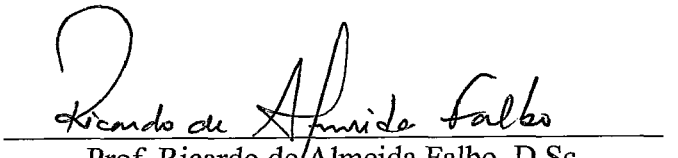
INSTANCIAÇÃO DE PROCESSOS DE SOFTWARE EM AMBIENTES CONFIGURADOS NA ESTAÇÃO TABA

Patricia Machado Berger

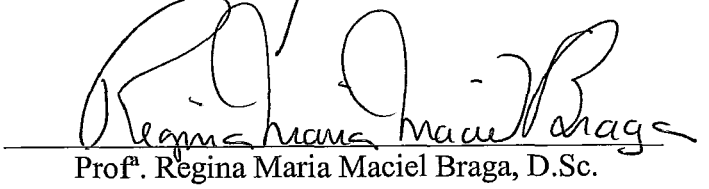
TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Profª. Ana Regina Cavalcanti da Rocha, D.Sc.



Prof. Ricardo de Almeida Falbo, D.Sc.



Profª. Regina Maria Maciel Braga, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
JULHO DE 2003

BERGER, PATRICIA MACHADO

Instanciação de Processos de Software em Ambientes Configurados na Estação TABA [Rio de Janeiro] 2003

VIII, 112 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2003)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Processo de Software

2. Ambientes de Desenvolvimento de Software Orientados à Organização

I. COPPE/UFRJ II. Título (série)

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

INSTANCIAÇÃO DE PROCESSOS DE SOFTWARE EM AMBIENTES CONFIGURADOS NA ESTAÇÃO TABA

Patricia Machado Berger

Julho/2003

Orientadora: Ana Regina Cavalcanti da Rocha

Programa: Engenharia de Sistemas e Computação

Organizações de software constantemente precisam aumentar a qualidade de seus produtos, ao mesmo tempo em que buscam diminuir o tempo e custo de desenvolvimento de projetos de software. Nesse contexto, ganha importância a questão da qualidade do processo utilizado para construir o software. Uma abordagem para garantia da qualidade do processo é a definição de um processo padrão, que descreve as atividades que devem ser realizadas no desenvolvimento de sistemas de software em todos os projetos de uma organização, garantindo a conformidade com os padrões de qualidade e procedimentos da organização. Por ser aplicável a todos os projetos, o processo padrão é genérico e precisa ser adaptado às necessidades específicas de cada projeto, o que não é uma tarefa simples. Portanto é necessário oferecer apoio ao gerente do projeto na realização desta tarefa, e este apoio pode ser beneficiado pelo uso de técnicas de Gerência do Conhecimento.

Este trabalho apresenta uma abordagem para a adaptação de processos de software para projetos específicos. A partir do processo adaptado é possível gerar um Ambiente de Desenvolvimento de Software Orientado à Organização para o projeto (ADSOrg). ADSOrg são ambientes que apóiam a atividade de engenharia de software em uma organização, fornecendo o conhecimento acumulado e relevante para esta atividade, e dando apoio ao aprendizado organizacional em Engenharia de Software.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

SOFTWARE PROCESS INSTANTIATION FOR SPECIFIC PROJECTS IN THE TABA WORKSTATION

Patricia Machado Berger

Julho/2003

Advisor: Ana Regina Cavalcanti da Rocha

Department: System and Computing Engineering

Software organizations must increase the quality of their products and, at the same time, decrease development costs and time to market. In this context, the quality of the software process becomes of great importance. An accepted approach to software processes is the definition of a standard process for the organization. The standard process defines the activities that must be performed in every project of the organization. However, the standard process is generic, and must be adapted to the specific needs of each project. The adaptation of the standard process for a specific project is not an easy task, therefore, it becomes necessary to offer support to the project manager using knowledge management techniques.

This thesis describes an approach to support the instantiation of software processes to specific projects. Based on the adapted process, an Enterprise-Oriented Software Development Environment (EOSDE) will be generated for the project. The EOSDE concept was defined aiming to support the knowledge management process in organizations that develop software. EOSDE support the software engineering activity in an organization, providing the accumulated and relevant knowledge for software development while supporting organizational learning.

À minha família

AGRADECIMENTOS

À minha família, por todo amor, apoio e incentivo. Em especial à minha avó Therezinha, por todo o carinho e dedicação.

A Karina, Gleison, Mariano, Sômulo e Sávio, pela ajuda e troca de conhecimentos.

Aos professores Ricardo Falbo e Regina Braga, por participarem da banca.

Ao pessoal administrativo da COPPE/Sistemas, Ana Paula, Claudia e Solange.

À CAPES pelo apoio financeiro.

À minha orientadora Ana Regina, pela dedicação, orientação e amizade.

ÍNDICE

CAPÍTULO I - INTRODUÇÃO

1.1 Motivação

Os processos e atividades relacionados à Engenharia de Software requerem a utilização constante de conhecimento, como por exemplo, conhecimento sobre o domínio de aplicação, conhecimento sobre Engenharia de Software, normas e práticas organizacionais, técnicas e métodos, experiência de projetos, dentre outros. Este conhecimento é produzido e utilizado durante a realização de projetos de software. Membros de equipes de projeto adquirem conhecimento a cada projeto, e a organização e indivíduos poderiam ganhar muito ao compartilhar este conhecimento. Entretanto, este conhecimento encontra-se disperso na mente de várias pessoas ou em documentos e as equipes de desenvolvimento não se beneficiam da experiência existente, repetindo os mesmos erros, embora alguns indivíduos na organização saibam como evitá-los. Como conseqüência, desenvolvedores de software, muitas vezes, falham em atender rápida e adequadamente às solicitações de seus clientes, porque o conhecimento de que necessitam foi perdido pela organização ou está em lugares desconhecidos ou inacessíveis. Sempre que o conhecimento requerido não está disponível, o desenvolvedor de software tem que partir do zero para a solução do problema, experimentar e construir novamente o conhecimento (VILLELA *et al.*, 2001b; RUS *et al.*, 2002).

Organizações de software constantemente precisam aumentar a qualidade de seus produtos, ao mesmo tempo em que buscam diminuir o tempo e custo de desenvolvimento de projetos de software (RUS *et al.*, 2002). Para atingir estes objetivos, é fundamental que as organizações sejam capazes de reter o conhecimento obtido em projetos anteriores e aplicá-lo em projetos futuros.

O conceito de Ambientes de Desenvolvimento de Software Orientados à Organização (ADSOrg) surgiu da necessidade de se gerenciar o conhecimento organizacional adquirido ao longo de projetos de software. Estes ambientes apóiam a atividade de engenharia de software em uma organização, fornecendo o conhecimento acumulado e relevante para esta atividade e dando apoio ao aprendizado organizacional em Engenharia de Software (VILLELA *et al.*, 2001a; VILLELA *et al.*,2001b).

O conhecimento sobre a definição de processos de software é um exemplo de conhecimento relacionado às atividades de Engenharia de Software. Acredita-se que a chave para a qualidade do software depende da qualidade do processo usado para construí-lo e, assim, muitos dos problemas de desenvolvimento de software podem ser resolvidos aperfeiçoando o processo (RAMAN, 2000; FUGGETTA, 2000).

Uma abordagem encontrada na literatura para garantia da qualidade do processo é a definição de um processo padrão para a organização. O processo padrão descreve as atividades que devem ser realizadas no desenvolvimento de sistemas de software em todos os projetos de uma organização, garantindo a conformidade com os padrões de qualidade e procedimentos da organização. Por ser aplicável a todos os projetos, o processo padrão é genérico e precisa ser adaptado às necessidades específicas de cada projeto. Logo, gerentes de projeto devem ser capazes de adaptar este processo para projetos específicos, o que não é uma tarefa simples (HENNINGER, 1999; KRUCHTEN, 1999; MAURER *et al.*, 1999; HENNINGER, 2001; NAGEL, 2001; HOLZ *et al.*, 2001, OLIVEIRA, 1999, MACHADO, 2000, BORGES, 2001).

Ambientes de Desenvolvimento de Software Orientados à Organização (ADSOrg) se propõem a apoiar a atividade de engenharia de software, possibilitando a gerência do conhecimento que pode ser útil aos engenheiros de software ao longo dos projetos de uma organização (VILLELA *et al.*, 2000). Portanto, este tipo de ambiente deve, também, ser capaz de gerenciar o conhecimento sobre a adaptação de processos de software. O trabalho aqui descrito se insere neste contexto.

1.2 Objetivo da Tese

O objetivo fundamental deste trabalho é a definição de uma abordagem para a adaptação de processos de software e geração de ADSOrg para projetos específicos, dentro do contexto de um Ambiente Configurado para uma organização na Estação TABA.

Para atingir este objetivo, foi definido um processo de instanciação de processos de software. Este processo é apoiado por uma ferramenta através da qual é disponibilizado ao gerente de projeto o conhecimento sobre instanciação de processos de software acumulado pela organização em projetos anteriores. À medida que novos projetos forem realizados, o conhecimento sobre instanciação de processos obtido pelos

vários gerentes de uma organização vai sendo acumulado e disponibilizado. Esta abordagem fundamenta-se, portanto, nos conceitos de Gerência do Conhecimento e de Ambientes de Desenvolvimento de Software Orientados à Organização.

A ferramenta *AdaptPro*, definida e implementada neste trabalho, está inserida no contexto da Estação TABA, e é disponibilizada em um Ambiente Configurado para uma organização específica. Os usuários da ferramenta são gerentes de projeto, responsáveis pela instanciação de processos para projetos específicos.

1.3 Estrutura da Tese

Esta tese contém, além desta Introdução, mais cinco capítulos e um anexo.

No segundo capítulo, é apresentado o estudo realizado sobre processo de software, descrevendo-se o estado da arte. São discutidos tópicos tais como modelos de ciclo de vida e critérios para sua seleção, as normas ISO/IEC 12207 e ISO/IEC TR 15504, o modelo de maturidade CMM e sua evolução para o CMMI, a definição de um processo padrão para uma organização e, a partir dele, a definição de processos especializados e instanciados.

No terceiro capítulo, é apresentada a revisão bibliográfica sobre Gerência de Conhecimento, discutindo-se os conceitos de conhecimento, aprendizado organizacional e memória organizacional. Também é discutida a aplicação da Gerência de Conhecimento no contexto da Engenharia de Software.

No quarto capítulo, é apresentado o conceito de Ambientes de Desenvolvimento de Software Orientados à Organização, discutindo-se seus objetivos, requisitos e infra-estrutura de conhecimento. A Estação TABA, que fornece a infra-estrutura necessária à instanciação de ADSOrg, é brevemente apresentada. Também é discutida a definição de processos no contexto de Ambientes de Desenvolvimento de Software Orientados à Organização. Por fim, é apresentado o processo de instanciação de processos de software objeto deste trabalho.

No quinto capítulo, é apresentada a abordagem proposta para a instanciação de processos de software em Ambientes Configurados na Estação TABA. Esta abordagem é apoiada pela ferramenta *AdaptPro*, que também é apresentada neste capítulo.

No sexto capítulo são apresentadas as considerações finais deste trabalho, ressaltando suas contribuições, limitações e perspectivas futuras.

O Anexo 1 apresenta as alterações realizadas no modelo de classes da Estação TABA.

CAPÍTULO II - PROCESSO DE SOFTWARE

2.1 Introdução

Embora diferentes projetos fracassem de maneiras diferentes, aparentemente a maioria deles fracassa devido a uma combinação de causas, dentre as quais estão a gerência de requisitos *ad hoc,* comunicação ambígua e imprecisa, e complexidade crescente. Se essas causas forem tratadas, a organização estará em boa posição para desenvolver e manter software de qualidade de maneira consistente e previsível, obtendo grande vantagem econômica (KRUCHTEN, 1999).

Para atender a essa demanda, ganha importância a questão da qualidade de software e, mais especificamente, o estudo e melhoria do processo através do qual o software é desenvolvido. Acredita-se que a chave para a qualidade do software depende da qualidade do processo de software usado para construí-lo. Assim, muitos dos problemas de desenvolvimento podem ser resolvidos aperfeiçoando o processo. (RAMAN, 2000; FUGGETTA, 2000).

Sem um processo bem definido, a equipe de desenvolvimento trabalha de maneira *ad hoc* e o sucesso depende dos esforços heróicos de indivíduos dedicados. Esta situação não é sustentável. Em contraste, organizações maduras empregando um processo bem definido podem desenvolver sistemas complexos de maneira consistente e previsível, independente de quem o produziu (KRUCHTEN, 1999; ROCHA *et al.*, 2001).

O processo pode ser melhorado a cada novo projeto, sofrendo refinamentos contínuos para aumentar sua capacidade de lidar com os requisitos e expectativas do mercado e da organização, e aumentando a eficiência e produtividade da organização como um todo. Portanto, o processo precisa ser continuamente avaliado e melhorado. Essas observações motivaram uma variedade de trabalhos dedicados à criação de modelos de qualidade e métodos para melhoria de processos de software (KRUCHTEN, 1999; FUGGETTA, 2000).

O desenvolvimento de software não é apenas uma questão de criar ferramentas e linguagens de programação, mas também é um esforço coletivo, complexo e criativo. Como tal, a qualidade de um produto de software depende fortemente das pessoas, da

organização e dos procedimentos usados para criá-lo e disponibilizá-lo (FUGGETTA, 2000).

2.1.1 Processo de Software

Processo de software pode ser definido como um conjunto de atividades, métodos, práticas e transformações que são usados para desenvolver e manter software e seus produtos associados, que podem incluir planos de projeto, documentos de projeto, código, casos de teste e manuais do usuário. À medida que uma organização amadurece, o processo de software se torna melhor definido e mais consistentemente implementado por toda a organização (PAULK *et al.*, 1997; RAMAN, 2000).

FUGGETTA (2000) define um processo de software como um conjunto coerente de políticas, estruturas organizacionais, tecnologias, procedimentos e artefatos que são necessários para conceber, desenvolver, disponibilizar e manter um produto de software.

Um processo de software possui quatro papéis (KRUCHTEN, 1999):

1. Guiar a ordem de atividades da equipe.
2. Especificar quais artefatos devem ser produzidos e quando.
3. Dirigir as tarefas de desenvolvedores individuais e da equipe como um todo.
4. Oferecer critérios de monitoração e medição dos produtos e atividades do projeto.

Um processo de software explora os seguintes conceitos (FUGGETTA, 2000):

- Tecnologia de desenvolvimento de software: suporte tecnológico usado no processo. Para realizar as atividades de desenvolvimento de software são necessárias ferramentas, infra-estrutura e ambientes. A tecnologia adequada permite a produção de software de alta complexidade de maneira viável financeiramente.
- Métodos e técnicas de desenvolvimento de software: diretrizes para uso de tecnologia e realização das atividades de desenvolvimento de software. O suporte metodológico é essencial para explorar efetivamente a tecnologia.

- Comportamento organizacional: a ciência de organizações e pessoas. Software é geralmente desenvolvido por uma equipe que deve ser coordenada e gerenciada dentro de uma estrutura organizacional.
- Mercado e economia: como qualquer outro produto, o software deve atender às necessidades reais dos clientes em uma situação de mercado específica.

Ainda segundo FUGGETTA (2000), o processo de software deve identificar:

- Atividades que devem ser realizadas para atingir os objetivos do processo.
- Papéis das pessoas no processo.
- Estrutura e natureza dos artefatos que devem ser criados e mantidos.
- Ferramentas a serem utilizadas.

A definição do processo de software pode ser feita em diferentes níveis de abstração (ABRAN, 2001). Frameworks de modelos de ciclo de vida servem como uma definição de alto nível das fases que ocorrem durante o desenvolvimento. Não são definições detalhadas, mas apenas a identificação das atividades de alto nível e do relacionamento entre elas. Na seção 2 descrevemos os principais modelos de ciclo de vida.

Definições de modelos de processos de ciclo de vida são mais detalhadas. As duas principais referências nesta área são as normas internacionais ISO/IEC 12207 (ISO/IEC 12207, 1995) e a ISO/IEC TR 15504 (ISO/IEC TR 15504, 1998), descritas na seção 3.

Outro aspecto importante a ser considerado é a medição e melhoria dos processos. Neste contexto dois paradigmas quais são úteis: o paradigma analítico, caracterizado por se apoiar em evidência quantitativa para determinar onde as melhorias são necessárias, e o paradigma de *benchmarking*, que envolve medir a maturidade de uma organização ou a capacidade de seus processos. Referências importantes neste contexto são o *Capability Maturity Model* (CMM) (ABRAN, 2001) e o CMMI (SEI, 2002). O objetivo desta tese não envolve aspectos de medição e melhoria de processos. Entretanto a definição de processos deve considerar, muitas vezes, as áreas-chave definidas no CMM. A seção 4 descreve as principais características do CMM e do CMMI.

No contexto deste trabalho três conceitos são, ainda, importantes: o de processo padrão, o de processo especializado e o de processo instanciado. Estes conceitos são descritos na seção 5.

2.2 Modelos de Ciclo de Vida

Um modelo de ciclo de vida define as diferentes etapas na existência de um produto de software. Tipicamente elas são: especificação e análise de requisitos, projeto, desenvolvimento, verificação e validação, operação, manutenção e descontinuação. Adicionalmente, um modelo de ciclo de vida de software define os princípios e diretrizes que vão guiar a realização destas etapas. Em geral, o ciclo de vida define a estrutura e a filosofia segundo as quais o processo de software tem que ser executado. Entretanto, ele não descreve um curso de ação preciso, uma organização, ferramentas e procedimentos de operação, políticas de desenvolvimento e restrições. Um ciclo de vida é um importante ponto de partida para definir como o software deve ser desenvolvido, mas a adoção de um ciclo de vida não é o suficiente para guiar e controlar um projeto de software na prática (FUGGETTA, 2000).

As características básicas comuns a todos os modelos são (CHRISTENSEN, 2001):

- descrever as principais fases do desenvolvimento;
- definir as principais atividades a serem realizadas durante cada uma das fases;
- especificar os produtos de cada uma das fases e insumos para o início das fases, e,
- fornecer um framework sobre o qual as atividades necessárias podem ser mapeadas.

2.2.1 Principais Modelos de Ciclo de Vida

A seguir serão descritos os principais modelos de ciclo de vida.

2.2.1.1 O Modelo Cascata

É o modelo mais antigo e o mais amplamente usado na engenharia de software, modelado em função do ciclo da engenharia convencional. Requer uma abordagem

sistemática, seqüencial ao desenvolvimento de software. O modelo em cascata é mais adequado em situações nas quais os requisitos são bem entendidos e o gerente do projeto confia na capacidade da equipe de desenvolver utilizando o processo (DAVIS, 1988; ALEXANDER, 1991; PRESSMAN, 2001; CHRISTENSEN, 2001).

O modelo cascata apresenta as seguintes vantagens (CHRISTENSEN, 2001):

- A fase única de requisitos leva à especificação antes do projeto e ao projeto antes da codificação.
- O uso de revisões ao fim de cada fase permite o envolvimento do usuário.
- O modelo permite que se imponha um controle de configuração.
- Cada passo serve como uma base aprovada e documentada para o passo seguinte.

O modelo cascata, entretanto, apresenta as seguintes desvantagens (CHRISTENSEN, 2001):

- Projetos reais raramente seguem o fluxo seqüencial que o modelo propõe.
- Os requisitos devem ser estabelecidos de maneira completa correta e clara logo no início de um projeto.
- A aplicação deve ser entendida pelo desenvolvedor desde o início do projeto.
- É difícil avaliar o progresso verdadeiro do projeto durante as primeiras fases.
- Muitas vezes os desenvolvedores ficam ociosos desnecessariamente, devido a estados bloqueadores (quando existem tarefas dependentes, membros da equipe devem aguardar que outros terminem).
- Uma versão executável do software só fica disponível numa etapa avançada do desenvolvimento e não é possível demonstrar a capacidade do sistema até este momento.
- Ao final do projeto, é necessário um grande esforço de integração e testes.

2.2.1.2 O Modelo Incremental

Neste modelo os requisitos são segmentados em uma série incremental de produtos, que podem ser desenvolvidos com certa independência. O processo se repete até que um produto completo seja produzido. A segmentação dos requisitos é realizada

antes do desenvolvimento da primeira versão. É adotado quando os requisitos são conhecidos no início do desenvolvimento, e é desejável que alguns incrementos sejam completados o mais rápido possível. No modelo Incremental, a cada incremento é produzida uma versão operacional do software (DAVIS, 1988; ALEXANDER, 1991; PRESSMAN, 2001; CHRISTENSEN, 2001).

O modelo incremental apresenta as seguintes vantagens (CHRISTENSEN, 2001):

- Menor custo e menos tempo são necessários para se entregar a primeira versão.
- Os riscos associados ao desenvolvimento de incrementos são menores, devido ao seu tamanho reduzido.
- O número de mudanças nos requisitos pode diminuir devido ao curto tempo de desenvolvimento da primeira versão.

O modelo incremental apresenta as seguintes desvantagens (CHRISTENSEN, 2001):

- Se o requisitos não são tão estáveis ou completos quanto se esperava, alguns incrementos podem precisar ser retirados de uso e retrabalhados.
- O gerenciamento de custo, cronograma e configuração é mais complexo.

2.2.1.3 O Modelo Evolutivo

No modelo Evolutivo, são desenvolvidas versões parciais que atendem aos requisitos conhecidos inicialmente. A primeira versão é usada para refinar os requisitos para uma segunda versão. A partir do conhecimento sobre os requisitos, obtido com o uso, continua-se o desenvolvimento, evoluindo o produto (ALEXANDER, 1991; CHRISTENSEN, 2001).

O modelo evolutivo apresenta as seguintes vantagens (CHRISTENSEN, 2001):

- O modelo pode ser usado quando os requisitos não podem ser completamente especificados de início.
- O uso do sistema pode aumentar o conhecimento sobre o produto e melhorar os requisitos.

O modelo evolutivo apresenta as seguintes desvantagens (CHRISTENSEN, 2001):

- É necessária uma forte gerência de custo, cronograma e configuração.
- O usuário pode não entender a natureza da abordagem e se decepcionar quando os resultados não são satisfatórios.

2.2.1.4 O Modelo RAD

É um modelo seqüencial linear que enfatiza o desenvolvimento rápido. A "alta velocidade" é conseguida através de uma abordagem de construção baseada em várias equipes trabalhando em paralelo quando o produto pode ser dividido em módulos. O modelo RAD (*Rapid Application Development*) é usado quando os requisitos são bem definidos, o escopo do sistema é restrito e a aplicação pode ser modularizada. Cada grande função pode ser alocada para uma equipe distinta e, depois, são integradas para formar o todo (PRESSMAN, 2001).

O modelo RAD apresenta como principal vantagem a possibilidade de se ter um ciclo de desenvolvimento extremamente curto. Entretanto, apresenta as seguintes desvantagens (PRESSMAN, 2001):

- Para projetos escaláveis, mas grandes, o RAD requer recursos humanos suficientes para criar um número adequado de equipes.
- RAD requer um comprometimento entre desenvolvedores e clientes para que as atividades possam ser realizadas rapidamente e o sistema seja concluído em um tempo abreviado. Se o comprometimento for abandonado por qualquer das partes, o projeto pode falhar.
- Não é apropriado quando os riscos são grandes.
- Não é apropriado quando o sistema precisar interagir com outros sistemas, isto é, quando existir um alto grau de interoperabilidade.

2.2.1.5 Prototipação

Protótipos podem ser utilizados para explorar requisitos que serão implementados posteriormente em um incremento funcional. Além de seu papel na exploração de requisitos, protótipos podem ser utilizados para determinar a viabilidade técnica, de custo e de cronograma para o projeto (CHRISTENSEN, 2001).

Um protótipo deve ser submetido a uma avaliação, geralmente feita pelo cliente. O fato de o cliente poder interagir com um protótipo ajuda a cristalizar suas necessidades funcionais e de desempenho. Baseado no *feedback* do usuário, os desenvolvedores podem implementar os requisitos.

Protótipos descartáveis são criados de maneira rápida, sem preocupação com a qualidade do software ou com a manutenibilidade, e devem ser descartados quando atingirem seu propósito. Alguns autores defendem a utilização de protótipos operacionais, que levam em consideração a qualidade do software e constituem versões operacionais do software, passando por refinamentos sucessivos até que o produto esteja completo (DAVIS, 1988; BERSOFF, 1991).

Existem riscos envolvidos no uso da prototipação, tais como: os clientes imaginam que a maior parte do trabalho já foi feita; o protótipo pode crescer de maneira não planejada, se tornando um incremento funcional; o protótipo pode ter um desempenho melhor do que um incremento funcional, pois não implementa toda a funcionalidade, causando frustração aos clientes quando o sistema completo é entregue (PRESSMAN, 2001; BERSOFF, 1991).

2.2.1.6 Modelo Espiral

O modelo espiral, proposto por BOEHM (1988) é um modelo de ciclo de vida evolutivo que combina a natureza evolutiva da prototipação ao modelo seqüencial linear. No modelo espiral, o software é desenvolvido em uma série de versões incrementais, cada vez mais completas (PRESSMAN, 2001).

O modelo reflete o conceito de que cada ciclo envolve uma progressão que realiza a mesma sequência de passos, para cada porção do produto e para cada nível de elaboração, desde o documento de conceito de operação até a codificação.

Cada ciclo da espiral começa com a identificação dos objetivos da parte do produto que está sendo elaborada, as alternativas de implementação do produto e as restrições impostas pela aplicação das alternativas. O próximo passo é avaliar as alternativas, identificando possíveis riscos para o projeto. Após os riscos terem sido avaliados, um protótipo é planejado e desenvolvido.

Uma vez que os riscos tenham sido resolvidos pelos protótipos anteriores, o próximo passo segue o modelo cascata (ou um modelo iterativo, caso seja mais

apropriado). Cada ciclo possui uma etapa de validação e de planejamento do próximo ciclo.

A principal vantagem do modelo espiral é a sua variedade de opções, que acomodam as boas características dos demais modelos de ciclo de vida, evitando muitas de suas dificuldades, através da abordagem baseada em riscos. Se o projeto tem baixo risco em áreas como interface com o usuário e restrições de desempenho e tem alto risco em relação ao orçamento e cronograma, a espiral se torna equivalente ao modelo cascata. Se o projeto tem baixo risco em relação ao orçamento e cronograma e tem alto risco em relação à interface com o usuário, a espiral se torna equivalente ao modelo evolutivo (BOEHM, 1988).

2.2.1.7 Modelo de Ciclo de Vida Associado ao RUP

O *Rational Unified Process* (RUP) é um processo de engenharia de software desenvolvido pela Rational Software, que possui um *framework* de processo que pode ser adaptado e estendido para atender às necessidades de uma organização que o adote. O desenvolvimento de software é feito de forma iterativa, o que oferece as seguintes vantagens (KRUCHTEN, 1999):

- Esta abordagem permite e encoraja o *feedback* do usuário, elicitando os requisitos reais do sistema.
- Inconsistências entre requisitos, projeto e implementação podem ser detectados rapidamente.
- A carga de trabalho da equipe é mais bem dividida por todo o ciclo de vida.
- A equipe pode compartilhar lições aprendidas, melhorando continuamente o processo.
- É possível oferecer evidências concretas do andamento do projeto durante todo o ciclo de vida.

Algumas características importantes da abordagem são (KRUCHTEN, 1999):

- No RUP, a abordagem iterativa é muito controlada e iterações são planejadas em número, duração e objetivos.

- RUP é uma abordagem orientada a casos de uso, o que significa que os casos de uso definidos para o sistema são a base para o resto do processo de desenvolvimento e desempenham papel significativo em diversos processos.
- RUP é um *framework* de processo que a organização pode modificar, ajustar e expandir para acomodar suas necessidades, características, restrições, história, cultura e domínio específicos.

As fases neste processo não são a tradicional sequência de requisitos, análise, projeto, codificação, integração e teste, mas são ortogonais a estas, envolvendo (KRUCHTEN, 1999):

- Concepção: especificação da visão do produto e seus casos de negócio e definição do escopo do projeto.
- Elaboração: planejamento das atividades e recursos necessários; especificação das características e projeto da arquitetura.
- Construção: construção do produto e evolução da visão, arquitetura e planos, até que o produto esteja pronto para entrega.
- Transição: transição do produto para seus usuários.

As quatro fases constituem um ciclo de desenvolvimento e produzem um versão do software. Um produto de software é criado em um ciclo inicial de desenvolvimento, e pode evoluir para sua próxima versão por uma repetição da mesma sequência de fases de concepção, elaboração, construção e transição, mas com uma ênfase diferente em cada fase. São os chamados ciclos de evolução.

Em cada fase, o progresso é alcançado iterativamente e cada fase consiste em uma ou mais iterações. Cada iteração segue um padrão similar ao modelo cascata, contendo atividades de especificação de requisitos, análise, projeto, implementação, integração e teste, mas de uma iteração para outra e de uma fase para outra, a ênfase das várias atividades muda (KRUCHTEN, 1999).

- Na fase de concepção, o foco é principalmente entender os requisitos gerais e determinar o escopo do esforço de desenvolvimento.
- Na fase de elaboração, o foco está nos requisitos, mas algum projeto e implementação são realizados, visando prototipar a arquitetura, mitigar

riscos e aprender a usar certas ferramentas e técnicas. É produzido um protótipo executável da arquitetura que serve como base para a próxima fase.

- Na fase de construção, o foco está no projeto e implementação. O protótipo é evoluído para o primeiro produto operacional.
- Na fase de transição, o foco está em garantir que o sistema tem o nível correto de qualidade para atingir os objetivos; são feitas correções, treinamento de usuários, ajustes e adição de elementos que estavam faltando. O produto final é produzido e entregue.

2.2.2 Seleção de um Modelo de Ciclo de Vida

A seleção de um modelo de ciclo de vida para um projeto é parte integrante da definição de um processo de desenvolvimento de software e deve ser feita com base nas características desse projeto (MACHADO, 2000; CHRISTENSEN 2001). A seleção de um modelo de ciclo de vida inadequado pode resultar em um sistema que não satisfaz às necessidades dos usuários, em um aumento do custo e maior tempo de desenvolvimento (ALEXANDER, 1991).

Para selecionar um modelo adequado ao projeto é necessário considerar as características do próprio projeto, da equipe de desenvolvimento e gerência, e dos próprios usuários. FALBO (1998) apresenta uma proposta para seleção do modelo de ciclo de vida baseada os seguintes critérios: paradigma de desenvolvimento adotado; necessidade do software ser colocado em uso rapidamente com funcionalidade total ou parcial; grau de definição do problema; tamanho e complexidade da aplicação; e nível de formação, atualização e experiência da equipe de desenvolvimento e de gerência.

ALEXANDER e DAVIS (1991) propõem 20 critérios específicos para aplicações e projetos que descrevem perfis de projetos adequados a cada modelo de ciclo de vida. Tais critérios são valorados e classificados em cinco categorias: critérios pessoais, do problema, do produto, de recursos e organizacionais A seleção de um modelo de ciclo de vida pode ser realizada criando um perfil para o projeto atual e comparando-o aos perfis dos modelos de ciclo de vida descritos. O modelo de ciclo de vida cujo perfil mais se assemelha ao perfil do projeto deve ser selecionado. O gerente de projeto deve ter um claro entendimento do projeto em questão para que o perfil possa ser criado corretamente.

MACHADO (2000) propôs uma combinação das abordagens de ALEXANDER e DAVIS (1991) e FALBO (1998) para seleção do modelo de ciclo de vida. Foram adotados 17 critérios adotados para a escolha de um modelo de ciclo de vida, cujos valores foram definidos a partir de uma adaptação do trabalho de ALEXANDER e DAVIS (1991).

2.3 Definição de Processos de Software

Com a preocupação mundial com processos de software, vêm surgindo uma série de padrões e modelos internacionais que têm como objetivo definir, avaliar e melhorar a qualidade dos processos de software. As organizações passaram a adotar tais padrões e modelos na definição de processos de software, buscando, assim, melhorar a qualidade de seus produtos, aumentar a produtividade de suas equipes e reduzir os custos e os riscos associados com o desenvolvimento de software (MACHADO, 2000).

2.3.1 A Norma ISO/IEC 12207 - Tecnologia de Informação - Processos de Ciclo de Vida de Software

A Norma Internacional ISO/IEC 12207 – Tecnologia da Informação – Processos de Ciclo de Vida de Software (ISO/IEC 12207, 1995) tem por objetivo auxiliar os envolvidos com a produção de software na definição de seus papéis, através de processos bem definidos, e desta forma proporcionar para as organizações que a utilizam um melhor entendimento das atividades a serem executadas nas operações que envolvem, de alguma forma, o software (ROCHA *et al.*, 2001).

A Norma classifica tais processos em: Processos Fundamentais, Processos de Apoio, Processos Organizacionais e Processos de Adaptação. Este último, em particular, trata exatamente da cultura e dos aspectos específicos da organização. Cada processo é definido em termos de suas próprias atividades e cada atividade é adicionalmente definida em termos de suas tarefas. Uma organização pode selecionar um subconjunto apropriado de processos, de acordo com seus objetivos, pois a Norma foi projetada para ser adaptada para uma organização, projeto ou aplicação específicos .

2.3.1.1 Processos Fundamentais

Os processos fundamentais constituem um conjunto de cinco processos que atendem as partes (pessoa ou organização) fundamentais durante o ciclo de vida do

software. Uma parte fundamental é aquela que inicia ou executa o desenvolvimento, operação ou manutenção dos produtos de software. Os processos fundamentais são: Processo de Aquisição, Processo de Fornecimento, Processo de Desenvolvimento, Processo de Operação e Processo de Manutenção.

O processo de desenvolvimento define as atividades do desenvolvedor, organização que define e desenvolve o produto de software. Este processo consiste nas seguintes atividades: (i) implementação do processo; (ii) análise dos requisitos do sistema; (iii) projeto da arquitetura do sistema; (iv) análise dos requisitos do software; (v) projeto da arquitetura do software; (vi) projeto detalhado do software; (vii) codificação e testes do software; (viii) integração do software; (ix) teste de qualificação do software; (x) integração do sistema; (xi) teste de qualificação do sistema; (xii) instalação do software; (xiii) apoio à aceitação do software.

2.3.1.2 Processos de Apoio

Os processos de apoio constituem um conjunto de oito processos. Um processo de apoio auxilia um outro processo como parte integrante, com um propósito distinto, contribuindo para o sucesso e qualidade do projeto de software. Um processo de apoio é empregado por outro processo, quando necessário. São eles: Processo de Documentação, Processo de Gerência de Configuração, Processo de Garantia da Qualidade, Processo de Verificação, Processo de Validação, Processo de Revisão Conjunta, Processo de Auditoria e Processo de Resolução de Problemas.

2.3.1.3 Processos Organizacionais

Constituem um conjunto de quatro processos, que são empregados por uma organização para melhorar continuamente a sua estrutura e os seus processos. São empregados tipicamente fora do domínio de projetos e contratos específicos; entretanto, ensinamentos desses projetos e contratos contribuem para a melhoria da organização. São eles: Processo de Gerência, Processo de Infra-estrutura, Processo de Melhoria e Processo de Treinamento .

2.3.1.4 Processo de Adaptação

O processo de adaptação define as atividades básicas necessárias para executar a adaptação da Norma para sua aplicação na organização ou em projetos. A adaptação deve ser executada baseando-se em alguns fatores que diferenciam uma organização ou projeto de outros, dentre os quais, a estratégia de aquisição, modelos de ciclo de vida de

projeto, características de sistemas de software e cultura organizacional. A existência desse processo permite que a Norma seja adaptável a qualquer projeto, organização, modelo de ciclo de vida, cultura e técnica de desenvolvimento (ROCHA *et al.*, 2001).

2.3.2 ISO/IEC TR 15504

ISO/IEC TR 15504 (ISO/IEC TR 15504, 1998) é o padrão internacional para avaliação de processos de software. Foi publicada inicialmente em 1998 como Relatório Técnico, estando atualmente em processo de revisão, e será publicado como norma ISO/IEC 15504 em 2003/2004. O desenvolvimento da ISO/IEC 15504 vem acontecendo em paralelo com estudos experimentais de seu uso realizados pelo projeto SPICE. Mais de 3000 avaliações foram realizadas em todo o mundo usando a ISO/IEC 15504.

