Universidade Federal da Paraíba
Centro de Ciências e Tecnologia
Departamento de Sistemas de Computação

Uma Ferramenta de Gerência de Projeto - GEPRO

Dissertação de Mestrado

por

José Belo Torres

Campina Grande, Dezembro de 1996

Universidade Federal da Paraíba
Centro de Ciências e Tecnologia
Departamento de Sistemas de Computação

José Belo Torres

Uma Ferramenta de Gerência de Projeto - GEPRO

Dissertação apresentada ao curso de mestrado em informática da Universidade Federal da Paraíba, como requisito parcial à obtenção do título de mestre em Informática.

Área de concentração : Sistema de Informação e Banco de Dados

Ulrich Schiel, Phd
(Orientador)

José Antão Beltrão Moura, Phd
(Có-Orientador)

Campina Grande - Pb
Dezembro de 1996

T693f Torres, Jose Belo
Uma ferramenta de gerencia de projeto : GEPRO / Jose Belo Torres. - Campina Grande, 1996.
61 f. : il.

Dissertacao (Mestrado em Informatica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Projetos de Software 2. Desenvolvimento de Softwares 3. Gerencia de Projeto - 4. Gerencia de Projeto - I. Schiel, Ulrich, Dr. II. Moura, Jose Antao Beltrao, Dr. III. Universidade Federal da Paraiba - Campina Grande (PB)

CDU 004.051(043)

UMA FERRAMENTA DE GERÊNCIA DE PROJETO - GEPRO

JOSÉ BELO TORRES

DISSERTAÇÃO APROVADA EM 16.12.96

PROF. ULRICH SCHIEL, Ph.D
Presidente

PROF. JOSÉ ANTÃO BELTRÃO MOURA, Ph.D
Examinador

PROF. JORGE CESAR ABRANTES DE FIGUEIREDO, Dr.
Examinador

PROF. DÉCIO FONSECA, Dr.
Examinador

CAMPINA GRANDE - PB

Agradecimentos

- Aos professores Ulrich e Antão por suas competentes orientações dadas ao projeto

- A Janete pela sua contribução na elaboração do projeto

- Ao Hermes pela competente revisão do trabalho

- Ao Washinton pela amizade, pelo incentivo e pela força durante o desenvolvimento do projeto

- A Laura pelo seu incentivo e motivação

- A Sônia e Augusto pelo apoio e colaboração na apresentação da defesa da dissertação

- Ao Reginaldo e Haroldo companheiros de discussões e de trabalhos acadêmicos

- Ao pessoal do NPD, em especial ao Caratti, Claúdia, Josin, Leandro e Régis pelas colaborações e discussões sobre o projeto

- Ao professor Pita, Alberto, Robson e todos aqueles que nos apoiaram para conclusão desse trabalho

Ao Professor Álvaro(In memorium) e

a Dona Alcência(In memorium)

Resumo

Os desperdícios, prazos esgotados e altos custos são constatados diariamente em diversos tipos de projetos.

Em projetos de Software, as metodologias estão preocupadas mais unicamente com o componente de construção, não se preocupando com os outros dois componentes: o de avaliação e o de gerência.

Os Ambientes de Desenvolvimento de Software automatizam os processos impondo aos projetos uma maior qualidade e produtividade no desenvolvimento de projeto de Software.

Em decorrência desses problemas é que nos motivamos a desenvolver uma ferramenta de gerência de projeto para um Ambiente de Desenvolvimento de Software, denominado DYNAMO, com o objetivo de diminuir os riscos de projetos e dota-los das características desejáveis de um ambiente, como também, fazer com que as metodologias possam ser utilizadas na sua totalidade, ou seja, possua os componentes de avaliação e de gerência.

Para desenvolvermos esta ferramenta, tivemos que criar uma rede, inspirada em Redes de Petri, para modelagem dos processos de desenvolvimento com o objetivo de controlar as múltiplas atividades de projetos de Software.

A ferramenta de gerência de projeto, GEPRO, utiliza-se dessa rede integrando esta ferramenta a uma metodologia com objetivo de controlar recursos humanos, recursos operacionais, prazos e custos, como também, avaliar e aperfeiçoar os produtos e processos de desenvolvimento.

O controle, avaliação e aperfeiçoamento serão vistos através dos módulos de especificação e controle da ferramenta com o apoio de métricas de Software.

Portanto acreditamos que a GEPRO contribua para simplificar e tornar eficiente os aspectos de gerência de projeto de Software.

Abstract

Wastes, missed deadlines and high costs are daily observed in several kinds of designs.

In Software Designing, methodologies are most concerned with the construction component, not focusing on other two components: management and evaluation.

The Software Engineering Environments automate processes by imposing, to the projects, better quality and productivity in Software Design Developing.

As a result of these problems, we have been motivated to develop a design management tool for a Software Development Environment, called DYNAMO. The objectives are reducing design risks, providing desirable features of an environment to the project, as well as allowing the methodologies to be used in their totality, that is, including the evaluation and management components.

To develop this tool, we had to create a network, inspired in Petri's Network, to model development processes in order to control many activities of Software Design.

The Design Management Tool, GEPRO, uses such network by integrating this tool to a methodology with the objective of controlling human resources, operational resources, deadlines, costs, as well as evaluating and improving the products and the processes of development.

The control, the evaluation and the improvement are observeded by the specification and control modules of the tool with the agreement of Software metrics. Therefore we believe that GEPRO contributes to simplify and to give more efficiency to management aspects of the Software Design.

Lista de Figuras

Capítulo 1

Introdução

1.1 Motivação

Atualmente, em diversas áreas, encontramos projetos onde os desperdícios, defeitos e prazos esgotados são constatados diariamente. Na informática, percebemos isso, até mesmo com mais ênfase, onde projetos são desenvolvidos sem nenhum critério de medição de qualidade, portanto, sem nenhum controle.

Um dos problemas práticos, encontrados em decorrência dos citados acima, é a dificuldade do gerenciamento de grandes projetos. Esses grandes projetos são os desafios que os cientistas de Computação têm enfrentado e procurado solucionar, apostando na integração de diversas áreas como Banco de Dados, Engenharia de Software e Inteligência Artificial.

Um problema específico em desenvolvimento e manutenção de grandes projetos de Software é que seus processos são difíceis de gerenciar. Uma idéia, com o objetivo de se controlar esses processos, faz-se através de um plano de projeto explícito, o qual orienta seus desenvolvedores a controlá-los de um modo bem compreensivo [Lot93].

Segundo Monarchl [Mon92], as metodologias para desenvolvimento de Software são divididas em três componentes distintos: **Construção**, **Avaliação** e **Gerência** dos Processos de Desenvolvimento.

As metodologias, hoje, disseminam-se. Atualmente, existem muitas metodologias e outras mais sendo desenvolvidas preocupadas com a construção, ou seja, com as técnicas de elaboração de produtos em cada fase de seu ciclo de vida, sem se importar com seus outros dois componentes, ou seja, a avaliação e a gerência.

Muitos ambientes têm sido desenvolvidos e muitas métricas têm sido criadas para apoiar à gerência de projeto. Mas, o que se sente é a falta de uma pesquisa integrada de ambientes e métricas, onde a engenharia de Software possa atuar realmente como uma disciplina de engenharia.

Em decorrência desses problemas, é que propomos um trabalho que contemple esse ponto muito seriamente: a introdução em um ambiente de desenvolvimento de Software de ferramentas que possam avaliar o progresso, a qualidade e a produtividade do projeto.

1.2 A Gerência, as Métricas e os Ambientes de Desenvolvimento de Software - ADSs

Os grandes projetos de Software envolvem cooperações entre múltiplos desenvolvedores que trabalham juntos com o objetivo de facilitar a integração das diversas atividades de projeto. É essencial que os ADSs permitam a esses múltiplos desenvolvedores trabalharem juntos no mesmo projeto concorrentemente, facilitando a gerência de projetos.

Não se pode controlar o que não se pode medir. Não se pode tomar decisões rápidas e precisas se as informações não estiverem disponíveis a qualquer instante durante o desenvolvimento de projeto. Não se pode obter qualidade, e portanto, produtividade, se os trabalhos dos engenheiros de Software não puderem ser compartilhados ou cooperados [Lot93].

Métricas de Software são úteis para o gerenciamento e monitoramento do projeto de Software. Caso os dados de métricas estiverem disponíveis para o gerente em uma freqüência básica, ações corretivas podem ser tomadas, incrementando a probabilidade de sucesso completo do projeto. O feedback mais cedo, quando da ocorrência de alguns desvios, é de grande importância para o bom andamento do projeto.

Os ADSs apóiam os desenvolvedores de projetos de Software na sua automatização através de trabalhos cooperativos, coleta de dados automática, troca de materiais, monitoramento on-line dos processos, para citar alguns exemplos.

ADSs estão sendo pesquisados com o intuito de automatizar os processos de desenvolvimentos de Software, objetivando alcançar uma maior qualidade em seus produtos e processos. As métricas são reconhecidas como um modo de se controlar o processo de desenvolvimento de projeto. A integração dessas áreas de engenharia de Software com outras, como Banco de Dados, é de fundamental importância para criação de grandes projetos de Software.

1.3 Objetivos

Esse trabalho apresenta uma ferramenta de gerência de projeto de Software para o ambiente DYNAMO (Dynamic Management of Object), denominada GEPRO, utilizando-se de métricas de Software com o objetivo de se controlar e aperfeiçoar os processos de desenvolvimento. O ambiente DYNAMO é um projeto em desenvolvimento pelo grupo de Sistema de Informação e Bancos de Dados do DSC/UFPb, para criação de um ADS com características temporais e ativas. Ele inclui duas metodologias de projetos de sistemas de informações: POKER e FADO [1] [Fu93].

[1] Ferramenta de Análise e Desenvolvimento Orientado a Objetos

FADO é uma metodologia de projeto de sistema de informação orientada a objetos e que tem como objetivo gerar uma especificação de um esquema conceitual no Modelo de Dados Orientado à Objeto e Temporal - TOM [Fu93].

A ferramenta GEPRO foi aplicada aos processos da metodologia FADO com o objetivo de controlar custos, prazos, recursos humanos, recursos operacionais, processos e produtos gerados no desenvolvimento de projeto de Software. A GEPRO pode ser aplicada a outras metodologias, basta que essas metodologias sejam acopladas a essa ferramenta.

Ela objetiva apoiar os desenvolvedores de projetos de Software na sua especificação e no seu controle. Históricos obtidos de projetos desenvolvidos anteriormente através do controle das atividades planejadas permitem que no futuro estimativas de prazos e custos possam ser elaboradas. A característica aberta da ferramenta possibilita a incorporação de novas metodologias para o ambiente.

O objetivo é buscar a qualidade e a produtividade no desenvolvimento de projeto de Software. Qualidade buscada quando da utilização de métricas disponíveis para o gerente de projeto de Software em uma freqüência básica. Produtividade quando da obtenção da qualidade de seus produtos, evitando trabalhos repetitivos ou manutenção nas diversas fases de seu ciclo de vida.

Portanto, este trabalho se propõe a desenvolver uma ferramenta de apoio à gerência de projeto com o objetivo de controlar/gerenciar e avaliar o componente de construção da metodologia FADO do ambiente DYNAMO, tornando-a assim, completa. Hoje esta metodologia possui somente o componente de construção. A ferramenta GEPRO objetiva, também, aperfeiçoar os processos de desenvolvimento de Software através de suas medidas.

1.4 A Organização do Trabalho

Neste capítulo mostramos alguns problemas que motivaram este trabalho, como também, levantamos os principais objetivos do trabalho.

Um Overview sobre gerência de projetos de Software, métricas de Software e Ambientes de Desenvolvimento de Software é apresentado no capítulo 2, considerando os aspectos da qualidade e da produtividade no desenvolvimento de projetos. Exemplos de algumas métricas existentes também são mostradas.

No capítulo 3, são levantados e analisados alguns ADSs desenvolvidos ou em desenvolvimento no meio acadêmico. Ferramentas de gerência para projetos em geral, no mercado, são citadas e analisadas. Também, são introduzidos o ambiente DYNAMO, a ferramenta FADO e o TOM.

Um modelo de rede para controlar as múltiplas atividades do projeto de Software, baseado em Redes de Petri, é definido no capítulo 4. Essa rede será a principal responsável no apoio à GEPRO no controle dos processos, produtos, recursos e equipe de projeto.

A ferramenta GEPRO é apresentada no capítulo 5. Os processos da metodologia FADO são modelados através da GEPRO dentro da definição de nossa rede. Esses modelos são utilizados pela GEPRO nos seus módulos de Especificação e Controle do projeto.

As conclusões, contribuições e trabalhos futuros foram levantados no capítulo 6. Novas metodologias poderão utilizar a GEPRO, como também, novas ferramentas poderão ser incorporadas ao ambiente. Com isso, uma atividade poderá ter como função fazer chamadas a essas ferramentas dando um maior grau de automatização ao ambiente.

Capítulo 2

A Qualidade e a Produtividade no Desenvolvimento de Projetos de Software

2.1 Introdução

Um dos maiores problemas práticos encontrados no desenvolvimento de grandes projetos de Software é o seu gerenciamento. O desenvolvimento de Software, grandes e crescentes, é o desafio que os cientistas de computação têm enfrentado e procurado solucionar.

Os grandes projetos de Software começaram a alcançar maiores níveis de complexidades nos meados dos anos 60. A complexidade dos grandes projetos os tornava de difícil entendimento e seus requisitos eram freqüentemente difíceis de serem entendidos. Como resultado, foram especificações freqüentemente não confiáveis e ineficientes, documentação pobre e muita manutenção após sua liberação. Métodos e ferramentas associadas suportavam o desenvolvimento de um projeto de Software, mas, forneciam pouco suporte para fazer frente aos desafios intelectuais de atender adequadamente aos grandes projetos. Isto foi largamente reconhecido como a crise do Software [PW88].

No inicio dos anos 70, novas técnicas apareceram como projeto top-down, modularidade, e encapsulamento [GM91]. Novas metodologias, utilizando-se dessas técnicas, como por exemplo, programação estruturada de Jackson, análise e projeto estruturado [GM91], foram desenvolvidas. Novos métodos de gerenciamento como organização dos processos dentro das fases do ciclo de vida do Software, aplicações de técnicas de caminhos críticos, estimativas de custos, e novas ferramentas automatizadas, como sistema de gerenciamento de configuração, analisadores de risco de teste e analisadores de fluxos de dados, apareceram também [GM91]. Embora existissem esforços para integração dessas ferramentas, essas foram desenvolvidas primeiramente voltadas para técnicas e ferramentas individuais. Às técnicas, métodos e ferramentas, foi dado o nome de Engenharia de Software, um termo que foi escolhido para enfatizar que os seus resultados teriam um custo efetivo, gerenciável, métodos auditáveis, bem suportado com um conjunto de ferramentas integradas [PW88].

Da crise do Software dos meados dos anos 60 para os dias de hoje, muitos conceitos, metodologias, linguagens, ferramentas e técnicas foram introduzidas com o objetivo de aperfeiçoar os processos de desenvolvimento de Software e seus produtos. O Software Engineering Institute (SEI) vem se preocupando com este problema através do desenvolvimento de um trabalho que se constitui no Modelo de Maturidade e Capacidade (CMM) que consta de cinco níveis que indicam, em uma ordem crescente, uma maior maturidade dos processos de desenvolvimento de Software. Para maiores informações ver Tate [TJR92]. A seguir as descrições dos seus cinco níveis:

1 Inicial - Desenvolvimento ad hoc o qual é ainda muito comum;

2 Repetitível - As tarefas são repetitivas e previamente controladas. É o inicio da gerenciabilidade;

3 Definido - É caracterizado por um processo de desenvolvimento bem definido;

4 Gerenciado - O processo é medido e controlado, e os relacionamentos entre atividades são quantitativamente entendidos;

5 Otimizado - Fornece uma coleção de dados para alteração, atualização e otimização dos seus processos;

No restante da seção 2.1, descrevemos sobre gerência, métricas e ambientes de desenvolvimento de forma independente. Na seção 2.2, damos uma visão de métricas no apoio à gerência de projetos de Software. Os ambientes no apoio à gerência de projetos de Software e a descrição de suas taxonomias e arquiteturas são vistas na seção 2.3. Na seção 2.4, elaboramos nossas conclusões.

2.1.1 A Gerência de Projetos de Software

Gerenciamento é um sistema de procedimentos, práticas, tecnologias e know-how que fornece o planejamento, organização, staffing, direção e o controle necessários para o sucesso da gerência de projetos [Tha87].

Gerenciamento é definido, também, como todas as atividades e tarefas entendidas por alguém ou mais pessoas através de uma proposta de planejamento e controle. As atividades devem ser coordenadas e realizadas independentemente [GM91].

Os princípios e técnicas de gerência são bastante gerais e não são específicos para Software. Entretanto, estimativas de custos, controle de qualidades de processos e produtos e métricas de complexidades são algumas técnicas e metodologias típicas da gerência de Software. Portanto, atividades da gerência podem ser suportadas por uma variedade de ferramentas, algumas adequadas para o gerenciamento em geral, outras específicas para projetos de desenvolvimento de Software [GM91].

As questões de gerenciamento crescem mais com a maior complexidade dos projetos. Assim, o aproveitamento das ferramentas de gerenciamento fica mais relevante em grandes projetos.

Uma instituição, que trabalha em grandes projetos de Software, tem observado que o maior gargalo encontrado no seu desenvolvimento é o seu gerenciamento efetivo. Esforços são realizados para localizar o problema e construir padrões bem definidos para sua gerência. Esses padrões incluem monitoramento e coordenação de etapas para serem realizados por membros do projeto, a progressão adequada de uma fase para outra, o gerenciamento efetivo dos produtos de desenvolvimento de Software, o uso correto e ótimo de métodos e ferramentas existentes, feedback de erros realizados por

membros de projetos, a propagação de informações (sumários e relatórios) para a pessoa certa no tempo certo [RS88].

Análise e controle de risco é um outro tópico da teoria do gerenciamento. Várias técnicas padrões existem para identificação de riscos de projetos, avaliando seus impactos, monitorando-os e controlando-os. O conhecimento dessas técnicas permite ao gerente de projeto aplicá-las quando necessário para aumentar as chances de sucesso de um projeto. As ausências dessas técnicas corre o risco de construção de produtos errados ou ter seus requisitos trocados durante o seu desenvolvimento. Uma abordagem efetiva para redução desses riscos é o modelo incremental de Barry Bohem [Boh81].

2.1.2 As Métricas de Software

O desenvolvimento produtivo de alta qualidade dependerá de como os processos são implementados e gerenciados. Isto dependerá de muitos fatores tais como qualidade e experiência do staff técnico, complexidade do projeto, uso de ferramentas de Software, confiabilidade requerida do produto, expectativas dos clientes e outras variáveis.

Métricas são instrumentos utilizados para dar apoio aos desenvolvedores de projetos de Software nos mais diversos aspectos. Um dos mais importantes é o que trata do aperfeiçoamento dos processos de desenvolvimento por métodos quantitativos. Um outro aspecto importante é o de quantificar os atributos de qualidades do Software. Portanto, sua aplicação é efetivamente apoiar à gerência de projetos de Software [Wei93].

A utilização de métricas é útil no apoio à gerência de Software, portanto, métrica não é uma solução mágica, é uma ferramenta para perseguir produtividade e qualidade nos processos de desenvolvimento de Software [MD93].

As métricas fornecem visibilidade e controle para o projeto de Software, reduzindo sua aparente complexidade. São valiosas por fornecer uma orientação para o aperfeiçoamento dos processos de desenvolvimento de Software.

Os dados de métricas capacitam o gerente de projeto a medir os efeitos de suas ações, e a corrigí-los. Esses dados de métricas servem para indicar o estado atual do projeto e o seu nível de qualidade.

A implementação de um programa de métricas (modelo ou conjunto de métricas bem definido) resultará em muitos benefícios para uma instituição. Esses benefícios incluirão menores custos de desenvolvimento, como resultado de uma alta qualidade e produtividade nos projetos de Software, devido a um maior controle sobre os processos [MD93].

2.1.3 Os Ambientes de Desenvolvimento de Software - ADSs

Ambientes referem-se a uma coleção de ferramentas de Hardware e Software que desenvolvedores de sistemas utilizam para apoiar-lhes na construção de projetos de Software. Como tecnologias aperfeiçoam-se e crescem, as expectativas dos usuários de um ambiente funcional tendem a se alterar com o tempo. Nos últimos vinte anos, o conjunto de ferramentas de Software disponíveis para desenvolvedores foi consideravelmente expandido [DRP87].

Eles emergiram de um esforço para localizar os problemas associados com o desenvolvimento e manutenção em projetos de Software em grande escala de uma forma mais automatizada. Uma das principais questões em pesquisa de ADS é como construir ambientes que são integrados e ao mesmo tempo flexíveis e extensíveis [BKG92].

ADSs apóiam os desenvolvedores nas avaliações de processos de Software usando regras diretas, procedurais, ou medição de dados. O objetivo de tais sistemas é aperfeiçoar o controle sobre o projeto de Software. O sistema ideal deve fornecer uma orientação baseada em medição nos modos definidos pelos usuários. Embora medição pareça oferecer o mais objetivo, racional e útil método para entendimento, orientação e aperfeiçoamento de processos, ela é também freqüentemente negligenciada por pesquisadores de ADS [Lot93].

Orientação para manutenir e aperfeiçoar processos de Software é feita através da determinação de que atividades podem ser realizadas em cada ponto no tempo e detectar desvios do plano. Orientação e controle intelectual podem ser ainda mais realçados pela coleta e utilização da medição de dados para orientar os processos. Os benefícios de orientação e medição de um processo de Software incluem o trabalho cooperativo de engenheiros de Software e gerentes de projetos através de uma interface amigável. Coletar e utilizar medidas de dados podem ser assistidos e suportados em muitos caminhos por um ADS [Lot93].

