UNIVERSIDADE DO EXTEMO SUL CATARINENSE – UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

DOUGLAS NIERO

APLICAÇÃO DE FERRAMENTAS DE GERENCIAMENTO DE PROJETOS NO CICLO DE VIDA DO SOFTWARE

CRICIUMA, JULHO DE 2008

DOUGLAS NIERO

APLICAÇÃO DE FERRAMENTAS DE GERENCIAMENTO DE PROJETOS NO CICLO DE VIDA DO SOFTWARE

Trabalho de Conclusão de Curso apresentado para obtenção do Grau de Bacharel em Ciência da Computação da Universidade do Extremo Sul Catarinense.

Orientadora: Profª. MSc. Ana Cláudia Garcia Barbosa

CRICIUMA, JULHO DE 2008

DOUGLAS NIERO

APLICAÇÃO DE FERRAMENTAS DE GERENCIAMENTO DE PROJETOS NO CICLO DE VIDA DO SOFTWARE

Submetido ao corpo docente do Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense como um dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

_____

Profa. Msc. Ana Cláudia Garcia Barbosa
Coordenadora do Curso de Ciência da Computação

Banca Examinadora:

_____

Profa. Msc. Ana Cláudia Garcia Barbosa (UNESC)
Orientadora

_____

Prof. Msc. Alessandro Zanini (Sistemas de Informação – UNISUL/UNIBAVE)

_____

Prof. Msc. Paracelso de Oliveira Caldas (UNESC)

Aos meus excelentes pais, Santos e

Cléia, sem eles, nada disso seria possível.

AGRADECIMENTOS

Agradeço, primeiramente a Deus por iluminar o
meu caminho durante esta caminhada e a
minha família que esteve presente em todos os momentos difíceis dando
força para a concretização dos meus objetivos.
Agradeço também
a minha orientadora Ana Cláudia Garcia Barbosa, por ter-me
aceitado como orientando e por toda a atenção e dedicação
oferecida a este trabalho de conclusão de curso.
A minha namorada Marluci,
e a todos os meus amigos e colegas, pelo incentivo e
companheirismo concedidos.

“Não tentes ser bem sucedido, tenta

antes ser um homem de valor.”

(Albert Einstein)

RESUMO

A presente pesquisa refere-se a análise e aplicação de técnicas de controle automatizadas no gerenciamento do ciclo de vida de um software na busca de um produto final com qualidade. Para tal finalidade, foram evidenciados conceitos presentes na Engenharia de Software referentes a modelos, processos, normas e padrões de desenvolvimento de software, com a utilização de ferramentas de auxílio em etapas específicas do desenvolvimento do produto. Dentre as etapas possíveis de controle automatizado priorizaram-se o gerenciamento do controle de versão do código fonte, atividades executadas durante o desenvolvimento do projeto e testes unitários, com o intuito de identificar pontos passíveis de falhas. Os resultados obtidos apresentaram-se de forma satisfatória, pois evidenciaram que o gerenciamento de qualidade sugerido proporciona subsídios fundamentais para a viabilização de um software menos propício a falhas e conseqüentemente confiável.

Palavras-chave: Engenharia de Software, Gerência de Projetos, Ferramentas de gerenciamento.

ABSTRACT

The present research mentions analysis and application to it of automatized techniques of control in the management of the cycle of life of a software in the search of an end item with quality. For such purpose, the models, processes, norms and standards of software development had been evidenced concepts gifts in the referring Engineering of Software, with the use of tools of aid in specific stages of the development of the product. Amongst the possible stages of automatized control the management of the control of version of the code was prioritized source, activities executed during the development of the project and unitary tests with intention to identify possible points of imperfections. The gotten results had been presented of satisfactory form, therefore they had evidenced that the suggested management of quality provides to basic subsidies for the construction of a less propitious the imperfections and consequently trustworthy software.

Word-key: Engineering of Software, Management of Projects, Tools of management.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
AMN	Associação do Mercosul de Normalização
CMM	*Capability Maturity Model for Software*
CMMI	*Capability Maturity Model Integration*
CMU	*Carnegie Mellon University*
COPANT	Comissão Panamericana de Normas Técnicas
IEC	*International Electrotechnical Comission*
ISO	*International Organization for Standardization*
MSF	*Microsoft Solution Framework*
OMT	*Object Modeling Technique*
OOSE	*Object-Oriented Software Engineering*
PBQP	Programa Brasileiro da Qualidade e Produtividade em Software
PMBOK	*Project Management Body of Knowledge*
PMI	*Project Management Institute*
PUCPR	Pontifícia Universidade Católica do Paraná
RUP	*Rational Unified Process*
SEI	Instituto de Engenharia de Software
UML	*Unified Modeling Language*
UNB	Universidade de Brasília
XP	*Extreme Programming*

SUMÁRIO

1 INTRODUÇÃO

O constante avanço das técnicas computacionais propiciou mudanças no que diz respeito ao controle da qualidade no desenvolvimento de software. As abordagens tradicionais aplicadas na maior parte em fases tardias do ciclo de vida, passaram a ser prioridade em todo o desenvolvimento da aplicação.

Com este propósito, foram desenvolvidos padrões e ferramentas para auxiliar a equipe de gerenciamento de projetos na definição de suas tarefas dentro do ciclo de vida estabelecido, por meio de processos bem definidos. Tal fator visa automatizar a forma manual de controle presente em grande parte das organizações da atualidade por meio da Engenharia de Software.

Tal fator surgiu da necessidade do desenvolvimento de aplicações complexas de forma rápida, estruturada e com previsibilidade dos resultados obtidos, podendo evitar possíveis falhas e garantir um produto satisfatório de acordo com as expectativas do cliente/usuário.

No entanto, existe por parte das organizações a dificuldade em determinadas situações para definir seus processos padrões pela falta de conhecimento de padrões de projetos ou ainda a não disponibilidade de treinamento suficiente em novas técnicas de desenvolvimento, não aplicando as normas e ferramentas de forma adequada a todo ciclo do projeto.