A ISO/IEC TR 15504 (ISO/IEC TR 15504, 1998) fornece um *framework* para a avaliação de processos de software, que pode ser utilizada por organizações envolvidas no planejamento, gerência, monitoração, controle e melhoria da aquisição, fornecimento, desenvolvimento, operação, evolução e suporte de software.

A ISO/IEC TR 15504 provê uma abordagem estruturada para avaliação de processos de software com os seguintes objetivos: (i) permitir o entendimento, por uma organização, do estado dos seus processos, visando estabelecer melhorias; (ii) determinar a adequação dos processos de uma organização para atender a um requisito particular ou classe de requisitos; (iii) determinar a adequação de processos da organização para um contrato ou classe de contratos.

2.3.2.1 O Modelo de Referência

A ISO/IEC TR 15504 define um modelo de referência para processos de software que forma a base para a avaliação. O modelo de referência é aplicável a qualquer organização de software que deseje estabelecer e melhorar sua capacidade de aquisição, fornecimento, desenvolvimento, operação, evolução e suporte de software. O modelo não determina uma estrutura organizacional, filosofia gerencial, modelos de ciclo de vida de software, tecnologias de software ou metodologias de desenvolvimento.

O modelo de referência, definido pela ISO/IEC TR 15504 possui duas dimensões:

- Dimensão de processo, caracterizada pelos propósitos do processo, que são os objetivos essenciais mensuráveis de um processo;
- Dimensão de capacitação do processo, caracterizada por uma série de atributos de processo, aplicáveis a qualquer processo, que representam características mensuráveis necessárias para gerenciar um processo e melhorar sua capacidade.

- **A Dimensão de Processo**

Os processos de software previstos na ISO/IEC TR 15504 são classificados em cinco categorias, de acordo com o tipo de atividade executada. A classificação é apresentada na tabela 2.1:

Tabela 2.1 - Classificação dos processos de software na ISO/IEC TR 15504

Categoria de Processo	Descrição
Cliente-Fornecedor	Processos que impactam diretamente o cliente, apóiam o desenvolvimento e transição do software para o cliente e proporcionam a correta operação e uso do produto ou serviço de software.
Engenharia	Processos que diretamente especificam, implementam ou mantêm o produto de software, sua relação com o sistema e a documentação para o cliente.
Apoio	Processos que podem ser utilizados por quaisquer outros processos (incluindo os próprios processos de apoio) em vários pontos do ciclo de vida do software.
Gerência	Processos que contêm práticas de natureza genérica que podem ser utilizadas por quem gerencia qualquer tipo de projeto ou processo que contenha um ciclo de vida de software.
Organização	Processos que estabelecem os objetivos da organização e desenvolvem processos, produtos e recursos que auxiliam no alcance destes objetivos.

Uma emenda à norma estende o conjunto de processos, criando uma categoria de processo adicional, a categoria Adquirente, que contém processos que impactam diretamente o adquirente.

- **A Dimensão de Capacitação**

A dimensão de capacitação define uma escala de seis níveis para medição da capacidade de qualquer processo definido no modelo de referência. Cada nível representa um avanço no desempenho do processo, estabelecendo um caminho natural para a melhoria do processo.

A determinação da capacitação é realizada por um conjunto de atributos de processos que medem se foi atingido um determinado aspecto da capacitação. Um atributo é uma característica mensurável da capacitação de um processo aplicável a todos os processos. Para cada atributo são associados uma descrição e um conjunto de características específicas. Cada atributo é avaliado em uma escala de 0 a 100%, representando o nível em que é atendido pelo processo.

A ISO/IEC TR 15504 sugere um modelo de avaliação compatível com os requisitos definidos no modelo de referência, mas qualquer outro modelo de avaliação pode ser utilizado, desde que atenda aos requisitos do modelo de referência. Os níveis de capacitação previstos na ISO/IEC TR 15504 são apresentados na tabela 2.2:

Tabela 2.2 - Níveis de capacitação previstos na ISO/IEC TR 15504

Nível	Nome	Descrição
0	Incompleto	O processo falha em atingir seu objetivo, produzindo pouco ou nenhum resultado.
1	Executado	O objetivo do processo é atingido de maneira geral, mas o processo pode não ser rigorosamente planejado ou acompanhado. O processo produz resultados que demonstram que o objetivo foi atingido.
2	Gerenciado	O processo produz resultados de acordo com procedimentos definidos e é planejado e acompanhado. Os resultados estão em conformidade com os padrões e requisitos.
3	Estabelecido	O processo é definido, executado e gerenciado de acordo com princípios da engenharia de software. Implementações individuais do processo utilizam versões aprovadas e customizadas de processos padrão documentados. Os recursos necessários para definição do processo estão disponíveis.
4	Previsível	O processo definido é executado consistentemente na prática, dentro de limites definidos para alcançar os resultados esperados. Medidas detalhadas de desempenho são coletadas e analisadas. Isto leva a um entendimento quantitativo da capacidade do processo e a uma melhor habilidade de prever e gerenciar o desempenho. O desempenho é gerenciado quantitativamente. A qualidade dos produtos é conhecida quantitativamente.
5	Otimizado	O desempenho do processo é otimizado para atingir aos objetivos de negócio atuais e futuros, e o processo atinge seus objetivos de negócio repetidamente. Objetivos quantitativos para o desempenho do processo são estabelecidos, baseados nos objetivos de negócio da organização. Monitoração constante do processo é obtida através de feedback quantitativo e a melhoria é atingida através da análise dos resultados.

2.4 CMM e CMMI

O CMM (*Capability Maturity Model*) foi criado pelo *Software Engineering Institute* (SEI) em resposta à falta de maturidade do processo de desenvolvimento de software, cujas conseqüências mais visíveis são a baixa qualidade e confiabilidade da maioria dos produtos de software atuais (BATISTA, 2000). A seguir, apresentamos o modelo do CMM para software e sua evolução, o CMMI (*CMM Integration*).

2.4.1 *Capability Maturity Model* (CMM)

O CMM é estruturado em cinco níveis de maturidade, de 1 a 5, onde o nível 1 é o menos maduro e o nível 5 é o mais maduro. A obtenção de um nível mais alto pode ser usado como demonstração da capacidade do processo. Cada nível de maturidade, com exceção do nível 1, é composto de várias áreas-chave de processo (*key process areas* - KPAs). Cada KPA é organizada em cinco características comuns, que especificam práticas-chave que, quando executadas em conjunto, alcançam os objetivos das KPAs (LEUNG et al., 2001).

Cada KPA indica o que deve ser realizado para alcançar um nível de maturidade. Uma organização que se encontre em determinado nível de maturidade deve realizar todas as KPAs deste nível, como também aquelas dos níveis inferiores. Para uma KPA ser satisfeita, todos os seus objetivos devem ser alcançados. A seguir, são apresentados os níveis de maturidade do CMM (PAULK *et al.*, 1997).

- **Nível 1: Inicial**

O processo de software é caracterizado como *ad hoc* e, eventualmente, caótico. Poucos processos são definidos e o sucesso depende de esforços individuais.

- **Nível 2: Repetível**

Os processos básicos de gerenciamento são estabelecidos para acompanhar custo, cronograma e funcionalidade. Os sucessos em projetos anteriores com aplicações similares são repetidos.

- **Nível 3: Definido**

O processo de software em relação às atividades de gerenciamento e desenvolvimento é documentado, padronizado e integrado em um processo de software

padrão para a organização. Todos os projetos utilizam uma versão aprovada e adaptada do processo padrão para desenvolvimento e manutenção de software.

- **Nível 4: Gerenciado**

São coletadas medições detalhadas do processo de software e da qualidade do produto. Tanto o processo de software como os produtos são quantitativamente entendidos e controlados.

- **Nível 5: Otimizado**

Melhorias contínuas nos processos são estabelecidas pelo retorno quantitativo dos processos e conduzidas pela introdução de idéias e tecnologias inovadoras.

O CMM visa a melhoria incremental de processos de software existentes. O nível Otimizado (nível 5) caracteriza as organizações cujos processos de software são melhorados de forma incremental e refinados através de monitoração, medição e análise de processos bem definidos e administrados. Entretanto, o CMM não oferece uma indicação específica sobre como melhorar processos de software, nem como definir novos processos (SCACCHI, 2000).

A figura 2.1 apresenta os níveis do CMM e suas áreas-chave.

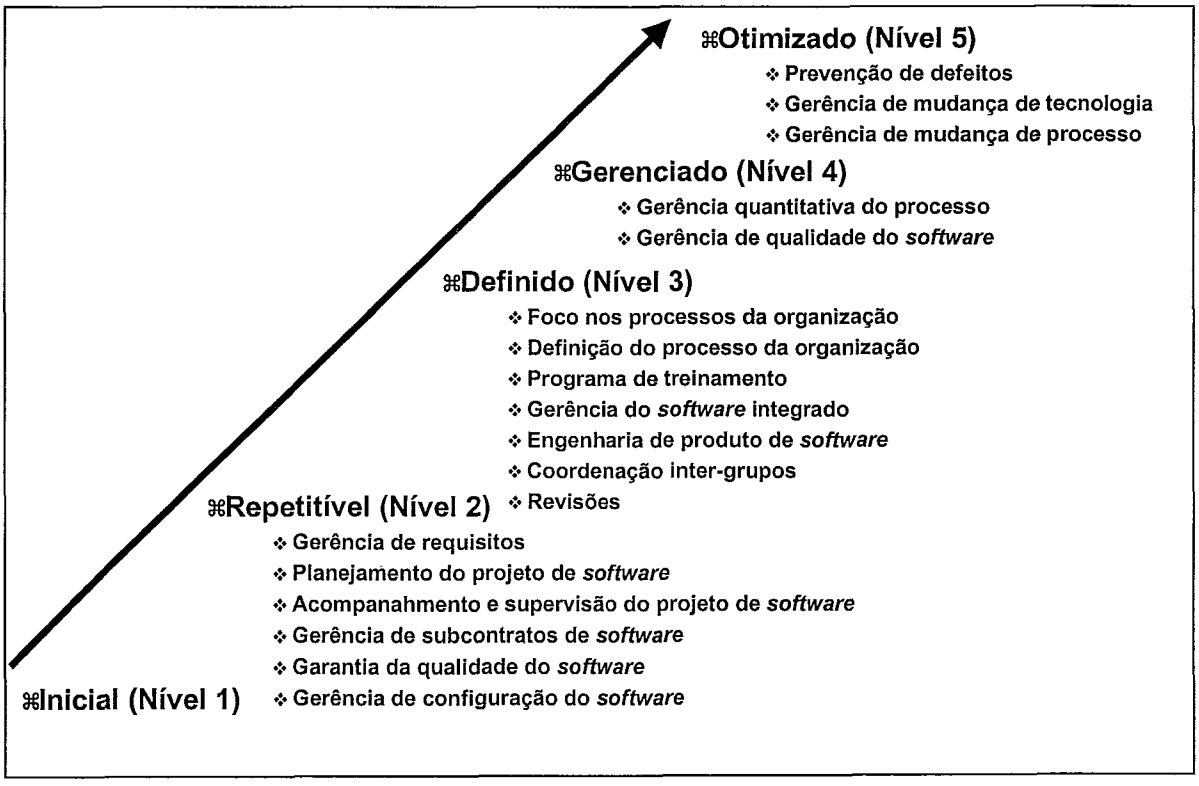


Figura 2.1 - Áreas-chave por nível CMM

2.4.2 *CMM Integration* (CMMI)

O modelo SEI-CMM tornou-se, ao longo de uma década de uso por organizações de software de diversos tamanhos e setores, um modelo de maturidade de processos conhecido, usado e respeitado. Baseado no sucesso do SW-CMM (CMM para software), outros modelos foram criados, procurando cobrir outras áreas de interesse. Assim surgiram os seguintes modelos (SEI, 2002):

- *Software Acquisition CMM* (SA-CMM): usado para avaliar a maturidade de uma organização em seus processos de seleção, compra e instalação de software desenvolvido por terceiros.
- *Systems Engineering CMM* (SE-CMM): avalia a maturidade da organização em seus processos de engenharia de sistemas, concebidos como algo maior que o software. Um sistema inclui o hardware, o software e quaisquer outros elementos que participam do produto completo.
- *Integrated Product Development CMM* (IPD-CMM): ainda mais abrangente que o SE-CMM, inclui também outros processos necessários à produção e suporte ao produto, tais como suporte ao usuário, processos de fabricação etc.
- *People CMM* (P-CMM): avalia a maturidade da organização em seus processos de administração de recursos humanos no que se refere a software; recrutamento e seleção de desenvolvedores, treinamento e desenvolvimento, remuneração etc.

Embora esses modelos tenham se mostrado úteis para muitas organizações, o uso de múltiplos modelos gerou alguns problemas, devido às diferenças de arquitetura, conteúdo e abordagem. A aplicação de diversos modelos não integrados em uma organização aumenta os custos de treinamento, das avaliações e das atividades de melhoria.

Por todas estas razões, o SEI iniciou um projeto chamado CMMI (*CMM Integration*), com o objetivo de apoiar a melhoria de processos e produtos diminuindo a redundância e eliminando a inconsistência que surge ao se utilizar diversos modelos independentes. Esse objetivo é atingido através da integração dos diversos CMMs numa estrutura única, todos com a mesma terminologia, processos de avaliação e estrutura. O

projeto também se preocupou em tornar o CMM compatível com a norma ISO 15504, de modo que avaliações em um modelo sejam reconhecidas como equivalentes aos do outro (SEI, 2002).

O CMMI oferece duas abordagens diferentes para a melhoria de processos. Estas duas abordagens são conhecidas como o "modelo contínuo" e o "modelo em estágios".

O SW-CMM é um modelo em estágios. Existem cinco níveis de maturidade e a organização é avaliada como estando em apenas um deles. O modelo da norma ISO/IEC TR 15504 usa um modelo contínuo. Neste caso, cada área-chave de processo possui características relativas a mais de um nível. Assim, uma área-chave que, no modelo em estágios, pertence exclusivamente ao nível 2, no modelo contínuo pode ter características que a coloquem em outros níveis. No modelo contínuo, cada área chave é classificada separadamente, de modo que a organização pode ter áreas no nível 1, outras no nível 2, ainda outras no nível 3 e assim por diante.

Para dar liberdade a organizações com realidades e necessidades distintas, o CMMI admite as duas abordagens. Ambas contém essencialmente as mesmas informações e a opção pelo modelo contínuo ou em estágios depende de cada organização. Cada modelo possui características que o tornam mais apropriado em uma situação ou outra (SEI, 2002).

O modelo em estágios oferece um caminho para melhoria de processos, indicando a ordem de implementação para cada área de processo, de acordo com os níveis de maturidade. Essa abordagem minimiza os riscos da melhoria de processos. A representação é indicada para organizações realmente comprometidas com a implantação do CMM em escala geral.

O modelo contínuo oferece uma abordagem mais flexível para a melhoria de processos, embora mais complexo de administrar. É indicado para organização que desejam dar prioridade à melhoria de uma área de processo ou conjunto de processos, de acordo com seus objetivos de negócio. Este modelo permite fácil comparação à ISO/IEC TR 15504, porque a organização das áreas de processo é derivada desta norma.

2.5 Processo Padrão, Processos Especializados e Processos Instanciados

Um elemento chave das normas e modelos de maturidade é a definição de um processo padrão que descreve as atividades que devem ser realizadas no desenvolvimento de sistemas de software em todos os projetos de uma organização. O uso de modelos de processo genéricos como base para o planejamento do processo de software específico para um projeto permite aos gerentes definir planos em conformidade com os padrões de qualidade e procedimentos da organização (MAURER *et al.*, 1999; HENNINGER, 2001).

Sobre a definição de processos de software, OLIVEIRA (1999) destaca as seguintes premissas:

- Nenhum projeto é igual ao outro e nenhuma organização é igual a outra. Desta forma, a norma ISO 12207, que estabelece os processos do ciclo de vida de um software, foi definida para um projeto genérico e deve ser adaptada para organizações e projetos específicos. Para a adaptação, é necessário identificar as características do ambiente do projeto.
- É preciso considerar a cultura da organização, o modo como ela desenvolve software e o problema crucial que é enfrentado em seus projetos de desenvolvimento de software, contemplando-os no processo.
- Como diferentes tipos de software (sistemas de informação, sistemas baseados em conhecimento, aplicações multimídia, etc.) podem ser desenvolvidos para um mesmo domínio e em uma mesma organização, torna-se, então, necessário definir diversos processos de desenvolvimento de software adequados para a construção dos diferentes tipos de software.
- Características do projeto, como requisitos, tamanho, criticalidade e quantidade de pessoas e partes envolvidas, também devem ser considerados em um processo de desenvolvimento de software para um projeto específico.
- Definir vários processos de software para uma mesma organização pode ser muito trabalhoso e ter como resultado processos completamente distintos, o que não é adequado. A padronização dos processos de software dentro de uma organização pode ser obtida através da definição de um Processo Padrão

para a organização, que serve de base para a definição dos processos de software adequados aos diferentes tipos de software e, a partir do qual, processos para projetos específicos podem ser definidos.

- Devido à alta complexidade da atividade de definição de processos de software, torna-se necessário oferecer apoio às pessoas responsáveis por ela, em geral, gerentes de projeto, que não necessariamente possuem grande conhecimento de engenharia de software. Para apoiá-los, portanto, é necessário tornar explícito o conhecimento de pessoas com experiência na definição de processos de software, utilizando-se técnicas de gerência de conhecimento.

Este tipo de esforço de padronização sofre, portanto, com um problema inerente: para acomodar todos os tipos de iniciativas de desenvolvimento em uma organização, o padrão vai inevitavelmente estar em um nível de abstração que atenda às necessidades de todos os projetos, mas não vai ser capaz de fornecer apoio específico às atividades individuais do projeto. A diversidade de projetos de tecnologia da informação frustra qualquer tentativa direta de sistematizar os processos usados para seu desenvolvimento. Não existe um único formato que seja adequado para efetivamente apoiar a realização de todos os projetos. É necessária uma abordagem mais flexível e configurável para definição do processo, uma maneira de adaptar o processo às necessidades específicas de cada projeto (HENNINGER, 1999; HENNINGER, 2001; CAMERON, 2002).

A Norma ISO/IEC 12207 reconhece esta necessidade no processo de adaptação e ao tratar da implementação do processo de desenvolvimento. Esta Norma é, portanto, flexível para diversas abordagens de engenharia de software, sendo utilizável com qualquer modelo de ciclo de vida, qualquer método ou técnica de engenharia de software e qualquer linguagem de programação. Estas questões são muito dependentes do projeto e do estado da arte da tecnologia, portanto estas escolhas são deixadas a critério dos usuários da Norma (ROCHA *et al.*, 2001).

Ao tratar da atividade de implementação do processo, a Norma ISO/IEC 12207 inclui na mesma as seguintes tarefas:

- Definir ou selecionar um modelo de ciclo de vida de software, apropriado às características do projeto e mapear as atividades e tarefas do processo de desenvolvimento na estrutura deste modelo de ciclo de vida.

- Selecionar, adaptar e utilizar padrões, métodos, ferramentas e linguagens de programação apropriados para executar as atividades do processo de desenvolvimento e de apoio.
- Desenvolver planos para conduzir as atividades do processo de desenvolvimento, incluindo padrões, métodos, ferramentas, ações e responsabilidades associados com o desenvolvimento.

CHRISTENSEN (2001) descreve os passos para realizar este mapeamento:

1. Comparar atividades ao modelo e identificar quais atividades não são necessárias para o modelo de ciclo de vida e requisitos do projeto. Este passo deve garantir que todas as atividades necessárias ao projeto sejam consideradas.
2. Identificar quantas instâncias de cada atividade são apropriadas. Caso o modelo de ciclo de vida selecionado seja iterativo, uma determinada atividade poderá ser realizada diversas vezes, de acordo com o número de iterações do modelo, enquanto no modelo cascata a mesma atividade será realizada uma única vez.
3. Colocar as atividades em sequência e checar o fluxo de informações. As dependências entre as atividades devem ser respeitadas, para garantir que o processo será realizado corretamente. Neste ponto ainda não é necessário atribuir datas para a realização das atividades.

Para a elaboração de um processo padrão para a organização, é preciso dispor de conhecimento sobre as atividades de desenvolvimento e manutenção de software que são sempre realizadas na organização independente de um projeto específico. A definição do processo padrão considera as características de desenvolvimento de software na organização. Considerando que os principais objetivos das organizações produtoras de software são definir, utilizar e estabelecer melhorias contínuas nos seus processos de software, torna-se fundamental, para alcançar estes objetivos, que tais organizações direcionem seus esforços não só para a aplicação de métodos e práticas de engenharia de software, como também passem a considerar características relacionadas ao ambiente de trabalho, ao conhecimento e experiência das equipes envolvidas, e à própria cultura e experiência da organização em desenvolver software (MACHADO, 2000).

A partir do Processo Padrão definido para a organização, diferentes processos de software podem ser especializados de acordo com as características de cada paradigma de desenvolvimento, podendo ser incluídas novas atividades, mas mantendo os elementos básicos definidos no Processo Padrão.

Para ser utilizado em um projeto específico, o processo especializado mais adequado deve ser instanciado para atender às características do projeto específico, devendo-se considerar o tamanho e complexidade do produto, as características da equipe de desenvolvimento, a expectativa de vida útil do software e demais características do projeto (MACHADO, 2000; BORGES, 2001).

A adaptação do processo às necessidades específicas de projetos requer um processo pelo qual desenvolvedores e gerentes sejam capazes de instanciar um processo adequado ao projeto em questão. Durante o planejamento de um novo projeto o gerente identifica e analisa as características principais deste projeto, que são usadas como base para a adaptação do padrão, gerando um processo instanciado para o projeto. Isso inclui a seleção de atividades e métodos do modelo de processo genérico que devem fazer parte do processo instanciado. Implica, também, na seleção de um modelo de ciclo de vida e no mapeamento das atividades do processo para este modelo (HENNINGER, 1999; KRUCHTEN, 1999; MAURER *et al.*, 1999; NAGEL, 2001; HOLZ *et al.*, 2001, OLIVEIRA, 1999, MACHADO, 2000, BORGES, 2001).

É uma tarefa difícil abstrair o modelo de processo a partir de um padrão e, então, instanciá-lo para um projeto específico. Em uma situação típica, a pessoa responsável pela definição do processo parte do processo padrão e mentalmente o mapeia para o modelo de ciclo de vida desejado. Durante ou depois da definição deste processo, é feita uma verificação manual dos processo, para garantir sua conformidade com o padrão. Este processo leva a um grande consumo de recursos (PURPER, 2000).

2.6 Conclusão

Um processo não deve ser seguido cegamente, gerando trabalho desnecessário e produzindo artefatos sem valor. O processo deve ser o mais simples possível, cumprindo sua missão de produzir rapidamente software de qualidade previsível. (KRUCHTEN, 1999).

O processo deve se tornar um recurso valioso em que desenvolvedores de software e gerentes de projeto possam confiar para apoio crítico e informações. A descrição do processo de software pode ser usada para gerar planos de projeto, apoiar estimativas de esforço e obter informações sobre como realizar as atividades, entre outros (HENNINGER, 2001; NAGEL, 2001).

A criação de metodologias padrão de desenvolvimento é território familiar para muitas organizações que já investiram recursos em padrões que não foram usados. Uma das razões mais citadas para não seguir o padrão da organização é que ele não atende às necessidades do processo. Para acomodar necessidades diversas, o padrão é descrito em um nível muito alto e não tem o nível de detalhe que pode realmente guiar um projeto. (HENNINGER, 1999; HENNINGER, 2001).

Para resolver estes problemas é necessário haver uma adaptação do processo padrão genérico para obtenção de um processo específico para o projeto. Adaptar para circunstâncias específicas deve ser a regra e deve ser parte integral de qualquer método e da maneira como ele é ensinado (NAGEL, 2001; CAMERON, 2002).

HENNINGER (2001) afirma, ainda, que, embora os *frameworks* de processo exijam o uso e manutenção de padrões, é de importância crítica que ferramentas e metodologias sejam criadas para apoiar o desenvolvimento, uso, evolução e adaptação do padrão para atender às necessidades dinâmicas das organizações de desenvolvimento de software modernas.

OLIVEIRA (1999) discute várias premissas relacionadas à definição de processos de software e, entre elas, a necessidade de apoiar esta definição utilizando técnicas de gerência do conhecimento. No próximo capítulo tratamos de aspectos relacionados à gerência do conhecimento.

CAPÍTULO III - GERÊNCIA DO CONHECIMENTO E ENGENHARIA DE SOFTWARE

3.1 Introdução

O desenvolvimento de software é uma atividade fortemente baseada em pessoas e conhecimento; é um campo com modificações constantes e, embora esteja lentamente amadurecendo, muitas atividades ainda são realizadas de maneira *ad hoc* e dependem de experiência pessoal. Devido ao rápido crescimento de organizações, o número de pessoas com experiência extensiva em projetos é pequeno e insuficiente. Atualmente, habilidades em gerência de projetos são um fator crucial nas contratações e estudos mostram que especialistas neste campo são muito procurados (BRANDT et al., 2001). Para lidar com estas restrições à medida que prazos e orçamentos diminuem, organizações de desenvolvimento de software precisam de assistência para realizar projetos cada vez mais críticos.

Para atingir seus objetivos, equipes de desenvolvimento de software precisam compreender e escolher os modelos e técnicas corretos para apoiar seus projetos. Em alguns casos existe conhecimento para apoiar estas decisões; em outros casos não. Assim, ao invés de se basear em conhecimento e experiência, os desenvolvedores passam a confiar apenas em sua intuição. Para oferecer apoio a esta atividade de tomada de decisão, devem ser desenvolvidos modelos de software baseados em experiências, cobrindo todos os aspectos, desde modelos de ciclo de vida de alto nível até técnicas de baixo nível, nos quais os efeitos do processo de decisão sejam bem entendidos (BASILI *et al.*, 2001b).

Nesse contexto, ganha importância a gerência de conhecimento, disciplina que visa elevar o conhecimento individual ao nível organizacional, capturando e compartilhando o conhecimento individual e transformando-o em conhecimento que a organização pode acessar (RUS *et al.*, 2002; KUCZA *et al.*, 2001).

A gerência do conhecimento é inerentemente inter-disciplinar, envolvendo gerência de recursos humanos e cultura organizacional, além de métodos e ferramentas da tecnologia da informação. A gerência do conhecimento pode melhorar a

produtividade de uma organização ao integrar aspectos tecnológicos com os aspectos humano e organizacional (O'LEARY *et al.*, 2001a).

SKYRME (1999) apresenta um *framework* para gerência do conhecimento formado por três níveis. No nível superior do *framework* estão os capacitadores, cujo principal papel é a liderança organizacional. A estrutura, cultura e ambiente da organização devem encorajar o desenvolvimento e compartilhamento de conhecimento. Sem esses capacitadores a maioria das iniciativas em conhecimento fracassa. O segundo nível do *framework* é composto por um conjunto de alavancas que facilitam a utilização do conhecimento. Entre elas estão incluídos processos que facilitam o fluxo de conhecimento e a manipulação eficaz das informações. O terceiro nível é composto pela infra-estrutura de comunicação que apóia a colaboração e pela infra-estrutura que desenvolve conhecimento através de habilidades e comportamento. Neste nível, o desenvolvimento rápido da tecnologia oferece um número crescente de ferramentas para captura, organização e compartilhamento do conhecimento (SKYRME, 1999).

Neste capítulo é feita uma discussão de vários tópicos relacionados à gerência do conhecimento. Na seção 2 é discutida a definição de conhecimento e suas formas de representação. A seção 3 trata do aprendizado organizacional e memória organizacional. Na seção 4 serão discutidas as atividades e tecnologias da gerência do conhecimento. A seção 5 apresenta a aplicação da gerência de conhecimento na área de Engenharia de Software. A seção 6 apresenta a conclusão do capítulo.

3.2 Conhecimento

Não existe uma definição universalmente aceita de conhecimento. BIGGAM (2001) estabelece três critérios subjetivos para o conhecimento:

- Ele deve ser verdadeiro;
- O observador deve acreditar que é verdadeiro;
- O observador deve estar em posição de saber que é verdadeiro.

Uma meta buscada pela gerência do conhecimento é a transformação de **dados** em **informação**, e da **informação** em **conhecimento**. Isto se baseia na crença de que obter conhecimento permite às organizações obterem vantagens competitivas e tornarem-se mais eficientes e produtivas (SAUNDERS, 2000).

Sobre a transformação de dados em informação, e da informação em conhecimento, SNOEK (1999) oferece a seguinte explicação. No nível mais baixo, existem caracteres, que não possuem significado algum sem uma sintaxe associada. Ao adicionarmos a sintaxe, os caracteres se transformam em dados. Dependendo do contexto no qual estão inseridos, os dados podem ser interpretados como informação. Se essa informação estiver ligada a uma rede de informações, ela se transforma em conhecimento.

Podemos, então, definir dados, informação e conhecimento da seguinte maneira (SNOEK, 1999 e SAUNDERS, 2000):

- Dados – fatos individuais fora de contexto.
- Informação – dados que são comunicados e possuem um contexto baseado nas intenções e percepções dos interlocutores.
- Conhecimento – um corpo organizado de verdades ou fatos conectados, entendido claramente e com certeza.

Com conhecimento, dados podem ser interpretados como informação. Novas informações podem ser desenvolvidas através da elaboração de informações existentes, também com o uso do conhecimento. Através do aprendizado (que é uma transformação baseada em conhecimento), a informação se transforma em conhecimento. Tal processo de aprendizado baseado em conhecimento pode obter novo conhecimento a partir de conhecimento existente (SNOEK, 1999).

3.2.1 Representação do Conhecimento

O conhecimento não deve se perder, portanto, ele deve estar em um formato que possa ser armazenado. Tal formato é chamado de representação explícita. Entretanto o conhecimento freqüentemente não está disponível em uma representação explícita, e sim sob a forma de conhecimento tácito (SNOEK, 1999).

SKYRME (1999) faz a seguinte distinção entre conhecimento tácito e explícito: conhecimento explícito é aquele que pode ser codificado em documentos, bases de dados e outras formas tangíveis; conhecimento tácito, por sua vez, existe nas mentes dos indivíduos, é pessoal e difícil de ser transmitido. A manipulação de conhecimento explícito se baseia em processos sistemáticos de manipulação de informação, enquanto a manipulação de conhecimento tácito envolve pessoas e o ambiente em que estas

trabalham. O benefício de conhecimento explícito é que ele pode ser armazenado e disseminado sem o envolvimento de seu originador, mas é necessário esforço considerável para produzi-lo (SKYRME ,1999; RUS *et al.*, 2002).

Este fato leva a um problema central de memória organizacional ou de gerência do conhecimento: o conhecimento tácito deve ser transformado em uma representação explícita que será armazenada para recuperação futura. Todos os tipos de transformações entre representações de conhecimento são possíveis: de tácito para explícito, tácito para tácito, explícito para explícito e explícito para tácito, como mostra a figura 3.1 (SNOEK, 1999).

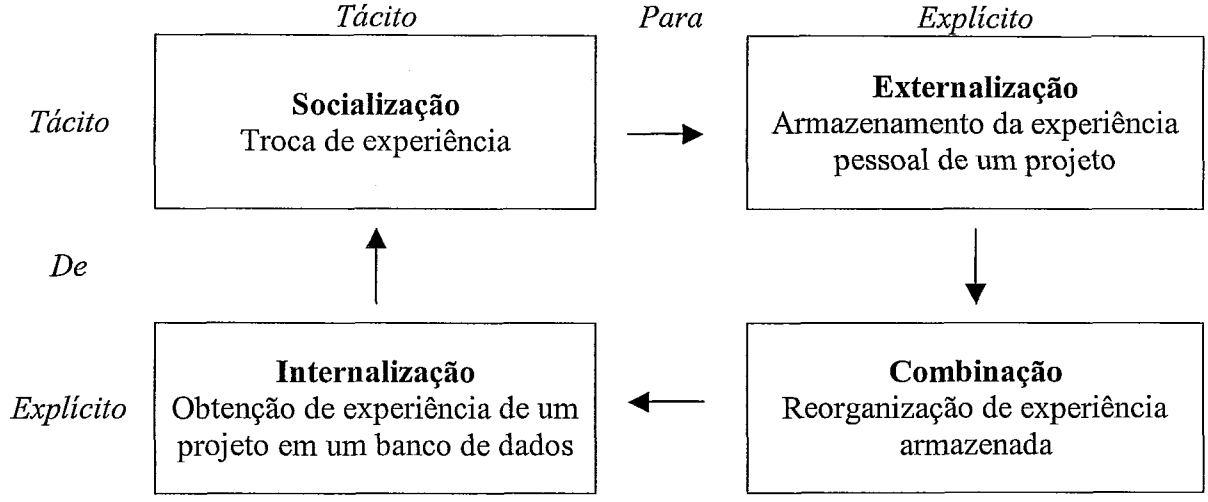


Figura 3.1: Tipos possíveis de transformações de representação de conhecimento (SNOEK, 1999)

3.3 Aprendizado e Memória Organizacional

Para se manter atualizada em um mundo de negócios cada vez mais acelerado, uma organização deve melhorar continuamente seus produtos e processos. Alcançar um nível significativo de melhorias requer o nivelamento do conhecimento de funcionários experientes e altamente educados. Uma organização em constante aprendizado estabelece maneiras de gerenciar o conhecimento e de transformar capital intelectual em lucro. Para isto, uma organização deve criar um ambiente que promova o aprendizado constante e a troca de experiências. Deve, também, ser capaz de melhorar a comunicação interna, documentar o conhecimento relevante e armazená-lo (para reuso) em uma memória organizacional (FELDMANN, 2001).

O conhecimento é criado por indivíduos e a organização deve apoiar e fornecer um contexto para que os indivíduos possam criar conhecimento e expandi-lo do nível

individual para o organizacional. Para ARENT *et al.* (2002), o aprendizado é organizacional na medida em que é realizado para atingir os objetivos da organização, é compartilhado e distribuído entre os membros da organização e os resultados do aprendizado sejam integrados aos sistemas, estruturas e cultura da organização.

Segundo SNOEK (1999), o aprendizado de uma organização pode ocorrer de duas formas: aprendizado *top-down* (uma área específica de conhecimento é considerada promissora, levando a ações para aquisição deste conhecimento) e *bottom-up* (baseado no aprendizado individual de empregados que distribuem as "lições aprendidas" pela organização). Tais lições podem ser qualquer experiência positiva ou negativa que possa ser usada para melhorar o desempenho da organização no futuro.

As lições aprendidas podem ser obtidas de três maneiras: aprendizado individual, aprendizado através de comunicação e aprendizado através do desenvolvimento de um repositório de conhecimento (SNOEK, 1999). No aprendizado individual, funcionários ganham experiência realizando seu trabalho diário. Essa experiência é usada de forma inconsciente para melhorar os processos de trabalho. Isto acontece quando surgem idéias de melhoria e aplicação das lições aprendidas. O aprendizado através de comunicação é mais eficiente, pois há a necessidade de articular as lições aprendidas para compartilhá-las com os colegas. O elemento de comunicação melhora o entendimento das lições. Podem ocorrer duas formas deste tipo de aprendizado. Na primeira, chamada de aprendizado sob demanda, um empregado identifica um problema e procura soluções entre seus colegas; a segunda, chamada aprendizado por oferta, descreve uma situação na qual o empregado encontra uma maneira de melhorar o processo de trabalho e comunica a seus colegas. Aprendizado através do desenvolvimento de um repositório de conhecimento se baseia em um repositório para armazenar lições aprendidas. O conhecimento pode ser recuperado, adaptado às necessidades atuais e novas experiências podem ser armazenadas.

HENNINGER (1999) ressalta que um repositório serve não apenas como meio de disseminar o conhecimento, mas também ajuda a organização de software a aprender o que funciona e o que não funciona no seu contexto de desenvolvimento de software. Entretanto, o uso de um sistema de aprendizado organizacional precisa ser formalizado de maneira que seu uso seja determinado em um processo documentado, para garantir que a informação correta está sendo coletada.