A infra-estrutura deve permitir um ambiente aberto; isto é, o ambiente não deve ter uma coleção fixa de ferramentas, mas, ser fácil de enriquecê-lo progressivamente pelas incorporações de novas ferramentas com o tempo.

2.1.4 A Gerência, as Métricas e os ADS

Como pudemos notar, as métricas e ambientes são de grande utilidade no gerenciamento de projeto. A utilização de um ambiente que contemple métricas auxiliará, ainda mais, o desenvolvimento de projeto. A má ou não utilização desses recursos podem acarretar os seguintes problemas em projetos de Software:

1 - Incertezas das estimativas;

2 - Esquema de progresso de monitoração frágil;

3 - Controles efetivamente negligenciados.

Em resumo, não se consegue atingir maiores níveis no CMM do SEI. Portanto é importante que os gerentes utilizem um ambiente de desenvolvimento de Software através do qual possam medir o progresso e a qualidade do projeto.

2.2 A Gerência e as Métricas como Fatores de Qualidade e Produtividade

O aperfeiçoamento contínuo dos produtos e processos é um objetivo básico para se chegar à qualidade e produtividade. Aperfeiçoar processos requer medidas quantitativas das qualidades. Existe, assim, a necessidade de medições dos processos de Software e dos produtos que se constroem.

As principais etapas em gerência de projetos são o **Planejamento** e o **Controle**. Ambas as etapas utilizam-se de medidas para obtenção do sucesso do projeto. Portanto, métricas de Software são ferramentas úteis no apoio ao desenvolvimento de projetos de Software nos aspectos quantitativos da qualidade, tais que suas informações possam aperfeiçoar seus processos e, em conseqüência, a qualidade do projeto. Essas informações servem para indicar o estado atual e o nível de qualidade do projeto.

Uma métrica é um número indicando qualidade. Para IEEE, "Métricas de Qualidade de Software" é uma função, cujos inputs são dados de Software e outputs é um simples valor (número) que pode ser interpretado como o grau de qualidade de um atributo de qualidade. Portanto, métricas utilizam-se de indicadores que relatam o grau de qualidade do desenvolvimento do projeto necessário para orientar os gerentes de projeto.

O indicador é uma unidade necessária para computar um valor de métrica. A métrica de complexidade de Mccabe [GM91] indica os números de caminhos de execução de um programa para mostrar sua complexidade. Um alto valor para o fator de Confiabilidade é um indicador onde riscos de vidas humanas existem, como construção de aeronaves, por exemplo.

Um indicador de métrica de produto é usado para medir as características da documentação e código fonte de um programa. Um indicador de métrica de processo é usado para medir características dos métodos, técnicas e ferramentas empregadas na aquisição, desenvolvimento, verificação e operação do Software. Aplicação de métricas sem uma clara proposta confunde, pois elas devem ser usadas para fornecer visibilidade para engenheiros e gerentes; para fixar aderência para padrões, orientações e princípios; para estimar, prever; e aceitar o produto [Wei93].

Elas são úteis, também, para o gerenciamento e monitoramento de projeto de Software. Se os dados de medidas estão disponíveis para o gerente de projeto de Software em uma certa freqüência, ações corretivas podem ser tomadas para que o plano de projeto tenha sucesso completo. Portanto, a aplicação de métricas de Software é uma ferramenta para efetivamente gerenciar o projeto e aperfeiçoar os seus processos.

Em resumo podemos observar:

✓ Métricas são usadas para aperfeiçoar os processos de desenvolvimento de Software;

✓ Você não gerencia o que você não mede;

✓ O que você mediu e está fora do plano sofrerá correções de ações;

✓ Métricas são usadas para estimativas de prazos e custos de projetos através de dados obtidos de projetos desenvolvidos.

A implementação de um programa de métricas resultará em muitos benefícios para a obtenção da qualidade do projeto. Esses benefícios incluirão menores custos de desenvolvimento, resultado de uma alta produtividade, elevadas vendas, devido ao pequeno ciclo de desenvolvimento de Software, e satisfação do cliente resultante da boa qualidade do produto [MD93].

2.2.1 A Qualidade e a Produtividade no Desenvolvimento de Software por Métodos Quantitativos

Qualidade de Software é o grau com o qual um projeto de Software possui uma desejada combinação de métricas.

A utilização de indicadores de métricas é um fator de aperfeiçoamento dos processos de desenvolvimento e da qualidade e produtividade da equipe de desenvolvimento.

Freqüentemente, usuários de Software expressam suas necessidades em termos qualitativos, tais como Confiabilidade, Manutenibilidade, Portabilidade, Eficiência, etc. Requisitos de qualidade de Software são estratégias utilizadas para localizar o grau ou valores dos atributos de qualidades desejados. Para atender níveis desejados de alguns atributos de qualidade de Software em um grande projeto, muitas pessoas utilizam programas que vislumbrem métricas de um modo quantificado [Wei93].

Um programa de métricas de qualidade de Software realiza, portanto, uma importante função fornecendo um caminho para medir o grau de qualidade dos atributos de seus produtos e dos seus processos de desenvolvimento.

Engenheiros de Software interessados no aperfeiçoamento da qualidade têm identificado atributos gerais que se aplicam através dos diversos processos de desenvolvimento.

A utilização de métricas, no futuro, será aplicada mais freqüentemente para a prevenção de falhas dentro de um mecanismo de feedback, identificando onde têm ocorrido os problemas, para que os processos sejam aperfeiçoados. Incorporação de métricas dentro de atividades associadas com prevenção de falhas reduzirão o tempo de atraso, melhorando a qualidade e produtividade. Isto pode ser referenciado como um

metaprocesso, um processo que aperfeiçoa os processos de desenvolvimento de Software [MD93].

Este metaprocesso incluirá mecanismos e procedimentos para produzir os aperfeiçoamentos dos processos identificados, monitorar e trilhar as ações, coletar e manter dados de métricas, manter os treinamentos e documentações necessárias para novamente ser introduzido o aperfeiçoamento de processos [MD93].

Um exemplo corrente do uso de métricas, que será mais freqüentemente praticado no futuro, é a prevenção de defeitos de Software. O seu uso põe mais ênfase na prevenção durante o desenvolvimento do produto de Software ao invés de só na detecção de defeitos através de revisões e testes. Essa abordagem utiliza a análise de defeitos para entender suas causas como uma oportunidade para aperfeiçoamento dos processos de desenvolvimento, como, também, prevenir falhas futuras.

2.2.2. Exemplos de Métricas de Software

A figura 2.1 mostra as diversas métricas utilizadas para o modelo em cascata.

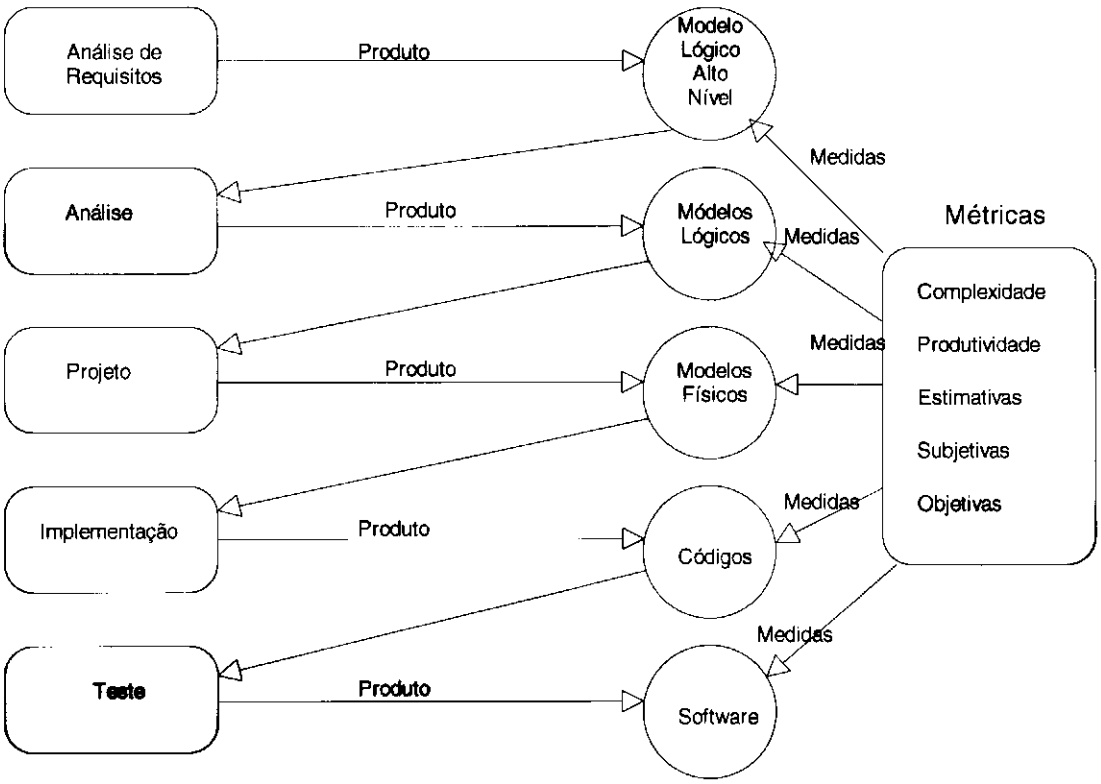


Figura 2.1 Métricas de Software

2.2.2.1 Métricas Para Estimativas de Prazos e Custos

Quanto tempo gastará um engenheiro de Software em uma dada tarefa é uma função da complexidade do problema, da habilidade do engenheiro, do projeto e as ferramentas

que estão disponíveis. Estimar a dificuldade da tarefa e quanto de cada tarefa cada engenheiro pode resolver é um problema da gerência de projetos de Software [MD93].

Medidas de projetos anteriores podem ser usadas, basicamente, como entradas para um modelo para estimar prazos e custos de projetos de Software. Essas são calculadas em função de parâmetros, como características dos projetos, ferramentas disponíveis, nível de qualidade do staff técnico, etc.

A maioria dos métodos de estimativas de custo de Software inicia prevendo o tamanho do Software e utiliza isto como entradas para estimativas dos esforços totais requeridos, distribuindo por cada fase do ciclo de vida. Isso é obtido através de dados históricos de projetos anteriores.

Se nós podemos prever o tamanho do Software antes do seu desenvolvimento, seu tamanho pode ser usado como a base para a determinação de quantos esforços são requeridos, quantos engenheiros são necessários, e quanto tempo gastarão para desenvolver o Software. Abaixo alguns exemplos de métricas para estimativas de prazos e custos.

1 - O Modelo COCOMO(Constructive Cost Model)

O modelo de estimativas COCOMO foi desenvolvido por Barry Bohem [Boh81] através de uma amostra de 63 projetos concluídos, com o objetivo de estimar esforços de desenvolvimento, prazos e tamanho para novos projetos. Ele é baseado no Número de Instruções Fontes (DSI - Delivered Sources Instructions) e distribui esforços e prazos por todo ciclo de vida do Software.

O COCOMO é baseado em estatísticas, mas não existe um modelo de acompanhamento para realimentação de dados históricos.

A dificuldade de estimar o tamanho do projeto em DSI é um fator negativo do modelo COCOMO, embora existam os fatores corretivos para diversas linguagens. Mas, tamanho de programas ou mais precisamente DSI não quer dizer complexidades. Um programa recursivo é pequeno e mais difícil que muitos programas convencionais.

Um outro fator negativo do COCOMO é que ele não leva em conta o trabalho desenvolvido novamente devido a erros de projetos. Isto é decorrente da falta de um modelo de acompanhamento dos projetos, e como conseqüência, a falta de aperfeiçoamento de processos de desenvolvimento de Software [GM91].

Um ponto positivo é a distribuição de custos nas diversas fases do ciclo de vida do desenvolvimento de Software.

2 - O modelo de Putnam

O modelo de Putnam utiliza-se da curva de Rayleigh para estimar o tamanho do projeto [Putnam78]:

S = E * K**1/3 * t**4/3, onde

S é o tamanho do Software em NCSS (Número de DSI sem comentários),
K é o custo de mão de obra (em homens-ano),
t é o tempo de desenvolvimento (em anos).

E é o fator de tecnologia ou fator de ambiente e pode ser calculado em função de projetos passados através da seguinte fórmula:

E = S/k**1/3 * t**4/3

Segundo ARIFOGLU [KFA90], o modelo de Putnam é um macro modelo para estimativas e dá bons resultados em planejamento e estimativas de grandes projetos de Software que requerem mais que 30 homens-ano.

2.2.2.2 Métricas Subjetivas

Os aspectos de qualidade podem ser medidos através de diferentes visões. Uma visão positiva dos aspectos de qualidade é responder a questão, se esse produto ou processo está bom ou não. Esta visão positiva de aspecto de qualidade pode levar à definição de métricas subjetivas. Ela está relacionada com os valores quantitativos dos atributos como Confiabilidade, Manutenibilidade etc.

A metodologia adotada no RADC - Rome Air Development Center - é um exemplo de métricas subjetivas. Ela se baseia nos atributos, em função da necessidade do cliente, conhecidos como fatores e critérios de qualidades.

O RADC possui três camadas. A quantificação ou grau de qualidade é informado através do entrelaçamento dos fatores e critérios de qualidades, conforme figura 2.2. Essas camadas são descritas abaixo:

1 - Métricas agregadas para representar fatores de qualidade no nível mais alto;
2 - Métricas agregadas para representar critérios no segundo nível; e
3 - Métricas agregadas compostas de elementos no terceiro nível.

Os valores são analisados pelo grupo responsável da qualidade conforme funções e características desejáveis do projeto ou através de um sistema especialista.

Inicialmente, relacionam-se os fatores e critérios necessários; após, quantificam-se seus fatores dentro de critérios; e por último, assegura-se seu cumprimento que faz parte do controle.

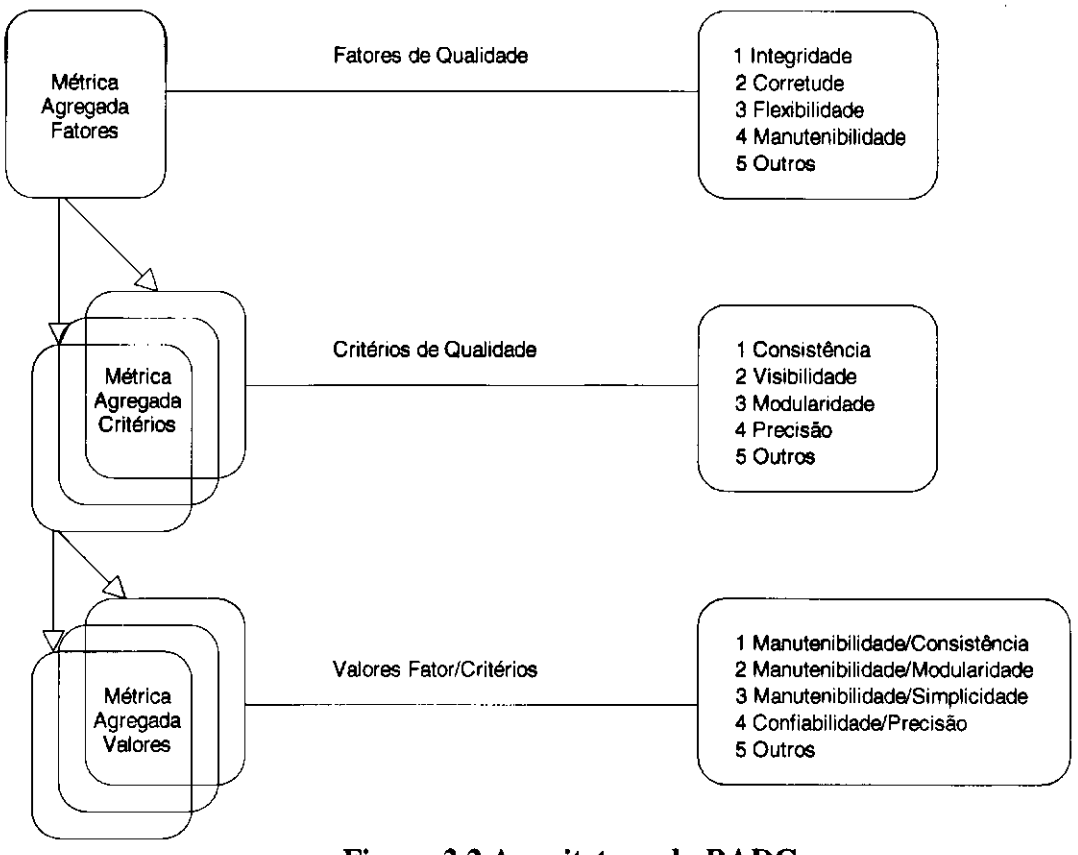


Figura 2.2 Arquitetura do RADC

2.2.2.3 Métricas Objetivas

A visão negativa dos aspectos de qualidade é reconhecida quando da ocorrência de erros nos produtos ou processos. Assim, alguém pode ver qualidade como o número de defeitos que estão contidos dentro do produto ou nos processos de desenvolvimento. Para muitos desenvolvedores, esta visão negativa pode ser mais fácil de quantificar que as visões positivas de qualidade. Esta visão negativa leva para a definição de métricas objetivas. Esses defeitos e suas origens podem ser localizados quando das revisões e testes através de inspeções.

Métricas de defeitos da UNISYS é um exemplo de métricas objetivas. Indicam ocorrências e origens de erros durante os processos de desenvolvimento de `Software`.

2.2.2.4 Métricas de Complexidade

A utilização das métricas de complexidade é importante na obtenção do aumento de produtividade no desenvolvimento de `Software`. Isto ocorre devido ao tratamento que se dá aos seus módulos, tornando-os mais simples. São utilizadas nas diversas fases do ciclo de vida.

As métricas de complexidades utilizam-se do princípio de dividir para conquistar. Uma métrica, figura 2.3, para medir complexidade dos produtos de projetos de `Software` pode ser vista em [ZD93]. Ela é baseada em métricas externas e internas. Para métricas

externas, a complexidade dos módulos gerados é função dos fluxos de dados e os seus `fan-in` e `fan-out` no projeto lógico. Para métricas internas, sua complexidade é função das chamadas de funções ou procedures, manipulação de estrutura de dados e os números de I/O. São utilizadas nas diversas fases da metodologia e mostra onde os cálculos de cada módulo do projeto deve ser calculado, Pe e Pi, e o ciclo devido à complexidade de cada módulo.

Se Pe é alto, indica que os produtos gerados na fase de análise são complexos. Com isso é necessário que se retorne ao início dessa fase e se divida os seus módulos em módulos menos complexos. O mesmo acontece em Pi, o processo retorna para início da fase de projeto.

Dentro desse contexto mostrado acima, módulos podem ser subdivididos em módulos menores com o intuito de tratar a complexidade tornando-os mais simples. Isto quer dizer, caso ocorra discrepância de um módulo para outro, ele deve ser revisto ou conduzido a um tratamento especial no seu desenvolvimento.

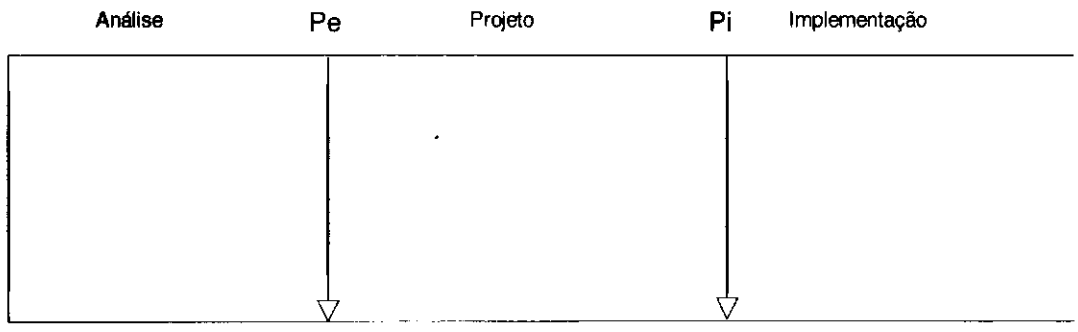


Figura 2.3 Métricas de Complexidade de Zagge

Métricas de Complexidade Ciclomática de McCabe (1976) e Ciência de `Software` de Halstead (1979) são outros exemplos de métricas de complexidade e podem ser vistas em GM [GM91].

2.2.2.5 Métricas de Produtividade

Vários são os fatores que afetam a produtividade, como por exemplo, capacidade de pessoal, complexidade do produto, Confiabilidade requerida, restrição de tempo, experiências em linguagens de programação.

Um dos requisitos básicos de gerenciamento em alguma disciplina é medir a produtividade das pessoas envolvidas na produção. As medidas são obtidas na fase de planejamento de um projeto através das métricas de estimativas. As métricas de produtividade fornecem a base para avaliação da performance de indivíduos, processos e ferramentas. A habilidade de quantificar o impacto de vários fatores na produtividade é importante no aperfeiçoamento de desenvolvimento. Evidentemente, aperfeiçoamento só se constata se pudermos medir produtividade quantitativamente.

A Análise de Ponto de Função (APF) de Albreth [KFA90] é um exemplo de métricas de estimativas de produtividade. O método de ponto de função foi derivado empiricamente

de uma quantidade limitada de experimentações de dados em aplicações de negócios ou sistemas de Informações.

Ela prevê quanto tempo gastará o projeto e quantas pessoas serão necessárias na fase de projeto do ciclo de vida do desenvolvimento de Software. A APF é um método de estimativas de esforços de desenvolvimento, prazos e tamanho da aplicação e da produtividade de programação.

Por exemplo, usando o ponto de função como uma métrica de produtividade, tem sido possível quantificar o aperfeiçoamento na produtividade quando da utilização de uma L4G em vez de uma L3G. Nós podemos medir produtividade de programadores em termos de número de pontos de função por unidade de tempo.