Neste sentido, Pressman (1995) afirma que os profissionais da área de software têm recebido pouco treinamento formal em novas técnicas para desenvolvimento de sistemas, baseando-se em experiências anteriores onde alguns programadores desenvolvem maus hábitos que resultam na deficiência da qualidade da aplicação.

Seguindo esta concepção, Capra (2007) afirma que uma das razões da falta de entendimento dos benefícios resultantes da aplicação de um modelo de qualidade nos processos de software das empresas, dá-se pelo fato da distorção da interpretação do modelo proposto, resultando no questionamento de sua utilidade na organização.

Desta forma, procura-se com este trabalho abordar aspectos referentes ao gerenciamento do ciclo de vida dos softwares, incluindo implementação de etapas referentes a atividades específicas como o controle de versão, testes de desempenho e gerenciamento de atividades a serem desenvolvidas por meio de técnicas da Engenharia de Software, com o auxílio de ferramentas associadas a padrões propostos.

1.1 OBJETIVO GERAL

Analisar e aplicar técnicas de controle automatizadas no gerenciamento do ciclo de vida de um software.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos desta pesquisa consistem em:

a) apresentar conceitos gerais da Engenharia de Software;

b) compreender os processos do ciclo de vida dos softwares;

c) identificar pontos passíveis de gerenciamento e proporcionar a automatização de etapas de controle do desenvolvimento do software;

d) compreender e utilizar ferramentas de auxílio ao processo de software.

1.3 JUSTIFICATIVA

A necessidade do desenvolvimento de aplicações complexas, de forma rápida e eficiente, para atender a necessidade das organizações resultou no aumento da procura por normas e técnicas de gerenciamento do ciclo de vida do software visando melhoria contínua dos processos envolvidos.

Segundo números da PBQP (2006), Programa Brasileiro da Qualidade e Produtividade em Software, foi comprovado um crescimento em relação ao conhecimento da Norma ISO/IEC 12207, aprovada no Brasil em 1995, cujo objetivo é estabelecer uma estrutura comum para os processos de ciclo de vida de software com terminologia bem definida. Conforme ilustra a Figura 1, os resultados apresentados passaram de 25% no ano de 1997 para 81% em 2005.

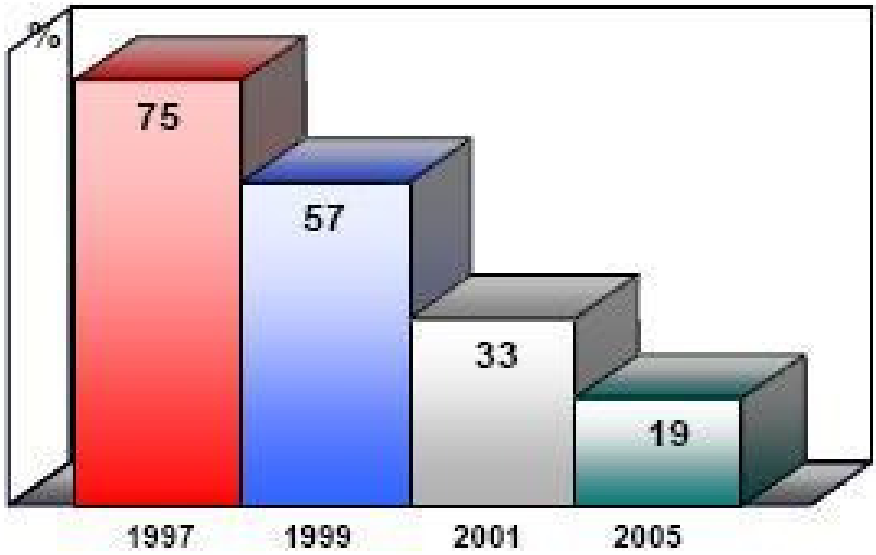

Figura 1. Desconhecimento da Norma NBR ISO/IEC 12207 (em percentual)
Fonte: PBQP (2006)

No entanto, a busca por novas tecnologias deve levar em consideração a compreensão de uma série de fatores para o funcionamento correto do controle dentro de uma organização. Devem-se abordar processos os quais envolvam todas as etapas do ciclo de vida do software para evitar problemas em fases tardias do desenvolvimento.

O controle ineficiente por parte de determinadas empresas ocorre pelo fato de não utilizarem padrões apropriados a estrutura da organização ou ainda por não aplicarem técnicas e ferramentas em etapas específicas fundamentais do desenvolvimento como, por exemplo, o controle de versão do código fonte, testes de software em etapas de desenvolvimento ou ainda gerenciamento das atividades a serem desenvolvidas.

Seguindo este contexto, Sommerville (2003) afirma que os padrões de processos podem causar dificuldades aos desenvolvedores caso não se adéqüem a sua maneira de trabalho ou não apresentem praticidade em uma de suas aplicações, pois não há necessidade de prescrever uma maneira específica de trabalhar se ela for apropriada para o projeto a ser desenvolvido.

Neste caso, pretende-se compreender os processos de ciclo de vida do software baseado na aplicação de técnicas e ferramentas de auxílio, que envolvam diversas etapas do ciclo de vida do projeto suprindo a deficiência de ferramentas que envolvem apenas o processo de modelagem de dados.

Dentre as etapas a serem implementadas, destacam-se o controle de versão do código fonte, mantendo o histórico das operações efetuadas, planejamento e execução de testes com o intuito de identificar pontos passíveis de falhas e o controle das atividades a serem desenvolvidas pelos membros da equipe, viabilizando um cronograma com tarefas e prazos definidos.

O gerenciamento das etapas citadas aborda a utilização de técnicas de desenvolvimento no processo do ciclo de vida do software em pontos específicos do projeto, utilizando conceitos de padrões de software e ferramentas de auxílio ao controle de determinadas etapas do modelo proposto.

Para tal finalidade, será utilizado o software de busca dos produtos em uma loja virtual, desenvolvido pelo acadêmico Jaison Niehues, com o intuito de adquirir informações a respeito do processo de análise de requisitos e propiciar a verificação de melhorias impostas ao mesmo com a utilização de recursos disponibilizados pela Engenharia de Software.