VAN HEIJST *et al.* (citado em SNOEK, 1999) descrevem uma série de requisitos que uma memória organizacional deve atender.

- O acesso do funcionário ao conhecimento na memória organizacional deve ser fácil para facilitar o aprendizado individual.
- Deve ser fácil para o funcionário decidir qual colega poderia ter o conhecimento necessário para uma atividade específica.
- Deve ser fácil para o funcionário decidir qual colega poderia ter interesse em uma lição aprendida.
- Deve ser fácil e recompensador para o funcionário submeter uma lição aprendida à memória organizacional.
- Devem haver critérios bem definidos para decidir o que é uma lição aprendida, como ela deve ser formulada e onde deve ser armazenada.
- Devem haver mecanismos para manter consistente a memória organizacional.
- A memória organizacional deve ter mecanismos para distribuir um conhecimento recém-inserido aos empregados que precisam dele.

No caso de organizações de software, é necessário haver uma expansão do conhecimento do nível individual para o organizacional, transformando a organização para que ela use experiências passadas para melhorar atividades de desenvolvimento futuras (MAURER *et al.*, 1999; ARENT *et al.*, 2002). Introduzir o aprendizado em uma organização de software muda profundamente como desenvolvedores e gerentes devem trabalhar. Desenvolvedores devem mudar suas mentalidades, habilidades e comportamento; gerentes devem mudar suas expectativas sobre o que é entregue e quando; e a organização deve reconsiderar suas abordagem de treinamento e seus processos. Se os efeitos da mudança não ocorrerem por toda a organização, melhorias não vão acontecer, e o custo e o esforço investidos terão sido desperdiçados (SCHNEIDER *et al.*, 2002).

3.4 Gerência do Conhecimento

A gerência do conhecimento busca gerenciar formalmente os recursos de conhecimento, visando principalmente facilitar o acesso e reuso do conhecimento,

tipicamente utilizando a tecnologia da informação. Trata-se de um processo de conversão do conhecimento a partir das fontes disponíveis a uma organização e conexão entre pessoas e conhecimento (O'LEARY, 1998b). WEI *et al.* (2002) definem a gerência de conhecimento como um processo sistemático e organizacional para reter, organizar, compartilhar e atualizar o conhecimento crítico para o desempenho individual e a competitividade organizacional. No contexto da Engenharia de Software, a gerência de conhecimento é definida como um conjunto de atividades, técnicas e ferramentas apoiando a criação e transferência de conhecimento de Engenharia de Software por toda a organização (KOMI-SIRVIÖ *et al.*, 2002).

Entre os objetivos da gerência do conhecimento estão: transformação do conhecimento individual em conhecimento coletivo, melhora do aprendizado e integração de pessoas novas na organização; disseminação de melhores práticas; melhora de processos corporativos, qualidade de produtos e produtividade; e redução do tempo de projeto de novos produtos (DIENG, 2000).

Os recursos de conhecimento geralmente incluem manuais, cartas, sumários de respostas a clientes, notícias, informações sobre clientes, processos de trabalho, documentos, documentação de projetos, tutoriais, FAQs e páginas amarelas (O'LEARY, 1998a; WANGENHEIM *et al.*, 2001). Uma grande variedade de tecnologias está sendo usada para implementar sistemas de gerência do conhecimento: e-mail; bancos de dados e data warehouses; sistemas de apoio ao trabalho em grupo; browsers e mecanismos de busca; Intranets e Internet; sistemas especialistas e baseados em conhecimento; e agentes inteligentes (O'LEARY, 1998a).

3.4.1 Atividades da Gerência do Conhecimento

Segundo ABECKER *et al.* (1998), as atividades básicas da gerência do conhecimento são:

Identificação e Aquisição: Uma infra-estrutura para gerência do conhecimento deve oferecer um serviço de descoberta de conhecimento. É necessário utilizar mecanismos para transformar o conhecimento tácito (o *know how* dos empregados) em conhecimento explícito, que possa ser armazenado e distribuído. Entre as informações que podem ser coletadas estão experiências em projetos (tanto de sucesso quanto de fracasso), melhores práticas, lições aprendidas e notícias.

Desenvolvimento: A infra-estrutura para gerência do conhecimento deve oferecer um serviço de associação de conhecimento, ou seja, deve ser capaz de explorar os relacionamentos entre os dados na base, gerando novo conhecimento a partir daquele que já foi coletado.

Disseminação: A disseminação do conhecimento é, em geral, feita através de *Intranets* ou da própria *Internet.* Entretanto, a infra-estrutura para gerência do conhecimento deveria oferecer um serviço de distribuição de conhecimento que atue de maneira pró-ativa, sendo capaz de distribuir o conhecimento relevante entre os usuários que possuam um perfil adequado para recebê-lo.

Uso: Uma infra-estrutura para gerência do conhecimento deve oferecer um serviço de busca de conhecimento, disponibilizando maneiras eficientes para o usuário encontrar uma informação no repositório de conhecimento. A busca pode ser feita através de *queries* à base de conhecimento, mas métodos de busca mais sofisticados, tais como máquinas de busca e agentes inteligentes são mais adequados. Usando um mecanismo de busca adequado, o usuário poderá, por exemplo, consultar uma base de lições aprendidas, obtendo informações pertinentes a um projeto atual.

Preservação: Uma organização deve ser capaz de garantir a qualidade do conhecimento armazenado em suas bases. Para tanto, a infra-estrutura para gerência do conhecimento deve oferecer um serviço de filtragem do conhecimento, para garantir que o conhecimento esteja completo e correto.

RUS *et al.* (2002) apresentam um ciclo de evolução do conhecimento em Engenharia de Software, composto das seguintes fases:

Origem/Criação: Membros da organização desenvolvem software através de aprendizado, resolução de problemas, inovação, criatividade e importação de fontes externas.

Aquisição: Membros adquirem e capturam informações sobre conhecimento em formato explícito.

Transformação/Organização: Organizações organizam, transformam ou incluem conhecimento em material escrito ou em bases de conhecimento.

Acesso: Organizações distribuem conhecimento através de educação, programas de treinamento, sistemas de bases de conhecimento automatizados ou redes de especialistas.

Aplicação: A organização aplica o conhecimento, que é disponibilizado de acordo com a necessidade pela gerência de conhecimento. Este é o principal objetivo da organização e a parte mais importante do ciclo.

KING *et al.* (2002), RAMASUBRAMANIAN *et al.* (2002) e WEI *et al.* (2002) alertam para a necessidade de se validar o conhecimento antes que ele seja armazenado em um repositório. Esta validação é, em geral, realizada por um especialista. Na ausência deste tipo de filtro de conhecimento, os sistemas de gerência de conhecimento podem ser inundados com submissões que não são relevantes ou legítimas.

3.4.2 Tecnologias

Várias tecnologias têm sido usadas para apoiar as atividades de gerência do conhecimento. Dentre elas ABECKER *et al.* (1999) ressalta:

1. Groupware, Workflow e CSCW são freqüentemente consideradas tecnologias centrais para a gerência do conhecimento, porque em tarefas que exigem conhecimento, a colaboração de vários especialistas e departamentos em uma organização é uma necessidade natural.
2. Sistemas de gerência, recuperação e filtragem de documentos são considerados soluções para gerência do conhecimento, uma vez que a maior parte do conhecimento estratégico disponível em uma organização está sob forma de documentos textuais (no melhor caso, semi-estruturados).
3. Técnicas de Inteligência Artificial, como ontologias, *data mining*, bases de casos e sistema especialistas, são adequadas para lidar com a representação, captura e processamento do conhecimento.
4. A Intranet de uma organização pode ser utilizada para disponibilizar conhecimento, como, por exemplo, modelos dos processos organizacionais, através uma interface Web (KOCK, 2002; RAMASUBRAMANIAN *et al.*, 2002)

Técnicas para construção de uma memória organizacional podem ser não-computacionais, baseadas em bancos de dados, documentos, conhecimento, casos, ou na Web; orientadas a produto ou a processos. A escolha depende do tipo de organização, suas necessidades, sua cultura e deve levar em consideração as pessoas, organizações e tecnologias (DIENG, 2000). A seguir discutiremos uma série de tecnologias capazes de oferecer apoio para a gerência de conhecimento.

3.4.2.1 Técnicas da Engenharia de Conhecimento

Segundo PREECE *et al.* (2001), em termos de tecnologia, a maioria dos sistemas de gerência do conhecimento atuais se baseia em sistemas de bancos de dados e Internet. Quando o conhecimento é armazenado de forma explícita, tipicamente são usadas tabelas (por exemplo, em bancos de dados relacionais) ou texto semi-estruturado (como no Lotus Notes). O uso de sistemas de representação do conhecimento é raro e poucas organizações possuem um processo sistemático para captura de conhecimento, ao invés de captura de informação. PREECE *et al.* (2001) afirmam que a gerência do conhecimento, como é praticada atualmente, sub-utiliza a tecnologia da engenharia de conhecimento.

PREECE *et al.* (2001) ressaltam dois processos da engenharia de conhecimento:

- O uso de processos de aquisição do conhecimento para captura sistemática de conhecimento estruturado, e;
- O uso de tecnologia de representação de conhecimento para armazenar conhecimento, preservando relações importantes que não são possíveis em bancos de dados tradicionais. Em geral, são utilizados datawarehouses, ontologias, bases de dados e de conhecimento.

PREECE *et al.* (2001) apresentam o processo da engenharia de conhecimento como composto de cinco fases: Análise de requisitos; Modelagem conceitual; Construção da base de conhecimento; Operacionalização e validação; Refinamento e manutenção. Qualquer sistema de gerência do conhecimento que envolva representação explícita de conhecimento é passível de ser desenvolvido usando pelo menos parte deste processo.

Repositórios de Dados

O'LEARY (1998a) afirma que, em muitas organizações, *data warehouses* são as primeiras ferramentas para gerência do conhecimento a serem utilizadas. *Data warehouses* agem como áreas de armazenamento central para os dados de uma organização e diferem de bancos de dados tradicionais por serem projetados para dar apoio à tomada de decisão, ao invés de meramente capturar dados de transações.

Ontologias

Ontologias são definidas por SWARTOUT *et al.* (1999) como um conjunto de conceitos ou termos que podem ser usados para descrever alguma área do conhecimento ou construir sua representação.

Ontologias são especificações explícitas de conceituações, que tipicamente descrevem uma taxonomia das tarefas que definem o conhecimento. Sistemas de gerência do conhecimento dependem de ontologias para facilitar a comunicação entre múltiplos usuários e para estabelecer *links* entre múltiplas bases de conhecimento. Uma ontologia oferece a estrutura básica sobre a qual uma base de conhecimento pode ser construída. (O'LEARY, 1998c e SWARTOUT *et al.*, 1999).

Bases de Conhecimento

Os dois principais componentes de sistemas de gerência do conhecimento são bases de conhecimento e ontologias. Ontologias e bases de conhecimento são intimamente relacionadas na gerência do conhecimento. Ontologias definem as características das bases de conhecimento, empregando modelos que são úteis na definição e acesso às mesmas. Bases de conhecimento, por sua vez, contêm o conteúdo de sistemas de gerência do conhecimento (O'LEARY, 1998c).

Sistemas de gerência do conhecimento empregam uma variedade de bases de conhecimento, compostas de dados numéricos e qualitativos (páginas da Web, por exemplo). Dentre elas, podemos destacar bases de conhecimento de lições aprendidas, melhores práticas e competências de funcionários.

- **Lições Aprendidas**

Repositórios de lições aprendidas permitem armazenar experiências anteriores, que podem ser usadas para aumentar o potencial de sucesso em projetos futuros, para apoiar operações ou gerar informações sobre as atividades da organização

(LIEBOWITZ, 2002; REIFER, 2002; KOMI-SIRVIÖ *et al.*, 2002). Um repositório de lições aprendidas pode conter lições informativas; lições de sucesso, que capturam soluções de problemas; e lições de problemas, que oferecem exemplos de experiências que deram errado e potenciais soluções para o problema. Um item neste repositório pode incluir a descrição de um problema ou fenômeno, análise, comentários e recomendações, informação de contato daqueles que contribuíram e diretrizes. As diretrizes podem servir como soluções ou estratégias de mitigação para os problemas. As lições obtidas a partir de um projeto variam consideravelmente, baseadas na experiência com métodos, domínios e clientes particulares. À medida que este repositório cresce, ele passa a refletir a experiência da organização (WEI *et al.*, 2002; NICK *et al.*, 2001; STATZ , 1999).

- **Melhores Práticas**

Bases de conhecimento de melhores práticas capturam informação e conhecimento sobre a melhor maneira de realizar atividades, e capturam conhecimento sobre processos, ao invés de artefatos. Segundo O'LEARY (1998a), são tipicamente geradas usando atividades de *benchmark* planejadas para descobrir a maneira mais efetiva e eficiente de realizar uma atividade. São particularmente complexas e difíceis de desenvolver, pois extraem informações de múltiplas fontes, dentro e fora da organização. Seu conteúdo deve ser classificado, abstraído, sintetizado e revisto.

Ontologias são particularmente importantes para garantir que bancos de dados de melhores práticas sejam capazes de comunicar ao usuário uma grande variedade de práticas e atividades e permitir ao usuário reconhecer quando uma prática pode ser usada em sua organização (O'LEARY, 1998a).

- **Competências de Funcionários**

A maneira mais eficaz de encontrar informações disponíveis freqüentemente não é utilizar máquinas de busca sofisticadas, mas perguntar a especialistas onde encontrá-las. E mesmo que eles não saibam a resposta, geralmente eles podem indicar alguém que saiba. Há várias vantagens no uso de especialistas humanos ao invés de mecanismos de busca. Eles não apenas podem dar indicações, mas freqüentemente podem qualificá-las, aconselhando sobre a qualidade e relevância de suas necessidades (SKYRME, 1995).

Por esse motivo, sistemas de gerência de capacidades devem permitir que uma organização saiba quem possui o conhecimento desejado e facilitar a identificação e localização de especialistas na estrutura da organização (LIEBOWITZ, 2002; REIFER, 2002; RAMASUBRAMANIAN *et al.*, 2002). Em geral, esses sistemas são bancos de dados de currículos estruturados e, portanto, podem ser implementados com bancos de dados disponíveis comercialmente. Uma base de conhecimento de recursos humanos sobre as competências e habilidades das pessoas pode incluir, ainda, educação, especialidades, experiências anteriores, informação de contato entre outras informações (PREECE *et al.*, 2001; WEI *et al.*, 2002; SANTOS, 2003; SCHNAIDER, 2003).

Raciocínio Baseado em Casos

Sistemas de repositório de lições aprendidas permitem que os usuários examinem experiências passadas, armazenando esta experiência como casos estruturados. Esses sistemas permitem *queries* sofisticadas de casos semelhantes. Embora sistemas simples possam utilizar bancos de dados convencionais, a funcionalidade completa só pode ser conseguida com o uso de software de raciocínio baseado em casos ou sistemas baseados em conhecimento (PREECE *et al.*, 2001).

Agentes Inteligentes

O problema do excesso de informação está se tornando grave para muitos profissionais. Agentes inteligentes podem ser treinados para vasculhar redes para selecionar e alertar os usuários sobre informações novas e relevantes. Além disso, eles podem ser usados para filtrar informações menos importantes de fontes de informação (SKYRME, 1998).

3.4.2.2 Técnicas Baseadas na Web

Uma organização pode explorar características da Web de diversas maneiras. Páginas HTML ou XML internas ou URLs externas podem ser usadas, tornando-as disponíveis por toda a organização. Fóruns de discussão, grupos de discussão e FAQs, tanto internos quanto externos, também incentivam a troca de informações. A Web pode ser a base para a distribuição uniforme de informações, independente de como elas estão armazenadas (DIENG, 2000).

Intranet e Internet

Freqüentemente pessoas em uma parte da organização "reinventam a roda" ou não conseguem resolver problemas de clientes rapidamente, porque o conhecimento de que precisam existe em alguma parte da organização, mas não é conhecido ou acessível por elas. Por isso, a primeira iniciativa de muitos programas de gerência do conhecimento é instalar ou melhorar uma Intranet e incluir bases de dados de melhores práticas e de competências (SKYRME, 1998). Uma Intranet permite a comunicação dentro da organização e compartilhamento de informações através da rede interna. É válido lembrar que as funções básicas de e-mail, listas de discussão e *newsgroups* freqüentemente têm o maior impacto a curto prazo (SKYRME, 1998).

Fóruns de Discussão

Sistemas de gerência do conhecimento permitem a existência de grupos de discussão com foco em questões ou atividades específicas (O'LEARY, 1998c). Sistemas de fóruns de discussão promovem a disseminação do conhecimento dentro de comunidades de prática. Comunidades de prática consistem em pessoas compartilhando uma prática ou domínio de interesse comum (FISCHER *et al.*, 2001). Usuários se inscrevem em fóruns de seu interesse, trocando perguntas e respostas, lições aprendidas, anúncios e notícias da indústria. Tais sistemas são facilmente implementáveis com software gratuito disponível na Web ou com produtos comerciais (PREECE *et al.*, 2001).

3.4.2.3 Técnicas de Trabalho Colaborativo

Técnicas que apoiam a colaboração entre vários especialistas e entre diferentes partes de uma organização são úteis em tarefas que exigem conhecimento. Logo, tornaram-se uma necessidade para a gerência do conhecimento. Técnicas de *groupware* e *workflow* permitem que equipes de trabalho compartilhem informações, documentos e aplicações.

3.4.2.4 Técnicas de Gerência de Documentos

Sistemas de gerência de documentos permitem aos usuários encontrar documentos existentes que sejam relevantes à tarefa que estão realizando. Em geral, são sistemas de busca e recuperação de informação que se integram à Intranet da organização (e podem ser estendidos à Internet) (PREECE *et al.*, 2001). SKYRME

(1998) afirma que documentos, e especialmente documentos estruturados, são o formato no qual grande parte do conhecimento explícito é compartilhado.

3.5 Gerência de Conhecimento e Engenharia de Software

Devido à evolução rápida no campo de desenvolvimento de software, organizações de software precisam realizar grandes esforços para se manter diante de desafios como: novas tecnologias de hardware e software, mudanças constantes nos requisitos de clientes, arquiteturas, métodos e ferramentas de software cada vez mais complexos e mudanças freqüentes na equipe (KOMI-SIRVIÖ *et al.*, 2002; WEI *et al.*, 2002).

Muitas organizações entenderam que, para obter sucesso no futuro, elas devem gerenciar e usar o conhecimento de maneira mais efetiva no nível individual, de equipe e organizacional. A criação, distribuição e reuso de conhecimento atualizado são fatores críticos de sucesso, mas são difíceis de se atingir na prática, pois o conhecimento em Engenharia de Software é diverso e em constante crescimento. Organizações têm problemas em identificar o conteúdo, a localização e o possível uso deste conhecimento. A principal motivação da gerência de conhecimento na área de engenharia de software é o melhor uso deste conhecimento (KOMI-SIRVIÖ *et al.*, 2002; RUS *et al.*, 2002).

Segundo (RUS *et al.*, 2002), a utilização da gerência de conhecimento pode ser considerada uma estratégia para prevenção de riscos freqüentes em projetos de software, tais como:

- Perda ou indisponibilidade de conhecimento:

  Muito do conhecimento referente ao negócio de uma organização está situado nas mentes dos indivíduos. Se não houver uma maneira de registrar este conhecimento, quando um membro da equipe deixa a organização, ela perde conhecimento substancial pertencente a este indivíduo (WEI *et al.*, 2002; WANGENHEIM *et al.*, 2001).

- Falta de conhecimento e longo tempo para adquiri-lo:

  Devido a constantes mudanças na equipe, é necessário nivelar o conhecimento de novos membros, que possuem diferentes níveis de conhecimento e habilidades. Tais membros devem adquirir rapidamente o conhecimento relevante para a realização de seu trabalho. Se este conhecimento estiver retido

em nível individual, torna-se difícil passá-lo adiante para que seja reutilizado pela organização (WEI *et al.*, 2002; WANGENHEIM *et al.*, 2001).

- Repetição de erros e retrabalho por falta de conhecimento do que foi aprendido em projetos anteriores:

  Devido à falta de informações, soluções que podem já ter sido obtidas na organização são reinventadas. Para evitar que isto ocorra, informação e conhecimento sobre soluções existentes devem ser armazenadas e facilmente acessadas para permitir seu uso. Adicionalmente, erros são repetidos devido à incapacidade das organizações em identificar e transferir lições aprendidas. Para prevenir a repetição de erros, conhecimento tácito sobre estratégias aplicadas no passado deve ser capturado explicitamente e estar acessível (WANGENHEIM *et al.*, 2001).

Organizações de software constantemente precisam aumentar a qualidade de seus produtos, ao mesmo tempo em que buscam diminuir o tempo e custo de desenvolvimento de projetos de software. Para atingir estes objetivos, é necessário evitar erros, reduzindo o retrabalho e repetir processos que obtiveram sucesso, aumentando a produtividade e possibilidade de sucesso futuro. Logo, organizações precisam reter o conhecimento sobre processos obtido em projetos anteriores e aplicá-lo em projetos futuros. Infelizmente, a realidade é que equipes de desenvolvimento não se beneficiam da experiência existente e repetem os mesmos erros, embora alguns indivíduos na organização saibam como evitá-los. Membros de equipes de projeto adquirem conhecimento individual valioso com cada projeto, e a organização e indivíduos poderiam ganhar muito ao compartilhar este conhecimento (RUS *et al.*, 2002).

3.5.1 Conhecimento Necessário

RUS *et al.* (2002) afirmam que uma organização de software tem as seguintes necessidades de conhecimento:

- Adquirir conhecimento sobre novas tecnologias: para gerentes, é difícil avaliar o impacto de uma nova tecnologia em um projeto, dificultando a estimativa de custos. Adicionalmente, torna-se difícil para desenvolvedores manter a mesma produtividade após a introdução de uma tecnologia que eles

não dominam. Assim, organizações de software devem adquirir rapidamente conhecimento sobre novas tecnologias.

- Acessar conhecimento sobre o domínio: o desenvolvimento de software exige acesso a conhecimento sobre o domínio para o qual o software está sendo desenvolvido. Uma organização deve adquirir conhecimento sobre um novo domínio e disseminá-lo por toda a equipe.
- Compartilhar conhecimento sobre políticas e práticas: novos desenvolvedores em um organização precisam de conhecimento sobre políticas, práticas e cultura organizacional. Organizações devem formalizar a disseminação deste conhecimento, mantendo o compartilhamento informal de conhecimento entre seus funcionários.
- Capturar conhecimento e localizar especialistas: saber o que seus funcionários sabem é necessário para que as organizações evitem a perda ou indisponibilidade de conhecimento. Saber quem possui qual conhecimento também é necessário para a alocação eficiente de pessoas a projetos e para identificar necessidades de treinamento.
- Colaboração e compartilhamento de conhecimento entre equipes: o desenvolvimento de software é uma atividade realizada em equipe, logo, membros das equipes precisam de uma maneira de colaborar e compartilhar conhecimento independentemente de tempo e espaço.

VILLELA *et al.* (2000) listam os tipos de conhecimento que podem ser úteis para organizações que desenvolvem e mantêm software, separando-os em três classes: conhecimento externo, conhecimento interno estruturado e conhecimento técnico informal:

- **Conhecimento externo:** ponteiros para sites na Internet; manuais, livros e materiais de treinamento disponíveis eletronicamente ou a informação de como obtê-los.
- **Conhecimento interno estruturado:** métodos de planejamento de projeto e de engenharia de software, modelos de documentos com exemplos reais de utilização, melhores práticas, componentes de software, relatórios de pesquisa, diretrizes e outras informações e comunicações específicas da

organização, competência dos funcionários, e informações de marketing e de resultados da organização.

- **Conhecimento técnico informal:** discussões e notícias.

3.5.2 Fatores para Sucesso

Vários fatores de sucesso para iniciativas de gerência de conhecimento em Engenharia de Software são descritos na literatura. Abaixo são relacionados alguns deles:

- Um programa de gerência de conhecimento requer comprometimento e apoio constantes por parte da gerência (KEMP *et al.*, 2001; LIEBOWITZ, 2002; WEI *et al.*, 2002).
- Muitos programas de lições aprendidas não obtêm sucesso, pois se baseiam em coleta e disseminação de conhecimento passivas. É necessário haver mecanismos pró-ativos, para enviar lições aprendidas apropriadas aos indivíduos baseado em seus perfis (LIEBOWITZ, 2002; KOMI-SIRVIÖ *et al.*, 2002; WEI *et al.*, 2002).
- Para um programa gerência de conhecimento funcionar bem, devem ser oferecidos incentivos ao reuso do conhecimento. Organizações devem encorajar e recompensar seus empregados por compartilhar seu conhecimento e buscar e utilizar o conhecimento de outros. Ligar a contribuição e aplicação do conhecimento à revisão de desempenho pode encorajar isso (LIEBOWITZ, 2002; REIFER, 2002; RUS *et al.*, 2002; HENNINGER *et al.*, 2001). Na abordagem descrita em (RAMASUBRAMANIAN *et al.*, 2002), tanto o autor quanto o revisor recebem uma compensação, que serve como mecanismo de incentivo ao compartilhamento e avaliação da qualidade dos itens no repositório.
- Existir uma cultura organizacional que valorize a criação e compartilhamento do conhecimento (WEI *et al.*, 2002; RUS *et al.*, 2002).
- Haver uma integração com a infra-estrutura tecnológica existente, para que o uso do repositório de aprendizado organizacional se torne parte integrante do processo de desenvolvimento (WEI *et al.*, 2002; HENNINGER, 1999).

- O uso do repositório e das ferramentas de gerência de conhecimento deve ser obrigatório. Gerentes de projeto estão ocupados lidando com problemas do dia-a-dia e podem optar por não utilizar as ferramentas disponíveis (KOMI-SIRVIÖ *et al.*, 2002; HENNINGER, 1999).
- O repositório deve conter informações atualizadas e relevantes para o desenvolvimento. Devem haver diretrizes tanto para o armazenamento quanto para a busca do conhecimento (KOMI-SIRVIÖ *et al.*, 2002; HENNINGER, 1999).

3.5.3 Benefícios Esperados

Os benefícios esperados com a utilização da gerência de conhecimento em organizações de software são: melhoria do desenvolvimento de software e do processo de software, aumentando a qualidade do software e a possibilidade de sucesso (SCHNEIDER *et al.*, 2002); reuso do conhecimento de projetos de software anteriores para apoiar projetos futuros (RAMASUBRAMANIAN *et al.*, 2002; KOMI-SIRVIÖ *et al.*, 2002; SCHNEIDER *et al.*, 2002; WANGENHEIM *et al.*, 2001); prevenir duplicação de esforços, evitar a repetição de erros comuns (HENNINGER *et al.*, 2001; WANGENHEIM *et al.*, 2001); reação rápida a requisições de clientes, gerando maior satisfação (RAMASUBRAMANIAN *et al.*, 2002); aumento da produtividade (LIEBOWITZ, 2002; RAMASUBRAMANIAN *et al.*, 2002; WEI *et al.*, 2002); redução do nível de defeitos com conseqüente diminuição de retrabalho e do custo de detecção e prevenção de defeitos (LIEBOWITZ, 2002; RAMASUBRAMANIAN *et al.*, 2002; KOMI-SIRVIÖ *et al.*, 2002); desenvolvimento de sistemas melhores, mais rapidamente (WEI *et al.*, 2002); oferecer a pessoas inexperientes processos e diretrizes para acelerar seu aprendizado e facilitar o compartilhamento de experiências e a obtenção informações de especialistas quando necessário (NICK *et al.*, 2001; WANGENHEIM *et al.*, 2001); melhora da comunicação do conhecimento na organização, apoiando a tomada de decisão e a criação de soluções inovadores e criativas (LIEBOWITZ, 2002; WANGENHEIM *et al.*, 2001); permite que a organização mantenha o conhecimento de especialistas e evite a perda de know-how dos empregados quando os mesmos deixam a organização (LIEBOWITZ, 2002; WANGENHEIM *et al.*, 2001); permite a consolidação de conhecimento através de toda a organização, diminuindo a curva de

aprendizado para novas tecnologias e permitindo que a organização se adapte rapidamente a novas oportunidades (WANGENHEIM *et al.*, 2001).

3.5.4 Projetos

O desenvolvimento de software é organizado principalmente sob a forma de projetos. Logo, a equipe do projeto torna-se uma comunidade de prática, onde os membros adquirem e desenvolvem novo conhecimento, habilidades e competências coletivamente (ARENT *et al.*, 2002; DECKER *et al.*, 2001).

A principal tarefa de um projeto de Engenharia de Software é o desenvolvimento de software. Entretanto, documentos como contratos, planos de projeto e especificações de requisitos e projeto também são produzidos durante o desenvolvimento de software. Estes documentos capturam o conhecimento que surgiu ao resolver os problemas do projeto e que pode ser reutilizado em projetos futuros. A partir de conhecimento de projetos anteriores, podem ser extraídos padrões, melhores práticas, diretrizes e manuais, ou seja, conhecimento mais aplicável (RAMASUBRAMANIAN *et al.*, 2002; RUS *et al.*, 2002).

É necessário garantir que o conhecimento adquirido durante um projeto não seja perdido. Isso pode ser feito se a organização registrar lições aprendidas sobre o que deu certo ou errado no projeto (WEI *et al.*, 2002; RUS *et al.*, 2002; SEPPANËM *et al.*, 2001).

Uma maneira de registrar as lições aprendidas é através de reuniões post-mortem. Reuniões post-mortem visam discutir e documentar o que foi aprendido durante a execução de um projeto, tanto problemas quanto sucessos, de forma que possa ser utilizado em projetos futuros. (RAMASUBRAMANIAN *et al.*, 2002; KOMI-SIRVIÖ *et al.*, 2002; RUS *et al.*, 2002)

É importante que os indivíduos tenham acesso à informação e ao conhecimento corretos, quando necessário, para realizar uma tarefa ou tomar uma decisão (RUS *et al.*, 2002). Embora projetos possam ser bem diferentes, o reuso de experiências geralmente segue o mesmo princípio durante o trabalho em um projeto. Antes do início do projeto, experiências anteriores, lições aprendidas e projetos similares são identificados. Estas experiências são recuperadas e adaptadas para a situação atual. Ao final do projeto, o conhecimento recém-adquirido é analisado e pode ser disponibilizado para reuso na

organização (RAMASUBRAMANIAN *et al.*, 2002; SEPPANËM *et al.*, 2001; HENNINGER, 1999; BRANDT *et al.*, 2001).

Organizações devem garantir que o conhecimento de seus especialistas em gerência de projetos se encontra disponível, que o reuso desse conhecimento por membros de outros projetos é possível e que, ao mesmo tempo, um processo de aprendizado organizacional contínuo é apoiado. São necessárias ferramentas, técnicas e métodos que capturem o conhecimento de maneira contínua e o disseminem de forma que possa ser utilizado em projeto futuros (HENNINGER *et al.*, 2001; BRANDT *et al.*, 2001).

3.5.5 Processos

Processos de software podem ser usados como força propulsora para a coleta e disseminação de conhecimento sobre o desenvolvimento de software (HENNINGER, 1999). Baseado neste fato, muitas iniciativas de gerência de conhecimento em engenharia de software são baseadas nos processos de software de uma organização (KING *et al.*, 2002; KARAGIANNIS *et al.*, 2000).

A maioria das experiências obtidas por funcionários de organizações de software pode ser ligada aos processos e suas atividades. O conhecimento é derivado dos processos de engenharia de software, uma vez que, durante a execução do processo, o conhecimento não só é utilizado, mas também é gerado. Sendo assim, é natural que a gerência de conhecimento seja apoiada, baseada e guiada pelos processos (KARAGIANNIS *et al.*, 2000; NAGEL, 2001).

Nesse contexto, a gerência de conhecimento pode ser encarada como uma abordagem para melhorar o desempenho organizacional, cujo objetivo é criar conhecimento explícito organizacional sob a forma de processos e diretrizes que permitam e apóiem os gerentes de projeto e desenvolvedores de software a melhorar sua prática de Engenharia de Software (ARENT *et al.*, 2002). Este conhecimento pode estar sob a forma de repositório de *templates*, *checklists* e diretrizes de processo, servindo para padronizar o processo utilizado em um projeto e seus produtos (RAMASUBRAMANIAN *et al.*, 2002).

Segundo MAURER *et al.* (1999), uma abordagem de gerência de conhecimento baseada em processos apresenta as seguinte vantagens:

- É possível associar a tarefa que deve ser realizada com o conhecimento necessário para realizá-la. A disseminação do conhecimento passa a ser estruturada em torno do processo e conhecimento factual, tal como diretrizes, regras de negócio e estudos, é ligado a processos. Este conhecimento é oferecido no momento adequado para apoiar a execução de uma atividade (KARAGIANNIS *et al.*, 2000; MAURER *et al.*, 1999).
- É uma maneira de transferir conhecimento e melhores práticas sobre processos de desenvolvimento de software para membros inexperientes da equipe de maneira não-intrusiva, uma vez que processos de gerência de conhecimento se encontram integrados às atividades diárias do empregado. Assim, o uso do conhecimento torna-se parte das atividades de trabalho rotineiras (LIEBOWITZ, 2002; HENNINGER *et al.*, 2001; MAURER *et al.*, 1999).

HENNINGER *et al.* (2001) afirmam que são necessárias ferramentas de gerência do conhecimento que sejam mais pró-ativas na entrega de informações aos desenvolvedores do que simples repositórios. Uma maneira de atingir esse objetivo é criar um processo customizável que ofereça informações sensíveis aos contexto para projetos de desenvolvimento e capaz de capturar experiências de desenvolvimento de maneira contínua.

MAURER *et al.* (1999) identificam três camadas de conhecimento relacionados a processos de software:

- Conhecimento genérico e reutilizável sobre o modelo do processo: o próprio modelo do processo, documentos contendo informação, URLs, acessos a repositórios.
- Planos de projetos que contém conhecimento sobre as tarefas a serem realizadas e o conhecimento relacionado a elas.
- Dados dinâmicos de projeto, que correspondem a informações e produtos criados durante a execução do processo.

A utilização de modelos de processos pode facilitar o entendimento e comunicação, guiar indivíduos na realização de processos, apoiar a melhoria de processos, apoiar a gerência de processos e automatizar a execução (MAURER *et al.*,

1999; DECKER *et al.*, 2001). MAURER *et al.* (1999) afirmam que descrições e modelos de processos são necessários para analisar a maturidade do processo e servem como base para iniciativas de melhoria. A utilização de gerência de conhecimento para apoiar a melhoria de processos é defendida por diversos autores. Iniciativas de melhoria têm maiores chances de sucesso quando baseadas no conhecimento detalhado do que é necessário e o que foi feito em um organização de desenvolvimento de software. (KOMI-SIRVIÖ *et al.*, 2002; RUS *et al.*, 2002; KUCZA *et al.*, 2001). Segundo SCHNEIDER *et al.* (2002), experiências são relacionadas aos ambiente e contexto em que elas ocorreram e, quando reutilizadas em seu contexto original, elas podem direcionar a melhoria de processos.

São questões relevantes sobre a criação de conhecimento organizacional sobre melhoria de processos: que tipo de conhecimento é necessário? Onde encontrar conhecimento sobre boas práticas? Como internalizar boas práticas em projetos? Como disseminar o conhecimento sobre boas práticas pela organização (ARENT *et al.*, 2002).

NAGEL (2001) descreve uma abordagem na qual as descrições dos processos de software são usadas para gerar planos de projeto, apoiar estimativas de esforço e obter informações sobre o que fazer em uma atividade específica. A base para isso é a adaptação do processo padrão para obter um processo específico para o projeto. Ao ligar as atividades do processo às experiências, temos de um lado a descrição padrão de como realizar uma atividade e do outro lado as experiências, que podem descrever como realizar a atividade ou oferecer uma solução (um padrão, uma ferramenta, um *template*) ou descrever problemas que podem ocorrer durante a realização da atividade. Os membros da organização recebem não apenas o padrão do processo e suas atividades, mas também exemplos e soluções de como realizá-lo. Se for detectado um método ou solução através do qual o processo é bem executado, este é considerado uma "melhor prática" e pode servir de exemplo de como o processo pode ser executado. Ela deve ser avaliada e, se aceita, representada pelo sistema de gerência do conhecimento e ligada ao processo. Melhores Práticas podem servir de exemplo para projetos futuros e serem usadas como base para melhoria dos processos, sendo consultadas em projetos futuros.