2.3 Os ADS e a Gerência como Fator de Qualidade e Produtividade no Desenvolvimento de Grandes Projetos de Software

A gerência de projetos de Software consiste no controle de produtos gerados nas diversas fases do ciclo de vida de desenvolvimento, como também, no gerenciamento do seus processos. Suporte para o gerenciamento de produtos inclui facilidades de versões, configuração, e padrões para realizações dessas tarefas consistentemente. Para os processos, incluem-se facilidades para gerenciamentos de projetos (planejamento e controle de projetos), de tarefas (ajudando desenvolvedores organizar e trilhar suas tarefas), de comunicações (conhecendo e controlando o modelo de comunicação na organização do projeto), e a modelagem do projeto [GM91].

Um ADS deve estar apto para suportar múltiplos usuários e/ou classes de usuários, ser fácil de usar, fornecer visibilidade de processos e também suportar o gerenciamento dos processos de Software.

Grandes projetos de Software freqüentemente envolvem cooperações entre membros da equipe de projetos. Conseqüentemente, é essencial que os ADSs permitam múltiplos desenvolvedores trabalharem juntos no mesmo projeto concorrentemente. O efeito dessas concorrências, entretanto, é permitir a esses múltiplos desenvolvedores acessar os vários componentes do projeto ao mesmo tempo.

Informações de quando uma ferramenta será requisitada e suas características, quais usuários estão utilizando o ambiente e quais as suas condições de usos, são importantes para segurança e integridade do ambiente de desenvolvimento.

O fundamental objetivo de um ADS é automatizar as tarefas de produção de Software. Um importante componente desta automação é o controle de ferramentas do ambiente. Ambientes modernos têm, entretanto, se concentrado no mecanismo de desenvolvimento para chamadas de ferramentas implicitamente, ao invés de forçar o usuário a interagir explicitamente com cada ferramenta. Isto garantirá que as ferramentas sejam chamadas em um tempo apropriado. Um exemplo simples ocorre, quando um usuário acaba a edição de um módulo, e o ambiente chama automaticamente um compilador para produzir uma versão do módulo [Lot93].

2.3.1 Características Desejáveis

Por ambiente de programação, nós designamos um ambiente que suporta somente a fase de codificação do ciclo de vida de desenvolvimento do sistema tal como edição e compilação. Por ADS, nós designamos um ambiente que aumenta ou automatiza todas as atividades compreendidas no ciclo de vida do `Software`, tais como gerenciamento de configuração e programação em grande escala, como também, gerenciamento de produtos, processos e equipe de projetos. Os processos de desenvolvimento de `Software` são freqüentemente descritos pelo modelo de ciclo de vida. Em geral os processos consistem de uma série seqüencial de fases separadas por revisões [DRP87].

Ambientes baseados em ferramentas isoladas têm uma arquitetura fracamente acoplada sem nenhum mecanismo para assegurar que as ferramentas sejam utilizadas adequadamente ou usadas no todo. Um assistente baseado no modelo Cliente/Servidor é uma proposta para obtenção de uma arquitetura mais fortemente acoplada, portanto mais adequada para apoiar os desenvolvedores de projetos [PW88].

Segundo Penedo [PW88], os ambientes devem possuir as seguintes características:

• Utilidade

Cobertura das funções dos usuários (programadores, projetistas, gerentes de projeto, garantia de qualidade de processos, etc) e das atividades do ciclo de vida (arquitetura de projetos, teste de aceitação, manutenção, etc);

• Usabilidade

Amigabilidade, consistência da interface de usuário, proximidade da prática existente, fácil de usar e suporte para as necessidades de projetos;

• Adaptabilidade

Grau no qual o ambiente pode ser customizado para encontrar as características de projeto desejáveis, procedimentos, e/ou preferência de usuários. Feito sob medida fornece suas capacidades mais eficientemente e/ou efetivamente;

• Grau de Automação

Extensão para os quais os usuários (projetos) recebem assistência automatizada para cumprir prazos, ou atividades do ciclo de desenvolvimento de `Software`;

• Grau de Integração

Facilidades de integração de ferramentas nos ambientes;

• Valores

Benefícios em termos de fatores tais como aumento de produtividade, qualidade de produto, decremento de prazo/custo em desenvolvimento de aplicações-fim, etc.

2.3.2 Arquiteturas

Segundo Penedo [PW88], as arquiteturas de ambientes podem ser do tipo:

- Arquitetura baseada em uma máquina virtual

A arquitetura baseada em uma máquina virtual organiza os componentes de um ambiente dentro de camadas de suporte de implementação, com camadas mais baixas suportando as mais altas;

- Arquitetura em rede

Esta arquitetura tem uma organização como uma rede de processos e seus componentes tipicamente interagem através de mensagens de comunicações;

- Arquitetura Centrada em Dados

Essa se utiliza de um repositório de informações na parte central e organiza os componentes em termos de fluxos de dados dentro e fora do banco de dados implementado;

- Arquitetura Centrada em Controle

A arquitetura Centrada em Controle localiza os subsistemas de supervisores internos do ambiente e a parte central e organiza os componentes em termos de fluxo de dados e controle entre eles (ambientes baseados em tecnologias de sistemas espertos exibem esse tipo de arquitetura).

2.3.3 Taxonomias

A taxonomia para ADS compreende cinco categorias, cada uma representando tendências, tendo um impacto significativo no ambiente. Abaixo, uma descrição sucinta das categorias, maiores informações ver em [PW88].

1 - Ambiente Centrado em Linguagem

Esses são construídos em cima de uma linguagem, entretanto fornecem um conjunto de características adequadas para aquela linguagem. Esses ambientes são altamente interativos e oferecem suporte limitado para programação em grande escala. Smaltalk é um exemplo deste tipo de ambiente;

2 - Ambiente Orientado à Estrutura

Os Ambientes orientados às estruturas incorporam técnicas que permitem os usuários manipularem estruturas de dados diretamente. A independência de técnicas de linguagem leva à noção de geradores de sistemas;

3 - Ambientes baseados em conjuntos de ferramentas

Esses fornecem uma coleção de ferramentas para tarefas de programação em grande escala com gerenciamento de configuração e controle de versão. Existe um controle definido de ambiente e gerenciamento das ferramentas utilizadas;

4 - Ambientes baseados em Processos

Os ambientes baseados em métodos incorporam suportes para um conjunto de atividades do processo de desenvolvimento de Software, incluindo tarefas tais como gerenciamento de projeto e de equipe;

5 - Ambientes Baseado em Medição [TVJ92]

Esses sistemas representam visíveis progressos em direção à realização de uma flexível coleção e uso de dados de medições para orientar os processos.

A utilização de uma categoria de ambiente não inviabiliza a utilização de uma outra.

2.4 Conclusão

Enquanto em disciplinas dominadas por produção pode ser justificável empregar o número mínimo de trabalhadores necessários e adicionar mais empregados se a produção atrasar, em engenharia de Software adicionar pessoas a um projeto que está atrasado pode atrasá-lo, até mesmo, ainda mais [Tha87].

Os princípios e técnicas de gerência são bastante gerais e não são específicos para Software. Entretanto, estimativas de custos de Software, controle de qualidades de processos e produtos e métricas de complexidades são algumas técnicas e metodologias típicas da gerência de Software. Portanto, atividades de gerenciamento podem ser suportadas por uma variedade de ferramentas, algumas adequadas para o gerenciamento em geral, outras específicas para projetos de desenvolvimento de Software [GM91].

Neste capítulo, foi dada uma visão geral sobre gerências, métricas e ADS e suas integrações com o objetivo de posicionar nosso trabalho, mostrando as principais tendências nas pesquisas de engenharia de Software.

As diversas métricas citadas poderão ser incorporadas em nossa ferramenta ou servir de referências para que se construa um programa de métricas. O COCOMO, por exemplo, pode ser incorporado a GEPRO se tivermos informações de diversos projetos de Software desenvolvidos anteriormente.

Os ADSs têm a responsabilidade de integrar diversas ferramentas como metodologias, modelos, métricas, etc. A nossa ferramenta vem com o intuito de enriquecer o ambiente DYNAMO e utilizar as suas diversas ferramentas que lá existem, como FADO e o TOM.

Dentro desse contexto, mostramos diversas ferramentas que são específicas para projeto de `Software` como métricas. Entre as ferramentas para projetos em geral, podemos citar os diagramas de PERT/CPM, GANNT [Pra88].

Capítulo 3

Os Ambientes de Desenvolvimento de Software e as Ferramentas de Projeto de Software

3.1 Introdução

Uma área da Engenharia de Software muito estudada nos ambientes de pesquisas acadêmicas é a medição com aperfeiçoamento de processos - métricas. Outra área é o ADS, o qual fornece serviços que permitem a integração de ferramentas individuais em diversos aspectos, como controle de dados e interfaces de usuários [DV92].

Assim, a união dessas duas áreas, métricas e ADS são vistos como uma nova área com o objetivo de produzir Software com qualidade [DV92]. É para a integração dessas duas áreas de pesquisa que esse trabalho procura, também, contribuir.

3.2 Os Ambientes de Desenvolvimento de Software

Nas duas próximas seções, levantamos alguns ambientes desenvolvidos e em desenvolvimento. Estes foram levantados em Process and measurement support in SEEs [Lot93]. Serão descritos somente as duas principais categorias de ambientes que estão sendo mais pesquisados hoje. Na seção 3.2.1, descrevemos os ambientes orientados a processos. Os ambientes orientados à medição serão abordados na seção 3.2.2.

3.2.1 Os Ambientes Orientados a Processos

Istar é um ambiente que integra ferramentas, uma base comum de objetos e uma interface de usuários. Existe uma linguagem específica para codificação de tarefas. Este ambiente monitora custos de processos pela demanda de tempo de um recurso.

O sistema **Workshop** é um ambiente para suportar equipes de programação. Processos e produtos são representados utilizando uma linguagem de representação de conhecimentos, SE-KRL. O conhecimento é armazenado em um banco de dados compartilhado. Regras de Produção da SE-KRL são manipuladas utilizando um sistema especialista para automatizar mecanismos de tarefas de desenvolvimento de Software e orientar equipe de projetos.

O **GRAPPLE** descreve um plano baseado em um assistente para trabalhos de desenvolvimento de Software. Este sistema implementa uma máquina de controle de trabalho que utiliza uma corrente de comandos de usuários e oferece um feedback em tempo real. Regras sofisticadas para descrições de ações de usuários são definidas e armazenadas em um sistema de manutenção. Essas regras definem níveis de

credibilidade para ações, permitindo o uso de raciocínio não atômico. Técnicas de IA de reconhecimento do plano são usadas pelo sistema em conjunto com as regras com o objetivo de decidir, o que os usuários do ambiente irão fazer e então oferecer ao assistente para distribuição de tarefas.

No **Arcadia,** o objetivo é fornecer um ambiente de Software flexível e extensível através de infra-estruturas. As infra-estruturas esquematizadas referem-se à linguagem de programação de processos, gerenciamento de objetos e de interfaces de usuários. Os modelos de processos são codificados em APP/A, uma linguagem para programação de processos.

No **Marvel,** os usuários trabalham na elaboração de códigos objetos através de uma interface Marvel, a qual permite o sistema estar a par de suas atividades. Processos são modelados utilizando a Marvel Strategy Language (MSL) com estratégias de pré e pós-condição. O trabalho é realizado quando o sistema detecta que ele satisfaz as regras de pré-condições. Utilizando essas regras, Marvel pode automatizar pequenas atividades formais tais como compilação. Marvel é limitado para controlar processos envolvendo chamadas de ferramentas.

O Assistente em Desenvolvimento escreve modelos em uma Linguagem de Modelagem Conceitual para processos, dados, ferramentas e especificação de usuários para um projeto. Esses modelos orientam o assistente. Regras são usadas para especificar pré e pós-condições para os processos. Este sistema fornece suporte ativo para uso de ferramentas em um ambiente. Todo trabalho é feito através da interface do usuário do assistente, o qual permite monitorar pessoas e integrar as ferramentas.

SLCSE (Software Life Cycle Suporting Environment) descreve um Ambiente de Suporte para o Ciclo de Vida do Software para o desenvolvimento de projeto que suporta a definição e uso de uma metodologia de desenvolvimento. O autor descreve este sistema como uma estrutura que pode ser usada para ambientes sob medida para necessidade de um projeto. O SLCSE é primariamente uma caixa de ferramentas que oferece uma interface de usuário unificada e a base de objetos comum. Usuários podem acessar um conjunto de ferramentas baseadas em regras que estão disponíveis para serem utilizadas em um projeto. SLCSE suporta codificação de conhecimento, em um conjunto de regras, acerca de uma metodologia. Isto é uma extensão para o qual o sistema pode definir um processo.

O sistema **Oikos** explora a possibilidade da utilização de um quadro de regras (modelo problema-solução) para aperfeiçoar processos. Processos são especificados utilizando uma extensão do Prolog compartilhado e são representados no sistema como agentes. Essas especificações de agentes baseados em dados, escritos em um sistema de quadro negro, são utilizadas para chamar ferramentas quando forem necessárias.

No **MELMAC,** processos são modelados e aperfeiçoados utilizando uma representação interna chamada rede FUNSOFT, que é uma extensão da Rede de Petri. Todas informações necessárias a respeito dos processos são armazenadas em uma representação interna. Visões são usadas para representação de processos a nível de usuários; especificamente, tipos de objetos, atividades, fluxo de dados, responsabilidade de staffs e feedback são todos definidos usando visões especiais dentro de uma

representação interna. Pré e pós-condições das atividades descrevem restrições do trabalho. A rede permite ao ambiente analisar como também executar um modelo de processo. Usuários são orientados no seu trabalho através de um interface comum de usuário.

Articulator é um sistema para modelagem e simulação de processos de engenharia de Software. Esse sistema utiliza técnicas de engenharia de conhecimento para entender processos e focaliza toda sua atenção no gerenciamento de processos. Utiliza uma linguagem baseada em objetos para especificar modelos de processos, um mecanismo baseado em conhecimento para responder pesquisas de usuários acerca de modelos e comportamento, e um simulador para testar o comportamento do modelo de processos. Processos são simulados usando uma máquina de estado finita onde cada estado é definido como uma posição no tempo.

MERLIN modela os aspectos de processos e cooperação de usuários através de regras. Definições de objetos e responsabilidades de usuários são expressos utilizando regras de estilo Prolog. Uma engenharia de inferência utiliza técnicas de forward e backward chaining para executar as regras e orientar múltiplos usuários em seu trabalho. Usuários interagem com o sistema via uma interface hipertexto que mostra o contexto de seus usuários, incluindo todos os objetos para os quais são responsáveis e as ações definidas para aqueles objetos. A base de regras é, entretanto, encapsulada dos usuários pelo ambiente. O Armazenamento persistente de objetos no sistema MERLIN é acompanhado por um esquema de Banco de Dados que é especializado para armazenar regras e facilitar o acesso e a sua seleção.

Estrutura de processos e fluxos de produtos, em **Weaver,** são representados utilizando editores gráficos, e o comportamento de processos é representado utilizando uma derivação da Rede de Petri. A linguagem de modelagem interna é completamente encapsulada dos usuários por visões. Estas são instrumentos que mostram e manipulam uma representação interna. No aperfeiçoamento, uma ferramenta "agenda" do sistema apresenta tarefas para participantes de processos e controla interações com ferramentas e informações. O sistema tem um suporte externo para interfaceamento para CASE e outras ferramentas de Software.

3.2.2 Os Ambientes Orientados à Medição

Ginger é um ambiente que coleta medidas de produtos não triviais automaticamente e usa essas medidas como uma simples orientação para processos. Dados de processos, tais como a quantidade de tempo utilizado por computador e os comandos que foram executados, são coletados pelo sistema operacional. Dados de produtos, tais como arquivo de código fonte e suas alterações, são automaticamente coletados em um intervalo regular pelo ambiente. Informações compiladas pelo Ginger são disponibilizados para programadores e gerentes.

Em um estudo aplicado no Ginger, o autor desenvolveu um modelo de performance de programadores baseado em erros acumulados no intervalo de tempo. Um erro é a quantidade de tempo de falhas permanente em um programa. Valores para período de vida de erros foram estimados a partir de registros alterados de códigos fontes e dados

de uso de terminais. Uma alteração para um conjunto contíguo de linhas de código fonte foi contada como remoção de erros. No seu modelo de performance, ao longo do desenvolvimento de projetos, baixos valores para erros acumulativos em um período de vida são desejáveis. Na avaliação experimental do modelo de performance de programadores, utilizando o ambiente, comparado a programadores que não tiveram tal orientação, resultou que o Ginger ajudou programadores a acompanhar seus objetivos mais efetivamente em programas que tinham utilizado o ambiente.

O **Amadeus** focaliza somente o serviço de fornecimento de medição de serviços para um sistema orientado a processos. Amadeus oferece caminhos flexíveis para especificar, coletar e avaliar medições de dados. Amadeus essencialmente oferece uma interface que realiza integração de medidas dentro de alguns ambientes orientados a processos. Isto ajuda a pensar do Amadeus como uma biblioteca de serviços para realização de medição e geração de `feedback` baseados em valores coletados.

Para usar o Amadeus, o usuário fornece um `script`, que consiste de um evento de processo, um agente e uma ação. Os três tipos de eventos são: eventos de processos, eventos de produtos e eventos de relógios; e tudo deve ser gerado pelo sistema orientado a processos. Um agente é uma função BOOLEANA e uma ação, usualmente uma chamada de uma ferramenta. Sobre o recebimento de um evento, se o agente é verdadeiro, o sistema Amadeus executa uma ação especificada. Essas ações podem coletar e analisar dados ou fornecer `feedback` baseadas em análises.

Amadeus é uma parte do ADS Arcadia, e está incluído aqui por causa deste foco de medição e capacidade para ser integrado a sistemas orientados a processos.

ES-TAME possui uma base armazenada em um repositório para organização de conhecimento acerca de desenvolvimento e manutenção de produtos de `Software`, expressada em termos de modelos de processos, produtos e qualidades. O sistema ES-TAME oferece uma metodologia, uma representação de conhecimento e uma estrutura de raciocínio para manipulação da informação na base de experiência.

No **SME** - `Software Management Environment`, existem armazenadas grandes quantidades de dados de projetos anteriores. Produtos, processos e outras propriedades são armazenadas no SEL - `Software Engineering Laboratory`. Dados são coletados de propriedades tais como raio de conhecimento, percentagens de casos de testes passados, esforços de pessoal e número de falhas encontradas. Para cada uma dessas propriedades, o SEL tem desenvolvido um modelo que descreve o comportamento típico de propriedades em seu ambiente. O SEL, também, captura experiências na forma de regras de gerenciamento. Essas regras usam dados de projetos e são avaliados por um simples engenho de experiência. Quando os dados de projetos desviam da linha básica, o sistema pode oferecer possíveis interpretações para esses desvios pela utilização dessas regras. Uma descrição de fenômenos que ocorreram no passado pode então ser oferecida para explorar a situação presente.

No projeto MVP, são usadas as anotações de processos para grandes modelagens de processos, incluindo especificações compreensíveis e desenhos de processos, produtos e recursos. A execução do ambiente **MVP-S** suporta a instanciação e execução dos modelos para a proposta de análises, simulação e orientação de projetos. O objetivo

primário do projeto é a representação explícita, coleção e uso de medição de dados. Na anotação MVP-S, os objetivos de um projeto são dados em cada modelo de processo usando critérios de entrada e saída, os quais são especialmente adaptados para incorporações de medidas de dados. Algoritmos de um projeto são opcionalmente utilizados por um processo individual usando uma linguagem. A execução do plano de projeto MVP-L para simulação e orientação é baseada nas noções de estados de projetos e transições de estados. Transições de estados ocorrem em resposta a interações de usuários, como por exemplo, iniciar um processo.

3.3 O Ambiente DYNAMO

O ambiente DYNAMO é um projeto em desenvolvimento pelo grupo de Sistema de Informação e Bancos de Dados do DSC/UFPb para criação de um ADS com características temporais e ativas. Ele inclui duas metodologias de projetos de sistemas de informação: POKER e FADO.

O DYNAMO objetiva apoiar desenvolvedores de `Software` a automatizar seus processos de desenvolvimento através do monitoramento de projetos e aperfeiçoamento de seus processos. A arquitetura do ambiente é mostrada na figura 3.1.

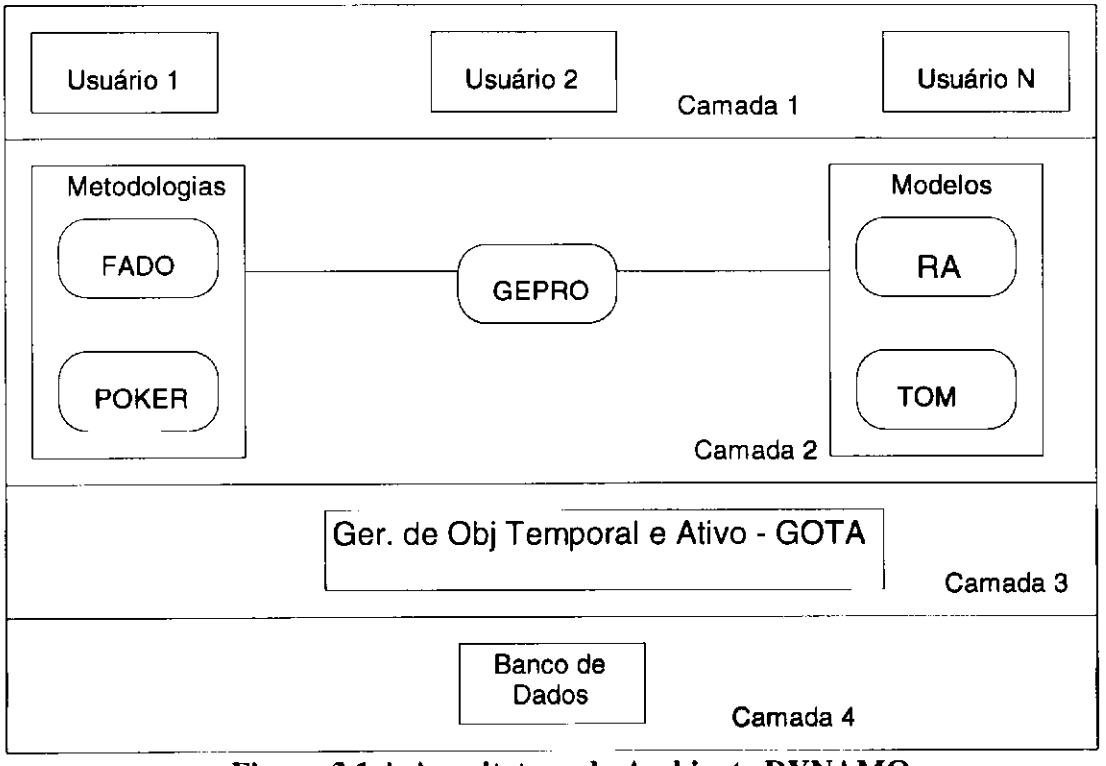


Figura 3.1 A Arquitetura do Ambiente DYNAMO

Ela mostra a interação de suas ferramentas e de seus usuários. O acompanhamento da geração dos seus produtos nas diferentes fases do projeto apoia a gerência na detecção de falhas e qualidades obtidas, portanto, identificando possíveis desvios que possam vir a ocorrer.