1.4 ESTRUTURA DO TRABALHO

Este trabalho é constituído de 8 capítulos, sendo o primeiro composto por uma introdução a pesquisa desenvolvida, seguido da descrição dos objetivos e justificativa proposta para a realização do trabalho desenvolvido.

O Capítulo 2 descreve uma breve introdução sobre a Engenharia de Software e parte do embasamento teórico referente ao ciclo de vida do software e modelos de desenvolvimento propostos na atualidade.

No Capítulo 3 são abordados os processos de desenvolvimento de software dos quais servem de base para o processo de desenvolvimento do projeto, fundamentais para esta pesquisa.

Na seqüência, o Capítulo 4 descreve as fases de desenvolvimento de software envolvidas nos processos propostos, especificando suas funcionalidades e importância de sua implementação.

O Capítulo 5 apresenta conceitos referentes a normas e padrões de qualidade, uma estrutura presente nos processos de desenvolvimento do ciclo de vida do software.

No Capítulo 6 são descritas técnicas automatizadas para o gerenciamento de projetos voltadas ao controle de atividades elaboradas, controle de código fonte e testes unitários, cujos objetivos são o aperfeiçoamento da qualidade do produto final.

O Capítulo 7 relaciona alguns trabalhos da área de desenvolvimento de software com qualidade, abrangendo técnicas, modelos e ferramentas utilizadas na atualidade para gerenciamento de projetos.

O trabalho desenvolvido é apresentado no Capítulo 8, sendo relatadas todas as etapas de metodologia utilizadas no desenvolvimento da pesquisa proposta, associados aos resultados obtidos.

Por fim, encontra-se a conclusão deste trabalho e a sugestão de temas referentes a pesquisas futuras na área de desenvolvimento de softwares de qualidade.

2 ENGENHARIA DE SOFTWARE

A Engenharia de Software é a ciência que abrange todas as etapas do desenvolvimento do software, desde os estágios iniciais de especificação do sistema até a manutenção, quando o mesmo se encontra em operação envolvendo a aplicação de teorias, métodos e ferramentas em situações adequadas (SOMMERVILLE, 2003).

Seguindo esta concepção, Pressman (1995) afirma que os métodos, ferramentas e procedimentos utilizados na Engenharia de Software possibilitam ao gerente do projeto um controle do processo de desenvolvimento do software, disponibilizando uma base para o profissional construir um software de alta qualidade no quesito produtividade.

2.1 HISTÓRICO

A evolução dos sistemas computadorizados que teve início a aproximadamente cinco décadas com o intuito do desenvolvimento de aplicações complexas para atender a necessidade dos usuários persiste até a atualidade.

Nas três primeiras décadas o principal objetivo estava voltado ao desenvolvimento de um hardware com menor tamanho e baixo custo de processamento e armazenamento de dados. Para tal finalidade os processadores a válvula foram substituídos por dispositivos microeletrônicos capazes de processar 200 milhões de instruções por segundo (PRESSMAN, 1995).

Segundo Sommerville (2003) o desenvolvimento do *hardware* de terceira geração com capacidade na época para processamento e armazenamento adequados, resultou na denominada “crise do software“ onde o surgimento de softwares maiores e

mais complexos em relação aos softwares precedentes resultou em uma série de fatores negativos como atrasos no prazo de entrega, difícil manutenção e desempenho inferior, exigindo uma abordagem formal de desenvolvimento. “Novas técnicas e novos métodos eram necessários para controlar a complexidade inerente aos grandes sistemas de software” (SOMMERVILLE, 2003, p. 4).

Com o objetivo de melhorar a qualidade de softwares complexos e conseqüentemente aumentar a produtividade, confiabilidade e satisfação profissional de engenheiros de softwares, surge o conceito de Engenharia de Software (REZENDE, 2005).

2.2 SOFTWARE

Um sistema de software, em termos gerais, consiste em um dos componentes do desenvolvimento de um sistema de informações, que compreende os módulos funcionais computadorizados dos quais interagem entre si para proporcionar aos usuários do sistema a automatização de diversas tarefas (BEZERRA, 2002).

Segundo Sommerville (2003) um sistema de software consiste em uma série de programas separados, documentação do sistema na qual descreve a estrutura do mesmo e documentação do usuário no caso de produto de software, o que contraria o conceito de pessoas que consideram o software apenas como programas de computador.

Desta forma, os softwares são construídos por engenheiros de software, personalizados para um cliente, com especificação usualmente controlada pela organização da qual adquire o software, ou de forma genérica para o mercado, controlada pela organização desenvolvedora do mesmo (SOMMERVILLE, 2003).

2.3 CICLO DE VIDA DO SOFTWARE

O ciclo de vida do software é uma seqüência de diferentes atividades aplicadas no decorrer do desenvolvimento do software, geralmente associadas a artefatos que normalmente são produtos de código fonte e manuais de usuários (GUSTAFSON, 2003).

Seguindo essa concepção, Bezerra (2002) classifica as atividades propostas como sendo as tarefas realizadas durante o processo de desenvolvimento do sistema, organizadas de acordo com a complexidade do projeto, caracterizando desta forma os modelos de ciclo de vida do software dos quais podem ser caracterizados como a representação de um sistema idealizado a ser construído.

2.4 MODELOS CICLO DE VIDA DO SOFTWARE

Um modelo de processo de software corresponde ao encadeamento específico das fases de desenvolvimento do software, diferenciando-se um dos outros pela forma das quais as diversas fases são encadeadas (BEZERRA, 2002).

Entre os modelos dos quais podem ser escolhidos de acordo com a natureza do projeto, Gustafson (2003) destaca os modelos cascata, prototipação, incremental, e espiral, como sendo os mais comuns, apresentados nas próximas seções deste documento.