3.6 Conclusão

Neste capítulo foi apresentada uma revisão da literatura sobre gerência do conhecimento. Os desenvolvimentos recentes na área tecnológica permitiram um grande

avanço no campo da gerência do conhecimento, uma vez que existem métodos e ferramentas disponíveis para dar apoio aos mais diversos tipos de conhecimento.

A gerência do conhecimento é fundamental para o sucesso futuro de organizações cujas atividades lidam com conhecimento de forma intensa, como as organizações de software. Quando realizada de maneira apropriada, a gerência do conhecimento pode trazer benefícios significativos, como o aumento da produtividade através do compartilhamento de conhecimento, melhor atendimento ao cliente pelo rápida acesso às informações e solução de problemas complexos pela reunião dos especialistas relevantes (SKYRME, 1999).

CAPÍTULO IV - AMBIENTES DE DESENVOLVIMENTO DE SOFTWARE ORIENTADOS À ORGANIZAÇÃO

4.1 Introdução

O conceito de Ambiente de Desenvolvimento de Software (ADS) surgiu no início da década de 70 e advém da identificação da necessidade de apoio integrado para as atividades de engenharia de software, ao longo do ciclo de vida do software (HARRISON *et al.*, 2000).

Um Ambiente de Desenvolvimento de Software é definido como sendo um sistema computacional que provê suporte para o desenvolvimento, reparo e melhorias em software e para o gerenciamento e controle dessas atividades (MOURA e ROCHA, 1992). Para tal, o ADS contém um repositório com todas as informações relacionadas com o projeto ao longo do seu ciclo de vida. Além disso, possui ferramentas que oferecem apoio às várias atividades técnicas e gerenciais passíveis de automação que devem ser realizadas no projeto. TRAVASSOS (1994) enfatiza que o ADS deve se preocupar com o apoio às atividades individuais e ao trabalho em grupo, o gerenciamento de projeto, o aumento da qualidade geral dos produtos e o aumento da produtividade, permitindo ao engenheiro de software acompanhar o projeto e medir a evolução dos trabalhos através de informações obtidas ao longo do desenvolvimento.

Com a definição de ADS e a preocupação de que software não deve ser desenvolvido de forma *ad hoc*, surgiram pesquisas em ambientes centrados em processo (GARG e JAZAYERI, 1995), onde se defende a idéia de ferramentas integradas e incorporadas a um processo de desenvolvimento de software específico. Um processo de desenvolvimento de software é um conjunto bem definido e ordenado de atividades, somado aos recursos utilizados e produzidos e ao conjunto de ferramentas e técnicas para apoio à realização destas atividades (PFLEEGER, 2001).

A COPPE possui um grupo de trabalho em ambientes de desenvolvimento de software responsável pela definição e criação da Estação TABA (ROCHA *et al.*, 1990). Este projeto surgiu no início da década de 90 e consiste de um meta-ambiente de desenvolvimento de software capaz de gerar, através de instanciação, outros ADS.

Posteriormente foram realizadas duas evoluções importantes nos ambientes gerados a partir da Estação TABA: os Ambientes de Desenvolvimento de Software Orientados a Domínio e os Ambientes de Desenvolvimento de Software Orientados a Organização.

O objetivo deste capítulo é apresentar os conceitos envolvidos no desenvolvimento da Estação TABA, desde a sua definição até o momento atual. A seção 2 apresenta o objetivo, as características e os requisitos da Estação TABA. A seção 3 comenta a evolução dos ambientes de desenvolvimento de software com a utilização de conhecimento sobre o domínio durante o processo de desenvolvimento. A seção 4 apresenta a evolução dos ambientes de desenvolvimento de software com a inclusão de conhecimento organizacional. A seção 5 apresenta a abordagem proposta para a implementação do processo de desenvolvimento em um ambiente Configurado na Estação TABA. A seção 6 apresenta a conclusão deste capítulo.

4.2 A Estação TABA

A Estação TABA (ROCHA *et al.*, 1990) é um meta-ambiente capaz de gerar, através de instanciação, ambientes de desenvolvimento de software adequados às particularidades de processos de desenvolvimento e de projetos específicos. ROCHA *et al.* (1990) definem meta-ambiente como um ambiente que abriga um conjunto de programas que interagem com os usuários para definir interfaces, selecionar ferramentas e estabelecer os tipos de objetos que irão compor o ambiente de desenvolvimento específico.

O projeto TABA foi criado no início da década de 90, a partir da constatação de que domínios de aplicação diferentes possuem características distintas e que estas devem incidir nos ambientes de desenvolvimento através dos quais os engenheiros de software desenvolvem aplicações. Desta forma, a Estação TABA tem por objetivo auxiliar na definição, implementação e execução de ADS adequados a contextos específicos.

Com o intuito de atender a este objetivo, quatro funções foram definidas originalmente para a Estação TABA (TRAVASSOS, 1994):

- Auxiliar o engenheiro de software na especificação e instanciação do ambiente mais adequado ao desenvolvimento de um produto específico a

partir do processo de software e/ou de uma definição do domínio de aplicação;

- Auxiliar o engenheiro de software na implementação das ferramentas necessárias ao ambiente definido;
- Permitir aos desenvolvedores do produto de software a utilização da estação através do ambiente desenvolvido;
- Permitir a execução do software na estação configurada para o seu desenvolvimento.

Na Estação TABA, um ADS é instanciado a partir da definição de um processo de desenvolvimento. Dois ambientes foram definidos e implementados de acordo com essa concepção: o ambiente Orixás (WERNECK, 1995), criado para apoiar o desenvolvimento de Sistemas Baseados em Conhecimento e o ambiente Memphis (WERNER *et al.*, 1997), criado para apoiar o desenvolvimento de software baseado em reutilização.

Ao longo dos anos o conceito de ADS evoluiu para a definição de ADS com suporte à utilização de informações sobre o conhecimento do domínio de aplicação durante o desenvolvimento e, mais recentemente, para a utilização de conhecimento organizacional. Para acompanhar essa evolução, a Estação TABA passou a instanciar, também, Ambientes Orientados a Domínio e Ambientes Orientados a Organização, descritos a seguir.

4.3 Ambientes de Desenvolvimento de Software Orientados a Domínio

Embora o uso de ambientes instanciados a partir da Estação TABA conduza o usuário através de um processo de desenvolvimento de software, um grande transtorno no desenvolvimento de software não foi abordado: o entendimento errôneo ou incompleto do problema que o software pretende resolver e que pode levar a produtos de software implementados corretamente, mas que resolvem os problemas errados. Um agravante desta situação, segundo FISCHER (1996), é o fato de problemas complexos exigirem mais conhecimento do que uma só pessoa pode possuir, o que faz com que a comunicação e a colaboração tornem-se fundamentais. A idéia central passou a ser, então, a de um entendimento compartilhado: os especialistas entendem a prática e os

projetistas de sistemas conhecem a tecnologia. Tal compartilhamento precisa ocorrer durante todo o processo de desenvolvimento, pois novos requisitos surgem durante o desenvolvimento porque não podem ser identificados até que porções do sistema tenham sido projetadas e implementadas.

Com a intenção de facilitar a comunicação e o entendimento do problema, OLIVEIRA (1999) propôs os Ambientes de Desenvolvimento de Software Orientados a Domínio (ADSOD). ADSOD são ambientes de desenvolvimento de software que apóiam os engenheiros de software em domínios específicos, permitindo o uso do conhecimento do domínio durante todo o processo de desenvolvimento, auxiliando o desenvolvedor no entendimento do problema (OLIVEIRA *et al.*, 1999a e 1999b). O domínio é uma área de aplicação na qual vários produtos de software serão desenvolvidos. De acordo com a proposta deste trabalho, para construção de ADSOD, o conhecimento do domínio é organizado em um modelo, denominado Teoria do Domínio, que utiliza ontologias do domínio.

Desta forma, o aspecto central dos ADSOD é a introdução e uso do conhecimento do domínio no ADS, provendo apoio aos desenvolvedores na identificação correta dos requisitos do sistema e sua descrição (OLIVEIRA *et al.*, 1999a). Como conseqüência, espera-se obter um aumento de produtividade no desenvolvimento. O uso do conhecimento do domínio durante o desenvolvimento de software tende a tornar o processo de entendimento do problema mais fácil e agradável tanto para os desenvolvedores quanto para os especialistas do domínio que, muitas vezes, não são profissionais da área de informática (FISCHER, 1994).

OLIVEIRA (1999) estendeu a Estação TABA de forma a permitir a instanciação de ADSOD. Alguns ADSOD foram instanciados utilizando essa infra-estrutura: CORDIS (OLIVEIRA, 1999), um ambiente orientado ao domínio de cardiologia desenvolvido para a Fundação Bahiana de Cardiologia; NETUNO (GALOTTA, 2000), um ambiente orientado ao domínio da acústica submarina desenvolvido para o Grupo de Sonar do Instituto de Pesquisas da Marinha (GS/IPqM), INSECTA (FOURO, 2002), um ADSOD para o domínio de entomologia desenvolvido para a Embrapa e SIDER (CARVALHO, 2002) um ambiente orientado ao domínio siderúrgico desenvolvido para a Companhia Siderúrgica de Tubarão (CST).

4.4 Ambientes de Desenvolvimento de Software Orientados à Organização

Para melhorar a maneira como as organizações desenvolvem e mantêm software, é fundamental melhorar a maneira como elas administram o conhecimento requerido para a realização desta atividade. Desenvolvedores de software precisam que todo o conhecimento relevante para a realização de suas atividades esteja facilmente disponível, o que inclui conhecimento sobre o domínio, diretrizes e melhores práticas organizacionais, técnicas e métodos de desenvolvimento de software, experiências anteriores com o uso destas técnicas e métodos e também com o processo de software (VILLELA *et al.*, 2000). ADSOD (OLIVEIRA, 1999) buscam assegurar a disponibilidade e o entendimento de conhecimento sobre domínios de aplicação. No entanto, o conhecimento necessário, para um esforço de engenharia de software eficiente, extrapola o conhecimento sobre o domínio.

Reconhecendo que as organizações que desenvolvem e mantêm software lidam de forma intensa com diversos tipos de conhecimento e que o conhecimento do domínio, embora fundamental, não é o único conhecimento que deve estar disponível durante o desenvolvimento de software, VILLELA *et al.* (2000) propuseram a construção de Ambientes de Desenvolvimento de Software Orientados à Organização (ADSOrg), que representam uma evolução dos ADSOD.

ADSOrg é definido por VILLELA *et al.* (2000) como uma classe de ambientes de desenvolvimento de software que apóia a atividade de engenharia de software em uma organização, fornecendo o conhecimento acumulado pela organização e relevante para esta atividade, ao mesmo tempo em que apóia, a partir de projetos específicos, o aprendizado organizacional em engenharia de software.

ADSOrg têm como objetivo apoiar o gerenciamento completo do conhecimento requerido em uma atividade de engenharia de *software*, evitando que este conhecimento fique disperso ao longo da estrutura organizacional e, conseqüentemente, sujeito a dificuldades de acesso e, mesmo, a perdas (VILLELA *et al.*, 2000).

A motivação para a construção de ADSOrg surgiu de duas constatações (VILLELA *et al.*, 2001b):

- duas ou mais organizações podem desenvolver software para um mesmo domínio com processos, interesses e características muito distintas, e,

- o conhecimento do domínio não é o único conhecimento importante para apoiar desenvolvedores de software em suas atividades. Outros conhecimentos também são extremamente importantes e úteis para os desenvolvedores. Deste modo, um ADSOrg inclui conhecimento não só sobre o domínio específico, mas também sobre, por exemplo, normas e diretrizes organizacionais, melhores práticas, relatos de experiências, lições aprendidas com o uso de processos, métodos e técnicas de *software,* entre outros.

Com a finalidade de evitar erros já cometidos e possibilitar a reutilização de soluções já aprovadas na execução de tarefas similares, foram definidos dois objetivos básicos para os ADSOrg (VILLELA *et al.*, 2000). O primeiro é prover, para os desenvolvedores de software, o conhecimento acumulado pela organização e relevante no contexto do desenvolvimento de software. O segundo é apoiar o aprendizado organizacional neste contexto.

Para atingir os objetivos estabelecidos, foram definidos os seguintes requisitos para ADSOrg (VILLELA *et al.*, 2000):

(i) ter a representação da infra-estrutura da organização;

(ii) reter conhecimento especializado sobre desenvolvimento e manutenção de software;

(iii) permitir a utilização deste conhecimento em projetos;

(iv) apoiar a atualização constante do conhecimento armazenado no ambiente, e;

(v) facilitar a localização de especialistas da organização que podem ser úteis em um projeto.

Para atender ao requisito (ii), um ADSOrg precisa dispor de conhecimento sobre as atividades de desenvolvimento e manutenção de software que são sempre realizadas na organização independente de uma projeto específico, permitindo a elaboração de um processo padrão para a organização. Além disso, o ADSOrg precisa armazenar a experiência desta organização em engenharia de software, o que envolve normas e diretrizes da organização; últimas notícias relativas às tecnologias utilizadas na organização; relatos de melhores práticas com relação à definição de processo e à

utilização de métodos e de linguagens de programação, com exemplos de projetos que as utilizaram; análise de ferramentas; bases de produtos intermediários com potencial para serem reutilizados; lições aprendidas em projetos anteriores; e medidas de desempenho da organização. Atividades de análise, projeto, implementação, teste e implantação podem ser apoiadas, por exemplo, através de notícias relativas às tecnologias utilizadas em cada atividade, de melhores práticas em relação aos métodos selecionados e da reutilização de produtos intermediários gerados por outros projetos (VILLELA *et al.*, 2000).

4.4.1 Configuração e Instanciação de Ambientes na Estação TABA

Para apoiar a construção de ADSOrg, a Estação TABA foi novamente estendida. Além de permitir a definição e instanciação de ADS, orientados a domínio ou não, a Estação TABA passou a permitir também a configuração de ambientes para organizações específicas. As funções originais da Estação TABA foram revistas e ampliadas. Suas funções atuais são (VILLELA *et al.*, 2001a):

(i) Auxiliar o engenheiro de software na configuração do ambiente mais adequado para apoiar o desenvolvimento e a manutenção de software em uma organização específica (Ambiente Configurado), considerando seu processo de software e a gerência do conhecimento organizacional relevante neste contexto;

(ii) Auxiliar o engenheiro de software na instanciação de ambientes de desenvolvimento de software para projetos específicos (caso a configuração de um ambiente para organização não seja possível ou considerada adequada);

(iii) Auxiliar os gerentes de projeto na instanciação de ambientes de desenvolvimento de software para projetos específicos a partir de um Ambiente Configurado;

(iv) Auxiliar o engenheiro de software de empresas cujo negócio é o desenvolvimento e a manutenção de software para diversos clientes na especialização de processos da sua empresa de acordo com as particularidades de um cliente específico;

(v) Auxiliar o engenheiro de software a implementar ferramentas necessárias aos ambientes;

(vi) Apoiar, através dos ambientes instanciados, o desenvolvimento e a manutenção de software, bem como a gerência destas atividades;

(vii) Permitir a execução do software na própria Estação, pelo menos para fins de teste.

A configuração de um ambiente na Estação TABA é feita a partir da definição de um processo de desenvolvimento que se caracteriza pela descrição de uma seqüência de atividades, suas ferramentas de apoio, produtos de software gerados e recursos consumidos (MACHADO, 2000). A definição do processo de desenvolvimento segue o esquema proposto por OLIVEIRA (1999): definição de um processo padrão para a organização, especialização de acordo com o paradigma de desenvolvimento e instanciação para o projeto específico (VILLELA et al., 2000).

OLIVEIRA (1999) propôs a definição de Processo Padrão, com o objetivo de estabelecer um processo de desenvolvimento comum a ser utilizado por uma organização, a partir do qual é feita a definição dos processos de desenvolvimento especializados e instanciados e, conseqüentemente, de ADS específicos para os projetos. A norma ISO/IEC 12207 (1998) é a base para a definição de qualquer processo na Estação TABA, podendo, também, ser considerado um dos modelos de maturidade (MACHADO, 2000).

Para definição do Processo Padrão no contexto da Estação TABA, foi estabelecido que cada atividade do processo de software deve ser descrita através da: (i) definição de seus objetivos principais, (ii) definição e descrição das sub-atividades ou sub-processos a serem realizados, (iii) definição dos produtos gerados na atividade, e, (iv) identificação dos responsáveis envolvidos na realização das atividades e sub-atividades (MACHADO, 2000).

A partir do Processo Padrão definido para a organização, diferentes processos de software podem ser especializados de acordo com as características de cada paradigma de desenvolvimento, podendo ser incluídas novas atividades, mas mantendo os elementos básicos definidos no Processo Padrão. Para ser utilizado em um projeto específico, o processo especializado mais adequado para um determinado projeto deve ser instanciado para atender às características do projeto específico, devendo-se

considerar o tamanho e complexidade do produto, as características da equipe de desenvolvimento, a expectativa de vida útil do software e demais características do projeto (MACHADO, 2000).

Segundo VILLELA et al. (2000), normas e diretrizes organizacionais podem auxiliar na definição do processo padrão para a organização; melhores práticas, experiências de projetos anteriores e últimas notícias sobre tecnologias podem auxiliar na especialização do processo padrão para um tipo de software específico; e experiências de projetos anteriores podem auxiliar na instanciação do processo para atender às características de um projeto específico.

O ambiente configurado contém o processo padrão da organização e os processos especializados para os paradigmas de desenvolvimento utilizados na organização. Este ambiente irá armazenar e prover todo o conhecimento capturado pela organização e importante para o desenvolvimento e a manutenção de software. Um ambiente configurado contém, ainda, a ferramenta para apoiar a instanciação de processos com a capacidade de gerar ambientes de desenvolvimento de software para projetos específicos, os ADSOrg, a partir dos processos instanciados. Os ambientes instanciados são os ambientes que apóiam o desenvolvimento dos produtos de software e contêm outras ferramentas de apoio ao desenvolvimento. Estes ambientes têm, também, acesso ao conhecimento da organização armazenado no repositório da organização. A figura abaixo mostra o esquema utilizado para geração de ambientes configurados para organizações específicas e instanciação de ADSOrg a partir da Estação TABA.

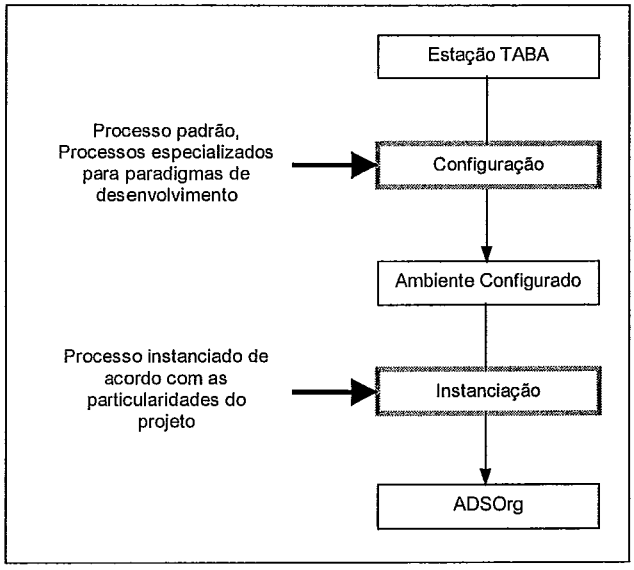


Figura 4.1 - Esquema para geração de ambientes a partir da Estação TABA

Facilidades para permitir o registro das organizações e dos projetos foram adicionadas à Estação TABA, uma vez que ambientes são configurados para uma organização e ADS convencionais e ADSOD são instanciados para um projeto específico. Tanto na configuração de ambientes quanto na instanciação de ADS convencionais e ADSOD, o engenheiro de software define as características do ambiente e solicita a sua criação. As características para a configuração de um ambiente incluem a organização para a qual o ambiente será gerado, o seu processo padrão e processos especializados, e as Teorias de Domínio a serem utilizadas. Para a instanciação de um ADS, o engenheiro de software define um projeto e o processo instanciado para o qual o ambiente será gerado, selecionando uma Teoria do Domínio, quando desejado. As ferramentas especificadas no processo instanciado são automaticamente incluídas no ambiente gerado.

4.5 Processo de Instanciação de Processos de Software

Não é possível garantir que o gerente de projeto, responsável pela instanciação do processo, tenha experiência na definição de processos, ou mesmo em engenharia de software. Portanto, torna-se necessário oferecer meios de apoiar a atividade de instanciação, para garantir a qualidade do processo.

Esta dissertação apresenta uma abordagem para a implementação do processo de desenvolvimento de software (atividade do processo de desenvolvimento da ISO 12207) em um ambiente configurado e propõe uma ferramenta cujo objetivo é apoiar o gerente de projetos durante a instanciação do processo, utilizando técnicas de gerência do conhecimento. Foi definido um processo para instanciação de processos de software, tornando explícito o conhecimento acumulado em diversos projetos.

Para ser utilizado em um projeto específico, o processo especializado mais adequado deve ser instanciado para atender às características do projeto. O processo de instanciação apresentado a seguir descreve as atividades necessárias para se realizar a adaptação (instanciação) de um processo especializado para um projeto específico, dentro do contexto de um Ambiente Configurado.

As atividades descritas são de responsabilidade do gerente do projeto. O resultado da instanciação é um processo instanciado que constitui o Plano do Processo

de Desenvolvimento para um projeto específico. O processo de instanciação proposto é composto das seguintes atividades:

Pré-Condição: Identificação de Atividades Obrigatórias

1. Caracterizar Projeto
    1.1 Definir Projeto
    1.2 Identificar Características do Projeto
2. Planejar Processo
    2.1 Incluir Atividades do Tipo de Software
    2.2 Definir Modelo de Ciclo de Vida
    2.3 Mapear Atividades do Modelo de Ciclo de Vida
        2.3.1 Verificar a Existência de Mapeamento Anterior
        2.3.2 Realizar Mapeamento
    2.4 Excluir Atividades Não Pertinentes
    2.5 Incluir Novas Atividades
    2.6 Selecionar Técnicas de Avaliação
    2.7 Selecionar Ferramentas
3. Instanciação de ADSOrg para o projeto

Pré-Condição: Identificação de Atividades Obrigatórias

Descrição:

A identificação das atividades obrigatórias é realizada durante a configuração do ambiente. Consiste em definir quais são as atividades essenciais para qualquer projeto da organização, independente de suas características (tamanho, complexidade, etc.). Estas atividades não poderão ser excluídas do processo durante a instanciação. Caso um modelo de maturidade tenha sido selecionado para o processo, as atividades que garantem a conformidade com o nível de maturidade desejado serão automaticamente marcadas como obrigatórias.

1. Caracterizar Projeto

Descrição:

Esta atividade visa identificar as características do projeto para o qual será realizada a instanciação, obtendo informações que servirão como base para a tomada de decisões na elaboração do plano do processo de desenvolvimento.

Sub-Atividades:

1.1. Definir Projeto

Esta sub-atividade consiste em definir o novo projeto a ser realizado na organização, para o qual será instanciado o processo.

Produto: Projeto definido

1.2. Identificar Características do Projeto

Esta sub-atividade tem como objetivo caracterizar o software a ser desenvolvido, a partir de informações sobre o seu escopo, tamanho, complexidade, tipo, domínio da aplicação, conhecimento da equipe sobre o domínio da aplicação, tecnologia de desenvolvimento, disponibilidade de recursos financeiros e de tempo, requisitos de qualidade e criticidade do software.

Produto: Projeto caracterizado

2. Planejar Processo

Descrição:

Esta atividade tem como objetivo a geração do Plano do Processo para o novo projeto, que é o processo instanciado. Quando um ambiente é configurado, são fornecidos o processo padrão e os processos especializados para as tecnologias de desenvolvimento utilizadas na organização. O processo especializado mais adequado ao projeto é selecionado de acordo com o paradigma de desenvolvimento. O processo de instanciação parte deste processo especializado e o adapta para adequá-lo às características específicas do projeto.

Sub-Atividades:

2.1. Incluir Atividades do Tipo de Software

Caso o tipo de software a ser desenvolvido não tiver sido definido durante a criação do Processo Especializado, ele deverá ser definido neste momento. Se o ambiente possuir o conhecimento das atividades que devem ser realizadas para o desenvolvimento deste tipo de software, estas deverão ser incluídas no processo.

Em um ambiente configurado para uma organização, duas situações podem ocorrer:

- A organização desenvolve apenas um tipo de software. Nesse caso o conhecimento sobre as atividades específicas para este tipo de software já foi incorporado ao processo padrão da organização, uma vez que tais atividades devem ser realizadas em qualquer projeto dessa organização. Quando esta situação ocorrer a atividade "Incluir Atividades do Tipo de Software" não será realizada.
- A organização desenvolve diferentes tipos de software. Nesse caso o conhecimento sobre as atividades específicas para cada tipo de software não será incorporado aos processos padrão e especializados, mas estará disponível no ambiente configurado e será incorporado ao processo instanciado para um projeto.

Produto: Processo especializado com atividades referentes ao tipo de software

2.2. Definir Modelo de Ciclo de Vida

Nesta sub-atividade o gerente do projeto seleciona o modelo de ciclo de vida que julgar mais adequado ao projeto.

Produto: Modelo de Ciclo de Vida para o projeto definido

2.3. Mapear Atividades do Modelo de Ciclo de Vida

As atividades previstas no processo especializado são organizadas de acordo com o modelo de ciclo de vida selecionado, o que implica em:

2.3.1. Verificar a Existência de Mapeamento Anterior

Consiste em verificar se já foi realizado um mapeamento em projeto anterior, do mesmo processo especializado para o mesmo modelo de ciclo de vida. Caso tal mapeamento exista, ele pode ser reutilizado.

Produto: Mapeamento anterior identificado

2.3.2. Realizar Mapeamento

Caso o mapeamento entre o processo especializado e o ciclo de vida ainda não tenha sido realizado, o gerente do projeto deve realizá-lo.

Produto: Mapeamento realizado

2.4. Excluir Atividades Não Pertinentes

O Gerente de Projeto avalia a pertinência das atividades não obrigatórias do processo para identificar se existem atividades que podem ou devem ser excluídas sem prejuízo para o projeto.

Produto: Processo com atividades desnecessárias ao projeto excluídas

2.5. Incluir Novas Atividades

O Gerente de Projeto pode incluir novas atividades no processo, caso as julgue necessárias ao desenvolvimento do projeto. O Gerente pode incluir atividades utilizadas em projetos anteriores ou pode definir uma nova atividade.

Produto: Processo com atividades incluídas

2.6. Selecionar Técnicas de Avaliação

Quando o projeto é caracterizado, o software é classificado de acordo com o tipo de aplicação e impacto em caso de falha. Com base nesta classificação são indicadas técnicas de avaliação da qualidade apropriadas para o projeto. Quanto maior for o risco de dano, mais rigorosas devem ser as técnicas de avaliação.

Produto: Procedimentos para avaliação da qualidade identificados

2.7. Selecionar/Associar Ferramentas

O Gerente de Projeto seleciona quais ferramentas serão efetivamente utilizadas para apoiar a execução das atividades do processo, baseado nas opções de ferramentas oferecidas pelo ambiente configurado. Com esta sub-atividade está finalizada a instanciação do processo.

Produto: Ferramentas associadas às atividades do processo e processo instanciado definido

3. Instanciação de ADSOrg para o projeto

Descrição:

Esta atividade tem como objetivo a instanciação de um Ambiente de Desenvolvimento de Software (ADSOrg) para a realização do projeto, a partir do processo instanciado.

4.6 Conclusão

Este capítulo apresentou os conceitos de Ambientes de Desenvolvimento de Software, Ambientes de Desenvolvimento de Software Orientados a Domínio e Ambientes de Desenvolvimento de Software Orientados a Organização. Também foram descritas as características e requisitos da Estação TABA e sua evolução para permitir a configuração de ambientes para organizações e a instanciação de ambientes de desenvolvimento de software para projetos específicos. Finalmente, foi apresentado o processo de instanciação, que parte de um processo especializado da organização, em um ambiente configurado, e o adapta, gerando um processo instanciado específico para um projeto. A partir deste processo instanciado será gerado um ADSOrg para o projeto.

CAPÍTULO V - ADAPTPRO: UMA FERRAMENTA DE APOIO À INSTANCIAÇÃO DE ADSORG

5.1 Introdução

No capítulo 4 descrevemos os ambientes configurados TABA e o processo de instanciação de ADSOrg. Para apoiar este processo foi desenvolvida a ferramenta *AdaptPro*, que, partindo de um processo especializado, apóia a adaptação deste processo de forma a se ter um processo de desenvolvimento adequado a um projeto específico. A partir deste processo adaptado ao projeto, *AdaptPro* gera um ADSOrg.

Neste capítulo descrevemos, na seção 5.2, a abordagem adotada para a implementação da ferramenta. Na seção 5.3 descrevemos a ferramenta com detalhes, mostrando a sua utilização para instanciação do ADSOrg para um projeto específico.

5.2 Abordagem Utilizada

A abordagem adotada neste trabalho para adaptação do processo está baseada no processo definido no capítulo anterior, em conhecimento obtido da literatura para apoiar as atividades do processo e em conceitos de gerência do conhecimento. Esta abordagem foi a base para a construção da ferramenta *AdaptPro* descrita na seção 5.3. A seguir comentamos como são realizadas as atividades do processo segundo esta abordagem.

5.2.1 Caracterização do Projeto

A atividade "**Caracterizar Projeto**" tem início com a sub-atividade "**Definir Projeto**". Em seguida é realizada a sub-atividade "**Identificar Características do Projeto**", que tem como objetivo caracterizar o software a ser desenvolvido. Consideramos que, para isto, devem ser considerados três tipos de características:

- **Características gerais**

O gerente do projeto deve caracterizar o projeto de acordo com um conjunto de critérios que servirão de apoio para a definição do modelo de ciclo de vida mais adequado ao projeto. Os critérios propostos nesta abordagem (tabela 5.1) são uma adaptação dos critérios propostos por MACHADO (2000), que por sua vez se baseou na abordagem proposta por ALEXANDER e DAVIS (1991).

Tabela 5.1 – Critérios de caracterização do projeto

Critérios relacionados aos usuários	**Valores**
Experiência dos usuários no domínio da aplicação	Baixa, Média ou Alta
Facilidade dos usuários em expressar requisitos	Baixa, Média ou Alta
Grau de acesso aos usuários	Baixo, Médio ou Alto
Nível de mudanças geradas no trabalho dos usuários	Baixo, Médio ou Alto
Critérios relacionados ao problema	**Valores**
Grau de maturidade do domínio da aplicação	Baixo, Médio ou Alto
Complexidade do problema	Baixa, Média ou Alta
Freqüência de mudanças nos requisitos	Baixa, Média ou Alta
Grau de magnitude das mudanças nos requisitos	Baixo, Médio ou Alto
Grau de modularidade do problema	Baixo, Médio ou Alto
Critérios relacionados ao produto	**Valores**
Tamanho da aplicação	Pequeno, Médio ou Grande
Grau de complexidade da aplicação	Baixo, Médio ou Alto
Critérios relacionados aos recursos	**Valores**
Disponibilidade de recursos humanos	Baixa, Adequada ou Alta
Disponibilidade de recursos financeiros	Baixa, Adequada ou Alta
Critérios relacionados à equipe de desenvolvimento	**Valores**
Experiência da equipe de desenvolvimento no domínio da aplicação	Baixa, Média ou Alta
Experiência da equipe de desenvolvimento em engenharia de software	Baixa, Média ou Alta
Nível de experiência da gerência	Baixo, Médio ou Alto
Capacidade de gerenciamento de múltiplas equipes	Sim ou Não
Critérios relacionados ao desenvolvimento	**Valores**
Há necessidade de entrega de produtos intermediários?	Sim ou Não
Há necessidade do software ser colocado em uso rapidamente com funcionalidade total ou parcial?	Sim ou Não
Grau dos riscos técnicos	Baixo, Moderado ou Alto
Necessidade de interface com sistemas existentes	Sim ou Não
Uso de tecnologia inovadora	Sim ou Não

- **Características de qualidade da ISO 9126**

A Norma ISO/IEC 9126 define seis características de qualidade para produtos de software que, entretanto, podem não estar todas presentes em um produto ou, então não

serem necessárias no mesmo grau. É, então, necessário determinar que características devem estar presentes no produto e em que grau.

A avaliação do grau em que cada característica de qualidade deve estar presente em um produto de software deve ser determinada através de avaliações de seus usuários.

OLIVEIRA (1999) definiu cinco graus de importância para as características de qualidade definidas pela Norma ISO/IEC 9126: Sem relevância, Pouco relevante, Relevante, Muito relevante e Imprescindível.

Durante esta atividade da instanciação, o gerente do projeto deve selecionar, entre os potenciais usuários do produto, um grupo de avaliadores. Estes avaliadores devem atribuir para cada característica de qualidade definida na Norma ISO/IEC 9126, o grau em que a referida característica é necessária na aplicação a ser desenvolvida. Com base nessas avaliações é, então, identificado o grau de relevância de cada característica para o projeto em questão.

Para se chegar a este grau de relevância, a partir do julgamento individual, é utilizada uma adaptação da proposta de BELCHIOR (1997) realizada por OLIVEIRA (1999), para tratamento dos dados coletados de especialistas na avaliação de cada característica de qualidade.

O consenso das diferentes opiniões é feito a partir de uma função *fuzzy* específica, envolvendo um cálculo do grau de concordância, a geração de uma matriz de concordância, a determinação da concordância relativa, o cálculo do coeficiente de consenso dos especialista e a determinação do valor *fuzzy* da característica de qualidade. A definição do perfil dos especialistas é utilizada para definir um peso para o especialista através de um Questionário de Identificação do Perfil de Especialista (QIPE). O QIPE possui um conjunto de questões, cujo objetivo é avaliar cada especialista e determinar sua importância definindo seu peso.

- **Nível de Garantia da Qualidade Necessário**

De acordo com BOEGH *et al.* (1993), os produtos de software podem ser classificados em níveis considerando-se o dano causado por falhas e, portanto, o grau necessário de garantia da qualidade. Tais níveis variam de A a D, sendo A o nível mais alto e D o nível mais baixo (tabela 5.2). Os níveis definem o grau de rigor com que a qualidade do produto deverá ser avaliada.

Tabela 5.2 – Classificação do projeto de acordo com o nível de avaliação (BOEGH *et al.*, 1993)

Nível	Ambiente	Pessoas	Economia	Aplicação
D	Pequeno dano a propriedades	Sem risco para pessoas	Perda econômica desprezível	• Lazer • Uso doméstico
C	Dano a propriedades	Poucas pessoas mutiladas	Perda econômica significativa	• Alarme de incêndio • Controle de processos
B	Dano recuperável ao ambiente	Risco para vidas humanas	Grande perda econômica	• Sistemas médicos • Sistemas financeiros
A	Dano irrecuperável ao ambiente	Muitas pessoas mortas	Desastre financeiro	• Controle de trens • Sistemas nucleares

5.2.2 Planejamento do Processo

Após a caracterização do projeto, tem-se a atividade "**Planejar Processo**", que envolve as seguintes sub-atividades:

- **Incluir Atividades do Tipo de Software**

Nesta atividade, caso as atividades de desenvolvimento específicas ao tipo de software a ser desenvolvido não tiverem sido incluídas durante a definição do processo padrão ou do processo especializado, estas deverão ser consideradas para inclusão neste momento. Se o ambiente possuir o conhecimento das atividades que devem ser realizadas no desenvolvimento deste tipo de software, estas deverão ser sugeridas ao gerente do projeto, que poderá incluí-las no processo. Para apoiar essa atividade, o gerente poderá consultar processos instanciados de projetos anteriores que desenvolveram o mesmo tipo de software e utilizaram o mesmo processo especializado, para verificar as atividades que foram inseridas no processo e em que ordem.

