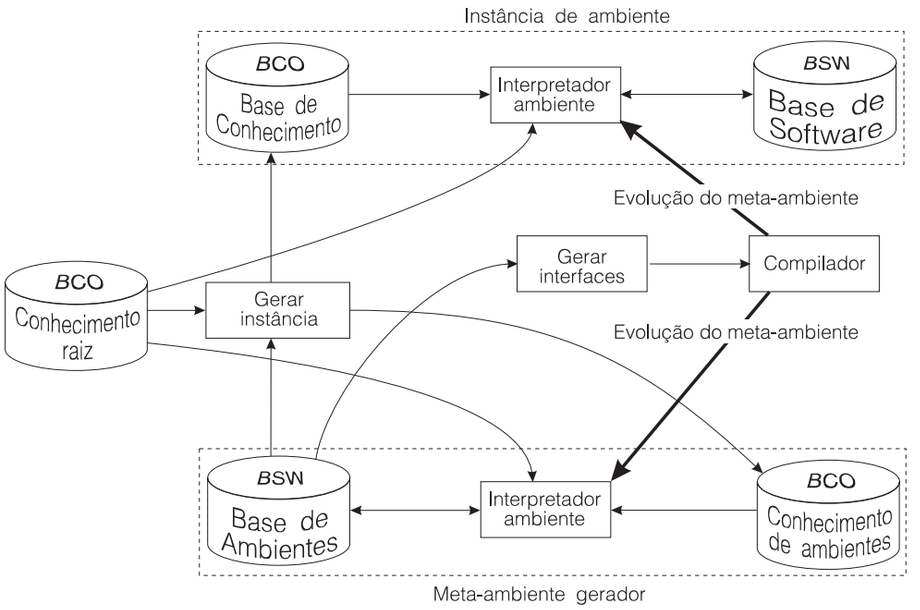

Figura 23. Manutenção das definições de ambiente com o meta-ambiente

4.7. Criação e manutenção do meta-ambiente

Como já havia sido mencionado, o código que constitui o meta-ambiente e todas as instâncias de ambiente é igual. A instanciação se dá via a geração de uma base de conhecimento, a base de instância. No entanto, para que o meta-abiente possa operar, ele precisa da meta-base. Esta tem, essencialmente, a mesma organização que a base de instância. Porém a sua criação não pode ser efetuada via os editores do meta-ambiente. Necessita-se, então, de um gerador "bootstrap".

O gerador "bootstrap" da base de conhecimento é um programa que converte tabelas "bootstrap" para o formato binário interno da meta-base, ver figura 22. Estas tabelas definem as mensagens, os diálogos, as linguagens de comando dos diversos editores, e o modelo da meta-base. As tabelas "bootstrap" são tipicamente arquivos de texto. A meta-base é gerada pelo programa *criar base de conhecimento*. O programa interpretador de ambientes precisa ser capaz de encontrar eficientemente os elementos de contidos na base de conhecimento. Para conseguir isto, ele necessita de chaves de acesso aos elementos contidos na meta-base. Estas chaves estão diretamente vinculadas às tabelas "bootstrap" a partir das quais é gerada a meta-base. Para evitar erros de transcrição no uso de chaves de acesso, é utilizado o programa *gerar chaves de acesso* para criar as tabelas de definição de chaves a partir das tabelas "bootstrap".

Uma vez criado o gerador "boot strap" pode-se criar uma instância de ambiente cuja finalidade é editar as definições variantes dos ambientes, ver figura 23. Este é o *meta-ambiente gerador*. O meta-ambiente gerador possui linearizadores que geram os dados necessários para o gerador de base de conhecimento. Desta forma fecha-se o ciclo de instanciação e manutenção de ambientes.

5. Epílogo

O projeto TOTEM visa contribuir para um aprimoramento substancial do processo de desenvolvimento de software. Para atingir este objetivo optou-se por desenvolver um meta-ambiente. O meta-ambiente, bem como as instâncias de ambiente devem possuir qualidade industrial, capacitando-os de serem efetiva e eficazmente utilizados por equipes encontradas em empresas. Para alcançar estes objetivos, o desenvolvimento e a industrialização dos resultados de pesquisa são realizados em cooperação com empresas. O processo de pesquisa e desenvolvimento utilizado baseia-se em um modelo de desenvolvimento incremental com aprendizado.