2.4.1 Modelo Cascata

No modelo cascata, as fases de desenvolvimento do software são realizadas de forma seqüencial, o que não impede eventualmente a ocorrência de uma retro-alimentação entre a fase atual e anterior. A partir desse princípio, o processo inicializa-se a partir do levantamento de requisitos, seguindo de forma seqüencial para as etapas de análise de requisitos, projeto, implementação, testes e implantação, como se observa na Figura 2 (BEZERRA, 2002).

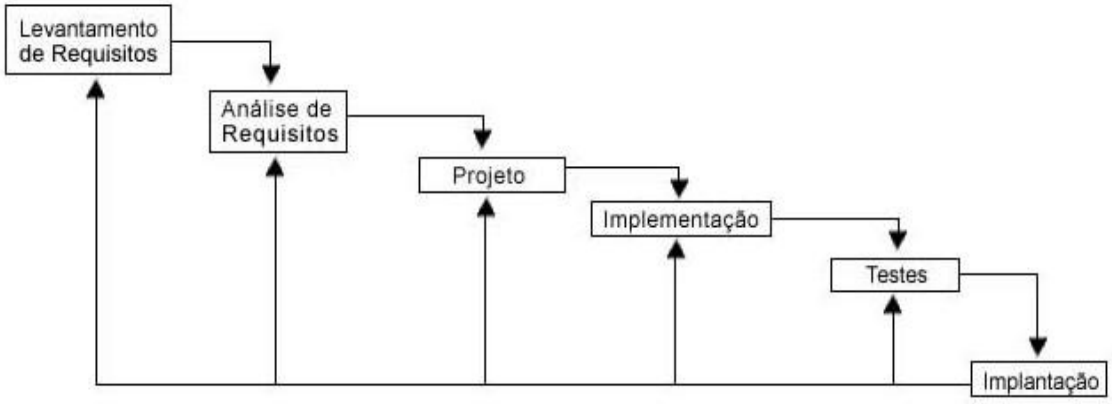

Figura 2. Representação Modelo Cascata
Fonte: BEZERRA, E. (2002)

Este modelo, segundo Sommerville (2003) apresenta problemas referentes a inflexibilidade na divisão do projeto nos estágios correspondentes. Os requisitos são levantados apenas na fase inicial do processo, impedindo eventuais mudanças necessárias no decorrer da implementação. Neste caso, o modelo cascata deve ser utilizado apenas em casos onde os requisitos forem compreendidos de forma correta.

2.4.2 Modelo de Prototipagem

Segundo Bezerra (2002) o modelo de prototipagem consiste na elaboração de protótipos[1] dos quais representam uma técnica que serve de complemento a etapa de análise de requisitos.

Após o levantamento dos requisitos, um protótipo é desenvolvido para validar as informações perante os usuários envolvidos, sendo modificado até que o mesmo atenda as exigências impostas (BEZERRA, 2002).

Neste sentido, Gustafson (2003) complementa que depois de aceito pelos clientes o protótipo pode ser descartado e inicializa-se o desenvolvimento do sistema conforme as fases do modelo cascata, com a vantagem de não conter características desnecessárias ao projeto.

2.4.3 Modelo Incremental

Este modelo consiste na divisão do desenvolvimento do software em ciclos compostos pelas etapas de análise, projeto, implementação e testes realizadas a cada ciclo. Esta característica possui o propósito de solucionar os problemas encontrados no modelo cascata, no qual as etapas citadas são desenvolvidas uma única vez antes do início do projeto (BEZERRA, 2002).

Seguindo esta concepção Sommerville (2003) afirma que o modelo incremental procura associar um modelo simples de gerenciamento que resulta em sistemas robustos e flexíveis a mudanças, características do modelo cascata, com a

---

[1] No contexto do desenvolvimento do software consiste em um esboço de alguma parte do sistema.

capacidade de preterir aspectos referentes a decisões do projeto desenvolvendo-o de forma estruturada e de fácil interpretação e manutenção.

Este abordagem deve levar em consideração a divisão dos requisitos em partes das quais são alocadas a um ciclo de desenvolvimento específico com grau de prioridade estabelecido pelos usuários e risco de cada requisito proposto. Tal fator permite que inconsistências de requisitos possam ser identificadas e solucionadas em etapas anteriores ao término do desenvolvimento de todo o sistema (BEZERRA, 2002).

2.4.4 Modelo Espiral

"Em vez de representar o processo de software como uma seqüência de atividades com algum retorno de uma atividade para outra, o processo é representado como uma espiral" (SOMMERVILLE, 2003, p. 44). Desta forma, um *loop* da espiral é considerado uma das fases do projeto, representado pelos seguintes setores:

a) **definição de objetivos:** são identificados os objetivos específicos, as restrições e riscos do projeto e elaborado um plano de gerenciamento detalhado;

b) **avaliação e redução de riscos:** são realizadas análises detalhadas dos riscos identificados no projeto e desenvolvidas alternativas para evitar esses riscos;

c) **desenvolvimento e validação:** validados os riscos encontrados, deve-se escolher o modelo a ser utilizado no desenvolvimento do projeto de acordo com as características dos erros em questão;

d) **planejamento:** o projeto deve ser revisto e se aprovado são traçados planos para a próxima fase do projeto seguindo o *loop* da espiral.

Pode-se afirmar que o modelo espiral não induz a um desenvolvimento em fases fixas, como por exemplo, especificação ou projeto, utilizando as características dos modelos apresentados anteriormente de acordo com a análise dos riscos encontrados (SOMERVILLE, 2003).

3 PROCESSOS DE DESENVOLVIMENTO

Um processo de software corresponde a uma atividade complexa cujo objetivo é compreender todas as etapas de desenvolvimento do software, desde sua concepção, com o intuito de minimizar os problemas relacionados a complexidade do desenvolvimento do projeto (BEZERRA, 2002).

Inthurn (2001) concorda e afirma que um processo de software adequado deve levar em consideração a relação entre todas as atividades, métodos e ferramentas envolvidas no decorrer do projeto, com a finalidade de proporcionar uma ampla visão do processo de desenvolvimento.