- **Definir Modelo de Ciclo de Vida**

Com base nas características do projeto, são oferecidas ao gerente do projeto as opções de modelo de ciclo de vida mais apropriadas ao projeto. O gerente do projeto deve, então, selecionar o modelo de ciclo de vida que julgar mais adequado.

Pela dificuldade de se determinar o modelo de ciclo de vida ideal para um projeto, recomenda-se a adoção daquele modelo que possuir o maior somatório de equivalências quando da adequação dos seus valores e critérios àqueles definidos para um projeto (MACHADO, 2000).

A tabela 5.3 mostra a avaliação dos critérios listados anteriormente (tabela 5.1) para alguns dos modelos de ciclo de vida encontrados na literatura.

Tabela 5.3 – Critérios e valores para escolha de um Modelo de Ciclo de Vida (MACHADO, 2000;ALEXANDER e DAVIS, 1991; PRESSMAN, 2001; PFLEEGER, 2001; FALBO, 1998)

	Modelo de Ciclo de Vida				
	Cascata	**Incremental**	**Evolutivo**	**RAD**	**Espiral**
Critérios relacionados aos usuários					
Experiência dos usuários no domínio da aplicação	Alta	Média ou Alta	Qualquer	Média ou Alta	Qualquer
Facilidade dos usuários em expressar requisitos	Alta	Média ou Alta	Qualquer	Alta	Qualquer
Grau de acesso aos usuários	Alto	Médio ou Alto	Médio ou Alto	Alto	Médio ou Alto
Nível de mudanças geradas no trabalho dos usuários	Baixo	Qualquer	Qualquer	Baixo	Qualquer
Critérios relacionados ao problema					
Grau de maturidade do domínio da aplicação	Alto	Médio ou Alto	Qualquer	Alto	Qualquer
Complexidade do problema	Baixa	Baixa ou Média	Qualquer	Baixa	Qualquer
Freqüência de mudanças nos requisitos	Baixa	Baixa ou Média	Qualquer	Baixa	Qualquer
Grau de magnitude das mudanças nos requisitos	Baixo	Baixo	Qualquer	Baixo	Qualquer
Grau de modularidade do problema	Qualquer	Médio ou Alto	Médio ou Alto	Alto	Qualquer
Critérios relacionados ao produto					
Tamanho da aplicação	Pequeno	Qualquer	Qualquer	Qualquer	Qualquer
Grau de complexidade da aplicação	Baixo	Baixo ou Médio	Qualquer	Baixo ou Médio	Qualquer

Critérios relacionados aos recursos					
Disponibilidade de recursos humanos	Alta	Qualquer	Qualquer	Adequada ou Alta	Qualquer
Disponibilidade de recursos financeiros	Adequada	Qualquer	Qualquer	Adequada	Qualquer
Critérios relacionados à equipe de desenvolvimento					
Experiência da equipe de desenvolvimento no domínio da aplicação	Média ou Alta	Média ou Alta	Qualquer	Média ou Alta	Qualquer
Experiência da equipe de desenvolvimento em engenharia de software	Qualquer	Qualquer	Qualquer	Alta	Qualquer
Nível de experiência da gerência	Qualquer	Médio ou Alto	Médio ou Alto	Alto	Médio ou Alto
Capacidade de gerenciamento de múltiplas equipes	Qualquer	Qualquer	Qualquer	Sim	Qualquer
Critérios relacionados ao desenvolvimento					
Há necessidade de entrega de produtos intermediários	Não	Sim	Sim	Não	Sim
Há necessidade do software ser colocado em uso rapidamente com funcionalidade total ou parcial?	Não	Qualquer	Qualquer	Sim	Qualquer
Grau dos riscos técnicos	Baixo	Baixo ou Moderado	Baixo ou Moderado	Baixo	Qualquer
Necessidade de interface com sistemas existentes	Não	Qualquer	Qualquer	Não	Qualquer
Uso de tecnologia inovadora	Não	Qualquer	Qualquer	Não	Qualquer

Caso o modelo de ciclo de vida seja iterativo, o gerente de projeto deve definir o número de iterações a serem realizadas (no caso do modelo evolutivo, o número de ciclos de desenvolvimento; no caso do modelo incremental, o número de incrementos; no caso do modelo RAD, o número de módulos a serem desenvolvidos em paralelo). Para apoiar essa definição, o gerente pode consultar o número de iterações realizadas em projetos anteriores que utilizaram o mesmo processo especializado e modelo de ciclo de vida.

- **Mapear Atividades do Modelo de Ciclo de Vida**

Neste momento as atividades do processo especializado e as atividades incluídas, nas atividades anteriores do processo, são organizadas de acordo com o modelo de ciclo de vida selecionado. Para isto deve-se:

- Verificar a existência, no repositório da organização, de um mapeamento feito para um projeto anterior, que considerou o mesmo processo especializado e o mesmo modelo de ciclo de vida. Caso tal mapeamento exista, ele é recuperado para ser reutilizado.
- Realizar Mapeamento, caso não tenha sido encontrado no repositório um mapeamento anterior. Um modelo de ciclo de vida define um série de etapas a serem realizadas para o desenvolvimento de um produto de software. As atividades do processo são organizadas de acordo com essas etapas, que podem ser estruturadas de maneira seqüencial e, nesse caso, são realizadas apenas uma vez, ou de maneira iterativa, sendo realizadas várias vezes. Neste momento deve-se definir as atividades que serão realizadas em cada uma dessas etapas e, quando apropriado, o número de iterações.

Cada um dos modelos de ciclo de vida descritos no capítulo 2 propõe uma maneira diferente de estruturar as etapas do desenvolvimento.

No modelo cascata as atividades do processo são executadas em uma única etapa seqüencial.

No modelo incremental há uma etapa inicial seqüencial, na qual pode ser realizada a especificação de requisitos e o projeto da arquitetura. Os requisitos são, então, segmentados e alocados a diferentes versões. Em seguida temos uma etapa iterativa, onde cada iteração corresponde a um incremento, ou seja, uma versão

operacional do software que implementa um subconjunto dos requisitos, de acordo com a definição feita na etapa inicial. Ao final do último incremento, todos os requisitos do software estarão implementados.

No modelo RAD há uma etapa inicial seqüencial, semelhante ao modelo Incremental, na qual são definidos os módulos que compõem o produto. A próxima etapa será realizada de forma paralela para cada um dos módulos do software. Uma equipe é alocada para cada módulo e o desenvolvimento de cada um deles é realizado de acordo com as atividades previstas no processo. Ao final do desenvolvimento dos módulos, é realizada uma nova etapa seqüencial, para integração dos módulos.

Caso o projeto seja desenvolvido segundo o paradigma orientado a objetos, o processo utiliza a estrutura do RUP. Nesse caso, as atividades do processo são estruturadas em quatro fases iterativas: Construção, Elaboração, Construção, Transição.

A partir do conhecimento sobre a estrutura de cada modelo de ciclo de vida, a ferramenta pode definir as atividades que compõem cada etapa, realizando um mapeamento inicial, que poderá ser modificado pelo gerente do projeto. Este mapeamento é, então, salvo para reutilização em novos projetos.

- **Incluir/Excluir Atividades**

Neste momento o gerente do projeto pode incluir novas atividades no processo, quando julgar necessário, baseando-se nas atividades incluídas em projetos anteriores ou definindo uma nova atividade. O gerente do projeto avalia, também, a pertinência das atividades não obrigatórias do processo especializado[1] para identificar se existem atividades que podem ou devem ser excluídas sem prejuízo para o projeto.

ODDO *et al.* (2003) estabeleceram uma correlação entre as características desejadas para o produto e as atividades e sub-atividades (tarefas) do processo de desenvolvimento. Para isto foram utilizadas avaliações de especialistas, considerando

---

[1] A identificação das atividades obrigatórias é realizada durante a configuração do ambiente. Consiste em definir quais são as atividades essenciais para qualquer projeto da organização, independente de suas características (tamanho, complexidade, etc.). Estas atividades não poderão ser excluídas do processo durante a instanciação. Caso um modelo de maturidade tenha sido selecionado para o processo, as atividades que garantem a conformidade com o nível de maturidade desejado serão automaticamente marcadas como obrigatórias.

que seu nível de conhecimento teórico e/ou experiência fornece a cada um deles subsídios que permitam um julgamento se não absolutamente preciso, ao menos cercado de um razoável grau de consistência. Os resultados deste trabalho são disponibilizados para consulta por parte do gerente do projeto, fornecendo apoio na seleção das atividades do processo instanciado.

Apresentamos, na tabela 5.4, o relacionamento entre as tarefas do processo de desenvolvimento da Norma NBR ISO/IEC 12207 e as características de qualidade apresentadas na Norma ISO/IEC 9126.

Tabela 5.4 – Relacionamento entre as tarefas do processo de desenvolvimento e as características de qualidade (ODDO, 2003)

Tarefas descritas na Norma NBR ISO/IEC 12207	Características da Qualidade de Produtos de Software (ISO/IEC 9126)					
	Funcionalidade	Confiabilidade	Usabilidade	Eficiência	Manutenibilidade	Portabilidade
Especificação dos requisitos conforme uso pretendido	**IMP**	MR	MR	MR	RL	RL
Estabelecimento de arquitetura de alto nível p/o sistema	MR	MR	RL	MR	MR	RL
Estabelecimento de requisitos de software conforme a ISO 9126	MR	**IMP**	MR	MR	MR	MR
Identificação de componentes e projeto de alto nível de interfaces e Banco de Dados	MR	MR	MR	MR	MR	MR
Projeto detalhado de componentes, interfaces e Banco de Dados e requisitos/cronograma de testes	MR	MR	MR	MR	MR	MR
Teste de cada unidade Software e Base de Dados e avaliação dos resultados	MR	**IMP**	MR	MR	RL	PR
Integração e teste de agregação das unidades de software	MR	**IMP**	RL	MR	RL	RL
Testes de qualificação dos itens de software	MR	MR	MR	MR	RL	RL
Integração itens de configuração de software, hardware, manuais, etc.	RL	MR	MR	MR	MR	RL
Teste de qualificação do sistema	MR	**IMP**	MR	MR	RL	RL
Desenvolvimento de Plano para Instalação do Software	PR	RL	RL	RL	PR	RL
Provisão de treinamento e suporte ao adquirente	RL	RL	**IMP**	RL	PR	PR
Desenvolvimento de Plano de Documentação do Ciclo de Vida	RL	RL	RL	RL	MR	RL
Produção e fornecimento e Manutenção de documentos conforme o Plano de Documentação	RL	RL	MR	RL	**IMP**	RL
Desenvolvimento de Plano de Gerência de Configuração	RL	MR	RL	RL	**IMP**	RL
Estabelecimento de sistemática para identificação de itens e versões	RL	MR	RL	RL	**IMP**	RL
Execução das atividades de controle da configuração	RL	MR	RL	RL	**IMP**	RL
Preparação de registros e relatórios da situação da configuração	RL	RL	PR	RL	MR	RL

Determinação. e garantia da completeza funcional/física itens software	**IMP**	MR	MR	MR	MR	PR
Controle formal da liberação e distribuição dos produtos de software	RL	RL	RL	PR	RL	PR
Desenvolvimento de Plano de Gerência da Qualidade conforme especificado	MR	MR	MR	MR	MR	MR
Garantia de aderência dos produtos e documentação a planos, contratos e requisitos	MR	MR	MR	MR	MR	RL
Garantia de aderência do processo aos planos, contratos e requisitos	MR	MR	RL	MR	MR	RL
Cumprimento, através da Gerência de Qualidade, dos requisitos da Norma ISO 9001	MR	MR	RL	RL	MR	RL
Execução de Verificação formal segundo critérios estabelecidos	MR	MR	RL	MR	MR	RL
Validação do produto de software	**IMP**	MR	MR	MR	MR	MR
Avaliação e discussão entre as partes da situação do projeto	MR	MR	MR	MR	RL	RL
Promoção de revisões conjuntas técnicas p/ avaliação de produtos e serviços de software	MR	MR	MR	MR	MR	RL
Condução de auditorias em marcos determinados	MR	MR	RL	MR	RL	RL
Estabelecimento dos requisitos de processo e de viabilidade e disponibilidade de recursos	RL	RL	RL	RL	RL	RL
Preparação de planos para a execução do processo	MR	MR	RL	RL	MR	RL
Monitoramento da execução do processo	MR	MR	RL	MR	MR	RL
Verificação dos resultados de avaliações, atividades e tarefas	MR	MR	RL	MR	MR	RL
Definição e Planejamento de infra-estrutura compatível com os requisitos do processo	RL	MR	RL	MR	MR	RL
Planejamento da alocação e treinamento de pessoal conforme requisitos do projeto	MR	MR	MR	MR	MR	RL
Implementação Plano de Treinamento garantindo disponibilização tempestiva de equipe treinada	MR	MR	MR	MR	RL	RL

Legenda: NR – Nenhuma Relevância PR – Pouca Relevância RL – Relevante MR – Muito Relevante IMP – Imprescindível

- **Selecionar Técnicas de Avaliação**

Quando o projeto é caracterizado, o software é classificado em um nível de acordo com o tipo de aplicação e impacto em caso de falha. Baseado neste nível, BOEGH *et al.* (1993) propõem que sejam utilizadas técnicas de avaliação da qualidade apropriadas para o projeto. Quanto maior for o risco e o impacto de dano, mais rígidas são as técnicas de avaliação. A tabela 5.5 apresenta as técnicas indicadas para avaliação de cada característica de qualidade, de acordo com o nível do produto. As técnicas são cumulativas, ou seja, um projeto classificado como sendo de nível A deverá contar com as técnicas indicadas para este nível e todos os níveis inferiores (B, C e D).

Tabela 5.5 – Técnicas de avaliação de acordo com o nível do projeto (BOEGH *et al.*, 1993)

Característica	Nível D	Nível C	Nível B	Nível A
Funcionalidade	Teste funcional	+ Inspeção de documentos	+ Teste de componentes	+ Prova formal
Confiabilidade	Facilidades da linguagem de programação	+ Análise da tolerância a falhas	+ Modelos de crescimento de confiabilidade	+ Prova formal
Usabilidade	Inspeção da interface com o usuário	+ Aderência a padrões de interface	+ Teste em laboratório	+ Modelos mentais do usuário
Eficiência	Medição do tempo de execução	+ Benchmark	+ Análise da complexidade de algoritmos	+ Análise de desempenho
Manutenibilidade	Inspeção de documentos	+ Análise estática	+ Análise do processo de desenvolvimento	+ Avaliação da rastreabilidade
Portabilidade	Análise da instalação	+ Aderência a normas de programação	+ Avaliação das restrições do ambiente	+ Avaliação do projeto de programas

Como etapas finais tem-se a seleção/associação de ferramentas, a visualização do processo instanciado, sua impressão caso desejado e, finalmente, a instanciação do ADSOrg.

Como se pode observar, a abordagem utilizada disponibiliza ao gerente do projeto o conhecimento sobre instanciação de processos acumulado pelos vários gerentes de uma organização, considerando projetos anteriores.

No contexto do Projeto TABA foi, também, construída a ferramenta *Acknowledge* (MONTONI *et al.*, 2002), que tem como objetivo apoiar todas as atividades do processo de aquisição do conhecimento, incluindo aquisição do conhecimento ao longo da execução de processos, filtro e empacotamento. Através de uma interface da ferramenta *AdaptPro* com *Acknowledge*, gerentes de projeto podem registrar idéias, dúvidas, sugestões e lições aprendidas durante a realização das atividades de instanciação do processo. Estas idéias e lições aprendidas serão, posteriormente, avaliadas (filtradas) para determinar se elas serão incorporadas ao repositório da organização. Em caso positivo são empacotadas e disponibilizadas. As dúvidas e sugestões são utilizadas para melhoria do processo. A partir da disponibilização do conhecimento no repositório da organização, este é disponibilizado ao gerente do projeto, durante a realização da atividade ao qual é pertinente.

5.3 A Ferramenta AdaptPro

Buscando-se apoiar a abordagem descrita na seção anterior, a ferramenta *AdaptPro* foi definida e implementada. *AdaptPro* apóia as atividades de caracterização do projeto, planejamento do processo e instanciação de um ADSOrg, definidas no processo de instanciação proposto.

A ferramenta é disponibilizada nos ambientes configurados para uma organização específica e, desta forma, possibilita a utilização do conhecimento organizacional armazenado no repositório da organização. Dentro do contexto de um ambiente configurado, os usuários da ferramenta são os gerentes de projeto que a utilizam quando um novo projeto é definido e torna-se necessário adaptar o processo especializado ao projeto e instanciar um ADSOrg.

As funcionalidades disponíveis na ferramenta são descritas nessa seção. O anexo 1 apresenta as classes que foram incluídas no modelo da Estação TABA para representar os conceitos envolvidos na adaptação de processos e instanciação de ADSOrg para projetos específicos.

AdaptPro baseia-se no processo de instanciação de processos de software definido no capítulo 4 e guia o usuário durante a realização das atividades deste processo.

Na interface da ferramenta, o lado esquerdo mostra o processo de instanciação, com suas atividades e sub-atividades, e o lado direito exibe a atividade que está sendo realizada pelo usuário. Os ícones localizados abaixo da barra de título permitem a consulta do conhecimento sobre o processo armazenado no repositório da organização e o registro de conhecimento, através de uma interface com a ferramenta de Aquisição de Conhecimento *Acknowledge* (MONTONI *et al.*, 2002).

A seguir são apresentados dois exemplos de utilização da ferramenta, passo a passo. O primeiro exemplo apresenta a instanciação de um processo para um projeto que utiliza o paradigma estruturado. O segundo exemplo apresenta a instanciação de um processo para um projeto que utiliza o paradigma orientado a objetos.

5.3.1 Projeto Estruturado

Neste exemplo, o ambiente que foi configurado pertence à Fundação Bahiana de Cardiologia e o novo projeto para o qual será instanciado o ADSOrg se chama CardioNet. Este projeto tem como objetivo construir um novo ambiente para o meta-ambiente CardioEducar[2]. Como o meta-ambiente, CardioNet será baseado na Web e, também, será desenvolvido em ASP usando o paradigma de desenvolvimento estruturado.

5.3.1.1 Caracterização do Projeto

A figura 5.1 mostra a tela inicial da ferramenta, na qual é feito o cadastro do novo projeto para o qual será adaptado um processo especializado e gerado um ADSOrg. O gerente preenche as seguintes informações a respeito do novo projeto: nome, descrição, tipo de software a ser desenvolvido e natureza do projeto. As opções disponíveis para o paradigma refletem as características do desenvolvimento de software na organização, ou seja, apenas os paradigmas selecionados durante a configuração do ambiente estarão disponíveis.

---

[2] CardioEducar é um meta-ambiente educacional para cardiologia desenvolvido pela COPPE/UFRJ, UERJ e Fundação Bahiana de Cardiologia/UFBA.

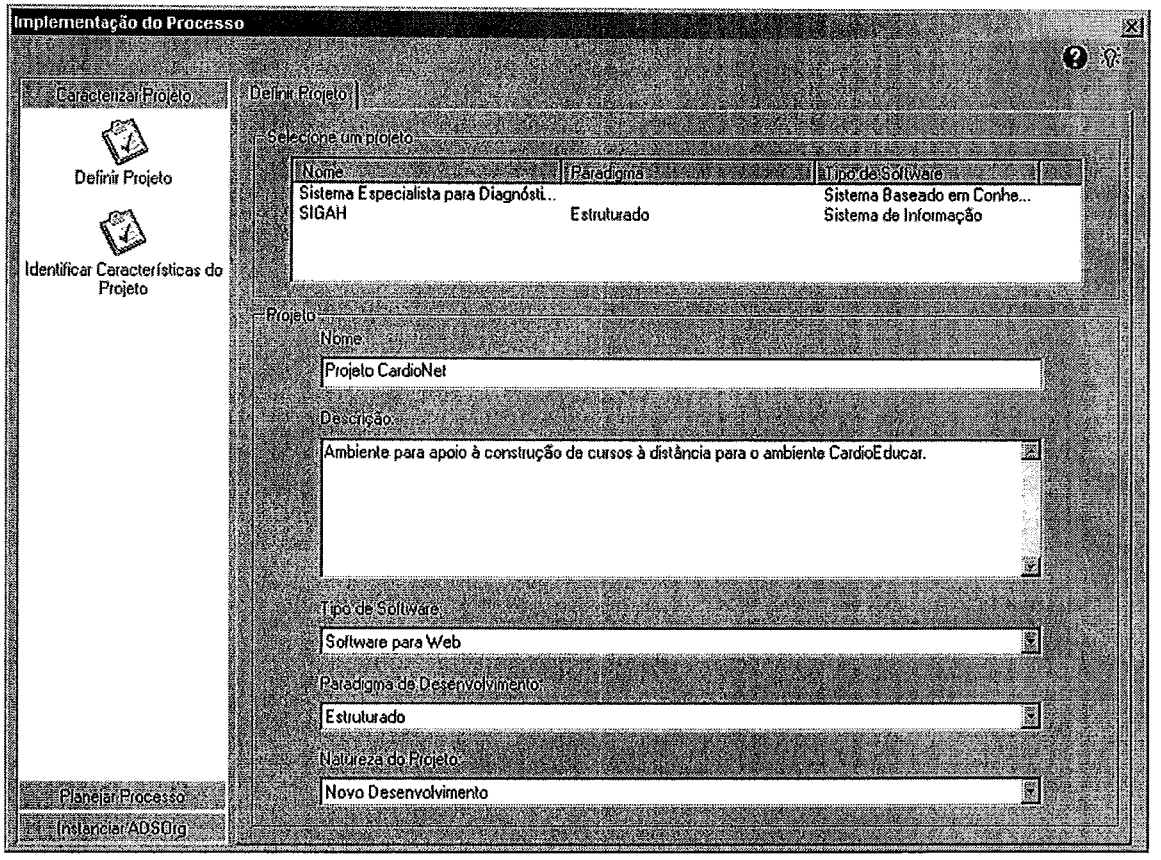


Figura 5.1 – Tela de cadastro do projeto CardioNet

A próxima atividade do processo é "Identificar Características do Projeto". Conforme a abordagem proposta, o usuário preenche um questionário composto de uma série de critérios que serão utilizados posteriormente para apoiar a definição de um modelo de ciclo de vida apropriado ao projeto: critérios relacionados aos usuários, critérios relacionados ao problema, critérios relacionados ao produto, critérios relacionados aos recursos, critérios relacionados à equipe de desenvolvimento e critérios relacionados ao desenvolvimento. Um dos aspectos considerados na abordagem proposta é a classificação do produto de acordo com o nível de garantia da qualidade necessário, considerando-se os possíveis danos causados por uma falha do software. Para identificar este nível, o gerente deve avaliar os possíveis danos causados a pessoas, ao ambiente e à economia em caso de falha.

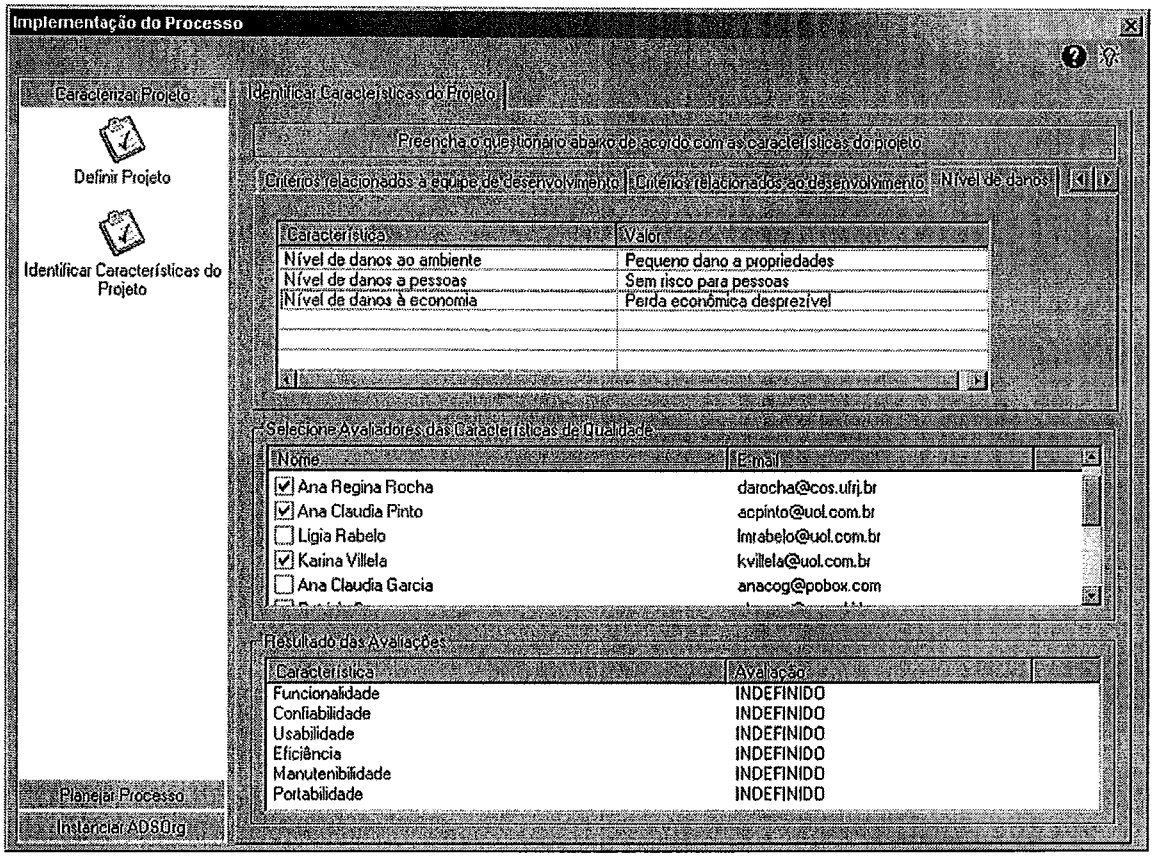


Figura 5.2 – Tela de caracterização do projeto e seleção de avaliadores

A seguir, o gerente seleciona, entre os indivíduos da organização, um grupo de avaliadores[3] que deverá identificar as características de qualidade necessárias ao produto. São consideradas apenas as características de qualidade descritas na norma ISO/IEC 9126. Neste momento ainda não foi realizada nenhuma avaliação e, portanto, a relevância das características de qualidade para o produto ainda é indefinida. A figura 5.2 mostra a tela que reflete esta situação.

Cada avaliador selecionado recebe um e-mail solicitando que acesse uma página Web, onde deverá ser registrada a sua avaliação e onde deve, inicialmente, preencher um questionário visando identificar o seu perfil, baseado na sua experiência com projetos de software e com qualidade de software (QIPE). Com base neste questionário,

---

[3] Para apoiar a seleção dos avaliadores, o gerente pode utilizar a ferramenta SAPIENS (SANTOS, 2003), que disponibiliza a descrição da estrutura organizacional, identificando competências e habilidades.

será atribuído um peso a cada avaliação, de forma que a opinião de avaliadores mais experientes tenha peso maior do que a de avaliadores com menor experiência. A figura 5.3 mostra o preenchimento do questionário de identificação do perfil do especialista.

Após o preenchimento do questionário, o avaliador seleciona o grau de relevância de cada característica para o projeto em questão, conforme mostra figura 5.4

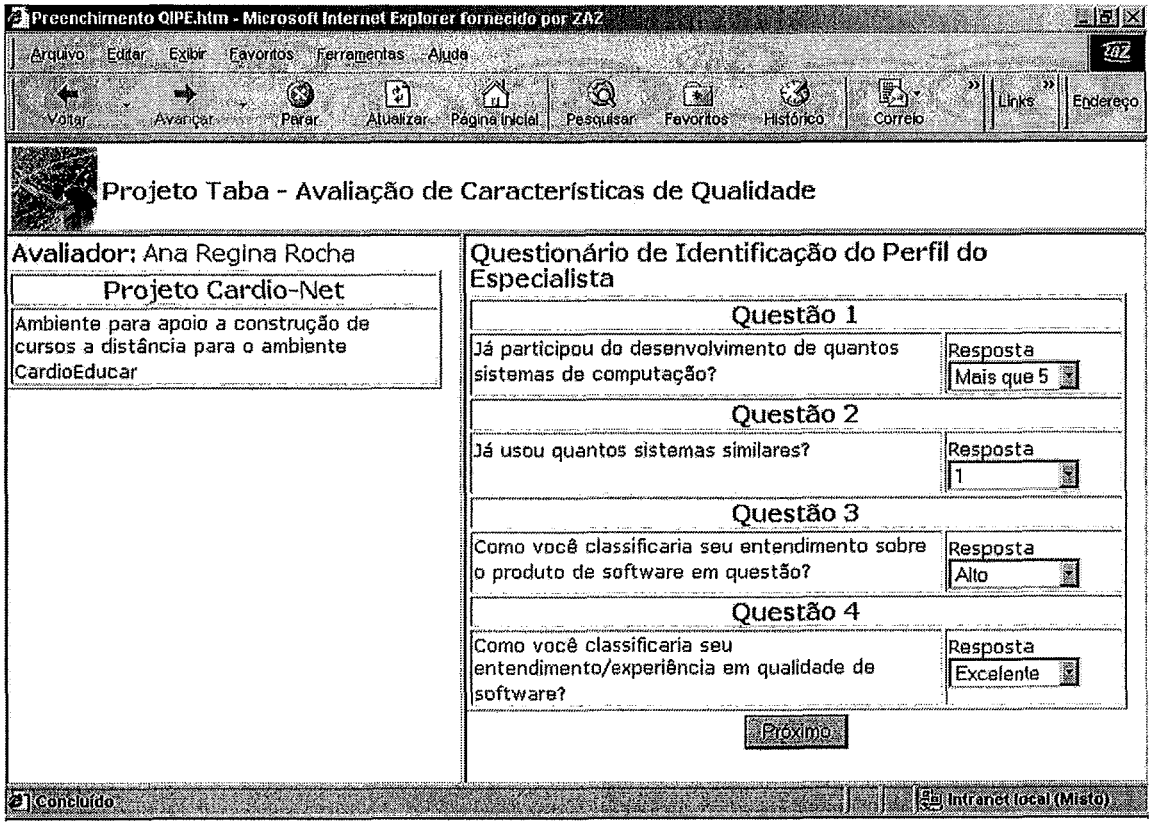


Figura 5.3 – Tela de preenchimento do QIPE

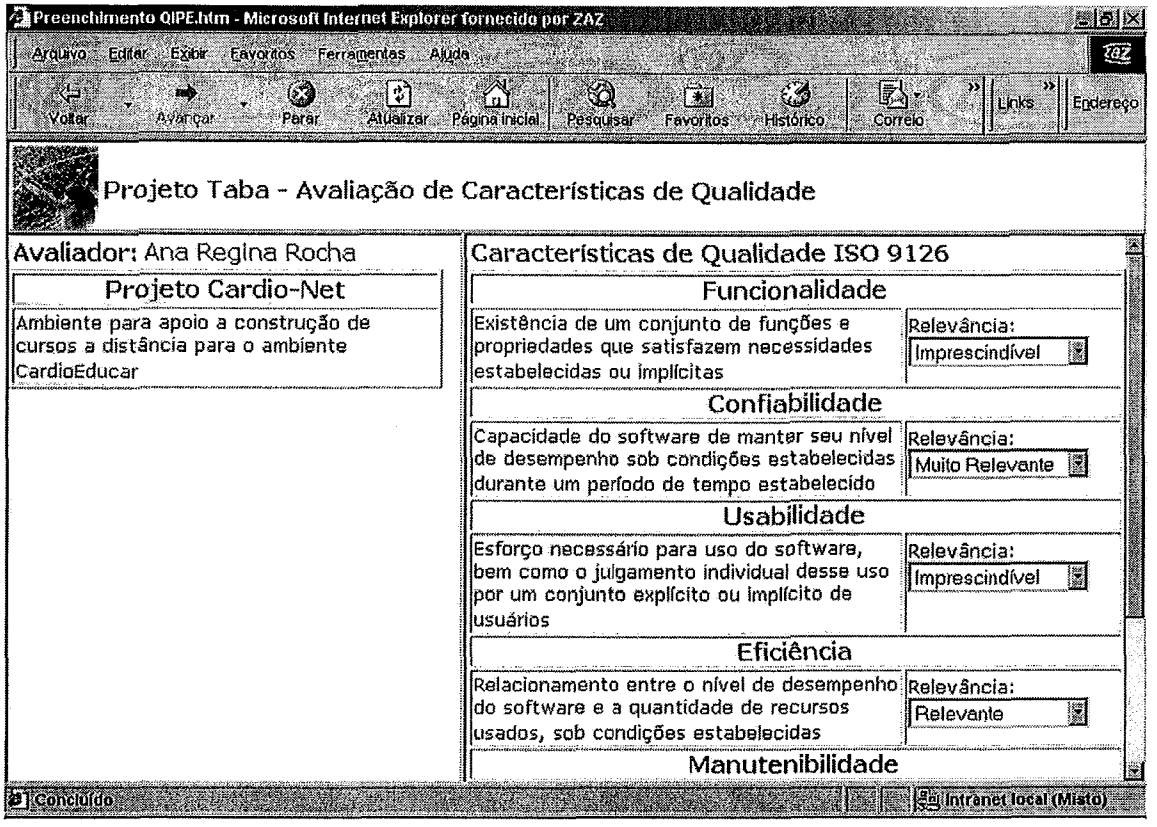


Figura 5.4 – Tela de avaliação das características de qualidade

Cada uma das avaliações registradas é considerada e, com base na experiência de cada um dos avaliadores, a ferramenta chega ao grau de relevância de cada característica. O gerente do projeto visualiza estes graus através da própria ferramenta *AdaptPro*, conforme mostra a figura 5.5.

5.3.1.2 Planejamento do Processo

A próxima atividade do processo é o planejamento do processo instanciado. Para isto, parte-se de um processo especializado e são feitas adaptações no mesmo para adequá-lo às necessidades do projeto. A ferramenta *AdaptPro* identifica o processo especializado adequado ao paradigma de desenvolvimento que será utilizado.

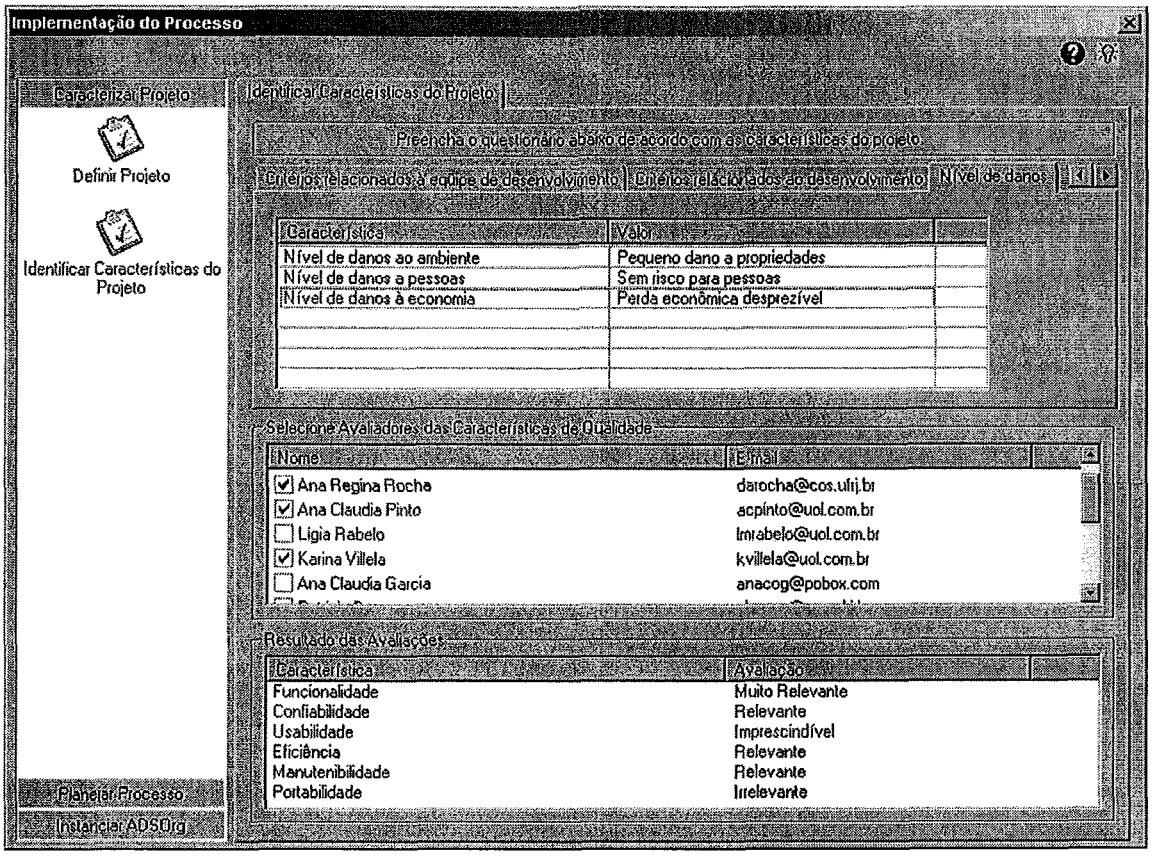


Figura 5.5 – Tela que ilustra o resultado das avaliações

A primeira sub-atividade corresponde à inclusão de atividades relacionadas ao tipo de software a ser desenvolvido[4]. A ferramenta exibe, à esquerda, as atividades indicadas para o desenvolvimento do tipo de software do projeto em questão e, à direita, as atividades do processo especializado já selecionado para o projeto. Para incluir uma atividade relacionada ao tipo de software no processo, o gerente seleciona a atividade e clica no botão de inclusão. Para apoiar a decisão sobre a inclusão destas atividades, é oferecida ao gerente a opção de consultar processos instanciados de projetos anteriores com o mesmo tipo de software e que utilizaram o mesmo processo especializado,

[4] Esta atividade é realizada apenas para as organizações que desenvolvem vários tipos diferentes de software. Nos casos em que a organização desenvolve apenas um tipo de software, as atividades relacionadas a este tipo de software já estão incluídas nos processos padrão e especializado da organização.

quando estes existirem. A figura 5.6 mostra a tela da atividade "Incluir Atividades do Tipo de Software".