A camada 2 do ambiente é constituída das metodologias POKER e FADO, na qual nossa ferramenta aplicará os mecanismos de gerência com auxílio dos modelos RA e TOM. A GEPRO e a Rede de Atividades (RA) estão sendo incorporadas ao ambiente pelo presente trabalho.

3.3.1 A Metodologia FADO

FADO é uma metodologia de projeto de sistemas de informação orientada a objetos e que tem como objetivo gerar uma especificação de um esquema conceitual no modelo TOM. Ela é inspirada nas metodologias de Booch e Rolland. A metodologia FADO é composta de diagramas, tabelas e formulários. A seguir faremos uma breve apresentação, para maiores detalhes ver [Fu93].

A figura 3.2 mostra os processos da metodologia FADO.

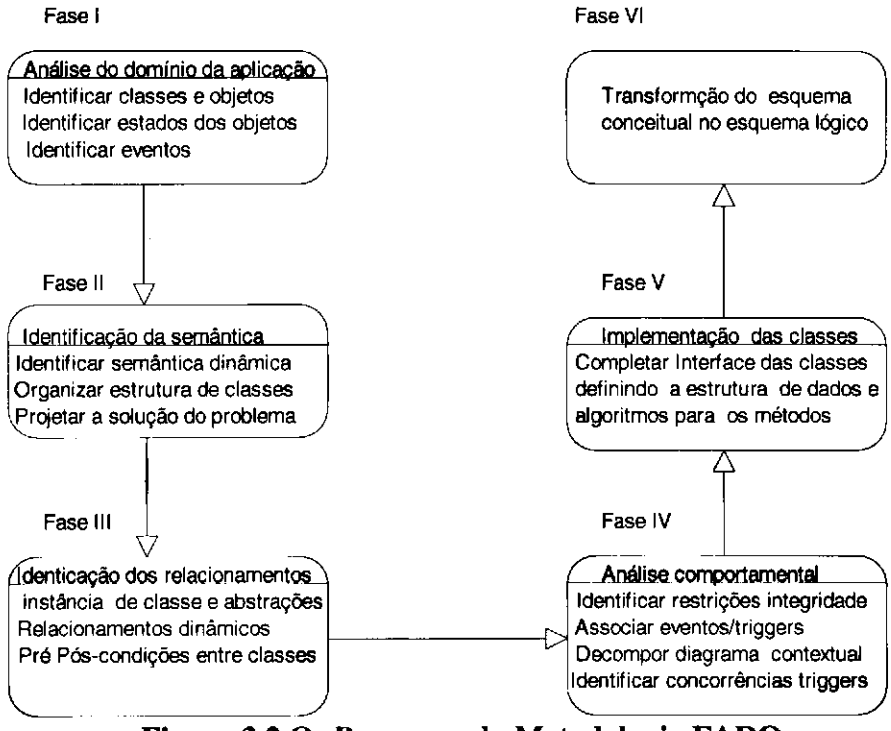


Figura 3.2 Os Processos da Metodologia FADO

Abaixo são descritos os diagramas gerados pelos processos mostrados acima:

- Diagrama Estático (DE)

É um diagrama que mostra as hierarquias de classes e os relacionamentos entre elas. Os tipos de relacionamentos representados são de instâncias, de classes e de abstrações. É obtido nas fases I, II e III.

- Diagrama Contextual (DCT)

Este diagrama fornece ao projetista uma visão geral das possíveis mensagens trocadas entre os objetos das classes, modelando o comportamento dinâmico do sistema como um todo. É obtido nas fases III e IV.

- Diagrama Dinâmico (DD)

É o diagrama que representa o comportamento detalhado das classes abordadas no diagrama contextual, em função dos eventos associados a ela, disparando ações ou falha-ações dos triggers, das classes que se comunicam, e das informações necessárias ao processamento de um determinado método. São obtidos na fase IV.

- Diagrama de Transição de Estados (DTE)

O diagrama de transição de estados representa a evolução dos estados dos objetos de uma classe, e os eventos que causam a mudança de um estado para outro. São obtidos na fase II.

3.3.2 O Modelo Orientado a Objeto e Temporal - TOM

TOM é um modelo de dados orientado a objetos e temporal [Schiel90]. A figura 3.3 ilustra os níveis de abstração do modelo TOM.

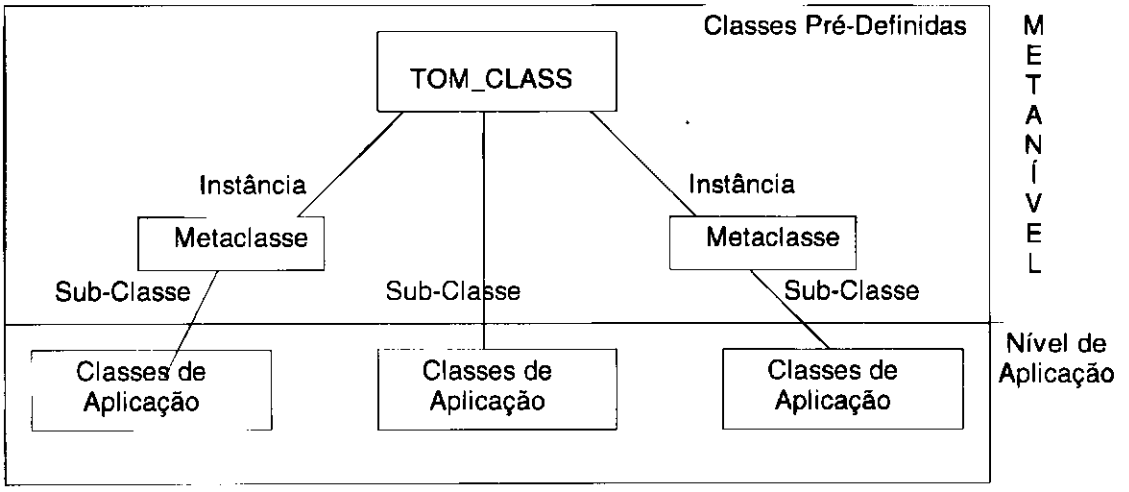


Figura 3.3 Níveis de Abstração do TOM

Existem dois níveis de abstração: o metanível e o nível de aplicação. No metanível, a metaclasse TOM_CLASS já é pré-definida e há um sistema de metaclasses que possibilita a definição e modelagem de qualquer conceito específico para uma determinada aplicação. O projetista de aplicação usa as metaclasses como conceitos do modelo necessários a um problema específico.

3.3.3 O DYNAMO e os Outros Ambientes

Os ADSs oferecem serviços de apoio aos processos de desenvolvimento de `Software` através de diversas técnicas.

Dentre os serviços prestados pelos ambientes descritos anteriormente, podemos citar: a integração de ferramentas, monitoramento de custos através de tempo de recursos de computadores, suporte à equipe de projetos, controle de processos, automatização de atividades, simulação de processos, interfaceamento com ferramentas CASE, medidas de qualidade de produtos, quantidade de tempo gasto, performance de programadores, métricas de dados para processos.

No oferecimento desses serviços, os ADSs utilizam diversas técnicas, como linguagem de processos, bases de conhecimentos, regras de pré e pós-condições, hipertexto, metodologias, modelos baseados em Redes de Petri.

Dentro desses serviços e técnicas utilizadas, notamos que, alguns ambientes oferecem determinados serviços e outros não. Podemos notar que os ambientes citados apresentam a maioria das características desejáveis de um ADS. A seguir analisamos as características de alguns ambientes:

• Quanto à Utilidade

As funções de suporte à equipe de projetos e as atividades do ciclo de vida são vistas em diversos ambientes. No Istar, por exemplo, através do monitoramento de custos e da codificação de tarefas por meio de uma linguagem de processos. Já para o Workshop o suporte à equipe é realizado através de um sistema especialista que manipula as regras de produção;

• Quanto à Usabilidade

Através de utilizações de suas interfaces, umas mais amigáveis que outras;

• Quanto à Adaptabilidade

Os ADSs podem escrever, através de suas linguagens de processos, novos processos para que o ambiente se adeque ou se adapte as novas características de projeto O Arcadia, por exemplo, fornece um ambiente flexível e extensível através de uma linguagem de programação de processos;

• Quanto à Automatização

Assistência através de `feedbacks` em tempo real para usuários de ADS são implementados por meio de sistemas inteligentes. O GRAPPLE utilizam de técnicas de IA e de planos de ações para decidir o que os usuários irão fazer para o cumprimento dos prazos estabelecidos.

• Quanto à Integração

Através chamadas implicitas de ferramentas, quando do término de alguma atividade, possibilita a integração das diversas ferramentas de um ambiente. O Assistente em Desenvolvimento fornece esses serviços através de regras de pré e pós-condições.

• Quanto a Valores

Benefícios são vistos através da qualidade e da produtividade do desenvolvimento de projeto. O Ginger, por exemplo, comprovou, através de experimentos, uma maior produtividade quando do uso de seu ambiente do que a sua não utilização.

O DYNAMO, também, oferece diversos serviços por meio de diversas técnicas, que foram descritas pelos ambientes por nós citados, através da incorporação das ferramentas GEPRO e RA. Com a incorporação da GEPRO ao DYNAMO, acreditamos que este implemente, como visto para os outros ambientes, as diversas características desejáveis para um ADS.

As vantagens de termos desenvolvidas as ferramentas GEPRO e RA para o DYNAMO são várias. Primeiro, é um projeto existente no DSC/UFPb e nada melhor que um ambiente feito sob medida. Segundo, a estrutura do TOM é aproveitada para definição da base de dados na GEPRO. Terceiro, a simplicidade da RA, pois possui somente três elementos para modelagem de processos. Quarto, a sua característica aberta possibilita a incorporação de novas ferramentas. Quinto, o desenvolvimento de nosso modelo baseado em Rede é mais extensível e de mais fácil implementação que uma linguagem de processos. Sexto, a sua adaptabilidade para outros ambientes, bastando que a base de objetos seja redefinida. Sétimo, a utilização dos modelos em rede e ECA e dos paradigmas orientados a objetos dão características de extensibilidade, de automatização, de adaptabilidade, de reusabilidade e de integração.

Abaixo comparamos os diversos serviços oferecidos pelo DYNAMO e os outros ambientes com suas diversas técnicas utilizadas.

A ferramenta GEPRO, como o Amadeus e o SLCSE, fornece um pacote de funções, através de sua rede, que podem ser inseridos em ambientes orientados a processos com o objetivo de coletar e avaliar medições de dados. Um aspecto positivo da GEPRO, em relação ao Amadeus, é a sua estratégia de modelos de processos baseada em uma rede, portanto um aspecto gráfico. Um outro aspecto é a questão de seus processos serem tratados como objetos, facilitando sua implementação. O SLCSE, o Amadeus e o GEPRO utilizam o controle de transição baseado em eventos.

A integração de ferramentas, uma base comum de objetos e uma interface de usuários, vista no Istar, são utilizadas também no DYNAMO.

Como no Workshop, o DYNAMO automatiza mecanismos de tarefas e dá `feedback` para a equipe de projeto.

Enquanto na Arcadia, Marvel e no Assistente em Desenvolvimento, emprega-se uma Linguagem de programação de processos no DYNAMO, utiliza-se um modelo de rede.

O MELMAC utiliza uma extensão de Redes de Petri. A sua rede é complexa e não implementa os processos como objetos.

Responsabilidades de usuários são descritas no DYNAMO como no MERLIN.

Editor gráfico e controle de produtos são contemplados no DYNAMO como são no WEAVER.

O Ginger é um ambiente com o objetivo de controlar programação. Para realizar isso, é necessário que o DYNAMO implemente um modelo de RA, onde, algumas atividades sejam chamadas de ferramentas que verificarão os códigos sendo construídos.

Descrições de ações (atividades) são utilizadas no DYNAMO como no GRAPPLE.

O SME e MVP utilizam-se de dados de projetos anteriores. O DYNAMO contemplará esse tipo de atividade no futuro, quando diversos projetos forem desenvolvidos e dados forem coletados.

3.4 As Ferramentas de Gerência de Projeto de Software

As principais ferramentas de gerência de projeto existentes no mercado como ACCENT GraphicVUE, AMS Schedule Publisher, Autoplan, Cobra, COMPASS, Easy Project, Ganttman, GECOMO, IslandPlan, MasterPlan 1.1X, Milestone, MS-Project, OpenPlan, Size Plus, SuperProject, Texim Project, Time-Line, XOPPS, Xplan, UltrePlanner, WinProject, são voltadas para o planejamento de projeto, ou seja, tratam os aspectos gerais de gerência de projeto, não se preocupando com os aspectos específicos de gerência de projeto de Software. Devido a isso, não realizamos uma análise mais detalhada das ferramentas de gerência de projeto.

O VIRITA com IPSW é uma outra ferramenta que possui a característica de um ambiente, que gerencia equipes de projetos e recursos de tempo automaticamente.

3.5 Conclusão

Neste capítulo, levantamos os principais ambientes desenvolvidos e em desenvolvimento, como também o ambiente DYNAMO. O desenvolvimento da ferramenta GEPRO para este ambiente, de um modo integrado com outras ferramentas, dá a característica de um ambiente orientado a processos e com medição.

Os ADSs estão mais preocupados com a automatização dos processos de desenvolvimento de Software que as ferramentas de gerência de projeto. Isto decorre do fato dessas ferramentas trabalharem isoladas, ao contrário do ambiente que está preocupado com a integração de várias ferramentas. Este é o fato de se ter dado mais ênfase aos ambientes em vez das ferramentas.

A FADO e o modelo TOM foram apresentados como partes do DYNAMO que irão interagir com a ferramenta GEPRO de modo integrado.

Capítulo 4

Rede de Atividades (RA) - Modelagem da Gerência Dinâmica de Processos de Desenvolvimento de Software

4.1 Introdução

É prática comum de Engenharia (não apenas de Software) os processos estarem organizados em níveis indesejáveis em relação o CMM da SEI. Em geral, atribui-se a responsabilidade da condução das diversas atividades a um *Gerente de Projeto,* a quem cabem freqüentemente as atribuições de estimar/cumprir prazos, aplicar o orçamento destinado e analisar/decidir sobre possíveis riscos.

Os ADSs empregam diversas técnicas de Software, para fornecer serviços específicos no apoio ao desenvolvimento de projeto de Software. Já as ferramentas de gerência de projeto dão suporte aos desenvolvedores nos aspectos mais gerais do gerenciamento.

Quanto aos aspectos em geral, o gerente de projetos de Software pode se valer de vários *pacotes para apoio automatizado à gerência,* a exemplo do MS-Project, CA-SuperProject, etc. (vide [PCW92] para uma discussão de alguns deste pacotes). Estes pacotes baseiam-se nos aspectos gerais da gerência de projeto e utilizam ferramentas como gráficos de Gantt ou de PERT, que freqüentemente não incluem *ajustes automáticos* em suas saídas (previsões de prazos, alarmes, etc.) a certos acontecimentos referentes aos processos como atrasos nos prazos ou conclusões antecipadas de atividades referentes aos processos.

Nos aspectos específicos, os gerentes se utilizam de ADSs como Arcadia, MELMAC, Weaver e outros. Estes utilizam técnicas como regras de pré e pós-condições, modelos baseado em rede, hipertexto, etc.

Neste capítulo, introduzimos um novo modelo para o ambiente DYNAMO, com o objetivo de descrever e controlar os processos, produtos e equipes de desenvolvimento de projeto de Software, denominado Rede de Atividades (**RA**), que não apresenta as limitações de ajustes dinâmicos ao comportamento sendo modelado.

A **RA** foi inspirada em três modelos básicos: Rede de Petri [Re82], os modelos orientados a objetos e o modelo de regras ECA (Evento-Condição-Ação) [DBM88].

Quanto às redes de Petri, inspiramo-nos em seu potencial de modelagem da sincronização entre muitas atividades de um ambiente complexo. O paradigma da orientação a objetos serviu de modelo para a realização dos diversos elementos de uma Rede de Atividades. Finalmente, a interação entre eventos e ações foi influenciada pelo modelo de regras ECA.

Quanto aos ambientes, o MELMAC [DV92] e o Weaver [FCKL92] nos inspiraram na utilização dos modelos baseados em Redes de Petri. Já para o Marvel [KFP88] nos modelos ECA. Os modelo orientados a objetos foram tratados como forma de coletar medidas automáticas através de seus elementos que foram definidos como objetos.

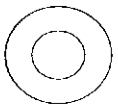
Uma Rede de Atividades permite modelar, como em um ambiente de desenvolvimento de `Software`, as múltiplas atividades que são mais ou menos (in)dependentes e sob quais condições novas atividades podem ser iniciadas. Assim, se for aplicada uma certa metodologia de desenvolvimento de `Software`, deve-se antes criar a RA desta metodologia e com isso a sua aplicação ficará submetida a um controle bem mais consistente. Mostraremos aqui como a RA foi utilizada para gerenciar os processos de desenvolvimento de projeto, aplicado à metodologia FADO.

A rede modela o elo de ligação entre os processos de desenvolvimento de `Software` de uma metodologia e a GEPRO. Portanto, qualquer metodologia pode ser utilizada com a GEPRO, bastando que essa metodologia e o seu elo com a GEPRO sejam modeladas por uma **RA**.

Este capítulo está organizado em mais cinco seções. Na seção 4.2, introduzimos o conceito da **RA**. A definição formal e a relação da **RA** com as Redes de Petri é dada na seção 4.3. A seção 4.4 cuida de um estudo de caso da aplicação da **RA** a metodologia de desenvolvimento de `Software` FADO e a seção 4.5 traz comentários quanto à implementação de uma **RA** como um sistema orientado a objetos. A seção 4.6 é dedicada a conclusões.

4.2 O Conceito da Rede de Atividades - RA

Uma **Rede de Atividades** é composta de três elementos interdependentes com os seguintes significados:



Atividade

Se estiver marcada no círculo central, significa que a atividade está sendo executada;
Se estiver marcada no anel externo, significa que a atividade está encerrada;
Se não estiver marcada, significa que a atividade está desativada;

Evento

Permite modelar condições necessárias para o início de uma atividade através de mensagens externas. É o elo de ligação;

Transição

Permite acionar atividades de saída quando todas as atividades de entradas estão encerradas e eventos associados já tiverem ocorridos.

Estes elementos podem ser ligados entre si. Pode haver arestas dirigidas de atividades e eventos para transições (ligações de entrada) e de transições para atividades e evento (ligações de saída). Entre o anel interno de uma atividade e uma transição pode haver arestas não-dirigidas ou dirigidas (da transição para a atividade). O significado das ligações é o seguinte:

- Uma ligação de entrada de um evento significa que a transição só pode ser acionada se o evento ocorrer;
- Uma ligação de saída para um evento significa que o evento modela exatamente a ocorrência desta transição.

Eventos que não são saída de nenhuma transição são denominados **eventos externos**; os outros são **eventos internos**.

- Ligações de entrada de atividades podem ser:
  se a ligação for do círculo externo, a transição só pode ocorrer quando a atividade estiver *encerrada*;
  se a ligação for do círculo interno, a transição só pode ocorrer quando a atividade estiver *em execução* e irá interromper a atividade em execução;
- Ligações de saída para atividades podem ser
  se a ligação for para o círculo externo, significa o início da atividade. Esta transição só pode ocorrer quando a atividade estiver *desativada*;
  se a ligação for para o círculo interno, a transição só pode ocorrer quando a atividade estiver *em execução* e significa o retorno de uma **SubAtividade**;
- Uma ligação (sem direção) do anel interno de uma atividade para uma transição significa a ativação de uma (sub)atividade sem que a atividade anterior seja encerrada.

No exemplo da figura 4.1, a atividade A1 está sendo executada, e o evento E1 já ocorreu. Nesta situação, a única mudança que pode ocorrer é o encerramento de A1 ou o evento E2. Com E2 ocorrendo, T2 poderá interromper A1. Com A1 encerrado, T1 poderá iniciar as atividades A2 e A3. A ocorrência de eventos e o encerramento de atividades são ocorrências externas que interferem no estado da rede.

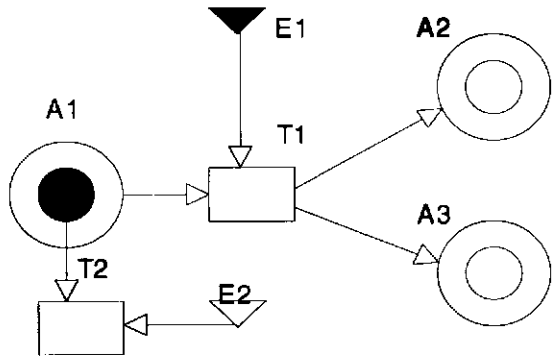


Figura 4.1 Exemplo de uma Rede RA

A rede mostra, portanto, uma característica dinâmica, onde suas atividades estão constantemente se alterando conforme utilização preestabelecida de uma metodologia.