Entre os modelos de processos utilizados no aperfeiçoamento do desenvolvimento do software, Martins e Silva (2004) destacam o *Rational Unified Process (RUP), Extreme Programming (XP), Microsoft Solution Framework (MSF) e Unified Modeling Language (UML).*

3.1 RATIONAL UNIFIED PROCESS

O *Rational Unified Process (RUP)* é uma metodologia completa criada pela *Rational Software Corporation*, com uma abordagem voltada à orientação de objetos cujo objetivo é viabilizar o desenvolvimento de projetos dinâmicos e complexos de forma eficaz em menor espaço de tempo (VIANNA, 2002).

"Sua meta principal é garantir a produção de software com alta qualidade satisfazendo as necessidades dos seus usuários, dentro de um cronograma e orçamento previsível" (VASCONCELOS, 2005, p. 1).

O *Rational Unified Process* engloba os modelos tradicionais de desenvolvimento do projeto de software, diferenciando-se na forma de organização das fases propostas, representadas pela Figura 3. O processo inicia-se na fase de concepção na qual é responsável pelo estudo de viabilidade, seguida pelas fases de elaboração caracterizada pela análise de requisitos e projeto, de construção que corresponde a codificação e testes e por fim a fase de transição referente as etapas de implantação e manutenção do sistema (WAZLAWICK, 2004). Essas fases são divididas em componentes dinâmicos, representados pelo tempo e aspectos que se modificam no decorrer do projeto como os ciclos, fases, iterações, e componentes estáticos, dos quais representam a equipe de desenvolvimento, atividades e artefatos (MARTINS, 2006).

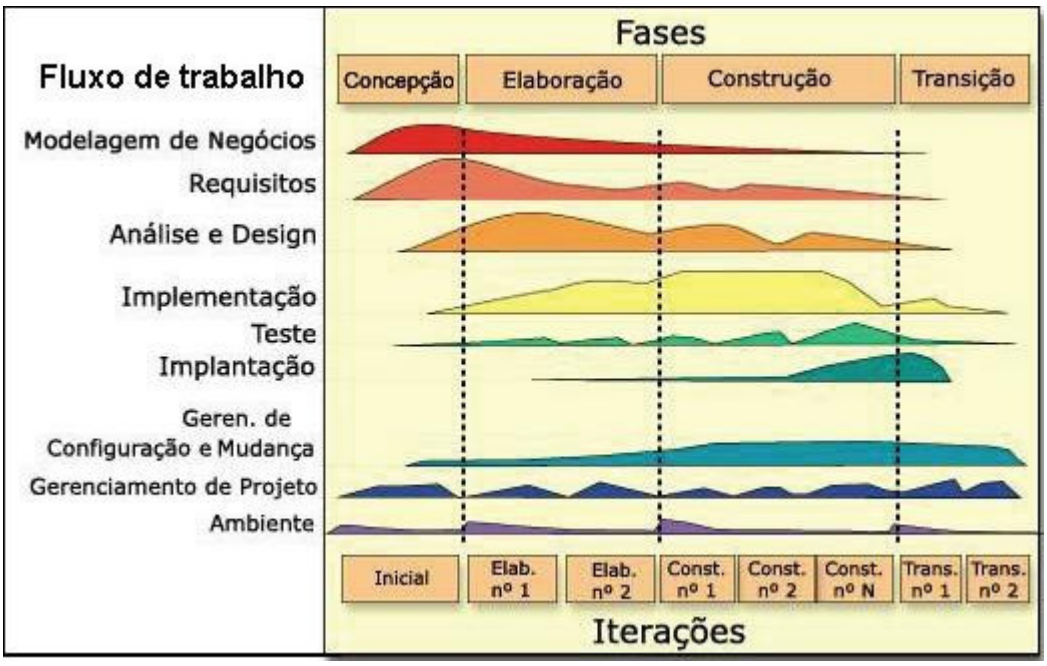

Figura 3. Processo básico do RUP
Fonte: VIANNA, M. (2002).

Segundo Martins (2006) e Vasconcelos (2005) o RUP é constituído das melhores práticas de desenvolvimento de software disponíveis na atualidade, das quais se podem destacar o desenvolvimento iterativo, gerenciamento de requisitos,

arquiteturas baseadas em componentes, verificação da qualidade e controle de mudanças de software:

a) **desenvolvimento iterativo:** condução do projeto em iterações onde são analisados e solucionados os riscos críticos ao final de cada iteração;
b) **gerenciamento de requisitos:** consiste em adquirir, planejar e documentar as funcionalidades necessárias ao sistema de forma flexível a mudanças;
c) **arquiteturas baseadas em componentes:** demonstra como projetar uma arquitetura flexível com reutilização de código;
d) **verificação constante da qualidade do software:** desenvolver atividades para solucionar problemas e conseqüentemente controlar e garantir a qualidade de software durante todo o ciclo de vida do projeto no que diz respeito à funcionalidade, confiabilidade e performance;
e) **controle de mudanças do software:** consiste no controle das mudanças dos requisitos e software modificados durante o projeto.

Para Scott (2003) as atividades apresentadas durante as fases de desenvolvimento do processo da *Rational Unified Process* podem ser customizadas, ocorrendo o acréscimo ou eliminação de atividades, de acordo com o objetivo proposto do projeto em relação a funcionalidades específicas ou recursos necessários disponíveis. Isso faz da RUP um processo com estrutura genérica, da qual realiza uso constante da *Unified Modeling Language* no auxílio a compreensão de fatores complexos essenciais envolvidos no sistema.

3.2 EXTREME PROGRAMMING

O *Extreme Programming (XP)* é um processo de desenvolvimento de software, criado no final da década de 1980, nos Estados Unidos, por Kent Beck. A técnica passou por refinamentos aplicados em diversos projetos ampliando o desenvolvimento de software adaptável e orientado a pessoas, com o intuito de produzir software com qualidade, de forma ágil e econômica (VASCONCELOS, 2005).