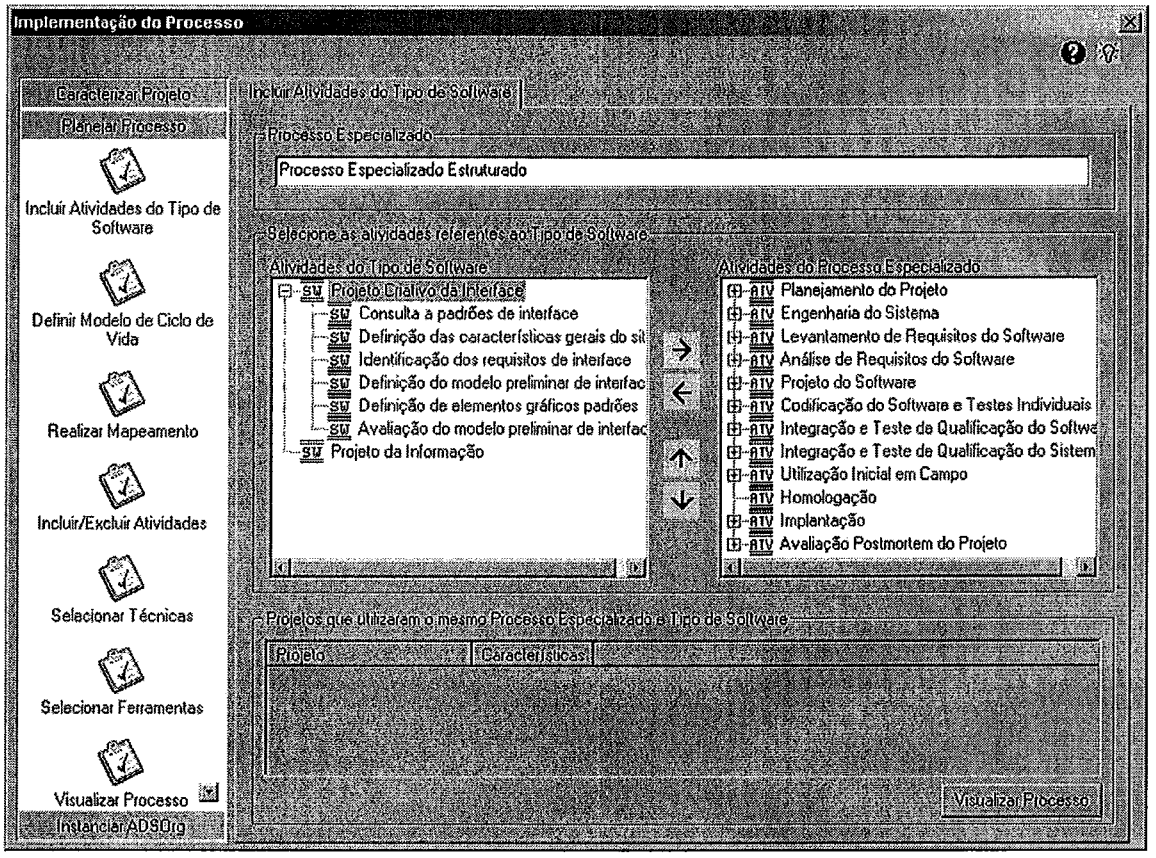


Figura 5.6 – Tela da atividade Incluir Atividades do Tipo de Software

Neste exemplo são exibidas as atividades indicadas para o desenvolvimento de software para Web e o processo especializado selecionado (processo estruturado). O gerente seleciona a atividade "Projeto Criativo da Interface" e a inclui na lista de atividades do processo.

A atividade seguinte corresponde à definição do modelo de ciclo de vida a ser utilizado no projeto. Com base na caracterização do projeto, a ferramenta determina a adequação ao projeto dos modelos de ciclo de vida disponíveis no repositório. A adequação de um modelo de ciclo de vida ao projeto representa o percentual de características do projeto que têm correspondência com as características do modelo, de acordo com a tabela 5.3. *AdaptPro* indica, ainda, quais são as características do projeto que tornam o modelo selecionado apropriado para ele.

Caso adequado para o modelo de ciclo de vida selecionado, o gerente deve informar o número de iterações a serem realizadas. Para apoiar essa decisão, a ferramenta exibe o número de iterações realizadas em projetos anteriores que utilizaram o mesmo processo especializado, tipo de software e modelo de ciclo de vida, (quando existirem). As características do projeto são apresentadas para que o gerente possa basear sua decisão em projetos similares ao projeto em questão.

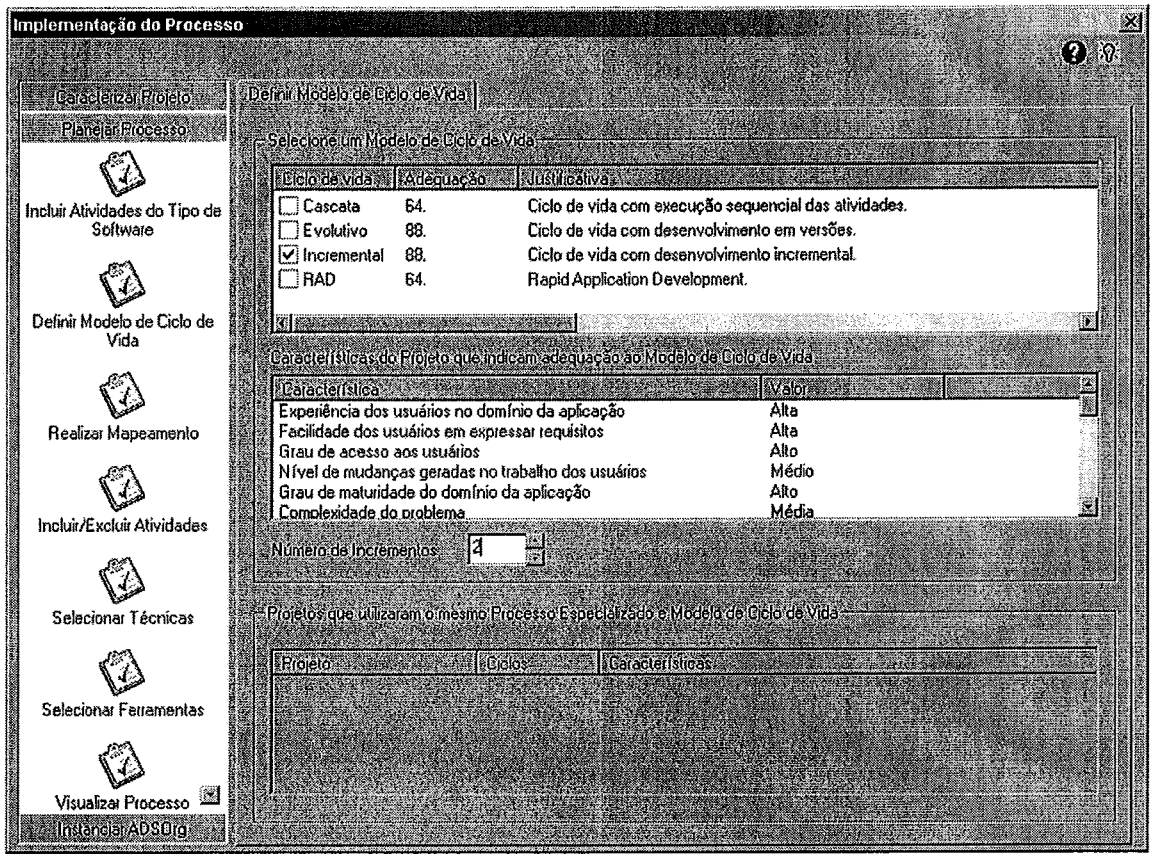


Figura 5.7 – Tela de definição do modelo de ciclo de vida

Neste exemplo o usuário selecionou o modelo de ciclo de vida incremental, que possui a mais alta adequação. As características do projeto que justificam essa escolha estão listadas. Em seguida o gerente informa o número de incrementos a serem realizados no projeto. Não existe nenhum projeto realizado pela organização que tenha utilizado o processo especializado estruturado e o modelo de ciclo de vida incremental. A tela 5.8 ilustra o exemplo.

O passo seguinte é a realização do mapeamento das atividades do processo para o modelo de ciclo de vida selecionado. A estrutura do modelo incremental descreve uma

etapa inicial (quando é realizado o planejamento do projeto) seguida por incrementos. Cada incremento implementa um subconjunto dos requisitos do sistema. Na atividade anterior, o gerente definiu que o software será desenvolvido em dois incrementos. Agora ele define os objetivos de cada incremento e que atividades serão executadas em cada etapa do projeto. A ferramenta busca, então, no repositório da organização por um mapeamento anterior realizado para o mesmo processo especializado e o mesmo modelo de ciclo de vida. Nesse caso, não existe um mapeamento anterior e a ferramenta, com base na estrutura do modelo incremental, propõe um mapeamento inicial, ou seja, propõe quais atividades devem ser executadas apenas no início do projeto e quais atividades devem ser realizadas em cada incremento.

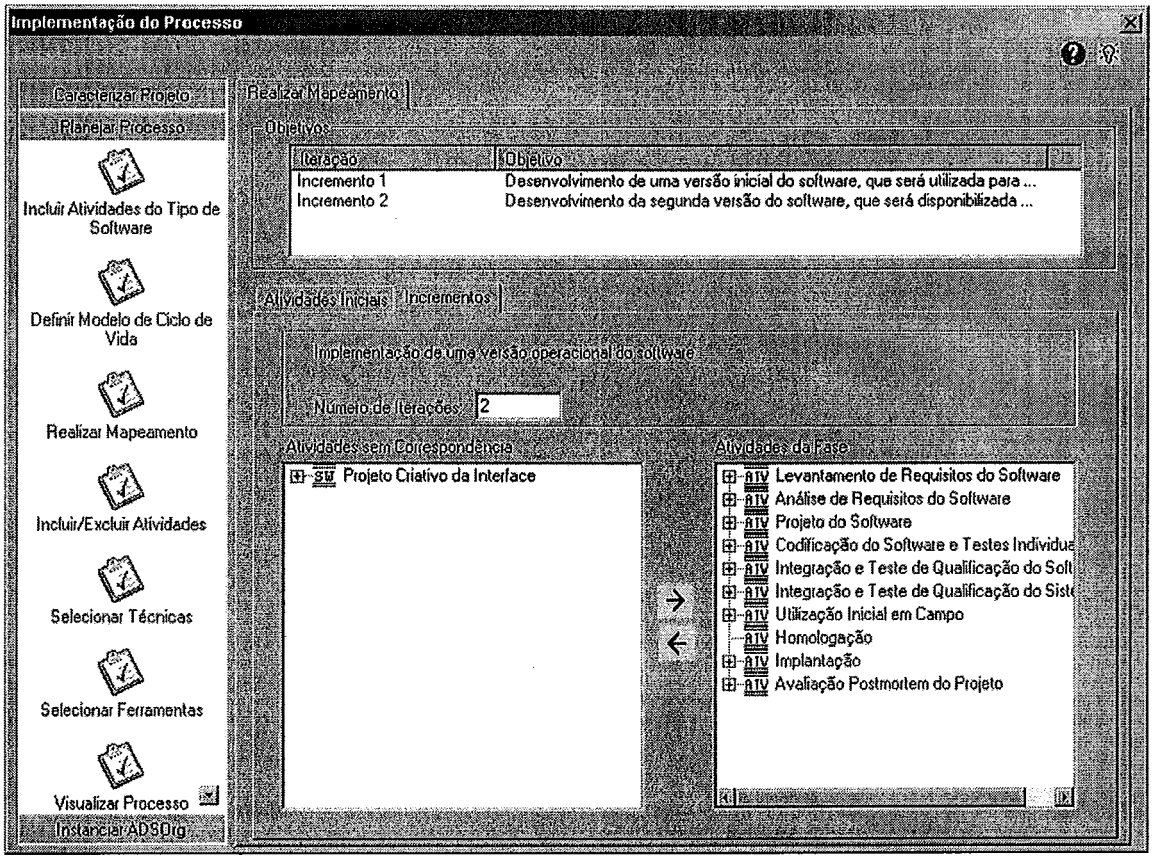


Figura 5.8 – Tela de mapeamento das atividades

Quando a ferramenta não for capaz de identificar em qual etapa será realizada uma determinada atividade, esta será exibida como uma atividade sem correspondência. O gerente pode alterar o mapeamento proposto, se julgar conveniente. Além disso, deve localizar no processo as atividades sem correspondência. A figura 5.9 exibe a tela com o

mapeamento inicial realizado pela ferramenta. Cada pasta representa uma etapa do processo. A figura 5.9 mostra o detalhamento da pasta incrementos, exibindo as atividades que serão realizadas em cada incremento. Serão realizadas duas iterações, conforme definido na atividade anterior.

No caso ilustrado por este exemplo, a ferramenta não foi capaz de identificar onde a atividade "Projeto Criativo Inicial" deveria ser realizada. Coube ao gerente determinar que esta atividade será uma das atividades iniciais do projeto. A figura 5.10 mostra a pasta "Atividades Iniciais", após a inclusão da atividade "Projeto Criativo da Interface".

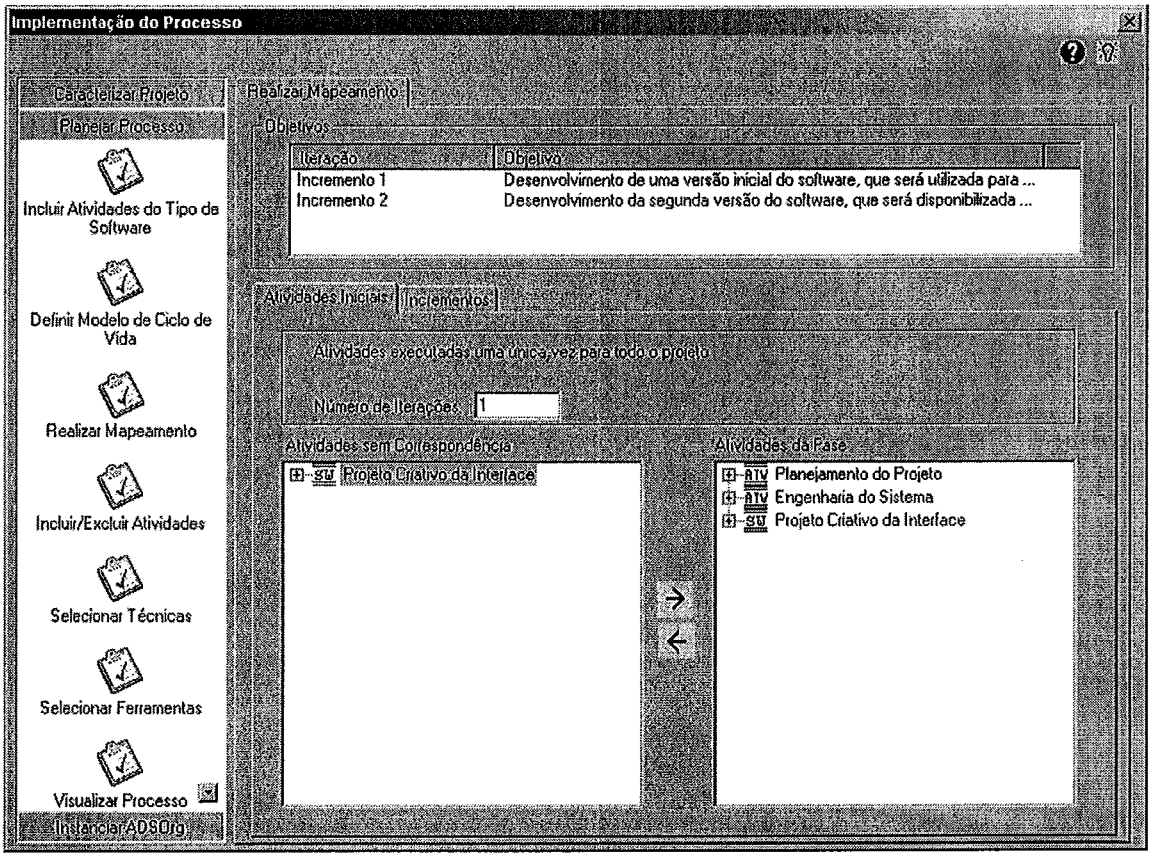


Figura 5.9 – Tela de mapeamento das atividades, após inclusão

O gerente pode, ainda, retirar uma atividade de uma determinada etapa, selecionando-a e clicando no botão de exclusão. Esta atividade passa para a lista de atividades sem correspondência e pode ser incluída em outra etapa caso desejado.

O mapeamento realizado pelo gerente é, então, armazenado no repositório da organização, de forma que, possa ser reutilizado em projetos futuros.

Após a realização do mapeamento, é realizada a atividade "Incluir/Excluir Atividades", caso o gerente julgue necessário incluir novas atividades ao processo, como por exemplo, avaliações, ou a construção de protótipos descartáveis. Para incluir uma nova atividade, o gerente clica no botão "Nova Atividade". A figura 5.11 mostra a tela da atividade "Incluir/Excluir Atividades".

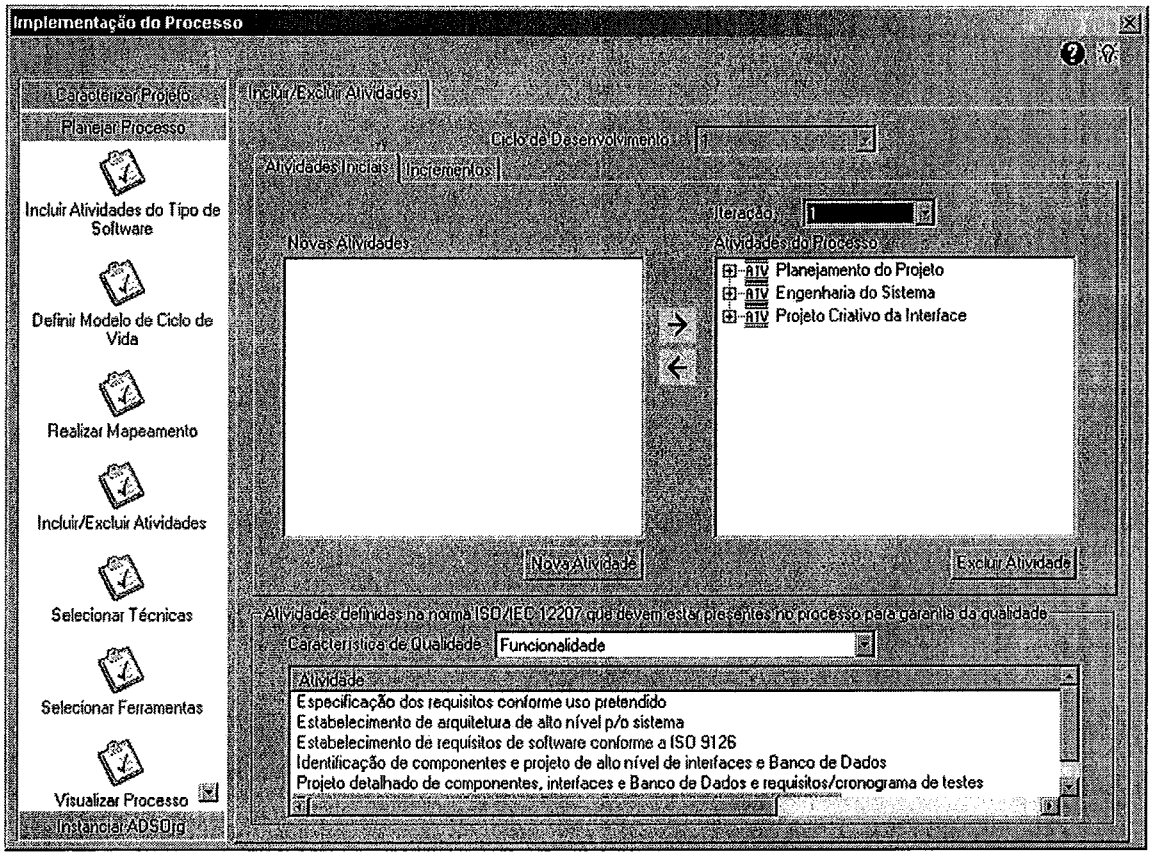


Figura 5.10 – Tela inicial da atividade "Incluir/Excluir Atividades"

Ao clicar no botão "Nova Atividade", é exibida ao gerente a tela de inclusão de atividade, para que o gerente preencha os dados da nova atividade. A figura 5.12 mostra a definição da atividade "Construção de um protótipo do software". A atividade criada será incluída na lista de novas atividades e o gerente poderá incluí-la no processo, utilizando o botão de inclusão.

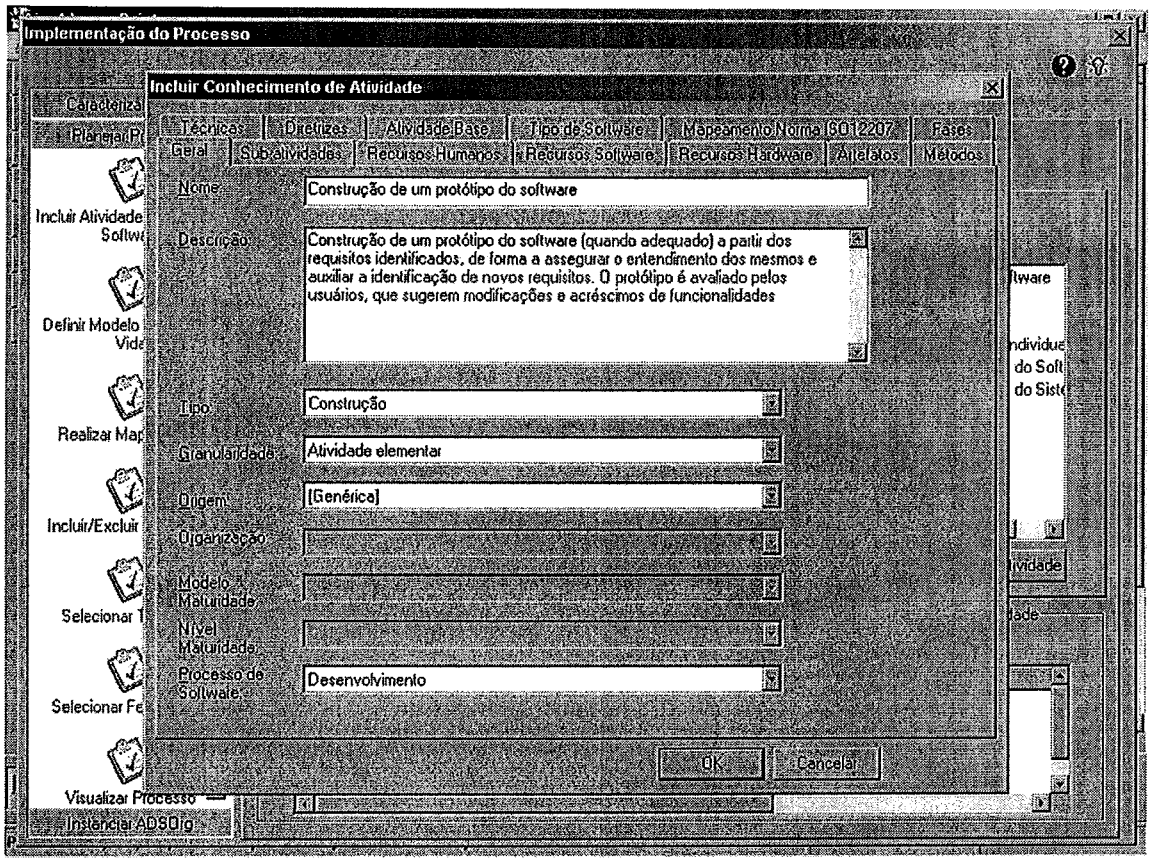


Figura 5.11 – Tela de criação de nova atividade

Em seguida a nova atividade é inserida como sub-atividade da atividade "Levantamento de Requisitos do Software", como mostra a figura 5.13.

O gerente pode, ainda, excluir atividades que não sejam pertinentes ao projeto em questão, desde que estas atividades não sejam consideradas obrigatórias. No exemplo, a atividade "Engenharia de Sistemas" foi considerada desnecessária e foi excluída das Atividades Iniciais. A figura 5.14 mostra o resultado de uma tentativa de excluir a atividade obrigatória "Projeto do Sistema".

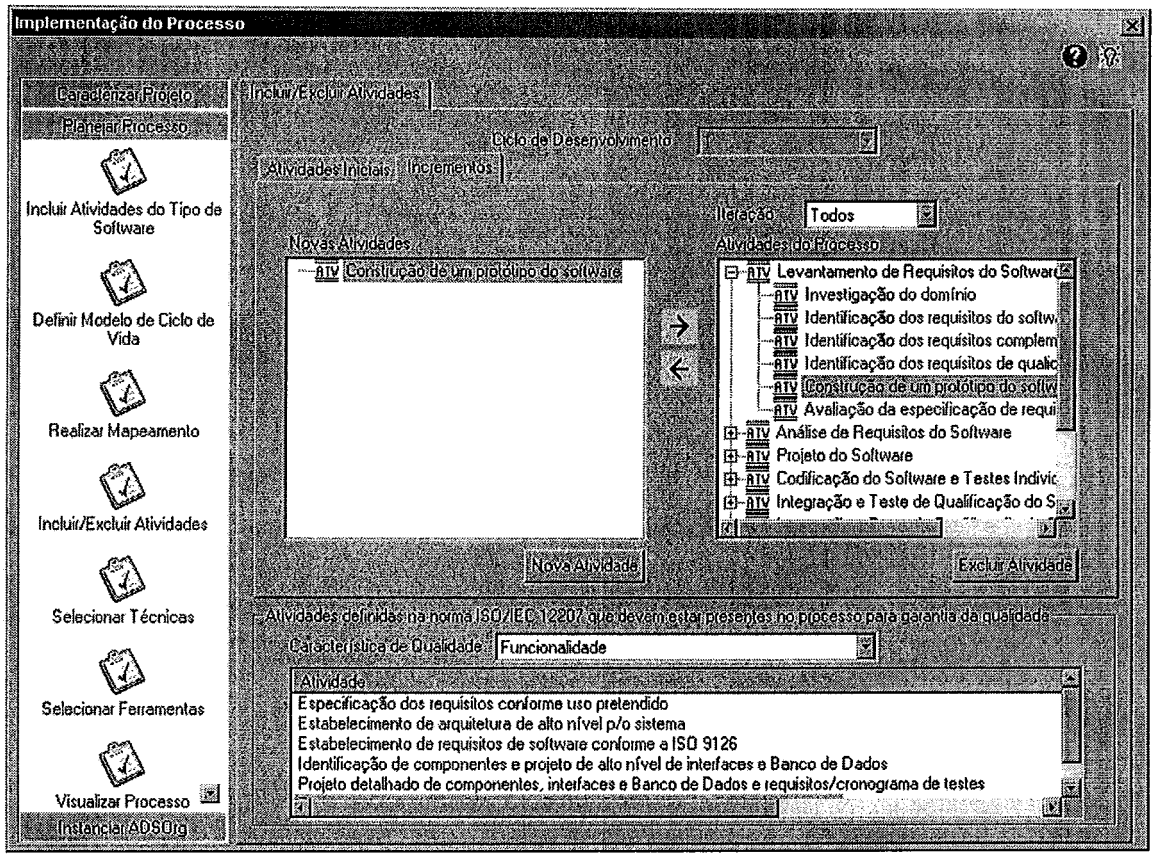


Figura 5.12 – Tela de inclusão de nova atividade ao processo

Para apoiar esta atividade, a ferramenta exibe, ainda, as atividades definidas na norma ISO/IEC 12207 e que são consideradas relevantes para a garantia da qualidade do produto. As atividades são exibidas de acordo com sua relevância para cada uma das característica de qualidade da ISO 9126 que foram consideradas relevantes para o produto em questão. A ferramenta considera apenas as características de qualidade avaliadas como relevantes para o processo. Neste caso, a característica Portabilidade não é considerada, uma vez que foi avaliada com sendo irrelevante para o produto em questão. A figura 5.14 mostra as atividades relevantes para se garantir a característica Funcionalidade.

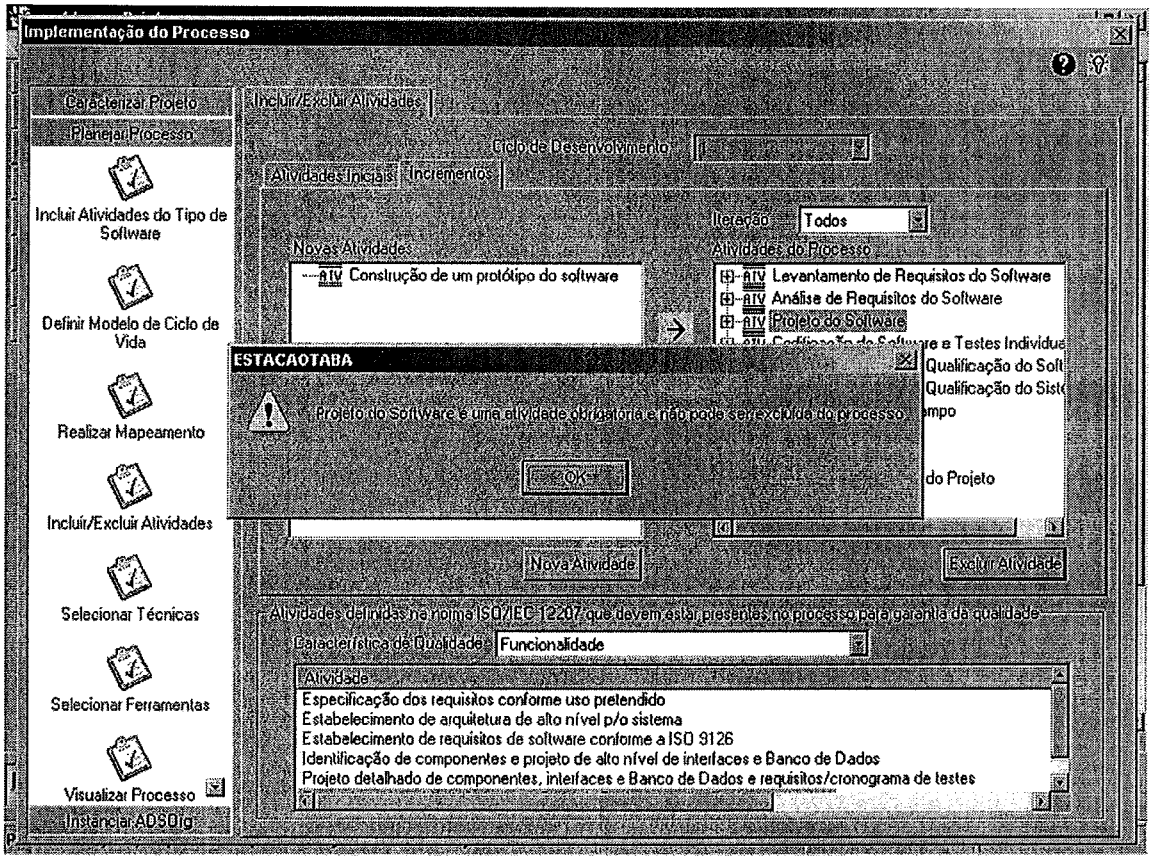


Figura 5.13 – Tela de ilustra uma tentativa de exclusão de atividade obrigatória

No contexto da Estação TABA, as atividades do processo são classificadas, em função de sua natureza, como atividades de gerência, atividades de construção e atividades de avaliação da qualidade. A próxima atividade consiste em selecionar técnicas de avaliação da qualidade, de acordo com o nível identificado anteriormente para o produto. As técnicas são exibidas de acordo com a característica de qualidade que avaliam e são associadas a atividades de avaliação da qualidade. Novamente a ferramenta considera apenas as características de qualidade avaliadas como relevantes para o produto.

O produto em questão foi classificado com nível D e não necessita de uma avaliação da qualidade rigorosa, uma vez que os possíveis danos causados por uma falha do software são pequenos. No exemplo ilustrado pela figura 5.15, o gerente selecionou a sub-atividade "Avaliação do modelo preliminar da interface" da atividade "Projeto Criativo de Interface" e a característica de qualidade Usabilidade. A ferramenta exibe a técnica "Inspeção da interface com o usuário", indicada para o nível de avaliação do projeto e o gerente a seleciona.