A execução de uma atividade pode ser automática ou manual. No primeiro caso, a própria Rede de Atividades pode estar ligada a uma ferramenta CASE ou a um Banco de Dados que irá executar a atividade e sinalizar seu encerramento.

Toda execução de uma atividade resulta na elaboração de algum produto, ou seja, mensagens, diagramas, códigos ou documentos em geral.

Pode ocorrer a situação, em certos ambientes de projeto, na qual a mesma atividade está sendo executada várias vezes ao mesmo tempo por membros de uma equipe de desenvolvimento. Para modelar esta situação, a rede é expandida para possibilitar múltiplas marcas nas atividades. Será chamada de **Rede de Atividades Expandida (RAE)**. Em uma RAE, como em uma Rede Predicado/Transição [JR91], as atividades recebem marcas individualizadas em seus círculos e as transições movem estas marcas das entradas para as saídas.

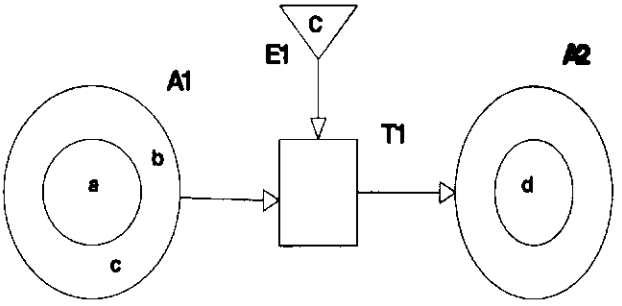


Figura 4.2 Rede de Atividades Expandida

Na figura 4.2, vê-se a atividade A1 sendo executada uma vez, produzindo *a*, e encerrada para *b* e *c*. Como E1 solicita a passagem de *c* para A2, através de C, o acionamento de T1 irá mover *c* para A2, onde C é um elemento de elaboração de c.

Portanto, marcações no círculo interno representam pessoas ou serviços executando determinadas atividades e elaborando documentos, enquanto as marcações no anel externo representam produtos de uma atividade aguardando para serem utilizados nas próximas atividades.

No modelo, não fica estabelecido o que significa uma atividade em execução. Assim uma atividade pode ser uma operação manual executada por uma pessoa, pode ser outra RA, ou Sub-RA, que será iniciada com a marcação da atividade. Ilustramos esta situação na figura 4.3. A passagem para a **SubAtividade** se dá por meio de uma ligação sem direção, o que caracteriza o fato de que a atividade superior continua *em execução.*

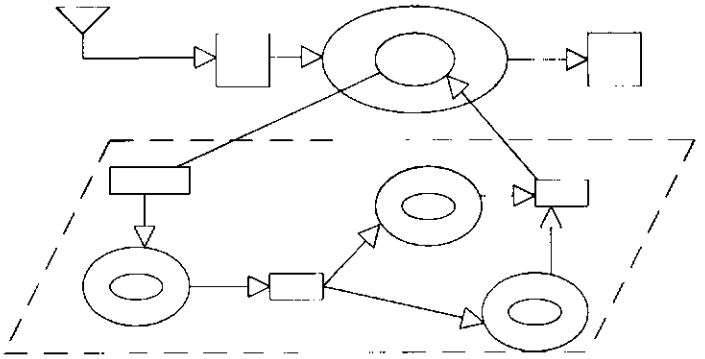


Figura 4.3 Uma RA Hierárquica

Cada atividade de uma RA pode ser refinada para detalhar seus procedimentos ou SubAtividades. No modelo Cascata, por exemplo, a atividade Teste poderá ser refinada através de uma sub-rede representando uma metodologia de teste. A figura 4.4 mostra este refinamento.

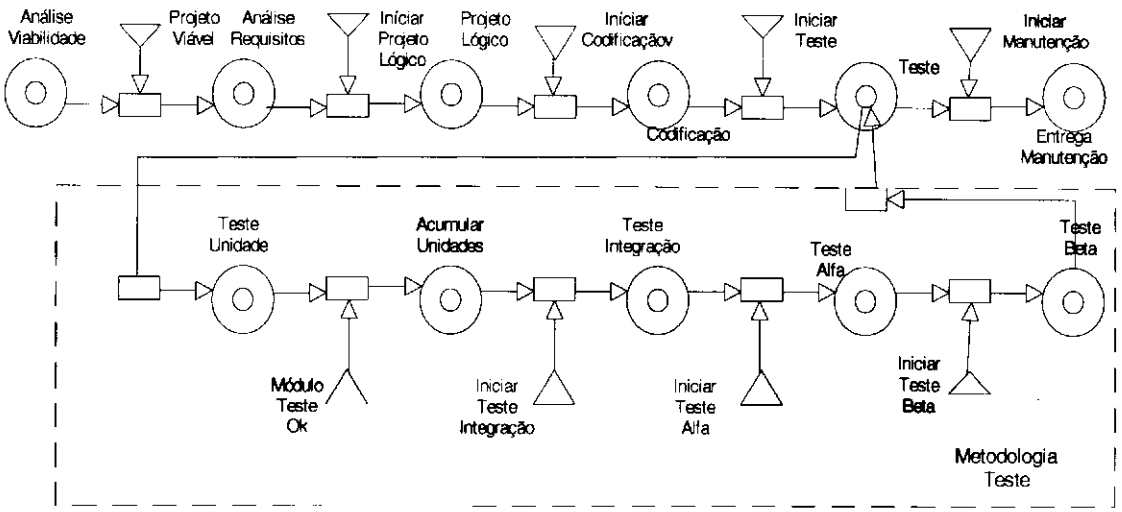


Figura 4.4 Exemplo de Refinamento da Rede

4.3 Definição Formal e Conversão em Rede de Petri

Esta seção apresenta uma definição formal das RAs e mostra como uma RA pode ser convertida em uma Rede de Petri, podendo-se, assim, aproveitar todo o aparato formal de análise de Redes de Petri existente.

Definição:

Uma **Rede de Atividades - RA** é um sistema

$<A, E, T, L, \alpha, \beta>$

em que
A é um conjunto de atividades;
E é um conjunto de eventos;
T é um conjunto de transições;
$L \subseteq ExT \cup AxT \cup TxE \cup TxA$ é um conjunto de arestas;
α: $A \rightarrow$ {d, a, e} é a função que dá o estado de uma atividade, podendo estar *desativada, em atividade* ou *encerrada,* respectivamente;
β: $E \rightarrow$ {0, 1} é a função de marcação de eventos.

Ao conceito de Rede de Atividade estão associados os conceitos de **transição habilitada** e **acionamento** de uma transição habilitada segundo uma **Regra de Transição,** que especifica as mudanças de estado válidas.

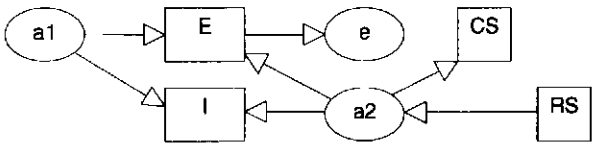
Uma transição está *habilitada* quando todas a atividades de entrada estiverem em estado *encerrado*, todos os eventos de entrada estiverem marcados (β(e)=1), todas atividades de saída estiverem *desativadas* e todos eventos de saída desmarcados. O *acionamento* de uma transição habilitada consiste em:
alterar as atividades de entrada de *encerrada* para *desativada*;
desmarcar os eventos de entrada;

marcar as atividades de saída como *em execução*; e
marcar os eventos de saída.

Uma **transição para SubAtividade** está habilitada quando há uma aresta não dirigida do círculo interno de uma atividade para a transição e esta atividade está *em execução*. Seu acionamento é similar ao acionamento de outras transições, exceto que a atividade *em execução* continua no mesmo estado. Uma **transição de retorno** é caracterizada por uma ligação dirigida para o anel interno de uma atividade. Ela está habilitada quando, além das outra condições regulares, esta atividade estiver *em execução*. O seu acionamento não altera o estado desta atividade.
Finalmente, existe uma mudança de estado de atividades de em execução para encerrada chamada **encerramento da atividade.**

Uma Rede de Atividades pode ser transformada em uma Rede de Petri do tipo Condição/Evento [Re82], segundo as seguintes regras:

R1: Todo evento da RA é uma condição da C/E;
R2: Um evento externo da RA é precedido de um evento de entrada da rede C/E, ou seja, não possue nenhuma pré-condição;
R3: Toda transição da RA é um evento da C/E;
R4: Toda atividade da RA representa uma rede da forma



sendo que *em atividade* significam as condições *a1* e *a2* marcadas, *e* significa *encerrado*, *E* é o encerramento da atividade, *I* é a interrupção da atividade, *CS* é a chamada de uma SubAtividade e *RS* é o retorno de uma SubAtividade. Caso não haja SubAtividades, *a2*, *CS* e *RS* podem ser desprezados. Uma transição de entrada para a atividade marcará *a1* e *a2* e a transição de saída estará ligada à condição *e*.

Por não fazer parte do escopo desse trabalho não mostraremos aqui a definição formal e transformação das Redes de Atividades Expandidas (RAEs), mas a principal diferença está nas funções α e β. A Rede de Petri adequada seria a Rede Predicado/Transição.

Observamos que na RA as atividades, que representam elementos dinâmicos de um processo, ficam nos elementos da Rede de Petri geralmente reservados para a parte estática de um sistema, os *S-elementos*. Com isso, fica claro que a utilidade da RA não consiste na modelagem de atividades em si, mas no controle da interação entre as múltiplas atividades de um processo de desenvolvimento. O conceito de atividade definido com seus três estados, parece-nos o mais adequado ao propósito da rede para descrever o controle e gerência de um processo complexo de desenvolvimento de sistemas.

4.4 O modelo Para a Metodologia FADO

Nesta seção mostraremos como uma RA pode ser utilizada para descrever uma metodologia de projeto de sistemas de informação. Utilizaremos, como exemplo, a metodologia FADO [Fu93, FS93]. A metodologia original será acrescida de elementos de gerência e controle do processo de desenvolvimento de `Software`. Maiores detalhes sobre as diversas fases e múltiplas atividades da FADO, aqui citadas, poderão ser vistos em [Fu93].

A figura 4.5(a, b) mostramos a RA das duas primeiras fases da metodologia FADO. Na Fase I (Análise do Domínio da Aplicação) são identificados classes, os possíveis estados dos objetos e uma primeira visão dos principais eventos do sistema. Com isto, obtemos como protótipo uma primeira solução do problema. Notamos que, se a atividade *Identificar Solução* não for satisfatória, retorna-se à identificação de novas classes.

Na segunda fase da metodologia (Identificação da semântica e dos relacionamentos) são definidos todos os relacionamentos estruturais entre os objetos, é testado um protótipo sobre a estrutura da aplicação e são definidas as mensagens entre os objetos (relacionamentos dinâmicos).

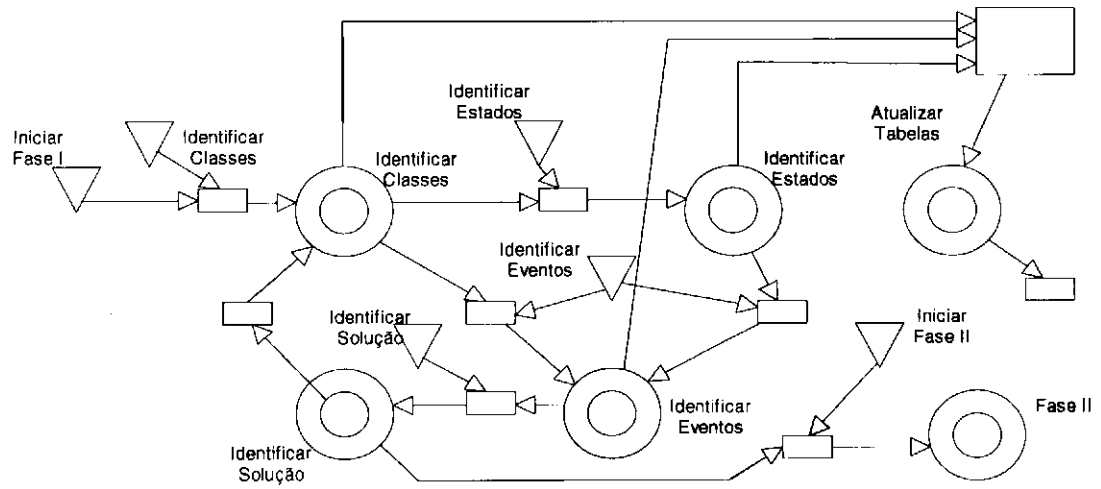


Figura 4.5(a) Fase I da Metodologia FADO

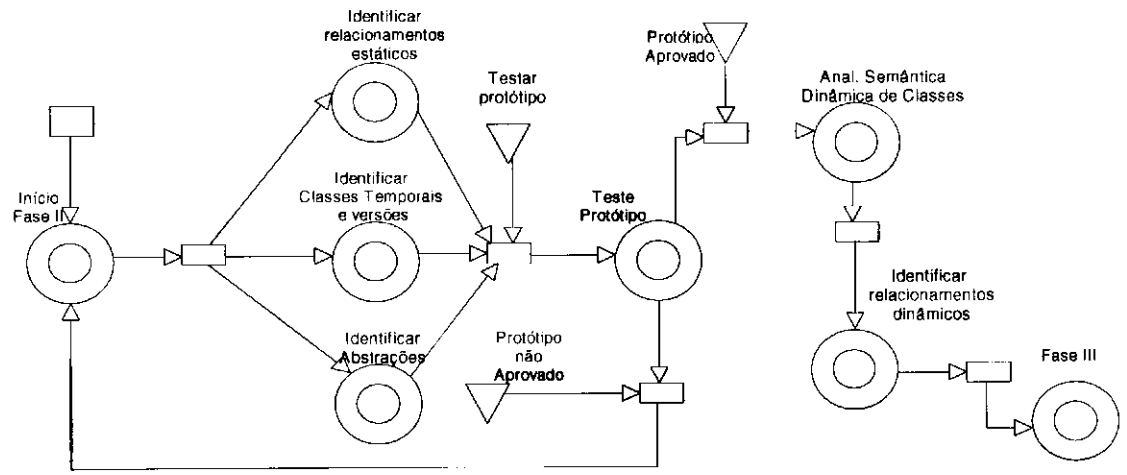


Figura 4.5(b) Fase II da Metodologia FADO

Na figura 4.6, mostramos o controle de qualidade de uma atividade específica de elaboração do *Diagrama Estático (DE).* Esta atividade é chamada em vários passos da metodologia.

Observamos que durante o *Desenhar DE* é feita uma análise da qualidade do DE, e se ele não estiver bom, voltamos ao *Desenhar DE.* Todo trabalho com o DE é registrado em uma Tabela de Estado do DE por *Atualizar Tabela Estado.*

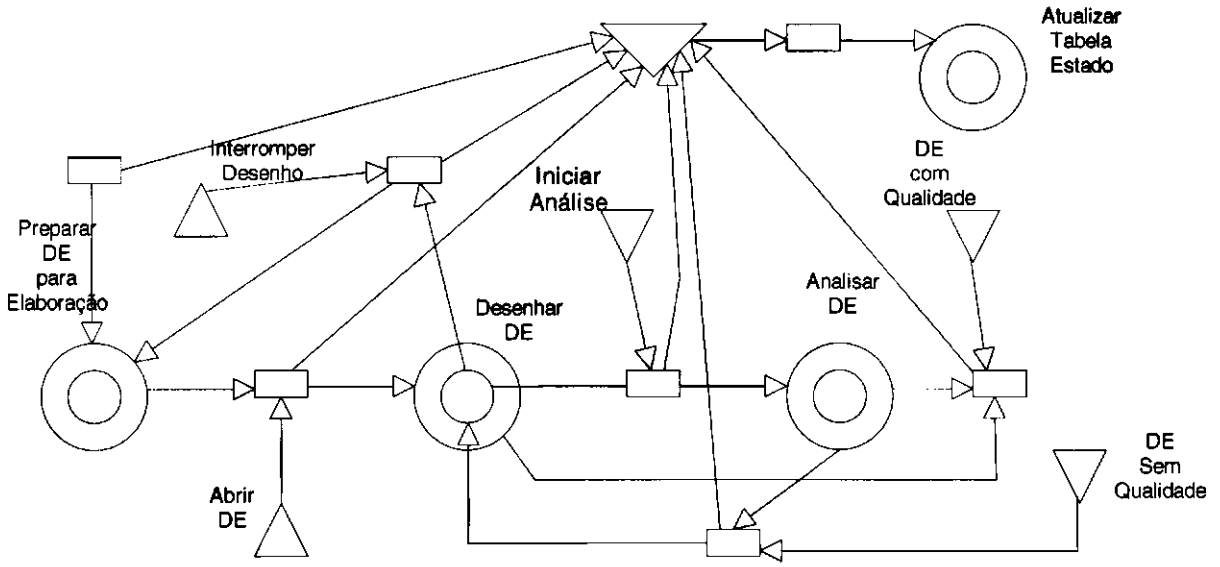


Figura 4.6 Controle de Qualidade de uma Atividade

Na figura 4.7, que mostra a última fase da metodologia antes da implementação das classes (não mostrada aqui), indicamos a verificação da qualidade do Diagrama Dinâmico (DD) de forma resumida em uma atividade.

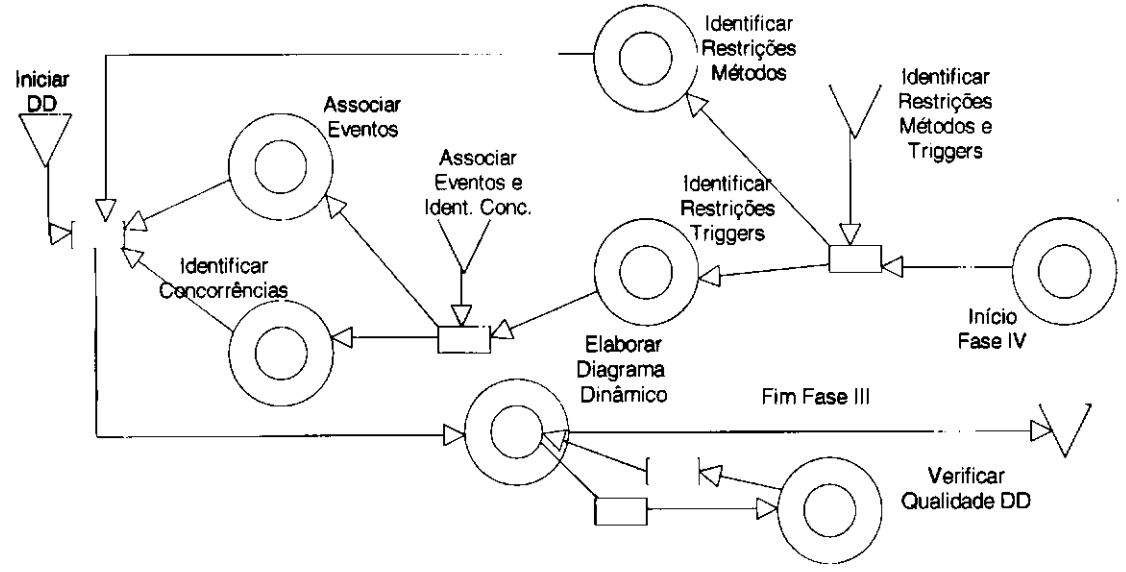


Figura 4.7 Fase III da Metodologia FADO

Encerramos esta seção com uma figura ilustrando um estado da RAE da Fase II em que os engenheiros Y,Z estão preparados para a elaboração de seu Diagramas Estáticos e um

engenheiro X terminou o seu Diagrama Estático mas o teste de qualidade resultou em um DE sem qualidade(Figura 4.8).

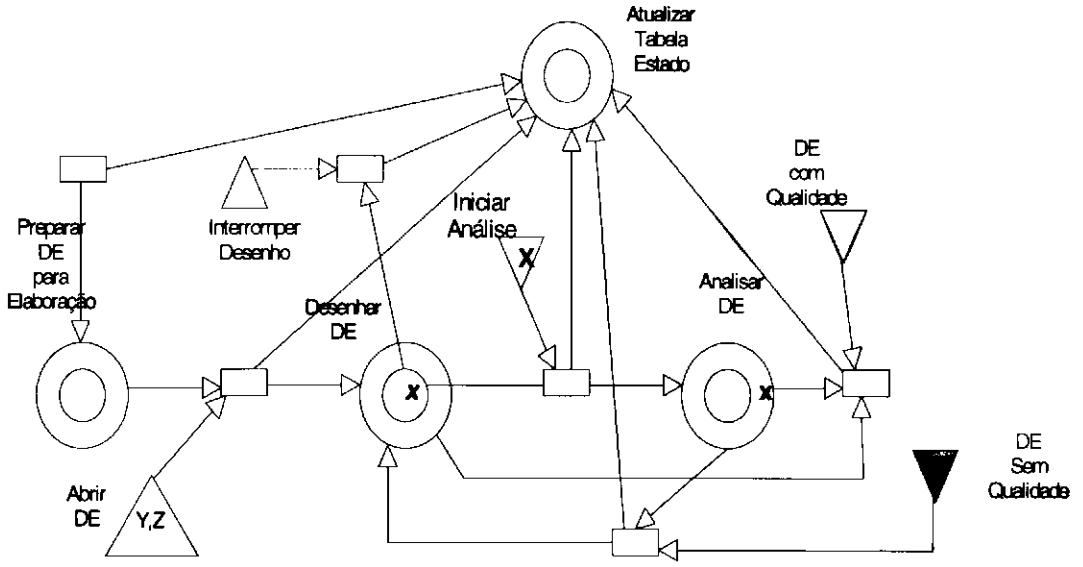


Figura 4.8 Rede de Atividades Expandida

O evento *AbrirDE* ocorrerá quando algum engenheiro utilizando a metodologia FADO, abrir um diagrama estático. Portanto, um engenheiro ao abrir um diagrama estático para sua elaboração, uma mensagem é enviada para o evento *AbrirDE* da RA informando sua ocorrência.