Seguindo este contexto, Caetano (2007) afirma que o XP envolve um processo simples, dividido em pequenas iterações das quais são realizadas o escalonamento dos requisitos levantados em contato direto com os clientes. Cada iteração por sua vez, é submetida a todas as fases do ciclo de vida do projeto, adicionadas de testes, com o intuito de identificar os problemas referentes a concepção ou programação e viabilizar sua correção em etapas iniciais de desenvolvimento com custos menores quando comparados a descoberta dos erros em fases tardias, conforme se observa na Figura 4.

Figura 4. Custo médio da correção de um defeito durante o desenvolvimento do software
Fonte: CAETANO, C. (2007)

Segundo Gervazoni (2005) e Vasconcelos (2005) a metodologia do processo XP envolve um processo de baixo risco com um ciclo relativamente curto, diferenciando-se, a primeira vista, dos processos convencionais, sendo eficiente em projetos nos quais as alterações de requisitos são fatores preponderantes.

Neste contexto, pode-se dividir o ciclo de vida da XP basicamente em quatro fases: planejamento, responsável pela aquisição dos requisitos extraídos juntamente a clientes, seguidas pelos testes, realizados após o termino da iteração atual, a codificação, realizada para atender aos testes executados, e por fim a reconstrução do projeto à medida que novas funcionalidades são incrementadas ao software (VASCONCELOS, 2005).

Segundo Vasconcelos (2005) independentemente da dimensão do projeto, a XP permite a elaboração de um planejamento referente a definição de cronogramas e custos em tempo hábil ao desenvolvimento do sistema com ausência de detalhes desnecessários, levando em consideração as constantes mudanças nos requisitos propostos. Desta forma, o XP vem obtendo o reconhecimento em relação a resultados obtidos na agilidade do método e enfoque na codificação, fatores que ressaltam aspectos fundamentais pouco enfatizados.

Dentre as práticas relacionadas a codificação, caracterizam-se o seguimento de padrões previamente definidos com a utilização da refatoração, responsável pelo aperfeiçoamento de um sistema, baseando-se na reestruturação do código sem alterar seu comportamento, a programação desenvolvida em pares de programadores e a construção de versões operacionais a cada tarefa finalizada pela equipe de desenvolvimento (SCOTT, 2003).

3.3 MICROSOFT SOLUTION FRAMEWORK

O MSF foi originado pela Microsoft, a partir do estudo dos procedimentos de desenvolvimento de seus produtos, abordando a utilização de práticas consideradas adequadas pela organização, flexíveis e com ausência de detalhes, com a finalidade de alcançar como resultado soluções satisfatórias. Mas este processo, não necessariamente, se prende a utilização de produtos da Microsoft (VIANNA, 2001).

Segundo Martins e Silva (2004) e Vianna (2001) a ausência de detalhes no MSF é uma característica que possibilitou uma abordagem simples, objetiva e de fácil compreensão entre os envolvidos no projeto das técnicas apresentadas, sendo constituídas de quatro elementos básicos: Princípios Fundamentais, Disciplinas, Modelo de Equipe e Modelos de Processo.

Os Princípios fundamentais são caracterizados por aspectos referentes ao entendimento do foco do negócio, com comunicação ampla entre membros da equipe e esclarecimentos a respeito das responsabilidades compartilhadas. Além do mais, cada indivíduo deve executar suas atividades de forma ágil, com investimento em qualidade para assegurar o desenvolvimento de um projeto eficiente, com custos e funcionalidades desejadas (GERVAZONI, 2005).

Para Gervazoni (2005) e Martins e Silva (2004) as disciplinas do MSF são áreas práticas que envolvem um conjunto de métodos e abordagens específicas, indispensáveis no decorrer do desenvolvimento do projeto, representadas pelo: gerenciamento de projeto, aplicando o processo *Standard Industry* de gerenciamento para princípios do processo com adição de recursos de auxílio para obtenção de sucesso; gerenciamento de riscos, cujo objetivo é a redução de riscos capazes de impossibilitar o funcionamento do projeto; e gerenciamento de aprendizado,

responsável pela identificação das habilidades requeridas pela equipe, alocando recursos necessários ao projeto.

O Modelo de Equipe *(Team Model)* tem por objetivo a atribuição das responsabilidades e definição dos papéis a serem seguidos pelos membros da equipe. Desta forma, Vianna (2001) afirma que todos os seis papéis da *MSF* devem ser representados por tratarem de objetivos necessários ao projeto, descrevendo os seguintes:

a) ***product management:*** possui o objetivo de representar a equipe ao cliente e vice-versa, com a finalidade de promover a satisfação do cliente;

b) ***program management:*** caracteriza-se pelo gerenciamento do desenvolver do projeto, propiciando alterações sugeridas pelo *Product Management* sem afetar o prazo e custos estipulados;

c) ***development:*** responsável pelo projeto em si e construção de soluções para atender as especificações impostas;

d) ***testing:*** elaborar testes referentes a construção de soluções a fim de abordar problemas antes da implantação do projeto;

e) ***user education:*** responsável pela otimização da performance do usuário por meio de treinamentos;

f) ***logistics management:*** planejamento e execução de logística com o intuito de assegurar uma implantação e manutenção satisfatórias.

Segundo Martins e Silva (2004) o modelo de Processo *(Process Model)* consiste na união dos benefícios alcançados por meio dos elementos básicos, baseando-se em fases de desenvolvimento divididas em: visão, planejamento, implementação, estabilização e instalação conforme a Figura 5.

Figura 5. Visão geral das fases do Modelo de processo
Fonte: MICROSOFT (2006).

3.4 UNIFIED MODELING LANGUAGE

A *Unified Modeling Language* (UML) é uma linguagem padrão de desenvolvimento de software, originada no ano de 1996, cujo objetivo principal era criar uma estrutura padronizada de modelagem de sistemas aceita pelos desenvolvedores de software. Sua fundamentação surgiu a partir do aproveitamento das melhores características extraídas dos métodos de *Booch (Booch Method)* de Grady Booch, o OMT (*Object Modeling Technique* - Técnica de Modelagem de Objetos) de James Rumbaugh e o OOSE (*Object-Oriented Software Engineering* - Engenharia de Software Orientada a Objetos), técnicas de modelagem orientadas a objeto propostas na época (BEZERRA, 2002).