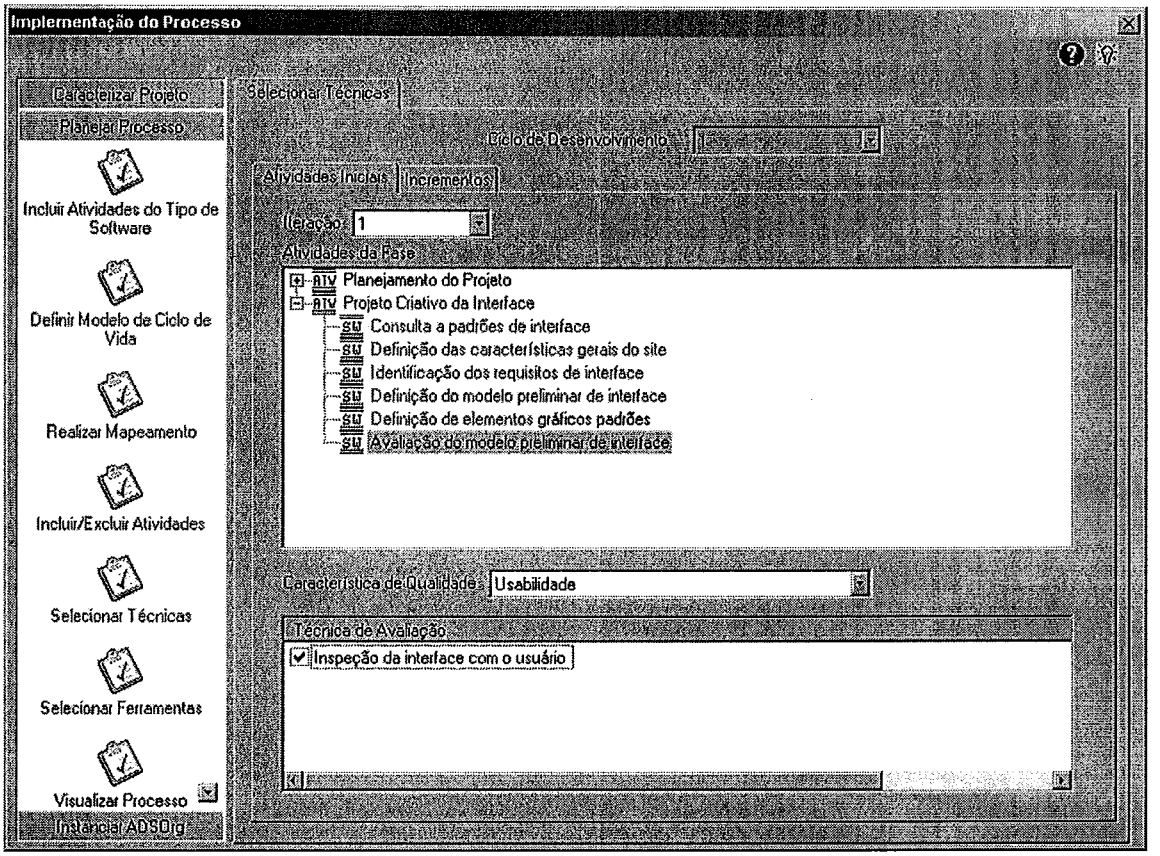


Figura 5.14 – Tela de seleção de técnicas de avaliação

Em seguida, o gerente do projeto deve selecionar ferramentas para apoiar a realização das atividades do processo. Para algumas atividades, a ferramenta utilizada já foi definida durante a configuração do ambiente, mas é possível que uma mesma atividade tenha mais de uma ferramenta associada e o gerente deve fazer a opção de qual ferramenta será efetivamente utilizada no projeto. A figura 5.16 mostra a tela de seleção de ferramentas para as atividades do processo. O gerente seleciona uma atividade do processo e, na parte baixo da tela, são exibidas as opções de ferramentas

disponíveis para a realização desta atividade. Em seguida, o gerente seleciona a ferramenta desejada, que será associada à atividade. O exemplo mostra a seleção da ferramenta "Word" para a atividade "Identificação dos requisitos do software".

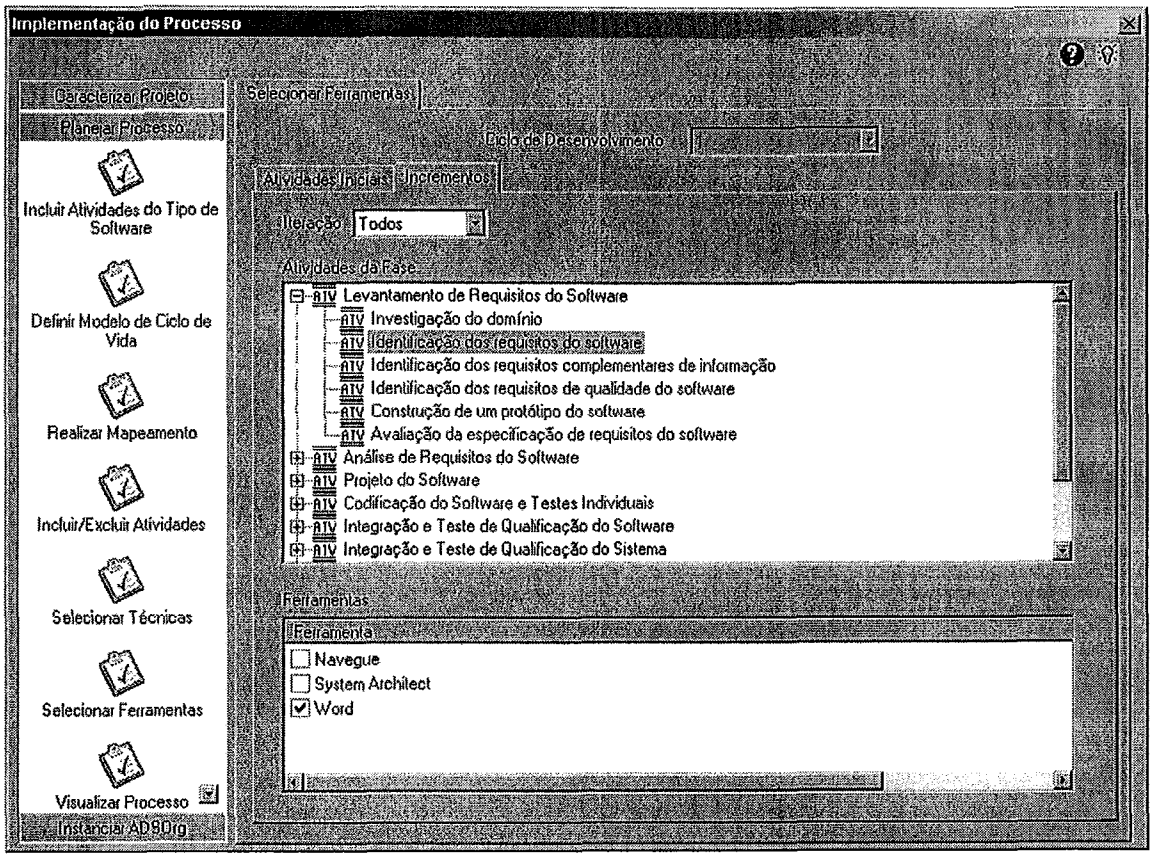


Figura 5.15 – Tela de seleção de ferramentas

Finalmente, após ter definido (planejado) o processo para o projeto em questão, o gerente do projeto visualiza o processo instanciado, selecionando a atividade "Visualização do Processo". A figura 5.17 ilustra a tela, que exibe as atividades do processo. Através desta tela, o gerente tem a opção de salvar o processo gerado no formato de um arquivo html. A ferramenta permite que o gerente selecione o diretório ou pasta onde este arquivo será salvo.

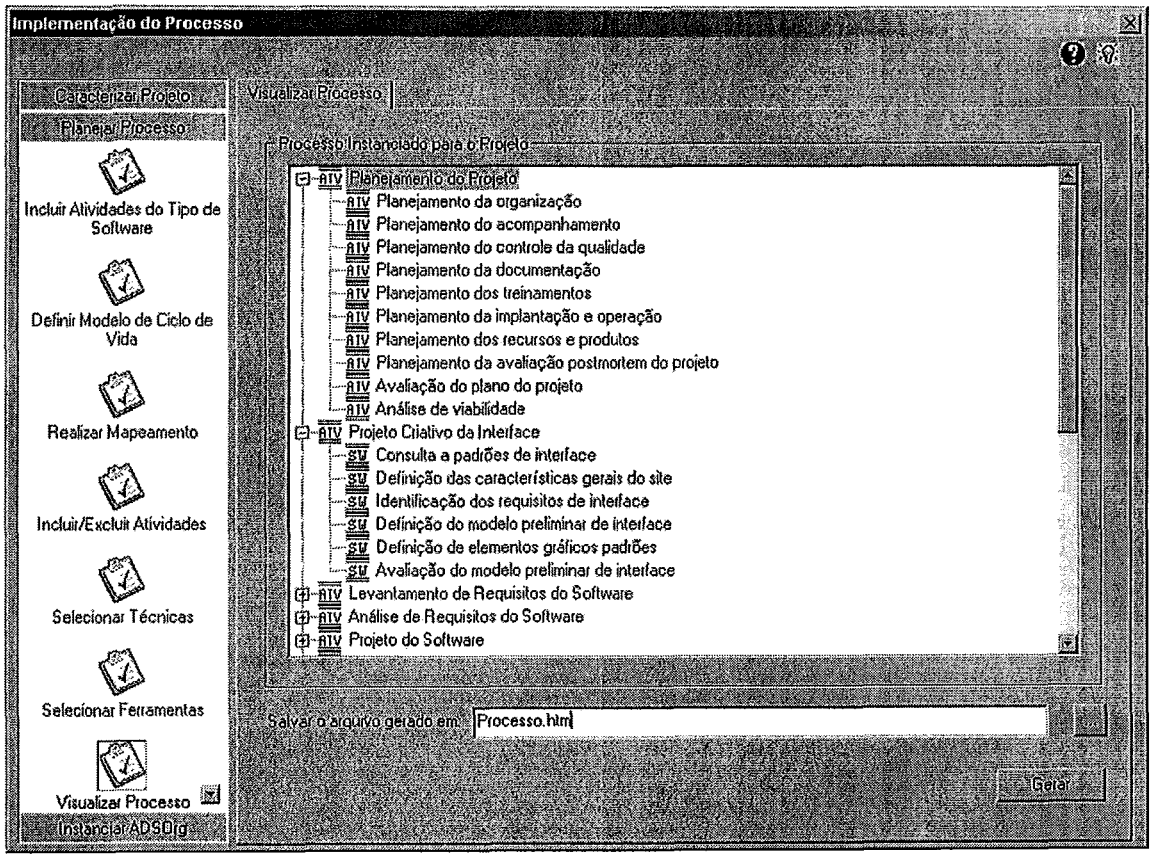


Figura 5.16 – Tela de visualização do processo

Quando o botão "Gerar" for clicado, o arquivo html será criado no diretório desejado e, em seguida, exibido na tela, para que o gerente possa conferir e, se desejar, imprimir o processo instanciado. A figura 5.17 mostra o arquivo html gerado a partir do processo instanciado para o projeto CardioNet.

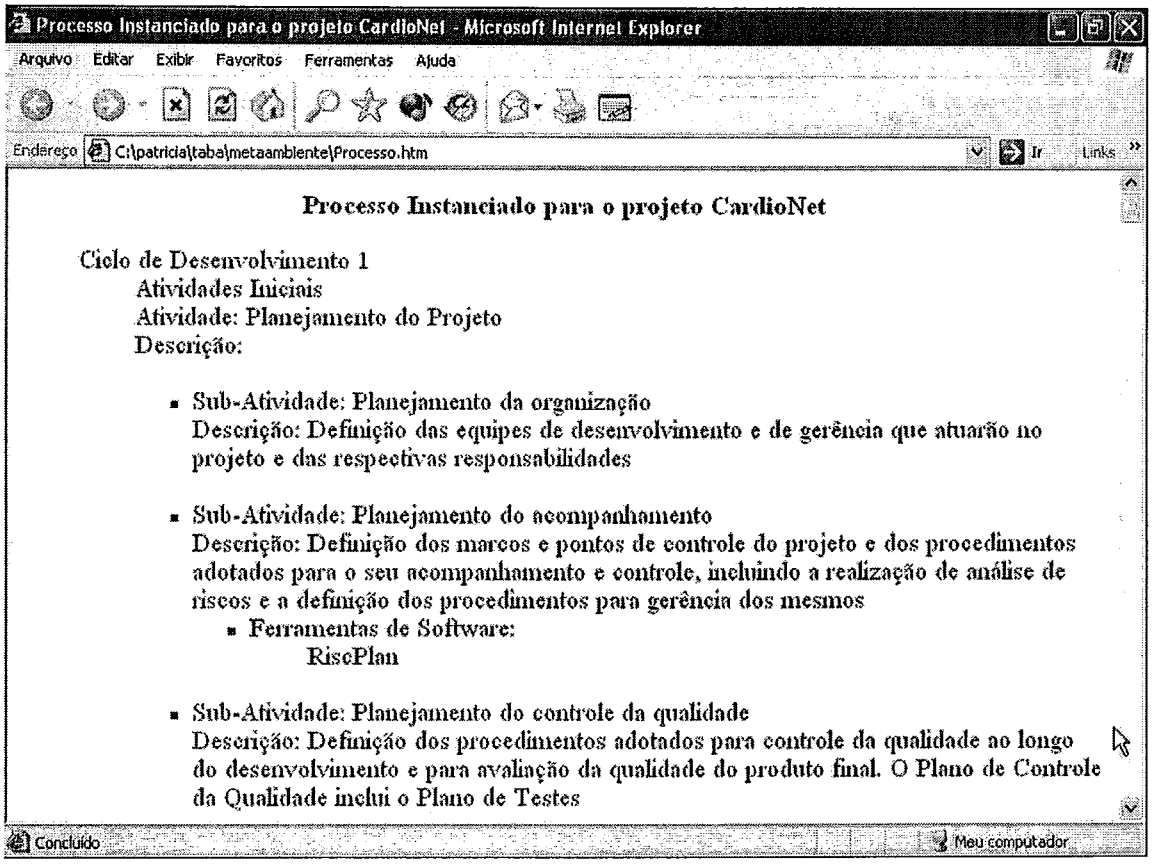


Figura 5.17 – Arquivo html gerado para o projeto CardioNet

5.3.1.3 Instanciação do ADSOrg

Uma vez que o processo foi definido, é instanciado um ADSOrg para o projeto. A figura 5.18 ilustra a tela de instanciação do ambiente para o projeto CardioNet. O gerente informa o nome e descrição do ambiente e, caso necessário, seleciona uma teoria do domínio. Em seguida, o gerente informa o caminho onde serão salvos os arquivos do ambiente. Finalmente, o gerente clica no botão "Instanciar" e o ADSOrg para o projeto é gerado.

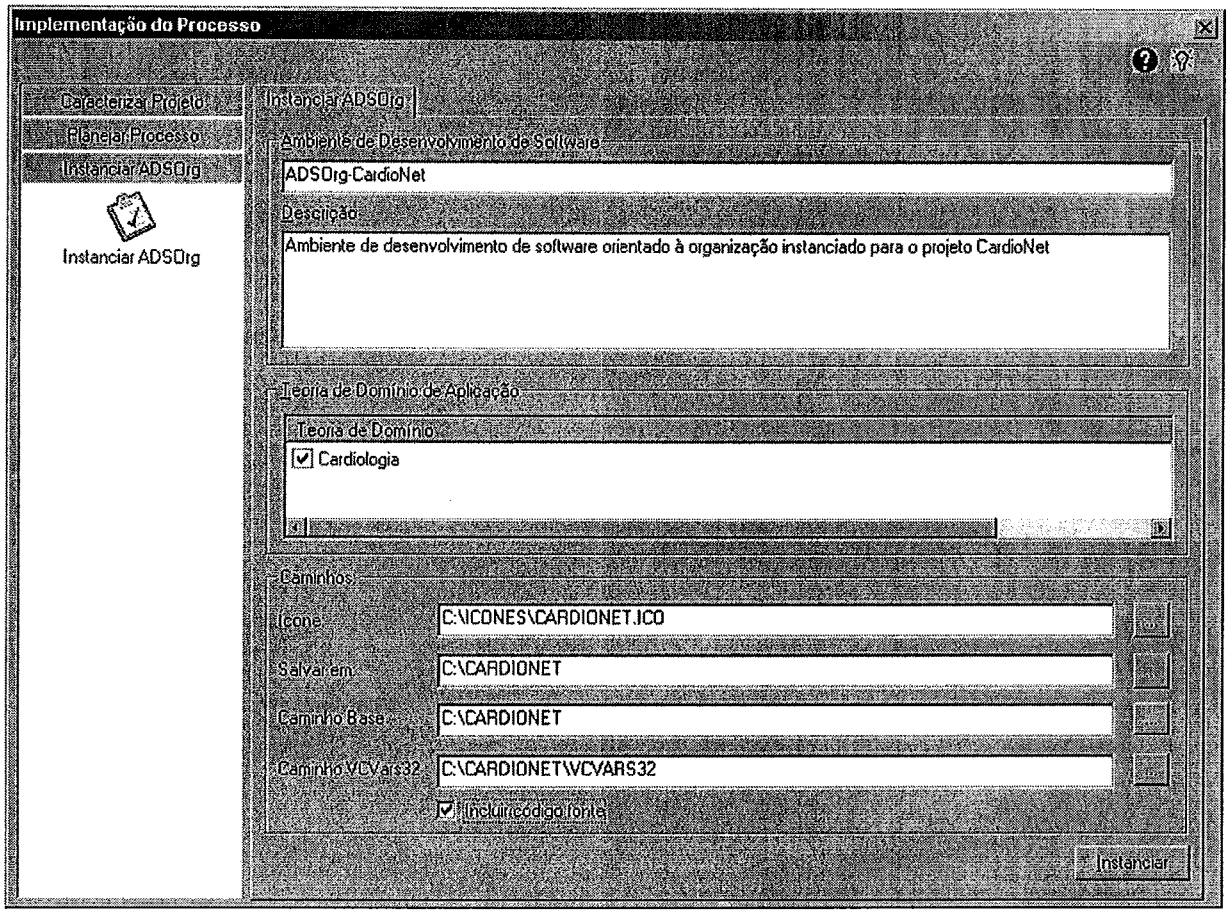


Figura 5.18 – Tela de instanciação do ADSOrg

5.3.2 Projeto Orientado a Objetos

A seguir será apresentado um exemplo de instanciação para um projeto que utiliza o paradigma de desenvolvimento orientado a objetos. Neste exemplo, o projeto para o qual será instanciado o ADSOrg se chama SIPLAG, e tem como objetivo a construção de um sistema de planejamento.

5.3.2.1 Caracterização do Projeto

A figura 5.19 mostra a tela de definição do projeto, onde foram preenchidas as seguintes informações sobre o sistema SIPLAG: nome, descrição, tipo de software a ser desenvolvido, paradigma de desenvolvimento e natureza do projeto.

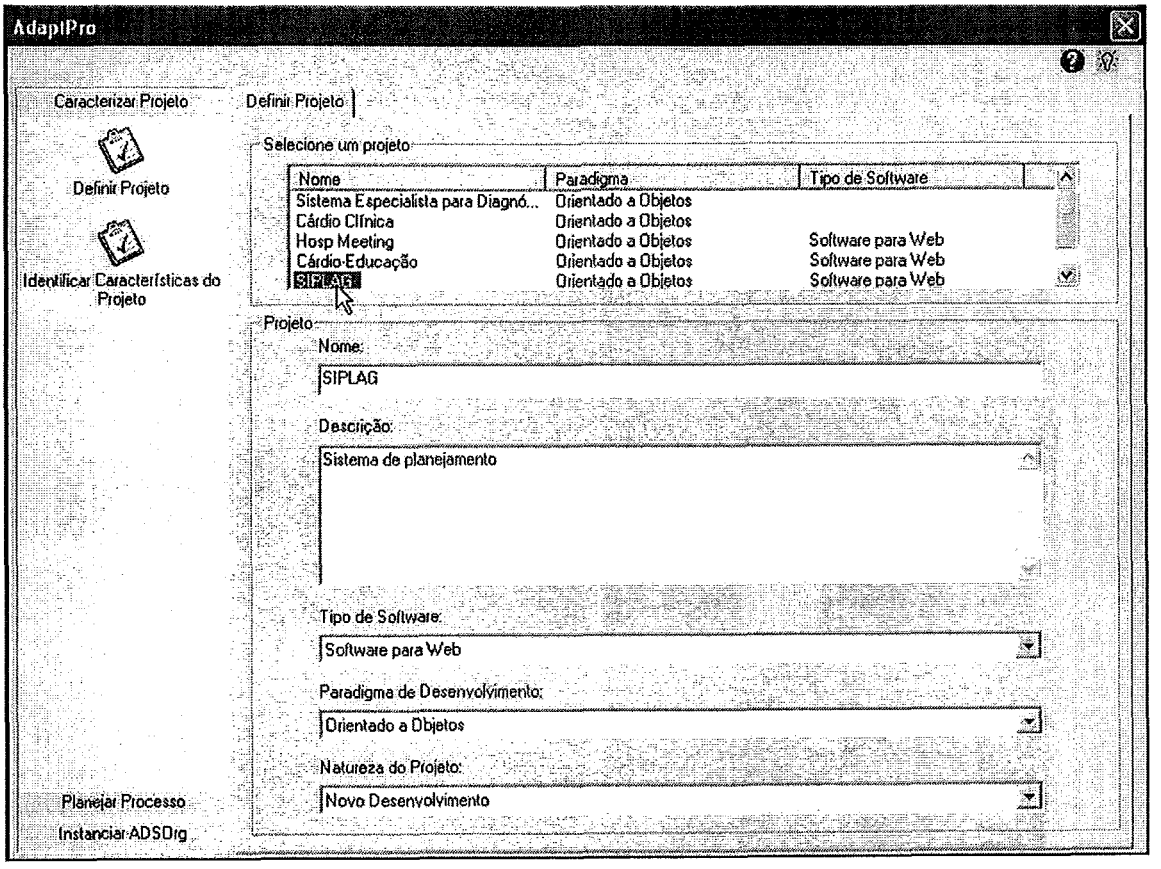


Figura 5.19 – Tela de cadastro do projeto SIPLAG

A figura 5.20 exibe a tela da atividade "Identificar Características do Projeto", na qual foi preenchido o questionário de caracterização do projeto em questão. A avaliação das características de qualidade para o projeto já foi realizada e os resultados também são exibidos nesta tela.

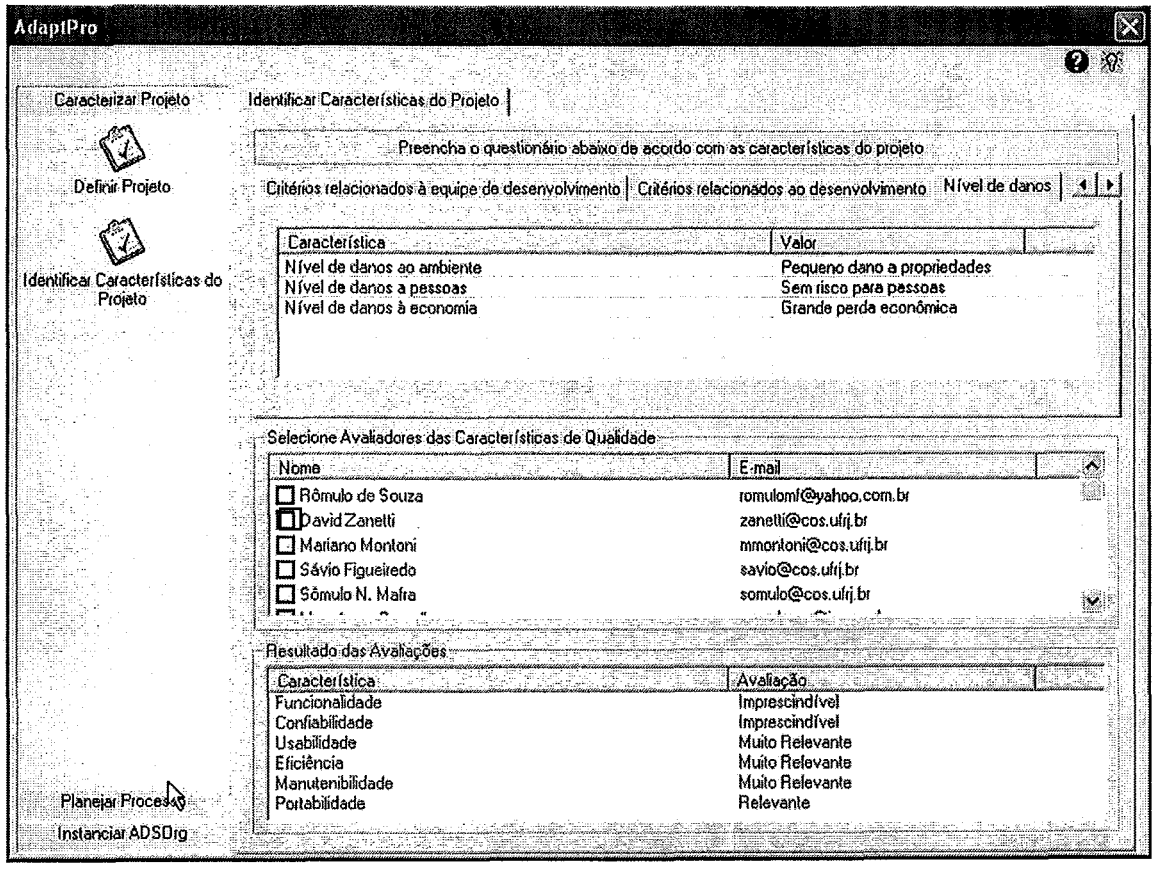


Figura 5.20 - Tela de caracterização do projeto

5.3.2.2 Planejamento do Processo

A figura 5.21 exibe a tela de inclusão das atividades do tipo de software. Neste exemplo existem projetos anteriores que utilizaram o mesmo processo especializado para desenvolver o mesmo tipo de software do projeto SIPLAG. Neste caso o usuário pode consultar os processos instanciados dos projetos anteriores, para apoiar a realização desta atividade.

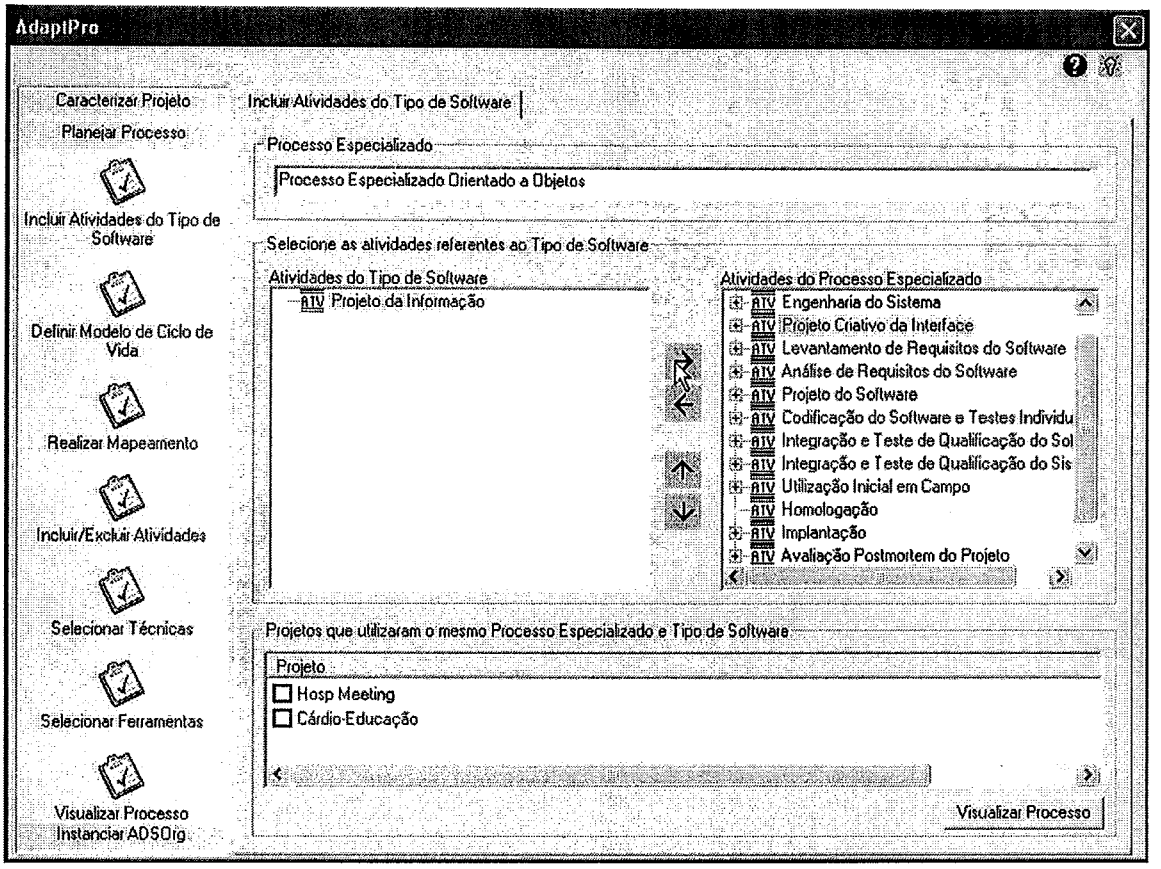


Figura 5.21 – Tela de inclusão das atividades do tipo de software

Por se tratar de um projeto que utiliza o paradigma orientado a objetos, a ferramenta *AdaptPro* indica o modelo de ciclo de vida do RUP. Em seguida, o gerente informa o número de ciclos de desenvolvimento a serem realizados, conforme mostra a figura 5.22.

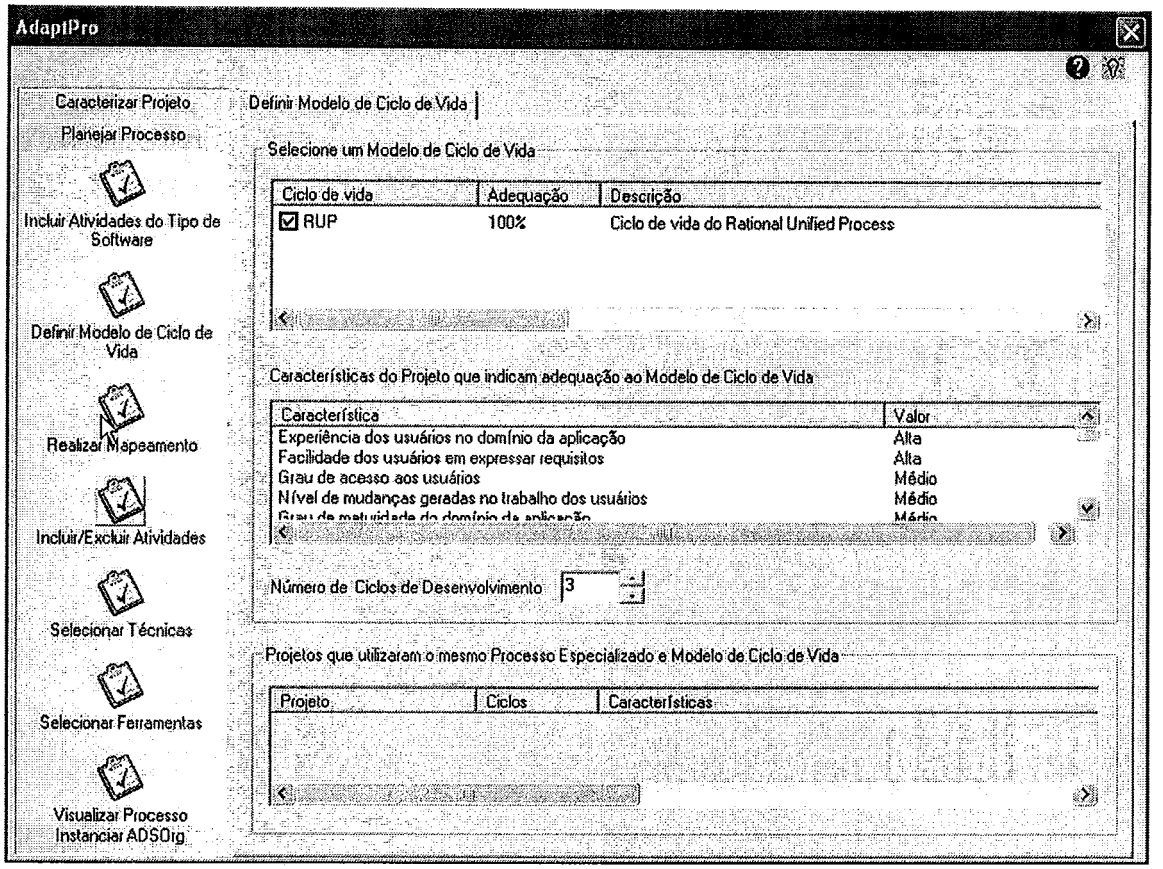


Figura 5.22 - Tela de definição do modelo de ciclo de vida

A próxima atividade é a realização do mapeamento das atividades do processo para o modelo de ciclo de vida selecionado. O modelo de ciclo de vida relacionado ao RUP possui quatro fases: Concepção, Elaboração, Construção e Transição. Para cada ciclo de desenvolvimento, o gerente deve indicar o percentual de casos de uso que será considerado em cada fase. Baseado no percentual de casos de uso selecionado para cada fase, *AdaptPro* realiza um mapeamento inicial, que pode ser modificado pelo gerente. O gerente pode, ainda, definir o número de iterações de cada fase do ciclo de desenvolvimento. A figura 5.23 mostra a tela da atividade "Realizar Mapeamento".

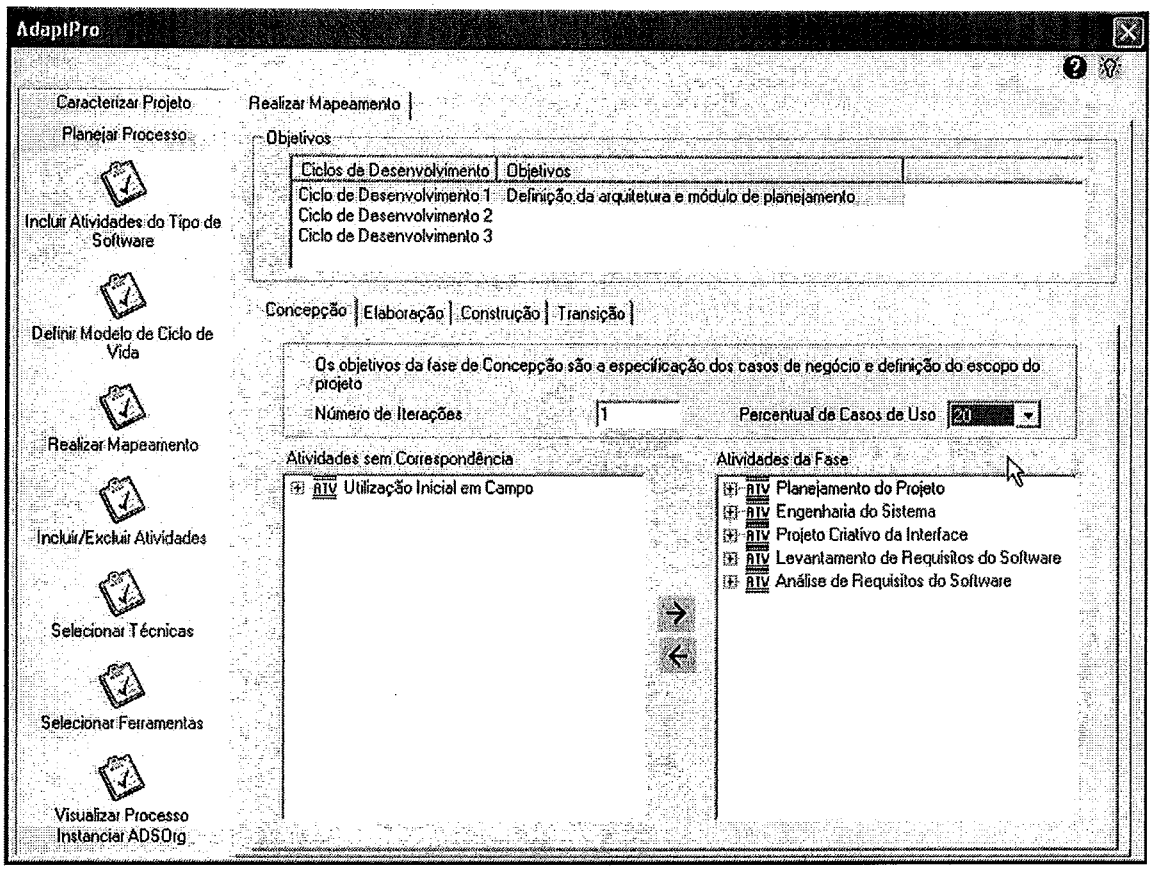


Figura 5.23 - Tela de mapeamento das atividades

Caso o gerente tenha alguma idéia sobre a realização de uma atividade do processo, ele pode registrá-la através da interface com a ferramenta de aquisição de conhecimento *Acknowledge* (MONTONI *et al.*, 2002). A ferramenta *Acknowledge* pode ser acessada através de um ícone localizado abaixo da barra de título. Da mesma maneira, o conhecimento armazenado pela organização sobre esta atividade do processso pode ser acessado. A figura 5.24 exibe o registro de uma idéia relacionada ao processo de instanciação.