Se ocorre que mais de um engenheiro abra um diagrama estático, implica que diversas pessoas estão realizando uma mesma atividade, por exemplo, *Desenhar DE*. Agora se um engenheiro termina a atividade *Desenhar DE* e ocorre o Evento *Iniciar Análise*, temos mais de uma atividade sendo realizada, *Desenhar DE* e *Analisar DE* simultaneamente. Portanto, percebemos a ocorrência de dois tipos de paralelismo: primeiro a realização de mais de uma atividade sendo executada ao mesmo tempo e segundo, engenheiros diferentes executando a mesma atividade ao mesmo tempo.

4.5 Aspectos de Implementação

Uma RA pode ser implementada (figura 4.9) em um sistema orientado a objetos, no qual cada elemento da rede é um objeto. Teremos, então as classes Evento, Transição e Atividade. Ilustramos o esquema de objetos do exemplo da figura 4.1.

Ao ocorrer um evento externo, o método *habilitar* envia uma mensagem para o método *preparar* da *transiçãoPosterior*; no exemplo, *E1* avisa *T1*. Quando *A1* é encerrada, seu método *encerrar* também avisa à transição. Se o método *preparar* de *T1* perceber que todas as condições de sua execução estão satisfeitas, ele ativa *executar(T1)*.

Uma vez construído o modelo, este servirá para gerenciar os processos de `Software` de uma metodologia modelada.

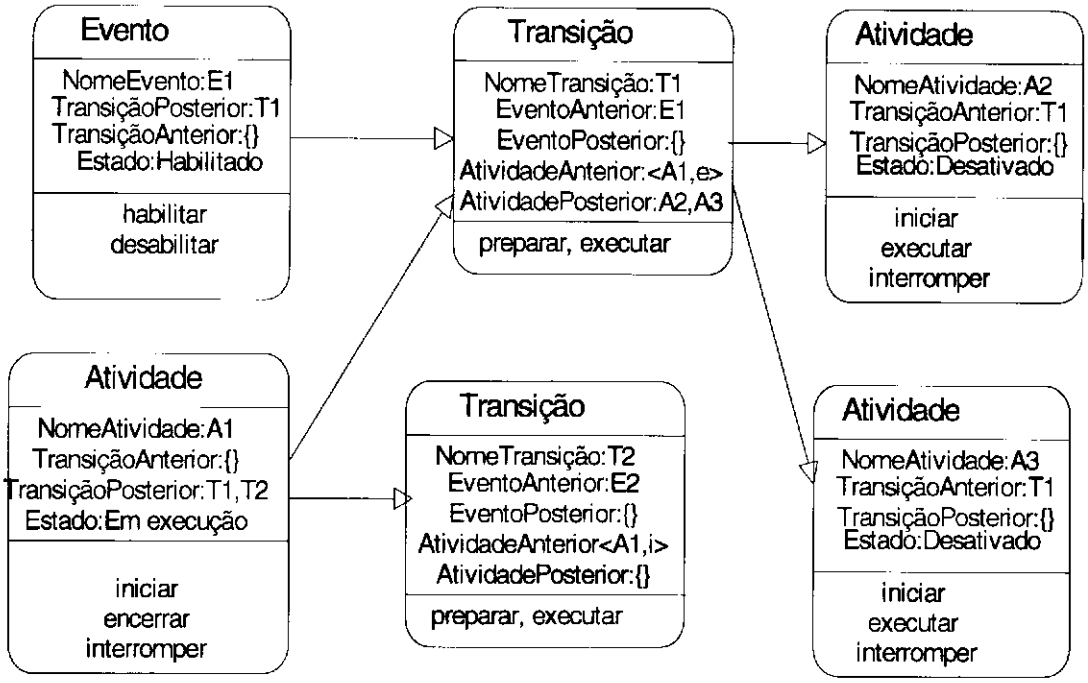


Figura 4.9 Esquema de Objetos de uma RA

Na figura 4.10, mostramos o Diagrama Dinâmico da metodologia FADO para a implementação de uma RA. Além das classes anteriores, acrescentamos uma classe para as Tabelas de Estado, que documentam o estado dos documentos elaborados (vide abaixo).

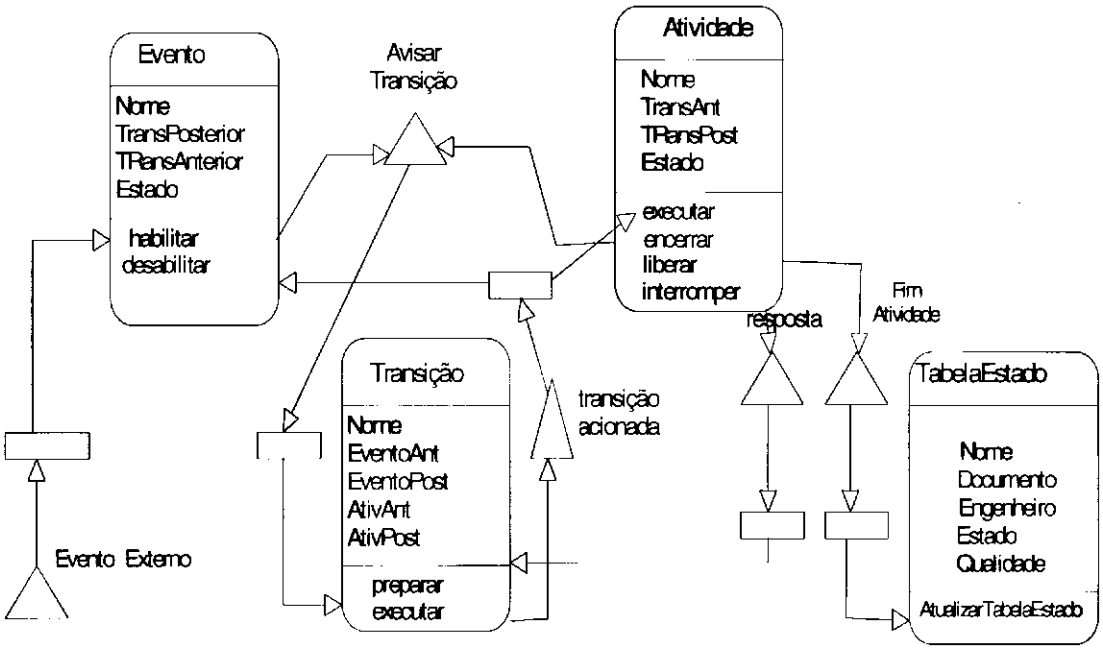


Figura 4.10 Diagrama Dinâmico de uma Rede RA

Supomos que as atividades consistem na elaboração de documentos como diagramas, tabelas, formulários, programas, etc. Para controlar a elaboração destes documentos, além das classes Atividade, Evento e Transição, haverá uma tabela denominada **TabelaEstado**, com os atributos **NomeAtividade**, **Documento**, **Engenheiro, Estado** e

Qualidade. O método **AtualizarTabelaEstado** é responsável pela atualização dos diversos atributos da tabela. Para o atributo Engenheiro supomos que vários engenheiros possam estar elaborando o mesmo documento. Neste caso, teremos uma Rede de Atividades Expandida (RAE) e um registro na TabelaEstado para cada engenheiro.

Exemplificamos para a Figura 4.6, supondo que a tarefa DesenharDE esteja sendo executada pelos Engenheiros X e Y, a construção da tabela 4.1(a).

Atividade	Documento	Engenheiro	Estado	Qualidade
Desenhar DE	DE	X	Executando	não verificada
Desenhar DE	DE	Y	Executando	não verificada

Tabela 4.1(a) - Tabela de Estados dos Diagramas

Agora se o evento **IniciarAnálise** ocorrer para o Engenheiro X, então o sistema passa a fazer uma primeira verificação da qualidade do diagrama, conforme mostra a tabela 4.1(b).

Atividade	Documento	Engenheiro	Estado	Qualidade
Analisar DE	DE	X	Executando	verificando
Desenhar DE	DE	Y	Executando	não verificada

Tabela 4.1(b) Tabela de Estados dos Diagramas

4.6 Conclusão

A RA foi criada com o intuito de ser utilizada como uma ferramenta que apoiasse a gerência de projeto de `Software`, controlando os processos de `Software` de uma metodologia especificada.

Nós utilizamos, como exemplo, sua aplicação à metodologia FADO, o que mostra sua utilidade para precisar a execução do processo de desenvolvimento de `Software` segundo uma metodologia definida. Também ilustra a possibilidade da inclusão de mecanismos de controle de qualidade, não necessariamente previstos na metodologia.

O conceito da nova rede é suficientemente amplo para poder ser aplicado a outros ambientes de desenvolvimento além da construção de `Software`.

A modificação efetuada nas Redes de Petri permite a concentração do projetista na modelagem da coordenação de múltiplas atividades, sem ter que se ocupar com tecnicismos das Redes de Petri. Por outro lado, sua transformação em Redes de Petri, muito bem conhecidas, é uma grande vantagem para análise formal de modelos de RAs.

Acredita-se que a RA contribua para simplificar e tornar eficientes os aspectos gerenciais da Engenharia de `Software`.

Capítulo 5

A Ferramenta de Gerência de Projeto - GEPRO

5.1 Introdução

Neste capítulo mostramos a especificação da ferramenta de gerência de projeto GEPRO para o ambiente DYNAMO.

A GEPRO apóia a gerência de projeto de `Software`, detectando possíveis desvios e aperfeiçoando os processos de desenvolvimento de uma metodologia, melhorando a qualidade e produtividade nos grandes projetos. Isto pode ser notado, em parte, quando da execução da RA.

A ferramenta foi desenvolvida, inicialmente, apoiando a metodologia FADO. A razão de se utilizar essa metodologia é devido aos seus processos gerarem objetos complexos, os quais não são suportados por modelos convencionais.

Um outro fator importante para utilização da metodologia FADO é devido a esta fazer parte do ambiente DYNAMO. Novas metodologias poderão ser incorporadas ao ambiente para serem apoiadas pela ferramenta GEPRO.

5.2 As Características da Ferramenta

A ferramenta é composta de um conjunto de classes interrelacionadas (figura 5.1) que serão incorporadas ao metanível do Modelo Orientado a Objeto e Temporal - TOM (figura 5.2). As classes definidas no metanível são inerentes a qualquer projeto. **ProjetoSoftware**, por exemplo, é uma meta classe que identifica todas as características de um dado projeto e **ProjetoAutomaçãoUniversitária** é uma subclasse da classe ProjetoSoftware no nível de aplicação.

A ferramenta GEPRO é descrita por uma RA que a integra a uma metodologia com o objetivo de controlar e aperfeiçoar seus processos de desenvolvimento. Essas metodologias têm que ser modeladas e validadas. A gerência de projeto é realizada pelos módulos de especificação e controle com apoio da metodologia modelada.

A RA da GEPRO está acoplada à RA da FADO, estabelecendo quais atividades de gerência deverão ser executadas em quais atividades de desenvolvimento de uma aplicação.

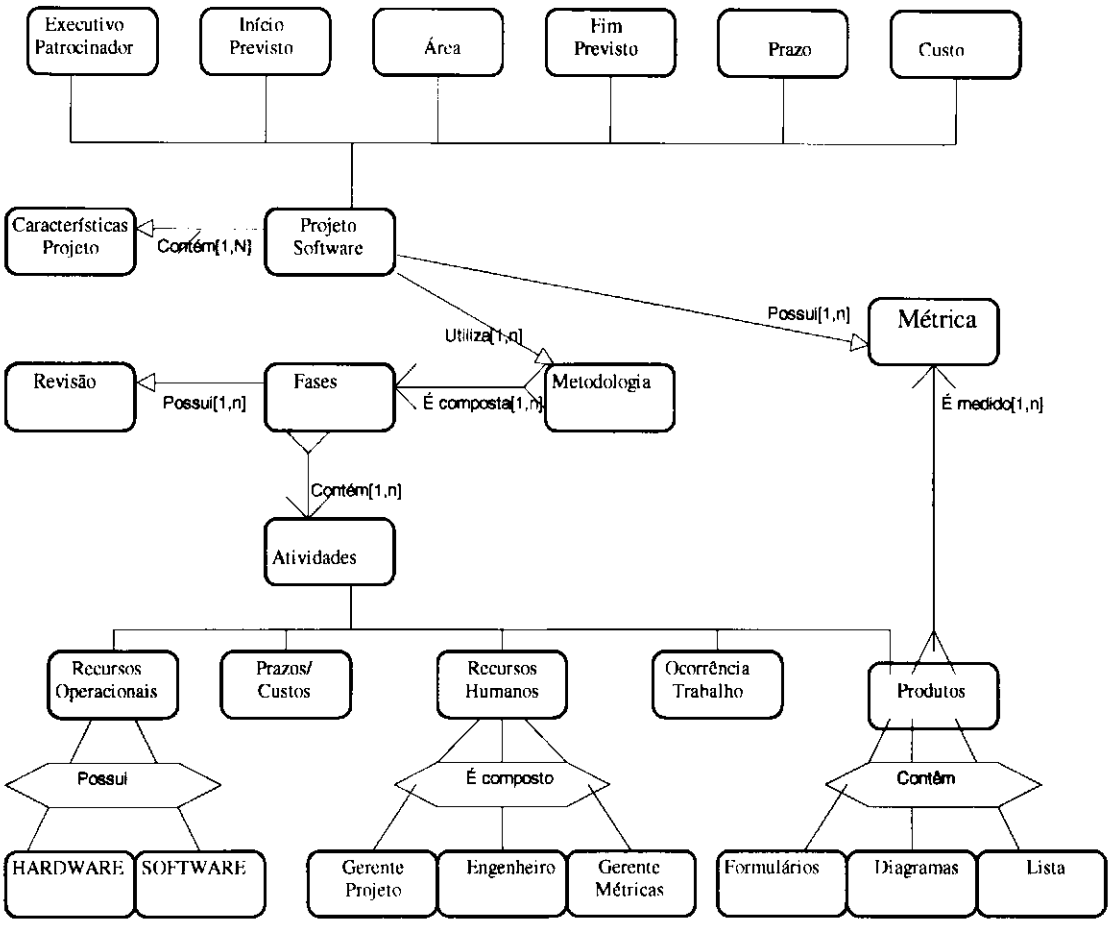


Figura 5.1 Modelo da Ferramenta GEPRO

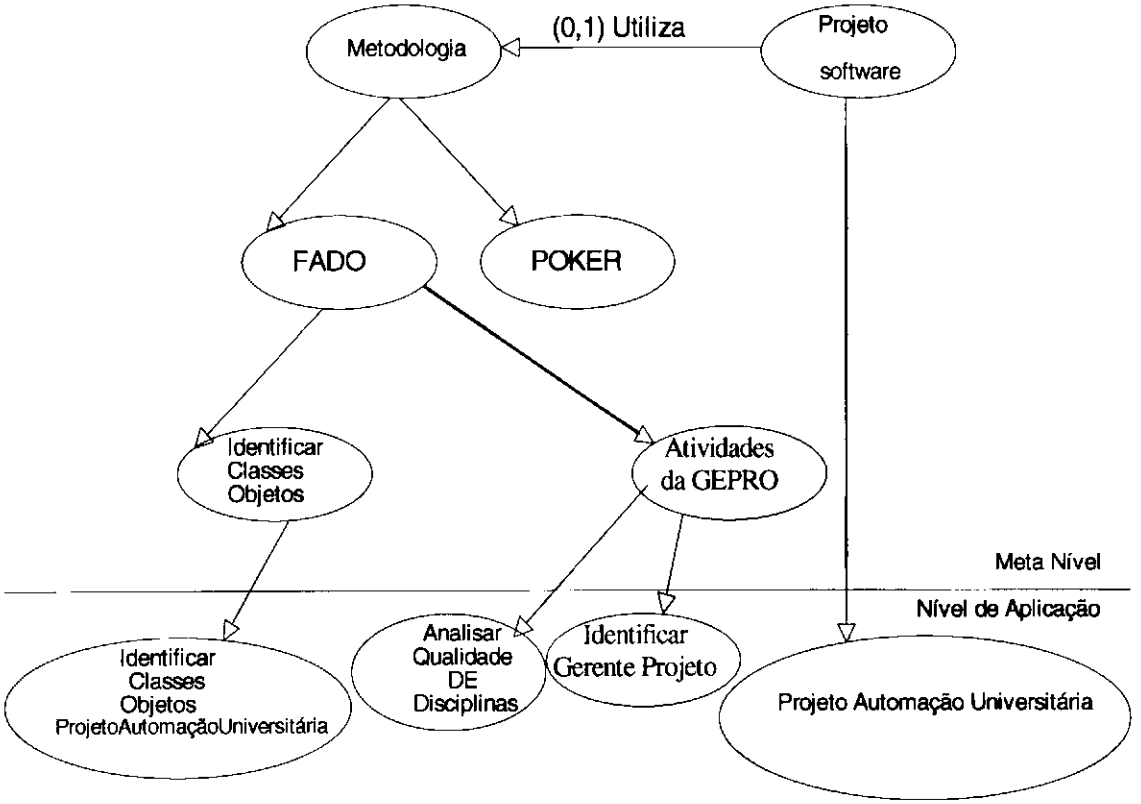


Figura 5.2 Meta-Esquema do Ambiente de Desenvolvimento

A figura 5.3 mostra o monitoramento do projeto através de dois níveis: o nível interno e o externo. O nível interno mostra a RA da GEPRO acoplada à RA da FADO. Já no nível externo, estão as interfaces das ferramentas.

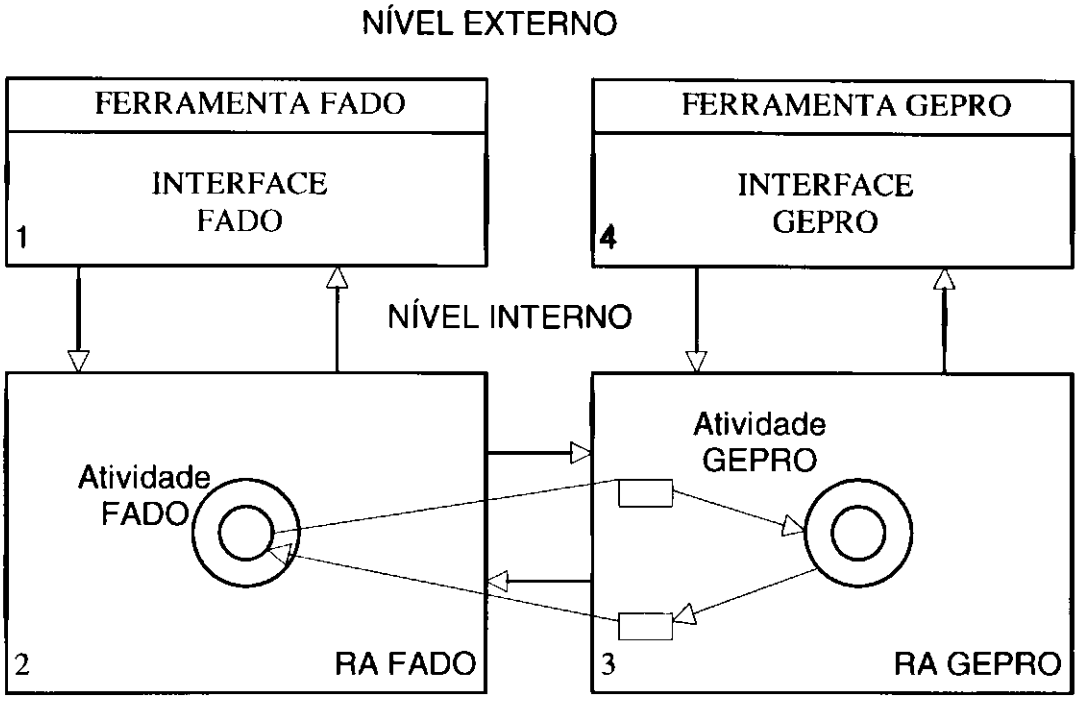


Figura 5.3 Níveis Internos e Externos do Monitoramento de Projetos

Na elaboração do projeto, um evento na interface FADO aciona uma atividade da RA da FADO que, por sua vez, está acoplada a uma atividade da RA da GEPRO. Esta atividade irá interagir com a interface da GEPRO para processar dados de E/S. O processo inverso se repete, informando o final de cada interação.

Com isso, fica claro que a substituição da FADO por outra metodologia não altera a RA da GEPRO, somente as conexões terão que ser estabelecidas.

Percebemos também que temos diversas RAs da GEPRO independentes dando suporte às diversas atividades, sejam ela com características próprias de gerência, sejam ligadas a uma atividade da RA da FADO.

Um programa de métricas onde os atributos de qualidade são valorados, é sugerido. Baseado em [GM91], o programa propõe que seus atributos sejam vistos em três diferentes visões: gerente, engenheiros e usuários e/ou através de qualidades internas ou externas. Qualquer um novo programa pode ser incorporado à ferramenta, bastando que se instale nos módulos de especificação e controle. Um exemplo é a metodologia RADC vista no capítulo 2.

5.3 A Arquitetura da Ferramenta GEPRO

A arquitetura da ferramenta é mostrada na figura 5.4. Mostramos, através dela, os relacionamentos dos seus diversos módulos e o banco de dados.