A UML consiste em uma linguagem gráfica na qual pode ser utilizada na visualização, especificação, construção e documentação dos elementos de sistemas de software complexos. Sua definição consiste basicamente em um conjunto de especificações das quais representam a modelagem do sistema proposto na forma de diagramas com grau de abstração ou detalhamento desejado (QUADROS, 2002).

Segundo Martins (2006) um diagrama pode ser caracterizado como uma apresentação gráfica de elementos de uma série de elementos relacionados entre si, dos quais permitem obter diferentes visões do software proposto. Dentre os principais tipos de diagramas suportados pela UML, podem-se destacar:

a) **diagrama de classes:** representam a classe, interface e o relacionamento entre elas;
b) **diagrama de objetos:** mostra os objetos e seus relacionamentos;
c) **diagrama de caso de uso:** mostra atores, casos de uso e seus relacionamentos;
d) **diagrama de interação:** modela a interação entre objetos com inclusão dos relacionamentos e as mensagens trocadas por ambos;
e) **diagrama de seqüência:** similar ao diagrama de interação com foco voltado as mensagens trocadas no tempo pelos objetos, ao invés de destacar o relacionamento entre eles;
f) **diagrama de estados:** mostra os estados, transições e eventos de um software com mudanças de estado num determinado objeto;
g) **diagrama de atividades:** demonstra o fluxo das atividades relacionadas a uma rotina;
h) **diagrama de componentes:** mostra a organização e dependências entre os componentes;

i) **diagrama de *deployment:*** visualiza a distribuição dos componentes através da organização;

No entanto, cabe ressaltar que a UML é uma linguagem de modelagem independente de linguagem de programação e forma de desenvolvimento a serem adotadas. Neste caso, ela pode ser utilizada em diferentes tipos de software dos quais envolvem diferentes abordagens de desenvolvimento (BEZERRA, 2002).

4 FASES DE PROCESSOS DE DESENVOLVIMENTO

Existem diversos processos de desenvolvimento, cada processo tem suas particularidades em relação à forma de encadear as atividades a serem seguidas, podendo distinguir atividades que, sofrendo ou não modificações, estão presentes na maioria dos processos existentes, como o levantamento e análise de requisitos, projeto, implementação, testes e implantação (BEZERRA, 2002).

4.1 LEVANTAMENTO DE REQUISITOS

A etapa do levantamento de requisitos possui a finalidade de identificar as necessidades do sistema a ser desenvolvido por meio de entrevistas com usuários, seguidas da elaboração de um plano contendo as informações referentes a limitações de recursos, prazos e orçamentos (INTHURN, 2001).

"A atividade de levantamento de requisitos corresponde à etapa de compreensão do problema aplicada ao desenvolvimento de software" (BEZERRA, 2002, p. 20). Com esta definição, Bezerra (2002) afirma que o objetivo principal desta etapa consiste no levantamento e definição das necessidades do sistema a ser desenvolvido, geralmente denominados requisitos, unificando a visão do problema entre desenvolvedores e usuários.

Os requisitos a serem levantados devem levar em consideração as funcionalidades do sistema, características de qualidade, como por exemplo, o tempo de resposta esperado para determinadas funções e declaração de restrições impostas a respeito do desenvolvimento como a adequação de custos e prazos a serem estipulados.

Atualmente, os requisitos devem apresentar um grau de volatilidade[2] para atender as mudanças cada vez mais rápidas decorrentes nas organizações, seja em relação a tecnologias aplicadas, expectativas de usuários ou regras de negócio, contrapondo o conceito usado no início da modelagem de sistemas, onde os mesmos eram utilizados em todas as etapas do desenvolvimento sem sofrerem alterações.

O levantamento de requisitos é de fundamental importância em termos de retorno em investimentos realizados para o desenvolvimento adequado do projeto devendo atender às necessidades dos clientes com custos e prazos definidos, aumentando o tempo do ciclo de vida do software.

4.2 ANÁLISE DE REQUISITOS

A análise de requisitos corresponde, em termos de sistemas de softwares, ao estudo detalhado por parte dos analistas das informações levantadas na etapa descrita anteriormente, com o intuito da construção de modelos dos quais representam o sistema a ser desenvolvido (BEZERRA, 2002).

O resultado obtido da análise de requisitos em um processo desenvolvido orientado a objetos não caracteriza apenas modelos dos quais retratam a estrutura de classes de objetos dos quais compõem o sistema como também modelos que especificam as funcionalidades do software proposto (BEZERRA, 2002).

Para Sommerville (2003) a análise de requisitos envolve diferentes tipos de pessoas em uma organização por se tratar da fase onde os membros da equipe de

---

[2] Pode sofrer modificações no decorrer do desenvolvimento do sistema.

desenvolvimento procuram extrair dos usuários as informações necessárias referentes ao software proposto.

Neste caso, a análise de requisitos é considerada um processo de difícil implementação pelo fato das informações obtidas por analistas não demonstrarem de forma objetiva o que se deseja alcançar do sistema computacional, seja pelo fato do cliente possuir conhecimento implícito apenas na sua área de atuação ou ainda expressar os requisitos em termos próprios os quais dificultam a compreensão (SOMMERVILLE, 2003).

4.3 PROJETO

Um projeto de software consiste em um conjunto de documentos, normalmente diagramas, desenvolvidos em *Unified Modeling Language* que expressam a forma na qual a aplicação deve ser construída descrevendo as etapas envolvidas no sistema e como elas devem ser montadas (BRAUDE, 2005).

Neste sentido, Bezerra (2002) relata que esta fase determina o futuro funcionamento do sistema por meio da produção de uma descrição computacional com as funcionalidades na qual o software deverá realizar levando em consideração aspectos físicos e a coerência com a análise de requisitos, caracterizada por duas atividades principais:

a) **projeto de arquitetura:** em um processo voltado a orientação a objetos, consiste na distribuição das classes dos objetos relacionadas em subsistemas e seus componentes, os quais são distribuídos fisicamente pelos recursos de *hardware* disponíveis;

b) **projeto detalhado:** consiste na modelagem das colaborações existentes entre objetos de cada módulo com o objetivo de aplicar as funcionalidades do módulo e na realização do projeto do banco de dados e interface com o usuário.