Bibliografia e referências bibliográficas

[Ambriola91] Ambriola, V.; Ciancarini, P.; Corradini, A.; DeFrancesco, N.; "Towards Inovative Software Engineering Environments"; *Journal of Systems and Software* 14; 1991; pags 17-29

[Balzer81] Balzer, R.; "Transformational Implementation: An Example"; *IEEE Transactions on Software Engineering* SE-7(1); janeiro 1981; pags 3-14

[Barstow86] Barstow, D.R.; Shrobe, H.E.; Sanderwall, E.; eds. *Interactive Programming Environments*; McGraw-Hill; 1986

[Bigelow88] Bigelow, J.; "Hypertext and CASE"; *IEEE Software*; março 1988; pags 23-27

[Boehm88] Boehm, B.; "A Spiral Model of Software Development and Enhancement"; *Computer* 1988; pags 61-72

[Boudier89] Boudier, G.; "An Overview of PCTE and PCTE+"; *SIGPLAN Notices* 24(2); fevereiro 1989

[Brerenton88] Brerenton, P. ed. *Software Engineering Environments*; Halstead, John Wiley; New York; 1988

[Brooks86] Brooks, F.P.; "No Silver Bullet - Essence and Accidents of Software Engineering"; *IEEE Computer* ; outubro 1986; pags 10-19

[Brown92] Brown, A.E.; Penedo, M.H.; "An Annotated Bibliography on Integration in Software Engineering Environments"; *Software Engineering Notices* 17(3); julho 1992; pags 47-55

[CASE90Rep] "CASE '90 REPORT: Results of the Fourth International Workshop on Computer Aided Software Engineering"; in *[CASE92]*; pags 289-326

[CASE92] Forte, G.; Madhavji, N.H.; Müller, H.A. eds.; *Proceedings of the Fifth International Workshop on Computer Aided Software Engineering*; Montreal; July 1992; IEEE 92CH3166-6;

[Charette86] Charette, R.N.; *Software Engineering Environments*; McGraw Hill; 1986

[Chorafas88] Chorafas, D.N.; Legg, S.J.; *The Engineering Data Base*; Butterworth; 1988

[Conradi86] Conradi, R.; Didricksen, T.M.; Wanwick, D.M. *Advanced Programming Environments*; Lecture Notes in Computer Science; Springer; Berlin; 1986

[Conte86] Conte, S.D.; Dunsmore, H.E.; Shen, V.Y.; *Software Engineering Metrics and Models*; Benjamin Cummings; 1986

[Cybulski92] Cybulski, J.L.; Reed, K.; "A Hypertext Based Software Engineering Environment"; *IEEE Software* maio 92; pags 62-68

[Dart87] Dart, S.A.; Ellison, R.J.; Feiler, P.H.; Haberman, N.; "Software Development Environments"; *Computer*; novembro 1987; pags 18-28

[Dowson87] Dowson, M.; "ISTAR An Integrated Project Support Environment"; *SIGPLAN Notices* 22(1); Janeiro 1987; pags 27-33

[Fickas85] Fickas, S.F.; "Automating the Transformational Development of Software"; *IEEE Transactions on Software Engineering*; novembro 1985; pags 1268-1277

[Fisher88] Fisher, A.S.; *Using Software Development Tools*; Wiley; New York; 1988

[Freeman84] Freeman, P.; Staa, A.v.; *Towards a Theory of Software Engineering*; Technical Report #242; Information and Computer Science; University of California, Irvine; novembro 1984;

[Forte92] Forte, G.; "Out of the Lab, onto the Shelf"; *IEEE Software* maio 92; pags 70-79

[Glass81] Glass, R.L., Noiseaux, R.A.; *Software Maintenance GuideBook*; Prentice-Hall, Inc.; New Jersey, 1981.

[Guedes93] Guedes, L.C.; Staa, A.v.; "Um Processo de Re-Engenharia Econômico e Eficaz"; VII Simpósio Brasileiro de Engenharia de Software; PUC-Rio; outubro 1993

[Hagerty91] Hagerty, M.; "Making a CASE for Software Design"; *Dr. Dobbs Journal*; janeiro 1991; pags 108-114

[Harrison89] Harrison, W.H.; Shilling, J.J.; Sweeney, P.F.; "Good News, Bad News: Experience Building a Software Development Environment Using the Object-Oriented Paradigm"; *OOPSLA 89*; outubro 1989; pags 85-94

[Harrison92] Harrison, W.H.; Kavianpour, M.; Ossher, H.; "Integrating Coarse-grained and Fine-grained Tool Integration"; in [CASE92]; pags 23-35

[Henderson87] Henderson, P.B.; Notkin, D.; "Integrated Design and Programming Environments"; *Computer*; novembro 1987; pags 12-16

[Humphrey89] Humphrey, W.S.; Managing the Software Process; Reading; Addison-Wesley; 1989

[Jones86] Jones, C.; *Programming Productivity*; McGraw-Hill; New York; 1986

[Katz90] Katz, R.H.; "Towards a Unified Framework for Version Modeling in Engineering Databases"; *ACM Computing Surveys* 22(4); dezembro 1990; pags 375-407