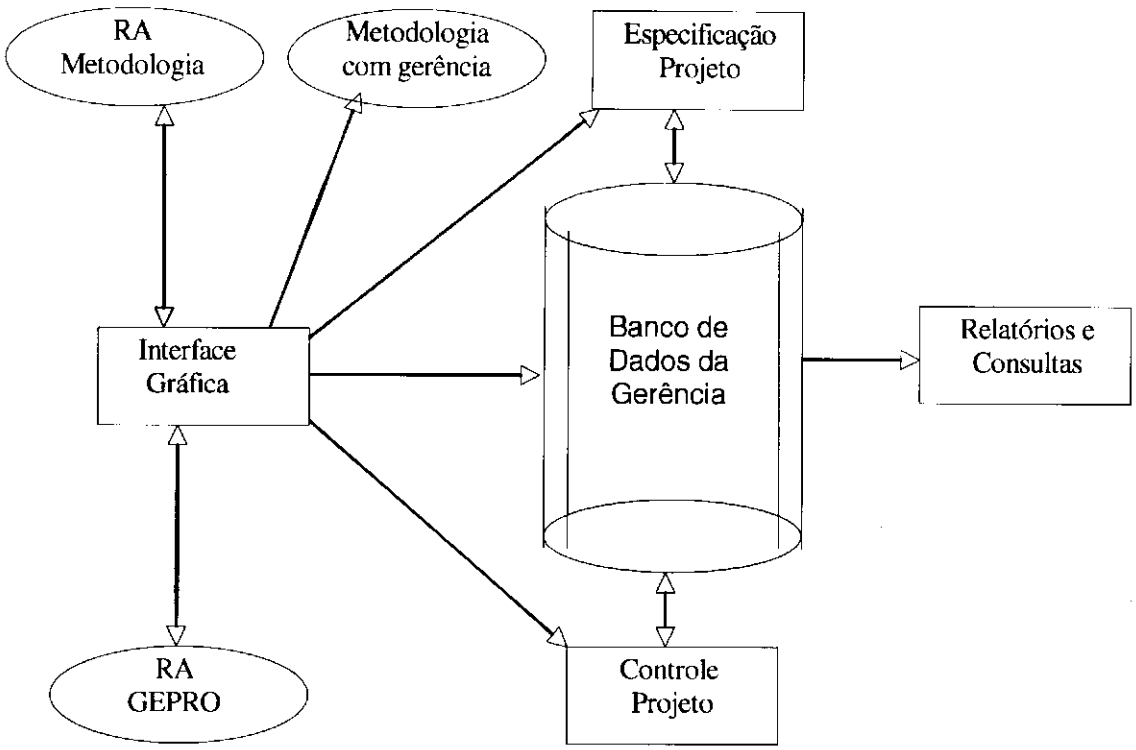


Figura 5.4 A Arquitetura da Ferramenta GEPRO

A interface gráfica elabora as RAs da FADO e GEPRO e seu acoplamento gerando uma metodologia com gerência, como também, as ligações. Os módulos de especificação e controle atualizam informações de projetos atuais no Banco de Dados que servirão de referências para futuros projetos. As informações obtidas pelo gerente são vistas pelo módulo de relatórios e consultas. Discutiremos, nas seções subseqüentes, cada módulo da GEPRO em maiores detalhes.

5.4 A Especificação do Projeto

A base comum de objetos, mostrada na figura 5.1, e definida no modelo TOM será utilizada para especificação de projeto.

Uma vez construídos os modelos de redes, através da interface gráfica, e a solicitação de um novo projeto for requerida, a ferramenta GEPRO estará pronta para apoiar os desenvolvedores de projetos de `Software` através da metodologia modelada. A especificação do projeto foi dividida em dois submódulos. O primeiro, para especificar os atributos da classe Projeto`Software;` e o segundo, para especificar os processos ou atividades. A figura 5.5 mostra os dois níveis para este módulo.

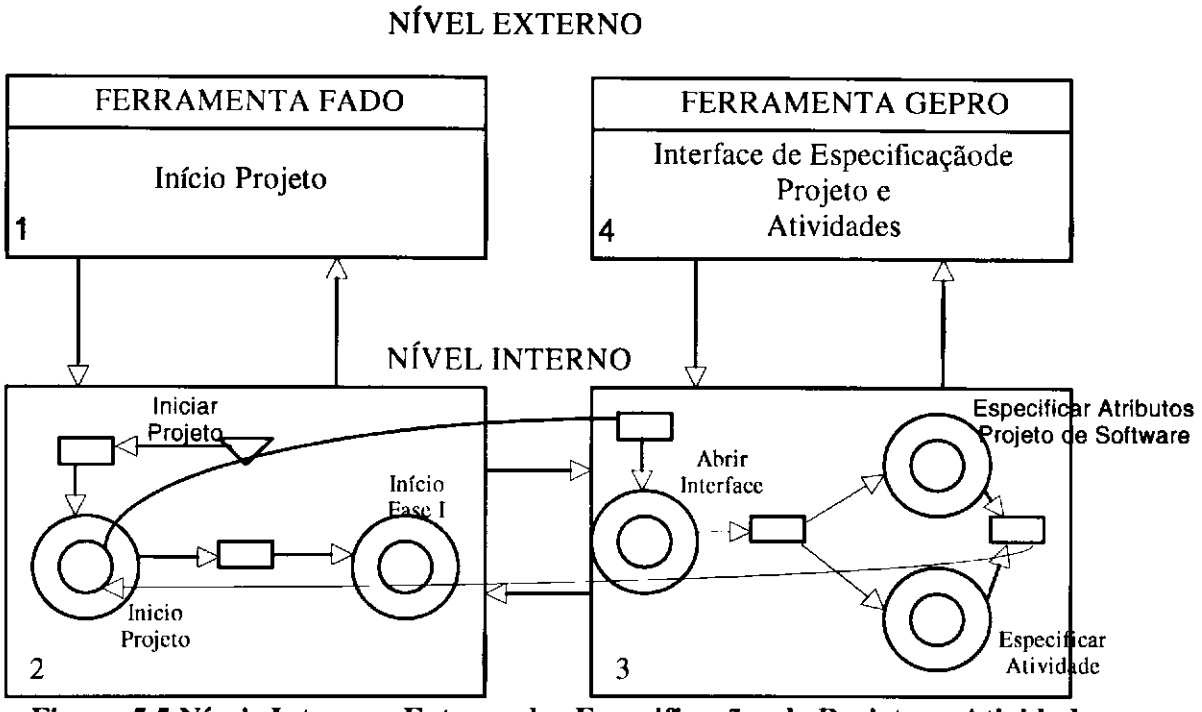


Figura 5.5 Níveis Interno e Externo das Especificações de Projetos e Atividades

5.4.1 Especificação dos Atributos de Projetos

Descrevemos abaixo as especificações da classe **ProjetoSoftware**. As características de projeto serão definidas através de suas interfaces e armazenadas na classe **ProjetoSoftware**. Seus atributos e subclasses são descritos abaixo.

Classe ProjetoSoftware
- IdentificaçãoProjeto:
- NomeProjeto:
- InicioProjeto:
- FimPrevistoProjeto:
- FimRealProjeto:
- TempoPrevistoProjeto:
- TempoRealProjeto:
- CustoProjeto:
- PrazoProjeto:
- MétricasProduto:
- Situação:
- Características:

IdentificaçãoProjeto: É a identificação do projeto, pode ser vista como um código;

NomeProjeto: É o nome do projeto, por exemplo, ProjetoAutomaçãoUniversitária;

InicioProjeto: É a data do início do projeto;

FimPrevistoProjeto: É a data prevista para conclusão do projeto;

FimRealProjeto: É a data real da conclusão do projeto;

TempoPrevistoProjeto: É o tempo, em horas, previsto;

TempoRealProjeto: É o tempo, em horas, real da duração do projeto;

MétricasProduto: São indicadores de atributos de qualidade valorados;

Situação: É a situação do projeto. Como: em elaboração, suspenso, atrasado, adiantado. Essas situações serão obtidas através do módulo de controle;

Características: Informa as características do projeto como: simples, complexo, desconhecido, de alto risco, nível da equipe, ferramentas disponíveis.

Quanto às métricas, abaixo, relacionamos os atributos que são vistos em relação às qualidades externas, e seus pesos. São dados valores de 1 a 5 para os seus atributos de qualidade conforme requisitos de projetos.

Atributos de Qualidade:

1 - Corretude;
2 - Confiabilidade;
3 - Robustez;
4 - Performance;
5 - User FriendliNess (Amigabilidade).

Pesos para os atributos:

1 : Péssimo
2 : Ruim
3 : Regular
4 : Bom
5 : Ótimo

Por exemplo, para um projeto de controle de aeronave, o atributo Confiabilidade e Robustez tem que ser máximo 5, não se admitindo qualquer falha.

5.4.2 Especificação de Processos ou Atividades

Os atributos da classe **Atividade**, também, serão especificados através de sua interface. Com o apoio do modelo, percorremos todas as suas atividades e preenchemos todas informações ou requisitos de atividades conforme necessidades do projeto. Abaixo, descrevemos os atributos e subclasses da classe **Atividade**.

Classe Atividade
IdentificaçãoAtividade:
TransiçãoAnterior:
TransiçãoPosterior:

NomeAtividade:
Recursos Operacionais:
RecursosHumanos:
InicioAtividadePrevista:
FimAtividadePrevista:
InicioAtividadaReal:
FimAtividadeReal:
TempoPrevisto:
TempoReal:
Produtos:
MétricasProcesso:
Ocorrências:
Métodos:
Status:

Aqui trataremos somente os atributos referentes ao projeto. Os atributos **IdentificaçãoAtividade**, **NomeAtividade** e **Métodos** são definidos na modelagem da metodologia vistos no capítulo anterior através dos aspectos de implementação. Métodos são ações tomadas por atividades pertencentes a RA. A linguagem de especificação dos métodos de uma atividade pode ser escrita em qualquer linguagem disponível como C++, C, Pascal conforme necessidades e experiências.

RecursosOperacionais: É um indicativo de que recursos, tanto de Hardware como de `Software`, são necessários para a realização de cada atividade;

RecursosHumanos: São os recursos humanos necessários para cada atividade;

InícioAtividadePrevisto e FimAtividadePrevisto: São as datas de início e fim previstas para uma atividade;

InicioAtividadeReal e FimAtividadeReal: São as datas de início e fim reais para uma atividade;

TempoAtividadePrevisto e TempoAtividadeReal: São os tempos gastos em horas previstos e reais para conclusão de uma atividade;

Produtos: São os produtos ou diagramas gerados a cada atividade; eles estão relacionados à classe métrica descrita abaixo;

Ocorrências: São informações de falhas ocorridas durante o projeto, e de suas origens;

MétricasProcesso: São vistas em relação às qualidades internas, tanto na visão do engenheiro como na visão do gerente. A definição de pesos é a mesma adotada para as métricas de projeto. Os atributos são mostrados abaixo:

1- Verificabilidade (Engenheiro);
2- Manutenibilidade (Engenheiro);
3- Reusabilidade (Engenheiro);
4- Portabilidade (Engenheiro);

5- Interoperabilidade (Engenheiro);
6- Produtividade (Gerente);
7- Visibilidade (Gerente);
8- Complexidade (Engenheiro);

Outros atributos poderão ser incorporados aos atributos existentes, ou, mais profundamente um programa de métricas.

As especificações de produtos serão feitas durante a elaboração do projeto e seus atributos são mostrados na tabela 5.1. A figura 5.6 mostra os níveis para as especificações de produtos.

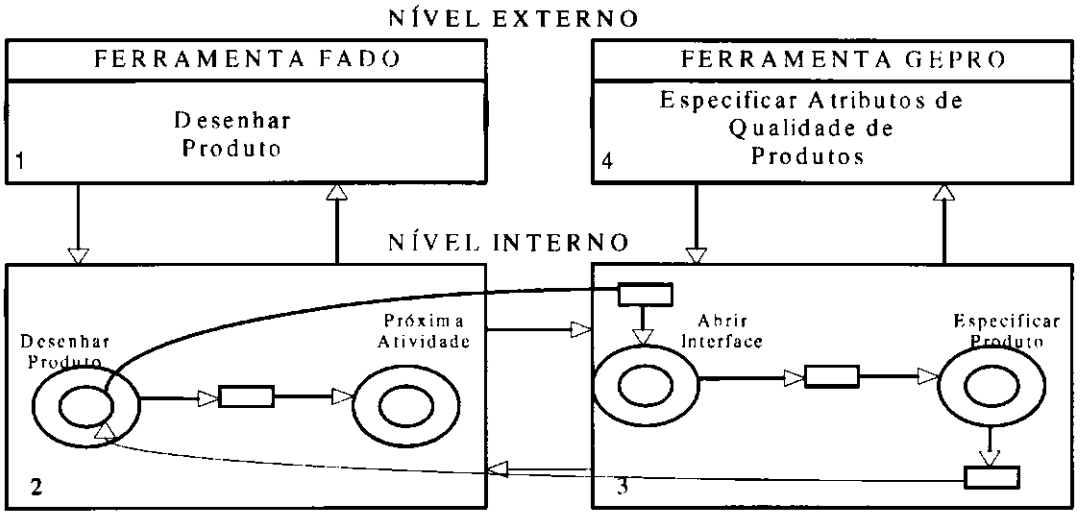


Figura 5.6 Níveis Interno e Externo das Especificações de Qualidade de Produtos

A definição dos pesos é atribuída pelo gerente de projeto. No futuro, conforme as características do projeto, podem ser desenvolvidas ferramentas que suportem a atualização automática desses valores.

Diagrama	Tipo Diagrama	Métricas	Valor Esperado	Valor Real
Projeto	DE	Complexidade	3	----------
Projeto	DE	Legibilidade	3	----------

Tabela 5.1 Tabela de Valores de Atributos de Qualidade

Para a especificação dos recursos humanos necessários, todas as atividades poderão ter um elo com a RA da GEPRO conforme figura 5.7.

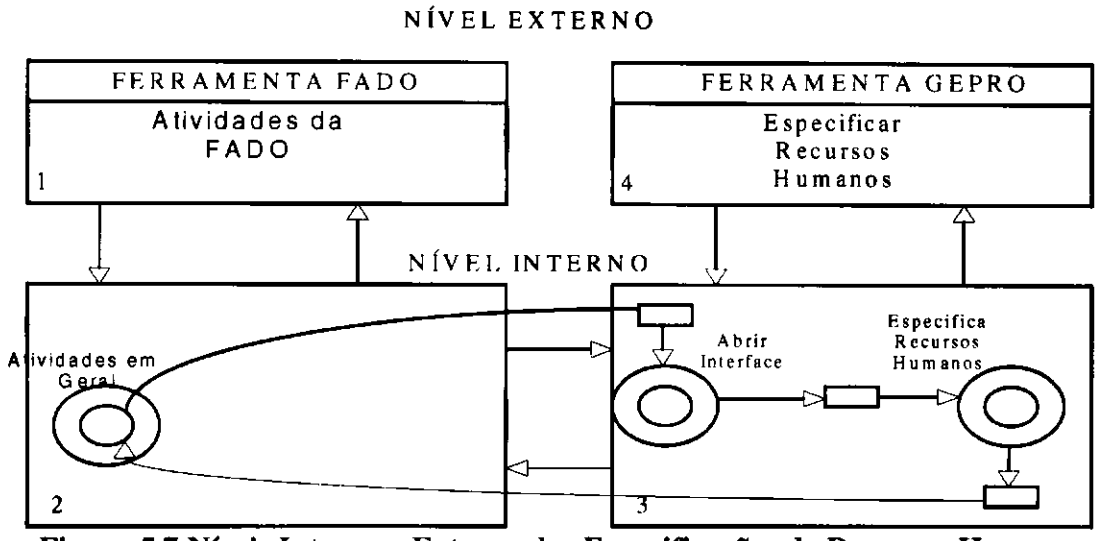


Figura 5.7 Níveis Interno e Externo das Especificações de Recursos Humanos

5.5 O Controle do Projeto

O controle é visto sob três ângulos: Produtos, Processos e Transição de Atividades.

5.5.1 O Controle de Produtos

O controle interno de produtos (diagramas, tabelas, códigos, etc) é realizado através de métricas especificadas que identificarão o estado de cada um conforme figura 5.8. A tabela de EstadoDiagrama, mostrado no capítulo anterior, é exemplificada através dos métodos de atividades chamados automaticamente. Os seu níveis são mostrados na figura 5.9.

Os valores reais são atualizados manualmente através de sua interface idênticos à especificação. A RA da FADO está relacionada a todas as atividades que possuem desenhos acoplados a uma RA da GEPRO. Essas atualizações manuais existirão até que se construam ferramentas "espertas" que nos dêem os valores reais de atributos de qualidade dos diversos produtos.

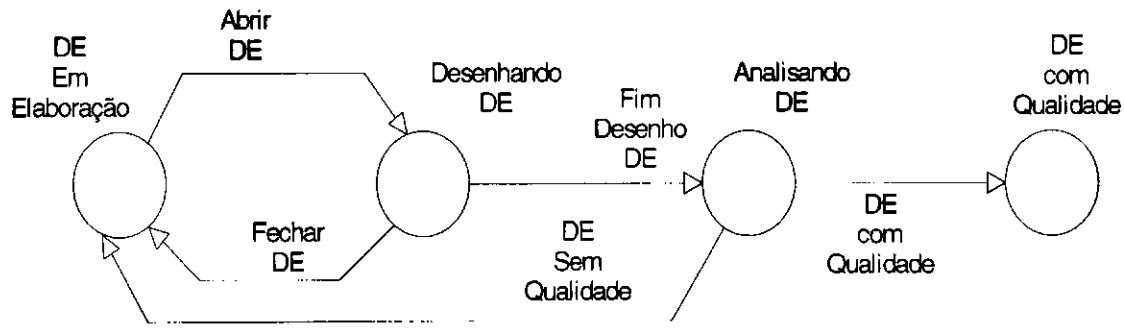


Figura 5.8 Estados dos Diagramas

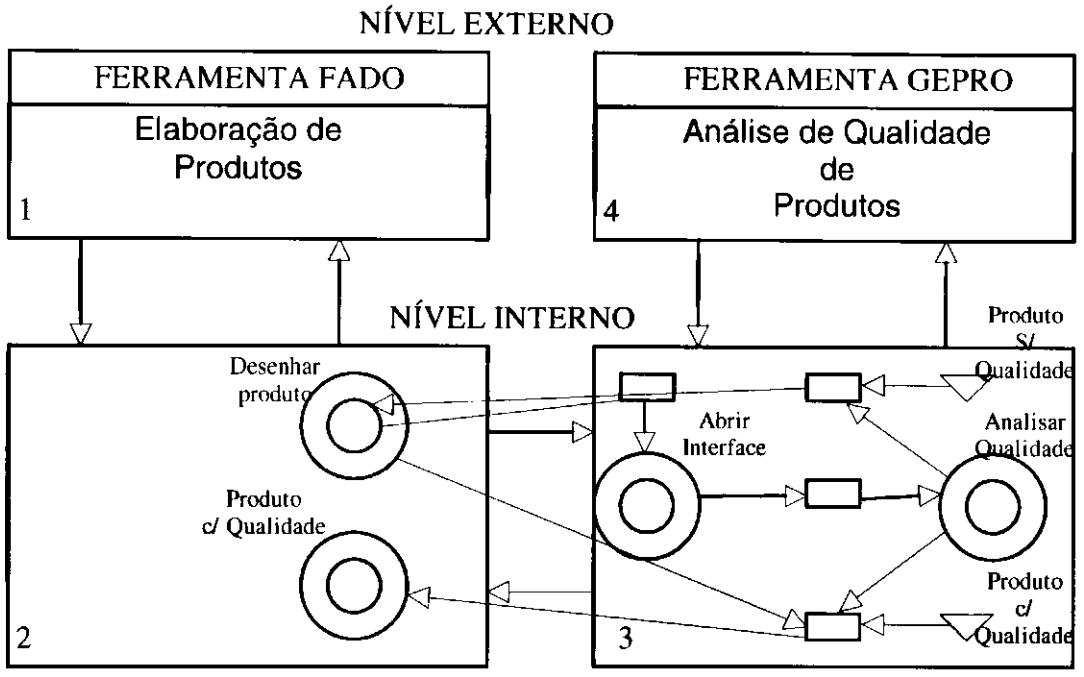


Figura 5.9 Níveis Interno e Externo do Controle de Produtos

5.5.2 O Controle de Processos ou Atividades

O controle de processos ou atividades é realizado através do controle de seus recursos humanos, recursos operacionais, custo, prazo, ocorrências e métricas.

No controle de recursos humanos, são observadas as pessoas alocadas conforme a especificação do projeto. Novas pessoas podem ser alocadas ou desalocadas conforme as novas necessidades do projeto. A figura 5.7 mostra os dois níveis da especificação dos recursos humanos necessários.

O controle de recursos operacionais é visto do mesmo modo que os recursos humanos, como também as decisões de alocar e desalocar recursos.

Em ocorrências indesejadas, são informados os erros em suas diversas atividades e a sua origem. Por exemplo, podemos estar em uma atividade de teste, após a implementação, e descobrir que não existe **Corretude** no projeto, ou seja, o seu comportamento não seguiu a especificação desejada. Os seus níveis são mostrados na figura 5.10.

Nas métricas de processos, serão controlados os atributos de qualidade do item 6 e 7 da especificação, ou seja, a produtividade e visibilidade.

A produtividade será vista, de um modo automático através das atualizações da classe Tabela, na forma de números de produtos (diagramas) elaborados. Por exemplo, para um sistema, se temos 20 DE a serem elaborados com uma previsão de 100 horas, concluímos que, para a elaboração de um diagrama, serão gastas em média 5 horas. Agora, se elaboramos 10 DE e gastamos 50 horas, isso implica que o projeto está caminhando dentro das metas estabelecidas, portanto com uma boa produtividade.

A visibilidade é função da elaboração dos seus documentos dentro de cada atividade. A utilização de ferramentas como GEPRO e FADO, indiretamente, já produz uma boa visibilidade.