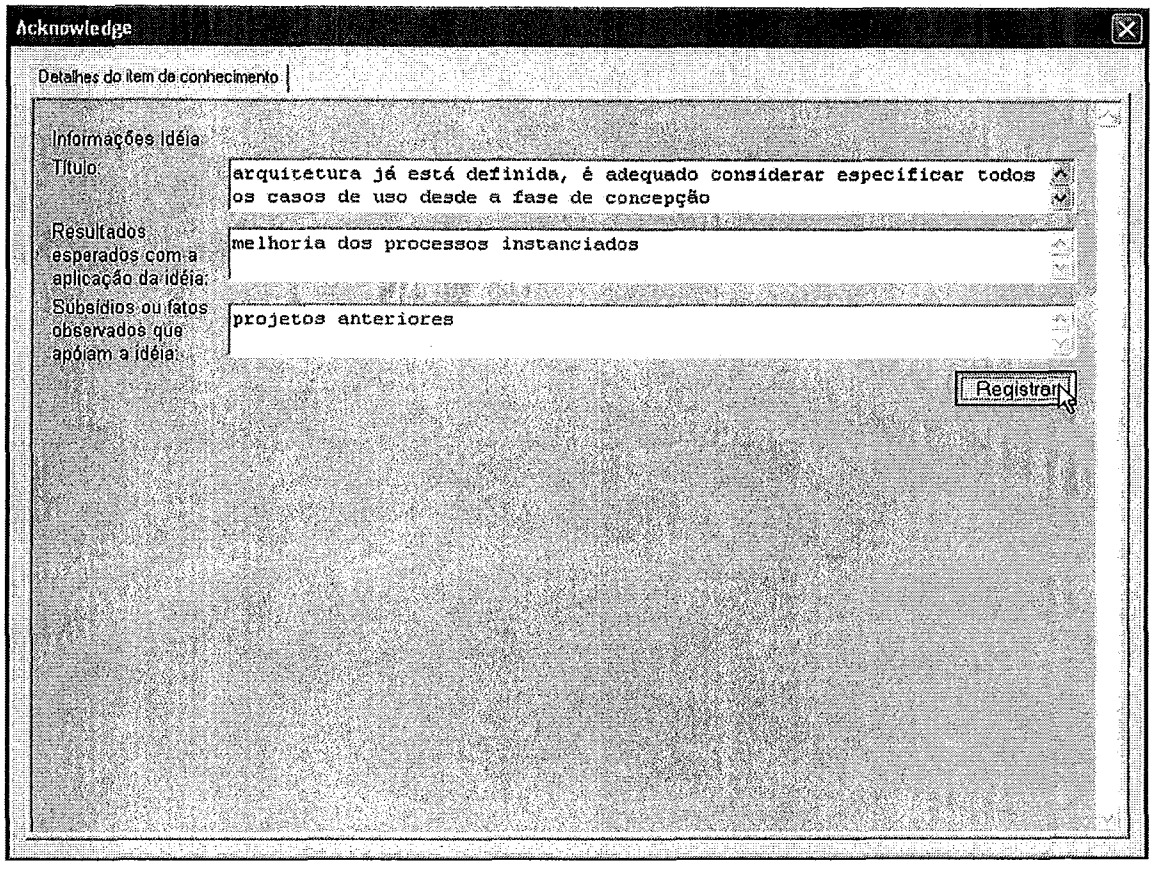


Figura 5.24 – Registro de idéia na ferramenta *Acknowledge*

As demais atividades do processo são realizadas de maneira semelhante ao primeiro exemplo.

5.4 Conclusão

Este capítulo apresentou a abordagem proposta neste trabalho e a ferramenta desenvolvida para apoiá-la, *AdaptPro*. Através desta abordagem é possível oferecer apoio ao gerente do projeto durante a execução das atividades do processo de instanciação, apresentado no capítulo 4. Este capítulo apresentou, ainda, um exemplo detalhado de utilização da ferramenta *AdaptPro*.

CAPÍTULO VI - CONSIDERAÇÕES FINAIS

6.1 Conclusões

Diante de um mercado cada vez mais competitivo, organizações de software constantemente precisam aumentar a qualidade de seus produtos, para garantir sua sobrevivência. Nesse contexto, a procura por abordagens que garantam a melhoria do processo de software tem sido uma preocupação constante.

A definição e utilização de padrões em Engenharia de Software são práticas importantes para se homogeneizar o desenvolvimento de software em projetos e organizações (MACHADO, 2000). A definição de um processo padrão que descreve as atividades que devem ser realizadas no desenvolvimento de sistemas de software em todos os projetos de uma organização tornou-se um elemento chave das normas e modelos de maturidade. Tanto a Norma ISO/IEC 12207, como os modelos de maturidade, estabelecem a definição de um processo padrão como condição para o estabelecimento de um processo de software disciplinado. Para que possa ser utilizado em um projeto, o processo padrão deve ser adaptado para atender às características específicas deste projeto, gerando um processo instanciado. Entretanto a adaptação do processo padrão para projetos específicos é uma atividade complexa que pode se beneficiar de apoio baseado em conhecimento.

Considerando a importância destas questões com base nos conceitos de Gerência do Conhecimento e Ambientes de Desenvolvimento de Software Orientados à Organização, este trabalho apresentou uma abordagem para a instanciação de processos de software para projetos específicos baseada na utilização do conhecimento organizacional. Através da disponibilização do conhecimento acumulado pelos vários gerentes de projeto de uma organização, espera-se apoiar a atividades de instanciação de processos.

Os conceitos da abordagem proposta estão implementados na ferramenta *AdaptPro*, que apóia a implementação do processo (atividade do processo de desenvolvimento da norma ISO IEC 12207) e é disponibilizada em um ambiente configurado para uma organização específica, no contexto da Estação TABA. Dentre as contribuições deste trabalho temos:

(1) Definição de um processo para instanciação de processos de software, tornando explícito o conhecimento acumulado em diversos projetos.

(2) Definição de uma abordagem para a instanciação de ADSOrg em um ambiente configurado TABA.

(3) Definição e implementação da ferramenta *AdaptPro*, que apóia a abordagem proposta e cujo objetivo é apoiar o gerente de projetos durante a instanciação do processo, utilizando técnicas de gerência do conhecimento.

Os benefícios da abordagem proposta poderão ser avaliados através da utilização da ferramenta *AdaptPro* em vários projetos e excede o tempo esperado para uma tese de mestrado. Portanto, a avaliação será realizada no contexto global do projeto ADSOrg. Incluiu-se, entretanto, no capítulo 5, um exemplo de sua utilização.

6.2 Perspectivas Futuras

Buscando-se melhorar e expandir a abordagem proposta, algumas perspectivas de trabalhos futuros são destacadas.

A ferramenta *AdaptPro* se utiliza de experiências obtidas em projetos anteriores para apoiar o gerente de projeto. Nesse contexto torna-se importante a capacidade de identificar, entre os projetos realizados pela organização, aqueles que possuem características semelhantes ao projeto atual. Atualmente a ferramenta realiza a busca por projetos similares através de critérios fixos (paradigma de desenvolvimento, tipo de software, modelo de ciclo de vida). Seria interessante haver a possibilidade de uma busca mais refinada, baseada em outras características do projeto, como tamanho e complexidade.

Através da utilização da ferramenta *AdaptPro* em projetos, será possível avaliar a abordagem proposta. À medida que os projetos forem realizados, os gerentes de projeto passarão a acumular experiências a respeito da atividade de instanciação e poderão, através da ferramenta de aquisição de conhecimento, registrar lições aprendidas. Também através de avaliações *post mortem* dos projetos realizados, será possível coletar experiências, tanto positivas quanto negativas a respeito dos processos instanciados. O conhecimento acumulado poderá ser usado para a melhoria contínua dos processos, garantindo a qualidade dos mesmos.

Atualmente *AdaptPro* utiliza um mecanismo de avaliação das características de qualidade da norma ISO 9126 que considera 5 níveis de relevância (Imprescindível, Muito Relevante, Relevante, Pouco Relevante, Irrelevante). Entretanto, ao oferecer apoio ao gerente de projeto, a ferramenta considera apenas se a característica é relevante ou não, não fazendo distinção entre uma característica meramente relevante e uma considerada imprescindível. Este mecanismo pode ser aprimorado para oferecer um apoio mais apropriado, de acordo com as características do projeto.

REFERÊNCIAS BIBLIOGRÁFICAS

ABECKER, A., BERNARDI, A., HINKELMANN, K. *et al.*, 1998, "Toward a Technology for Organizational Memories", *IEEE Intelligent Systems*, v. 13, n. 3 (May/Jun), pp. 40-48.

ABECKER, A., BERNARDI, A., SINTEK M., 1999, "Developing a Knowledge Management Technology", IEEE 8th International Workshop on Enabling Technologies Infrastructure for Collaborative Enterprises.

ABRAN, A., MOORE, J., 2001, Guide to the Software Engineering Body of Knowledge, IEEE Computer Society.

ALEXANDER, L. C., DAVIS, A. M., 1991, "Criteria for Selecting Software Process Models". In: *Proceedings of the Fifteenth Annual International Computer Software & Applications Conference – COMPSAC 91*, pp. 521-528, Tokyo, Japan, September.

ARENT, J., NORBJERG, J., PEDERSEN, M., 2000, Creating Organizational Knowledge in Software Process Improvement, Second International Workshop on Learning Software Organizations (LSO 2000), pp.81-92.

BASILI, V., LINDVALL, M., COSTA, P., 2001a, "Implementing the Experience Factory concepts as a set of Experience Bases", SEKE 2001.

BASILI, V., LINDVALL, M., COSTA, P. *et al.*, 2001b, "Building an Experience Base for Software Engineering: A report on the first CeBASE eWorkshop".

BATISTA, J., FIGUEIREDO, A.D., 2000, "SPI in a Very Small Team: a Case with CMM", *Software Process Improvement and Practice*, 5:243-250.

BERSOFF, E. H., DAVIS, A. M., 1991, "Impacts of Life Cycle Models on Software Configuration Management". In: *Communications of the ACM*, v. 34, n. 8, pp.104-117, August.

BIGGAM, J. 2001, Defining Knowledge: an Epistemological Foundation for Knowledge Management. Hawaii, USA, Proceedings of the 34th Hawaii International Conference on System Sciences.

BORGES, L. M. S., 2001, *Uma Ferramenta para Instanciação de Processos de Software e Apoio ao Compartilhamento de Experiências*, Tese de Mestrado, UFES, Vitória, ES, Brasil, Dezembro.

BRANDT, M., NICK M., 2001, Computer-Supported Reuse of Project Management Experience with an Experience Base. In: K.-D. Althoff, R.L. Feldmann, W. Müller (Eds.): LSO 2001, LNCS 2176, pp. 178-189-. Springer-Verlag Berlin Heidelberg 2001.

CAMERON, J., 2002, "Configurable Development Processes", *Communications of the ACM*, Março, Vol. 45, No. 3, pp. 72-77.

CARVALHO, A. L., SIDER: Um Ambiente de Desenvolvimento de Software Orientado ao Domínio Siderúrgico, Dissertação de Mestrado, Mestrado em Informática, UFES, Julho de 2002.

CRISTENSEN, M. J., THAYER, R. H., The Project Manager's Guide to Software Engineering Best Practices, IEEE Computer Society.

DAVIS, A. M., BERSOFF, E. H., COMER, E. R., 1988, "A Strategy for Comparing Alternative Software Development Life Cycle Models". In: *IEEE Transactions on Software Engineering*, v. 14, n. 10, pp. 1453-1461, October.

DECKER, B., JEDLITSCHKA, A., 2001, The Integrated Corporate Information Network iCoIN: A Comprehensive, Web-Based Experience Factory, In: K.-D. Althoff, R.L. Feldmann, W. Müller (Eds.): LSO 2001, LNCS 2176, pp. 192-206. Springer-Verlag Berlin Heidelberg 2001.

DIENG, R., 2000, "Knowledge Management and the Internet", *IEEE Intelligent Systems*, (May/Jun), pp. 14-17.

FALBO, R. A., 1998, *Integração de Conhecimento em um Ambiente de Desenvolvimento,* Tese de D. Sc. , COPPE/UFRJ, Rio de janeiro, R.J., Brasil.

FELDMANN, R. L., ALTHOFF, K.-D., 2001, On the Status of Learning Software Organizations in the Year 2001. In: K.-D. Althoff, R.L. Feldmann, W. Müller (Eds.): LSO 2001, LNCS 2176, pp. 163-177. Springer-Verlag Berlin Heidelberg 2001.

FISCHER, G., OSTWALD, J., 2001, "Knowledge Management: Problems, Promises, Realities, and Challenges", *IEEE Intelligent Systems*, (Jan/Feb), pp. 60-72.

FOURO, A. M., 2002, *Apoio à Construção de Base de Dados de Pesquisa em Ambientes de Desenvolvimento de Software Orientados a Domínio*, Tese de M. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

FUGGETTA, A., 2000, Software Process: A Roadmap. In: Future of Software Engineering, A Finkelstein (ed).

GALOTTA, C., 2000, *Netuno: Um Ambiente de Desenvolvimento de Software Orientado ao Domínio de Acústica Submarina*, Tese de M. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

GARG, P., JAZAYERI, M., 1995, "Introduction". In: Garg, P., Jazayeri, M. (eds), *Process-Centered Software Engineering Environments*, chapter 1, IEEE Computer Society Press.

HENNINGER, S., 1999, Using Software Process to Support Learning Software Organizations, First International Workshop on Learning Software Organizations (LSO 1999), pp.99-114.

HENNINGER, S., 2001, "Turning Development Standards into Repositories of Experiences", Software Process Improvement and Practice 2001, 6: 141-155.

HENNINGER, S., SCHLABACH, J., 2001. A Tool for Managing Software Development Knowledge, pp. 182-195, Third International Conference PROFES 2001, Kaiserlautern, Alemanha.

HOLZ, H., KÖNNECKER, A., MAURER, F., 2001, Task-Specific Knowledge Management in a Process-Centred SEE. In: K.-D. Althoff, R.L. Feldmann, W. Müller (Eds.): LSO 2001, LNCS 2176, pp. 163-177. Springer-Verlag Berlin Heidelberg 2001.

ISO/IEC 12207, 1995, *Information Technology – Software Life-Cycle Processes.*

ISO/IEC TR 15504, 1998, *Parts 1-9: Information Technology - Software Process Assessment.*

KARAGIANNIS, D., TELESKO, R., 2000. The EU-Project PROMOTE: A Process-oriented Approach for Knowledge Management. Proc. Of the Third Conference on Practical Aspects of Knowledge Management (PAKM2000). Basel, Suíça.

KEMP, L., NIDIFFER, K., *et al*, 2001, "Knowledge Management: Insights from the Trenches", *IEEE Software*, (Nov/Dez), pp. 66-68.

KING, W., MARKS, P., MCCOY, S., 2002, "The Most Important Issues in Knowledge Management", Communications of the ACM, Setembro 2002/Vol. 45 No. 9, pp 93-97.

KOCK, N., 2002, "Managing With Web-Based IT in Mind", Communications of the ACM, Maio 2002/Vol. 45 No. 5, pp 102-106.

KOMI-SIRVIÖ, S., MÄNTYNIEMI, A., SEPPANËM, V., 2002, "Toward a Practical Solution for Capturing Knowledge for Software Projects", *IEEE Software*, (Mai/Jun), pp. 60-62.

KRUCHTEN, P., The Rational Unified Process – An Introduction, Addison-Wesley.

KUCZA, T., NÄTTINEN, M., PARVIAINEN, P., 2001 Improving Knowledge Management in Software Reuse Process, pp. 141-152, Third International Conference PROFES 2001, Kaiserlautern, Alemanha.

LEUNG, H.K.N., YUEN T.C.F., 2001, "A Process Framework for Small Projects", *Software Process Improvement and Practice*, 6:67-83.

LIEBOWITZ, J., 2002, "A Look at NASA Goddard Space Flight Center's Knowledge Management Initiatives", *IEEE Software*, (Mai/Jun), pp. 40-42.

MACHADO, L.F.D., 2000, *Modelo para Definição de Processos de Software*, Tese de Mestrado, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, Março.

MAURER, F., DELLEN, B., HOLZ, H., 1999, Process Support for Virtual Software Organizations, First International Workshop on Learning Software Organizations (LSO 1999), pp.87-98.

MONTONI, M. A., ROCHA, A.R., 2002, "Aquisição de Conhecimento em Processos de Negócios", *Workshop de Teses – XVI Simpósio Brasileiro de Engenharia de Software*, Gramado – RS, Outubro.

NAGEL, C., 2001. Processes and Knowledge Management: A Symbiosis, pp. 153-166, Third International Conference PROFES 2001, Kaiserlautern, Alemanha.

NICK, M., ALTHOFF, K-D, 2001, Engineering Experience Base Maintenance Knowledge, In: K.-D. Althoff, R.L. Feldmann, W. Müller (Eds.): LSO 2001, LNCS 2176, pp. 222-236. Springer-Verlag Berlin Heidelberg 2001.

ODDO, M., ROCHA, A.R., 2003, Relating Process Activities to Software Quality Characteristics, trabalho submetido à revista Software Quality Journal.

O'LEARY, D. E., 1998a, "Enterprise Knowledge Management", *IEEE Computer*, v.31, n.3 (Mar), pp. 54-61.

O'LEARY, D. E., 1998b, "Knowledge Management Systems: Converting and Connecting", *IEEE Intelligent Systems*, v. 13, n. 3 (May/Jun), pp. 30-33.

O'LEARY, D. E., 1998c, "Using AI in Knowledge Management: Knowledge Bases and Ontologies", *IEEE Intelligent Systems*, v. 13, n. 3 (May/Jun), pp. 34-39.

O'LEARY, D. E., STUDER R., 2001a, "Knowledge Management: An Interdisciplinary Approach", *IEEE Intelligent Systems*, (Jan/Feb), pp. 24-25.

O'LEARY, D. E., 2001b, "How Knowledge Reuse Informs Effective System Design and Implementation", *IEEE Intelligent Systems*, (Jan/Feb), pp. 44-49.

OLIVEIRA, K.M., 1999, *Modelo para Construção de Ambientes de Desenvolvimento de Software Orientados a Domínio,* Tese de D. Sc. , COPPE/UFRJ, Rio de janeiro, R.J., Brasil.

OLIVEIRA, K. M., ROCHA, A. R., TRAVASSOS, G. H. *et al.*, 1999a, "Using Domain-Knowledge in *Software* Development Environments", In: *Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering*, pp. 180-187, Kaiserlautern, Alemanha, Jun.

OLIVEIRA, K. M., ROCHA, A. R., TRAVASSOS, G. H. *et al.*, 1999b, "O uso da Teoria do Domínio no Processo de Desenvolvimento de *Software*", In: *Anais da X Conferencia Internacional de Tecnologia de Software*, pp 223-235, Curitiba, Brasil, Mai.

PAULK, M. C., WEBER, C. V., CURTIS, B., CHRISSIS, M. B. (eds), 1997, *The Capability Maturity Model: Guidelines for Improving the Software Process.* Carnegie Mellon University, Software Engineering Institute, Addison-Wesley Longman Inc.

PFLEEGER, S. L., 2001, *Software Engineering Theory and Practice*, 2th edition, Prentice Hall PTR.

PREECE, A., FLETT, A., SLEEMAN, D., 2001, "Better Knowledge Management through Knowledge Engineering", *IEEE Intelligent Systems*, (Jan/Feb), pp. 36-43.

PRESSMAN, R. S., 2001, *Software Engineering: A Practitioner's Approach*, 5th edition, McGraw-Hill.

PURPER, C., 2000, "Transcribing Process Model Standards into Meta-Processes", 7th European Workshop on Software Process Technology, EWSPT 2000, pp.55-68, Austria.

RAMAN, S., 2000, "It Is Software Process, Stupid: Next Millennium Software Quality Key", *IEEE AES Software Magazine*, Junho, pp. 33-37.

RAMASUBRAMANIAN, S., JAGADEESAN, G., 2002, "Knowledge Management at Infosys", *IEEE Software*, (Mai/Jun), pp. 53-55.

REIFER, D., 2002, "A Little Bit of Knowledge Is a Dangerous Thing", *IEEE Software*, (Mai/Jun), pp. 14-15.

ROCHA, A. R. C., AGUIAR, T. C., SOUZA, J. M., 1990, "TABA: A Heuristic Workstation for *Software* development", In: *Proceedings of COMPEURO 90*, Tel Aviv, Israel, Maio.

ROCHA, A.R. C., MALDONADO, J.C., WEBER, K.C., 2001. Qualidade de Software. São Paulo. Prentice Hall.

RUS, I., LINDVALL, M., 2002, "Knowledge Management in Software Engineering", *IEEE Software*, (Mai/Jun), pp. 26-38.

SANTOS, G., 2003, *Representação da Distribuição do Conhecimento, Habilidades e Experiências através da Estrutura Organizacional*, Tese de M. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

SAUNDERS, V. M., 2000, "Collaborative Enterprise Environments – Enterprise-Wide Decision Support & Knowledge Management", IEEE Aerospace Conference, pp. 321-330.

SCACCHI, W., 2000, "Understanding Software Process Redesign using Modeling, Analysis and Simulation", *Software Process Improvement and Practice*, 5:183-195.

SCHNAIDER, L., 2003, *Planejamento da Alocação de Recursos Humanos em Ambientes de Desenvolvimento de Software Orientados à Organização*, Tese de M. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

SCHNEIDER, K., HUNNIUS, J-P., BASILI, V., 2002, "Experience in Implementing a Learning Software Organization", *IEEE Software*, (Mai/Jun), pp. 46-49.

SEI, 2002, "Capability Maturity Model Integration, Version 1.1 Continuous Representation". URL: http://www.sei.cmu.edu

SEPPANËM, V., KOMI-SIRVIÖ, S., MÄNTYNIEMI, A., *et al.*, 2001. Contexts of KM based SPI: The Case of Software Design Experience Reuse, pp. 57-67, Third International Conference PROFES 2001, Kaiserlautern, Alemanha.

SKYRME, D., 1995, "Global Intelligence Networking: Technological Opportunities and Human Challenges", The Journal of AGSI, Volume 4 - Issue 3, (Nov), pp. 106-115.

SKYRME, D., 1998, "Knowledge Management Solutions - The IT Contribution", *SIGGROUP Bulletin*, v. 19, n. 1 (Apr), pp. 34-39.

SKYRME, D., 1999, "Knowledge Management: Making It Work", *The Law Librarian*, Vol. 31, No. 2, pp.84-90.

SNOEK, B. 1999. Knowledge Management and Organizational Learning. Diploma Thesis, Fraunhofer/IESE.

STATZ, J., 1999, "Leverage Your Lessons", *IEEE Software*, (Mar/Apr), pp. 30-32.

SWARTOUT, W., TATE, A., 1999, "Ontologies", *IEEE Intelligent Systems*, (Jan/Feb), pp. 18-19.

TIWANA, A., RAMESH, B., 2001, "Integrating Knowledge on the Web", *IEEE Internet Computing*, (May/Jun), pp. 32-39.

TRAVASSOS, G. H., 1994, *O Modelo de Integração de Ferramentas da Estação TABA*, Tese de D. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

VAN HEIJST, G., VAN DER SPEK, R., KRUIZINGA, E., 1997, "Corporate Memories as a Tool for Knowledge Management", *Expert Systems With Applications*, 13(1):41-54. in ref. (SNOEK, 1999).

VILLELA, K., ROCHA, A. R., TRAVASSOS, G.H., 2000, *Ambientes de Desenvolvimento de Software Orientados à Organização*, Publicação Técnica COPPE/UFRJ - ES530/00 Rio de Janeiro, RJ, Abril.

VILLELA, K., SANTOS, G., GALOTTA., C., 2001a, "Estendendo a Estação TABA para a criação de Ambientes de Desenvolvimento de *Software* Orientados a Organização", *Caderno de Ferramentas - XV Simpósio Brasileiro de Engenharia de Software*, pp. 359-362, Rio de Janeiro, RJ, outubro.

VILLELA, K., SANTOS, G., GALOTTA., C., 2001b, "Cordis-FBC: um Ambiente de Desenvolvimento de *Software* para Cardiologia", *I Workshop de Informática Médica*, Rio de Janeiro, RJ, outubro.

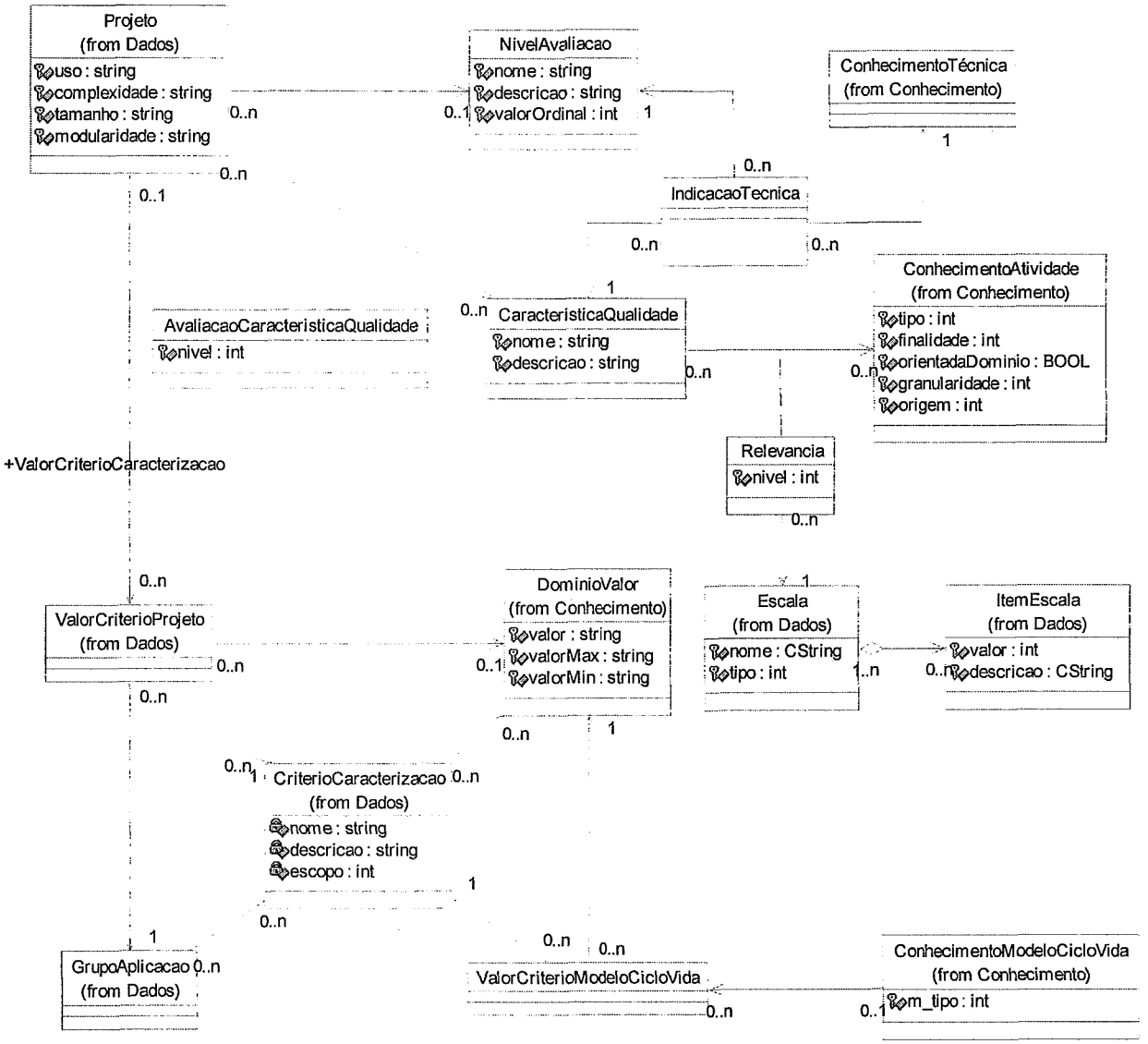
WANGENHEIM, C., LICHTNOW, D., *et al*, 2001, Supporting Knowledge Management in University Software R&D Groups. In: K.-D. Althoff, R.L. Feldmann, W. Müller (Eds.): LSO 2001, LNCS 2176, pp. 52-66. Springer-Verlag Berlin Heidelberg 2001.

WEI, C-P., HU, P., CHEN, H-H., 2002, "Design and Evaluation of a Knowledge Management System", *IEEE Software*, (Mai/Jun), pp. 56-59.

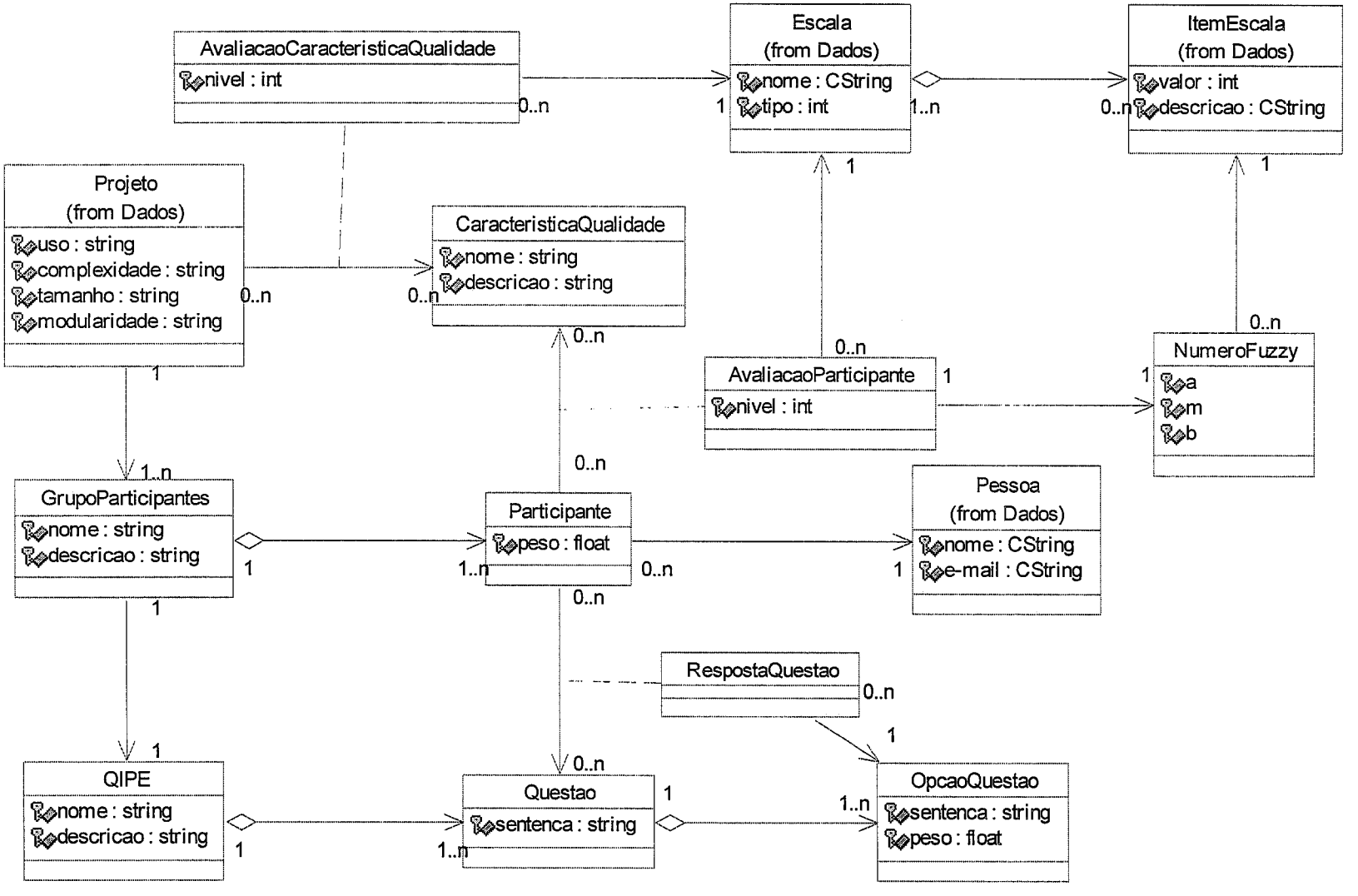
WERNECK, V. M., 1995, *Ambiente para Desenvolvimento de Software Baseado em Conhecimento*, Tese de D. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

WERNER, C. M. L., TRAVASSOS, G. H., DA ROCHA, A. C. *et al.,* 1997, "Memphis: A Reuse Based O. O. *Software* Development Environment", In: *Proceedings of TOOLS*, Beijing, China, Sep.

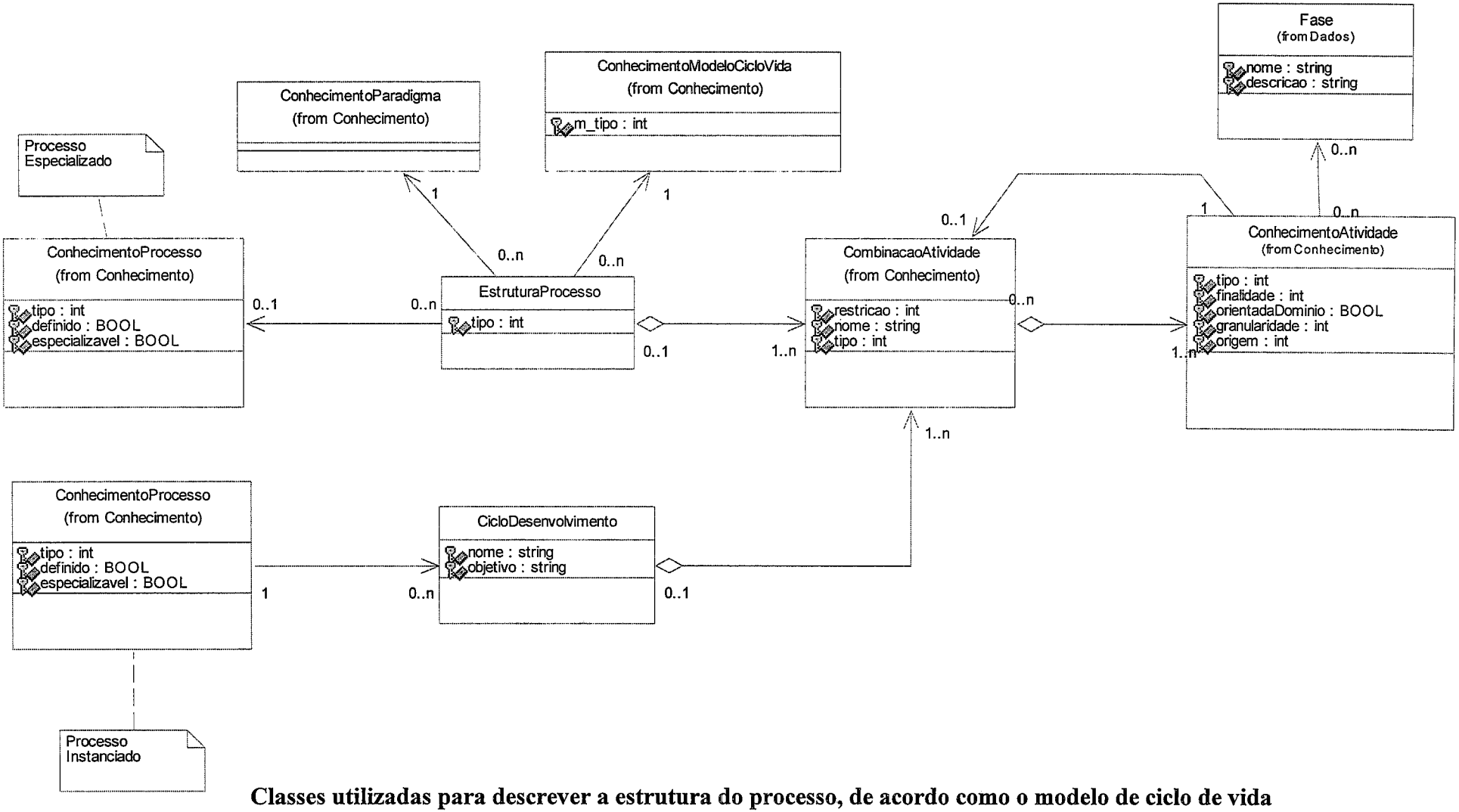
ANEXO I – ALTERAÇÕES NO MODELO DA ESTAÇÃO TABA



Classes utilizadas para descrever a caracterização do projeto e a indicação de modelos de ciclo de vida



Classes utilizadas para representar a avaliação das características de qualidade (OLIVEIRA, 1999)



Classes utilizadas para descrever a estrutura do processo, de acordo como o modelo de ciclo de vida