[Kempe89] Kempe, M.; "Hyperbook: An Experiment with PCTE"; *Software Engineering Notes* 14(5); julho 1989; pags 55-61

[Kishida88] Kishida, K.; Katayama, T.; Matsuo, M.; Miyamoto, I.; Ochimiziu, K.; Saito, N.; Sayler, J.H.; Torii, K.; Williams, L.G.; "SDA: A Novel Approach to Software Environment Design and Construction"; *IEEE*; 1988; pags 69-79

[Lai93] Lai, R.; "The Move to Mature Processes"; IEEE Software 14(7); julho 1993; pags 14-17

[Lamsweerde88] Lamsweerde. A.V. et al.; "Generic Life-Cycle Support in the ALMA Environment"; *IEEE Transactions on Software Engineerng* 14(6); 1988

[Lehman85] Lehman, M.M.; Belady, L.; *Program Evolution - Processes of Software Change*; Academic Press; 1985

[McLure88] McLure, C.; *CASE is Software Automation*; Prentice Hall; 1988

[Notkin88] Notkin, D.; "The Relationship Between Software Development Environments and the Development Process"; *Software Engineerng Notes* 13(5); 1988

[Oesterweil87] Oesterweil, L.; "Software Processes are Software Too"; *Proceedings of the International Conference on Software Engineering*; 1987

[Page89] Page, T.W.; Berson, S.E.; Cheng, W.C.; Muntz, R.R.; "An Object-Oriented Modeling Environment"; *OOPSLA 89* Proceedings; outubro 1989; pags 287-296

[Parker90] Parker, T.; "CASE on the PC: The State of the Art"; *Computer Language* 7(1); janeiro 1990; pags 89-97

[Partridge86] Partridge, D.; *Artificial Intelligence - Applications in the Future of Software Engineering*; Halstead, John Wiley; New York; 1986

[Penedo93] Penedo, M.H.; Riddle, W.; "Process-sensitive SEE Architecture (PSEEA), Workshop Summary"; *Software Engineering Notes* 19(5); julho 1993;

[Pintelas91] Pintelas, P.; Tragoudas, S.; "A Comparative Study of Five Language Independent Programming Environments"; *Journal of Systems and Software* 14; 1991; pags 3-15

[Rosenblatt89] Rosenblatt, W.R.; Wileden, J.C.; Wolf, A.L. "OROS: Toward a Type Model for Software Development Environments"; *OOPSLA 89*; outubro 1989; pags 297-304

[Perry88] Perry, D.E.; Kaiser, G.E.; "Models of Software Development Environments"; *IEEE*; 1988; pags 60-67

[Sinson89] Sinson, W.; "Views of Software Development Environments: Automation of Engineering and Engineering of Automation"; *Software Engineering Notes* 14(5); julho 1989; pags 108-117

[Staa85] Staa, A.v.; *Projeto MOSAICO, Especificação de Requisitos*; Monografias em Ciência da Computação, Departamento de Informática, PUC/RJ; 1985

[Staa85] Staa, A.v.; Freeman, P.; *Requirements for Software Engineering Languages*; Technical Report #85-08; Information and Computer Science; University of California, Irvine; janeiro 1985;

[STAA92a] STAA Informática; *Ambiente de Desenvolvimento de Software TALISMAN - Manual de Referência*; Versão 4.2; Rio de Janeiro; 1992

[STAA92b] STAA Informática; *Ambiente de Desenvolvimento de Software TALISMAN - Resumo Técnico*; Rio de Janeiro; 1992

[Stenning87] Stenning, V.; "On the Role of an Environment"; *Proceedings of the International Conference on Software Engineerng*; 1987

[Taylor90] Taylor, R.N. ed.; Proceedings of the Fourth ACM SIGSOFT Symposium on Software Development Environments; *Software Engineering Notes* 15(6); dezembro 1990

[Taylor90] Taylor; R.N.; Belz, F.C.; Clarke, L.A.; Osterweil, L.; Selby, R.W.; Wileden, J.C.; Wolf, A.L.; Young, M.; "Foundations for the Arcadia Environment Structure"; in [Taylor90]; pags 1-13

[Thomas92] Thomas, I.; Nejmeh, B.A.; "Definitions of Toll Integration for Environments"; *IEEE Software* maio 92; pags 29-35

[Tsai88] Tsai, J.J.P.; Ridge, J.C.; "Intelligent Support for Specifications Transformation"; *IEEE Software*; novembro 1988; pags 28-35

[Wileden86] Wileden, J.C.; et alii (editors); Proceedings of an International Workshop on the Software Process and Software Environments; *Software Engineering Notes* 11(4); 1986