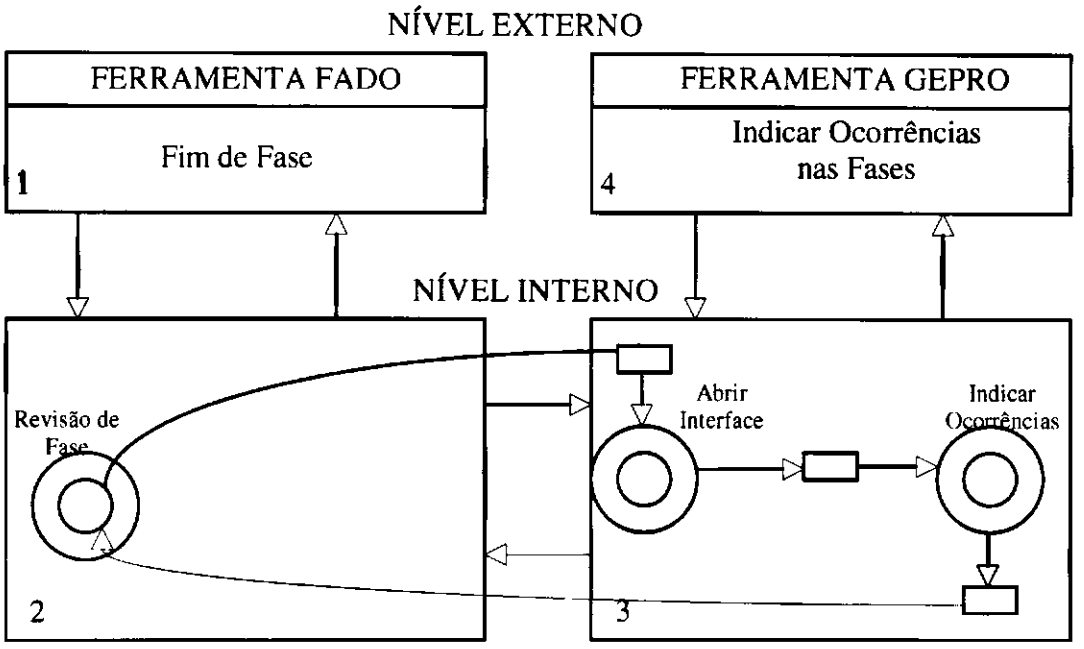


Figura 5.10 Níveis Interno e Externo das Ocorrências de Erros nas Fases de Projeto

5.5.3 O Controle de Transições de Atividades

O controle de transições de atividades é realizado de um modo automático através de sua RA. Ela foi descrita com detalhes no capítulo anterior. É visto através do monitoramento de projeto com apoio da RA.

5.5.4 Relatórios e Consultas

Estes são vistos durante o desenvolvimento de projeto sobre o módulo de controle (Produtos e Processos). Diversas são as informações que servirão de `feedback` para o gerente, para que decisões possam ser tomadas. Abaixo, descrevemos alguns tipos de informações que poderão ser obtidas.

. Quanto aos Processos:

- Relação de atividades sendo elaboradas com recursos humanos alocados;

- Relação de atividades sendo elaboradas com recursos operacionais;

- Relação de atividades com datas de início e fim previstas;

- Relação de atividades com tempos previstos e reais gastos;

- Relação de atividades com suas ocorrências;

- Relatório gráfico baseado em GANNT e mapa de PERT/CPM;

. Quanto aos Produtos:

- Relação de estados de diagramas;

- Relação de diagramas e medidas previstas e reais;

- Relação de engenheiros elaborando diagramas.

5.5 Conclusão

A nossa ferramenta é constituída de uma interface gráfica e dos módulos de especificação, controle e relatórios/consultas.

Um fato importante foi a forma de ligação entre as duas ferramentas FADO e GEPRO. Com isto novas metodologias poderão ser apoiadas pela GEPRO, bastando, para isso, estabelecer as conexões das atividades desta metodologia com os controles da GEPRO.

No controle de processos, as transições de atividades são controladas e os seus aperfeiçoamentos serão realizados através de informações de ocorrências que servirão como experiências para novos projetos.

O controle sobre produtos é baseado em medições, indicando se o mesmo possui os atributos de qualidade desejados. Isto poderia ser de um modo automático caso se desenvolva uma ferramenta "esperta". Por exemplo, um analisador de complexidade pode ser desenvolvido para identificar a complexidade de determinado produto (diagrama, códigos, etc).

Os desvios podem ser vistos no módulo de consultas ou visões. Os aperfeiçoamentos são realizados através de experiências de projetos desenvolvidos através de valores obtidos e das ocorrências indesejáveis relatados no controle de processos e produtos.

No módulo de especificação e controle, podem ser desenvolvidas diversas ferramentas que darão ao ambiente um maior suporte de automação.

A interface gráfica pode ser aproveitada, também, como uma ferramenta para modelar outros tipos de trabalhos, sendo utilizada, portanto, como uma metodologia para desenvolvimento de projeto de `Software`. Por exemplo, as restrições de integridade da ferramenta FADO poderiam ser modeladas utilizando esta ferramenta.

Capítulo 6

Conclusão

6.1 Introdução

Ambientes de Desenvolvimento de Software estão sendo pesquisados com o intuito de automatizar os processos de desenvolvimento, objetivando alcançar uma maior qualidade e produtividade em seus produtos e processos. As métricas de Software são reconhecidas como um modelo de controlar, através de medidas, os processos e produtos de projeto de Software. A integração dessas áreas de Engenharia de Software com outras, como Banco de Dados e Inteligência Artificial, é de fundamental importância na construção de grandes projetos de Software.

Um outro fator importante é a utilização de modelos Orientados a Objeto, pois, com isso, o gerente de projetos pode, mais naturalmente, focalizar os elementos de um Software complexo a ser construído, e realizar seus controles.

O nosso grande esforço era encontrar uma solução, através de uma ferramenta de gerência de projeto de Software, para controlar os processos de desenvolvimento de uma forma dinâmica.

A utilização da Rede de Petri torna difícil a modelagem de processos de uma metodologia deixando complexa a sua legibilidade. O mapa de Pert/CPM modela somente os aspectos estáticos e isto não era o que nós desejávamos.

Um fator importante era quanto a sua implementação. Tornando os elementos da RA como objetos, eles seriam mais fáceis de implementar, como também, guardariam suas próprias informações, podendo ser implementada por qualquer linguagem orientada a objeto.

6.2 Considerações Finais

O modelo da GEPRO tem como sua principal característica ser um sistema aberto, onde novas atividades podem ser incorporadas ao modelo de uma metodologia com o objetivo de aperfeiçoar os processos de desenvolvimento de Software. A sua característica aberta, também, proporciona a utilização de novas metodologias, bastando que sejam modeladas.

Uma rede de atividades foi definida para apoiar a gerência dos processos de uma metodologia através de sua modelagem. Um conjunto de objetos interrelacionados foi definido e incorporado ao TOM com objetivo de facilitar as especificações de projetos e apoiar os desenvolvedores de projetos de Software através de suas informações.

Nós apresentamos, neste trabalho, uma ferramenta de gerência para um ambiente com o objetivo de gerenciar projetos de Software. Abaixo, levantamos alguns tópicos que foram desenvolvidos por este trabalho:

✓ Necessidade de se definir uma ferramenta que pudesse controlar uma metodologia de projeto de Software;

✓ Definição de uma rede, baseada em Redes de Petri, para descrever a interdependência das atividades em um projeto de software e os respectivos controles;

✓ Necessidade da especificação, aplicação, refinamento e execução dos processos de uma metodologia de desenvolvimento de Software através de uma RA;

✓ Necessidade de adoção de feedback;

✓ Necessidade de elaboração de várias RAs para GEPRO;

✓ Necessidade do acoplamento entre as RAs FADO e GEPRO.

O enriquecimento do ambiente DYNAMO através do desenvolvimento da RA e da GEPRO possibilitou a este ambiente obter diversas características apresentadas anteriormente. No capítulo 3 mostramos também as restrições e deficiências do nosso ambiente.

6.3 Contribuições

Abaixo relacionamos algumas contribuições dadas pela ferramenta de gerência de projeto de Software GEPRO;

✓ Complemento à metodologia FADO, pois essa tinha somente o suporte para o componente de construção;

✓ A elaboração de uma rede onde se dará suporte a projetos que utilizam de controle de processos;

✓ Discussão para chegar ao melhor nível no CMM da SEI;

✓ Contribuição de pesquisas nas áreas de ADS e métricas na UFPB/CG;

✓ Análise de Ambientes de Desenvolvimento de Software;

✓ Maior automatização do ambiente DYNAMO.

6.4 Trabalhos Futuros

Um problema enfrentado é a comunicação das interfaces da metodologia e a ferramenta GEPRO quando várias pessoas estiverem realizando alguma atividade. Uma restrição à

ferramenta é a simplicidade ou a falta de um programa de métricas sugerido, como métricas para estimar prazos e custos.

Portanto, poderíamos, em função desses problemas, indicar os seguintes trabalhos futuros:

✓ Aperfeiçoar os processos de desenvolvimento de `Software` através de elaborações de novas métricas para a ferramenta;

✓ Ferramentas automáticas de análise de qualidade chamadas através de uma atividade;

✓ Aplicar à RA para novas aplicações de controle de processos;

✓ Desenvolver métricas de custos e prazos automáticas devido à coleção de dados anteriores para o módulo de especificação de projeto;

✓ Aplicar a RA para utilização de projetos em geral;

✓ Integrar várias ferramentas em um ambiente através de uma rede RA;

✓ Implementar a comunicação das interfaces das duas ferramentas. Uma solução, para esta implementação, seria mandar as mensagens, que seriam ações, para uma caixa eletrônica do gerente. O gerente, ao observar essas mensagens, poderia chamá-las, e então, sua interface ser acionada;

✓ Desenvolver ferramentas que contemplem os diversos tipos de métricas a serem incorporadas ao ambiente;

✓ Desenvolver a interface da GEPRO.

REFERÊNCIAS BIBLIOGRÁFICAS

[APC90] Ambriola, V., P. Ciancarini, e C. Montagero. Software Process Enhactement in Okos. ACM SIGsoft Software Engineering Notes, 15(6):183-192, December 1990.

[Ari93] Arifoglu, Ali. A methodology for Software de Cost Estimation. ACM SIGSOFT Engineering Notes 18(2):96-105, Abril de 1993.

[Bar92] Barghoutit Naser S. Suporting cooperation in the Marvel process-centered SDE. ACM Software. pg 21-30, Dezembro de 1992.

[Bap92] Baptista A., "Uma Metodologia de Análise e Projeto De Sistema Orientado a Objetos". Dissertação de Mestrado em Informática UFPE Departamento de Informática 1992.

[BKG92] Ben-Shaul, Israel Z., Kases, Gail E. e George T. Heinerman. An Architecture for multi-user Software development environment. ACM Software. pg 149-158, Dezembro de 1992.

[BB88] Basili, Victor R. e H. Dieter Rombach. The TAME project: towards improvement-oriented Software environments. IEEE Transactions Software Engineering, SE-14(6):758-773, june 1988.

[BJ92] Brown, Alan W., John A. McDermil. Learning From IPSE's Mistakes. IEEE Transactions Software Engineering, SE-14(6):758-773, june 1988.

[Boh81] Bohem, B. W., Software Enginneering Economic. Englewood Cliffsa, NI Prentice - Hall 1981.

[Boh84] Bohem B. W., "Verifying and Validating Software Requirements e Designs Specifications". IEEE Software, Janeiro 1984.

[Boh88] Bohem, B. W., "Understanding and Controlling Software Costs". IEEE Transaction on Software Engineering. Vol 14 No 10. Outubro 1988.

[BCH92] Brockers, Alfred, Christopher M. bott, H. Dieter Rombach e Martin Verlage. NVP Language Report. Technical Report 229/92, Fachbereich Informatik, Universität KaiserSlautern, Kaiserslautern, Germany, December 1992.

[Bro78] Brooks, Frederick P.Jr. The Mytical Man-Month. Addison Wesley, 1978.

[CE93] Cerqueira Neto, Edgard Pedreira de, "Gestão de Qualidade, Principios e Métodos". Ed. Pioneira São Paulo, 1993.

[Che76] Chen P., "The Entity-Relationship Model Towards a Unified View of Data". ACM Transactions on Database Systems. Março 1976.

[CK92] Cybulky, Jacob L., Karl Reed. A Hipertext Basead Software Engineering Environment. IEEE Software. pg 62-68, Março de 1992.

[Cle88] Clemm, Geoffrey M. The Workshop System - a practical knowledge based Software environment. ACM SIGSOFT Software Engineering Notes. 13(5):55-64, November 1988.

[DBM88] U. Dayal, A.P. Buchanan, D.R. McCarthy, "Rules are Objects Too: A Knowledge Model for Active Object-Oriented Database Systems", Proc. 2nd Intl. Workshop on Objec-Oriented Database Systems, LNCS 334, Springer Verlag, 1988, pp. 129-143.

[DRP87] Dart Susan A., RobertJ.Ellison, Peter H. Feiler, e A. Nico Habermann. Software Development Environment, IEEE Computer, pages 18-28, November 1987.

[DV90] Deiters, Wolfang e Volker Gruhn. Managing Software process in the environment MELMAC. ACM SIGSOFT Software Engineering Notes, 15(6):193-205, December 1990.

[DV92] Deiters, Wolfang e Volker Gruhn. The FUNSOFT Net approach to Software process management. Technical Report 2/92, Fraunehofer Institut for Software e System Engineering, University of Dortmund, Dortmund Germany, 1992.

[DeM82] DeMarco, Tom. Controle de Projetos de Software, gerenciamento avaliação e estimativas. Rio de Janeiro. Ed. Campus, 1982.

[Dow87] Dowson, Mark. Integrated project support with STAR. IEEE Software, pages 6-15, November 1987.

[Eji93] Ejiogu, Len O. Five principles for the formal validation of models of Software metrics. ACM SIGPLAN 28(8):67-76, agosto de 1993.

[Fai85] Fairley, R.E., Software Engineering Concepts. McGraw-Hill. Séries in Software Engineering and Technology 1985.

[FCKL92] Fernstrom, Christer, Kjell-Hakan Narfelt e Lennart Ohlsson. Software factory principles, architeture, e experiments. IEEE Software, pages 36-44, march 1992.

[FS93] Furtado, E., Schiel U,. "Uma Metodologia para projeto de Banco de Dados Temporal Orientado a Objetos". VII Simpósio Brasileiro de Banco de Dados. Campina Grande, Maio 1993.

[Fu93] Maria Elisabeth S. Furtado "Uma metodologia para projeto de banco de dados temporal orientado a objetos" Diss. de Mestrado, COPIN/UFPB, 1993.

[GE90] Garlan, David e Ehson Ilias. Low-Cost, Adaptable Tool Integration Policies for Integrated Environment. ACM SIGSOFT pg 1-10, 1990.

[GM91] Ghezzi, C. Jazayeri, M. Mandili, D."The Fundamentals of Software Engineering". Prentice-Hall Internacional, 1991.

[JR91] K. Jensen & G. Rozenberg High-Level Petri Nets, Theory and Applications, Springer Verlag, 1991.

[Kad92] Kadia, R., Issues Encountres in building a flexible Software development environment. ACM SIGSOFT, pg 169-180, 1992.

[KKTK91] Kasumoto, Shinji, Ken-Ichi Matsumoto, Tohru Kikuno e Koji Torii. On a measurement envoironment for controlling Software development activities. IEICE Transaction, E74(5):1051-1054, May 1991.

[KA90] Khoshafian, s., Abnous R., "Object Orientacion, Concepts, Languages, Databases, Userinterfaces". Copyright 1990.

[KB95] Krishnamurthy Balchander, David S. Roseblin. Yeast: A general purpose Event-Action system. IEEE Transaction on Software Engineering 21(10):845-857, Outubro de 1995.

[KFA90] Kugler, J. L., Fernandes, Aguinaldo Aragon. "Gerência de Projeto de Sistemas Uma Abordagem Prática". 2 ed.Rio de Janeiro. LTC Livro Técnico e Científicos 1990.

[KFP88] Gail E. Kaiser, Peter H. Feiler, and Steven S. Popovich. Intelligente assitance and maintenance. IEEE Software, pg 40-49, Maio 1988.

[Lai93] Lai, Robert. The move to mature process. IEEE 10(4):14-15, julho de 1993.

[LJ94] Lorent, Mark e Jeff Kidd. Object-Oriented Software Métrics. Prentice Hall, Inc; Ney Jersey, 1994.

[Lot93] Lott Christopher M. Process and measurement support in SEEs. ACM SIGSOFT Software Engineering Notes. 18(4):83-93, Agosto de 1993.

[LT87] Lehman, M. M., W. M. Turski. Essential properties of IPSEs. ACM SIGSOFT Software Engineering Notes, 12(1):52-55 1987.

[Mat90] Matsumoto, Ken-Ichi. A programmer performance model and its measurement environment. PhD thesis, Osaka University, Toyonaka, Osaka 560, Japan, August 1990.

[MD93] Möller, K. H., D-J Paulish. Software Metric, A practioners's guide to improved product. IEEE Computer Society Press. Chapman @ Halll Computing, 1993.

[MMJP93] Milet, Paulo Barreira; Milet, Evandro Barreira; Jr, Paulo Jorge C. Pereira. "Os Princípios da Qualidade Total aplicadas a área de Informática. Rio de Janeiro; LTC - LivrioTécnico e Científico Ed. 1993 56pp.

[MMJP93] Milet, Paulo Barreira; Milet, Evandro Barreira; Jr, Paulo Jorge C. Pereira. "Indicadores de Qualidade e Produtividade para área de Informática. Rio de Janeiro; LTC - Livro Técnico e Cientifico Ed. 1993. 56p.

[MMJP93] Milet, Paulo Barreira; Milet, Evandro Barreira; Jr, Paulo Jorge C. Pereira. "O usuário que não sabe o que quer" Rio de Janeiro; LTC Livro Técnico e Cientifico Ed. 1993. 56p.

[Mon92] Monarchl D., Puhr G., A research typology for object-oriented Analysis and Design. Comunication of the ACM. vol 35, no 9. Setembro de 1992.

[MSTK93] Matsmuto, Ken-Ichi, Shinji Kusumoto, Tohru Kikuno e Koji Torii. A new framework of measuring Software development processes. In Proceedings of the 1. International Software Metrics Symposium, pages 108-118. IEEE Computer Society Press, May 1993.

[Not88] Notkin, David. The relationship between Software development environments and the Software process. ACM SIGsoft Engineering Notes, 13(5):107:109, February 1988.

[OV92] Oivo, Markku e Victor R. Basili. Representing Software engineering models. The TAME goal orientes approach. IEEE Transactions on Software Engineering, 18(10):886-898, October 1992.

[PBMC93] Paulk, Mark C., Bill Curtis, Mary Beth Chrissis, Charles v. Weber. Capability Maturity Model, Version 1.1. IEEE Software vol. 10 july 1993.

[PCW92] PC World Magazine, dezembro 1992, pp. 60-68.

[PW88] Penedo, Maria H. e William E. Riddle. Guest Editor's introduction to Software engineering environment architectures. IEEE Transactions on Software Engineering, 14(96):689-695, June 1988.

[PR90] Porter, Adam A. e Richard W. Selby. Empirically guided Software development using metric-based classification trees. IEEE Software, pages 46-54, March 1990.

[Pra88] Prado, Darci. Administração de projetos com Pert/CPM. 2 - ed. Rio de Janeiro LTC; Belo Horizonte : Editora UFMG, 1988. 126p.

[Re82] Reisig, W. Petri Nets - An Introduction, Springer Verlag, 1982.

[Rep87] Reps, Thomas. Language Processing in Program Editors. IEEE computer, pages 29-40, November 1987.

[Rob89] Robach, H. Dieter. The Role of measurement in ISEEs. In Carlo Ghezzi e John McDermid, editors, Proceedings of the 2. European Software Engineering Conference, pages 65-85. Lectures Notes in Computer Science Nr. 387, Springer Verlag, September 1989.

[RS88] Ramanathan Jaysashree e Soumitra Sarkar. Providing customized assistance for Software lifecycle approaches. IEEE Transactions on Software Engineering, 14(6):749-757, June 1988.

[Rib87] Ribeiro, C. A. S. "I Simpósio Brasileiro de Engenharia de Software".SBC - Sociedade Brasileira de Computação, UFRJ. Universidade Federal do Rio de Janeiro, COPPE/SISTEMA - NCE, 1987. Pp 55-59.

[Rot93] Rothery, Brian. ISO 9000. So Paulo. Makron Books, 1993.

[SAJ91] Selby, W. Richard, Adam C. Schmidt e Ji Berney. Metric-driven analysis and feedback systems for enabling empirically guided Software development. In Proceedings of the 13. International Conference Software Engineering, pages 288-298. IEEE Computer Society Press, May 1991.

[TFLL88] Taylor, Richard N., Frank C. Belz, Lori A. Clarke, Leon Osterwweil, Richard W. Selby, Jack C. Wileden, Alexander L. Wolf e Michal Young. Foundations for the arcadia environment architeture. In Peter Henderson, (editor), Proceedings of 3. ACM SIGsoft/SIGPLAN symposium on Practical Software Development Envoronments, pages 1-13, November 1988. Appeared as ACM SIGSOFT Software Engineering Notes 13(5), November 1988.

[Tha87] Thayer Richar H., Sofware Engineering Project Management A Top-Down View. IEEE Software, Pg - 15-51, 1987.

[TJR92] Tate, Grahan, June Verner e Ross Jeffrey. CASE: A testbed for modeling, measurement e management. Communication of the ACM, 35(4):65-72, April 1992.

[Wei93] Weinberg, Gerald M. "Software com qualidade: pensando e idealizando sistemas". São Paulo: Makron Books, 1993.

[You92] Coad P., Yourdon E., "Análise Baseado em Objetos". Campus 1992.

[ZD93] Zage, Wayne M. e Dolores M. Zage. Evaluation design metrics on large-scale Software. IEEE Software 10(4):75-81, julho de 1993.