Ao final desta etapa é realizada a especificação do projeto no qual possui o objetivo de relatar o impacto do projeto inteiro em relação a fatores do ambiente operacional referentes a arquitetura de *hardware* e estrutura do banco de dados necessária para o software desenvolvido (INTHURN, 2001).

4.4 IMPLEMENTAÇÃO

Em termos gerais, a implementação consiste na codificação do sistema, onde é realizada a tradução da descrição computacional da fase de projeto em código executável por meio do uso de linguagens de programação (BEZERRA, 2002).

Seguindo esta concepção Inthurn (2001) afirma que no decorrer da implementação devem ser realizados testes iniciais com o propósito de eliminar possíveis erros de lógica voltados a programação e análise do resultado da execução do programa, com o intuito de verificar se o mesmo atende as especificações impostas em fases anteriores.

4.5 TESTES

"O objetivo do processo de teste é avaliar a qualidade do produto, verificando e corrigindo problemas e má interpretação dos requisitos" (MARTINS, 2006, p. 211).

Com esta definição, Martins (2006) diz que é necessário elaborar testes durante todo o ciclo de vida do projeto para proporcionar controle e garantia do sistema em relação a quesitos como funcionalidade, confiabilidade e performance.

Martins (2006) concorda com Inthurn (2001) e afirma que os custos referentes a eliminação dos problemas após a codificação são relativamente maiores se comparados a remoção dos mesmos no início do projeto, ressaltando a importância da realização de testes desde as etapas iniciais do desenvolvimento. Ainda complementa que a realização de testes visa reduzir a margem de erros e não eliminar todo o risco do desenvolvimento de softwares com falhas.

4.6 IMPLANTAÇÃO

A fase de implantação consiste na instalação do software no ambiente operacional do cliente, seguido da importação dos dados para o sistema e treinamento dos usuários com a finalidade de propiciar a utilização adequada do produto (BEZERRA, 2002).

Seguindo este conceito, Leme Filho (2003) afirma que as duas principais atividades da fase de implantação correspondem a migração de dados, quando o sistema proposto vem a substituir um já existente, e treinamento de usuários, que caso tratado de forma inadequada pode ocasionar a não utilização de projetos.

5 QUALIDADE DE SOFTWARE

A qualidade de um software é um assunto complexo no qual consiste em um conjunto de procedimentos aplicáveis a serem satisfeitos de modo que o projeto atenda as necessidades de seus usuários de forma adequada. Para tal acontecimento, deve-se salientar a importância da especificação e modelagem das informações referentes ao projeto desde as fases iniciais de desenvolvimento com o envolvimento de todos os membros da equipe (INTHURN, 2001).

Para Pressman (2002) a qualidade de projeto refere-se as características que os projetistas determinam para um determinado item, onde os fatores ligados a tolerâncias e especificações de desempenho contribuem de forma significativa para a qualidade do projeto.

Rezende (2005) concorda com Presmann (2002) e afirma que o conceito de qualidade consiste em aspectos voltados a conformidade com os requisitos, adaptabilidade ao uso e atendimento confiável em tempo hábil das necessidades dos usuários. Assim, após a implantação do projeto na organização o usuário/cliente obterá qualidade nas informações de forma precisa e conseqüentemente confiável.

Segundo Sommerville (2007) o uso de novas técnicas de gerenciamento de qualidade provenientes da manufatura de software vem se aprimorado de forma significativa nos últimos 15 anos. No entanto, o produto desenvolvido deve não somente atender as necessidades impostas pelos clientes, como também requisitos dos quais não se encontram incluídos na especificação, como por exemplo, a facilidade na manutenção do produto.

5.1 IMPORTÂNCIA DO GERENCIAMENTO DE QUALIDADE

O desenvolvimento de software de qualidade é uma atividade complexa que se tornou um fator diferencial na competitividade entre as organizações presentes em um mercado da atualidade cada vez mais exigente, tornando-se indispensável para sustentar o software ativo no mercado.

Atingir o nível elevado de qualidade do produto é o objetivo da maioria das organizações. “Atualmente não é mais aceitável entregar produtos com baixa qualidade e reparar os problemas e as deficiências depois que os produtos foram entregues ao cliente” (SOMMERVILLE, 2003, p. 458).

Desta forma, “As empresas mais competitivas são as empresas que trabalham sob a ótica da melhoria contínua dos processos para aumentar a qualidade do processo de desenvolvimento e, conseqüentemente, aumentar a qualidade do produto final” (CAETANO, 2006, p.1).

O gerenciamento de qualidade formalizada é de fundamental importância para as equipes de desenvolvimento de sistemas amplos e complexos. Tal fator permite, à medida na qual o software é desenvolvido, a análise das atividades implementadas e correção de não conformidades decorrentes de suposições incorretas atingindo a garantia de qualidade exigida do produto (SOMMERVILLE, 2007).

Segundo Martins (2006) e Sommerville (2007) o gerenciamento de qualidade do software pode ser estruturado em três atividades principais denominadas de garantia de qualidade, planejamento de qualidade e controle de qualidade:

a) **planejamento de qualidade:** busca definir os procedimentos e padrões de qualidade adequados dos quais precisam ser seguidos. Seu objetivo é

estabelecer um plano capaz de descrever as atividades a serem exercidas pela equipe do projeto associadas ao cumprimento de custos e prazos estabelecidos, com o intuito de garantir a qualidade do software e satisfação perante o cliente/usuário.

b) **garantia de qualidade:** visa relatar as dificuldades e problemas diagnosticados e assegurar a consistência do projeto com os padrões e objetivos da organização. A equipe de garantia de qualidade deve ser independente da equipe de desenvolvimento para que se possa obter uma visão objetiva do software sem comprometer a qualidade por orçamentos de curto prazo.

c) **controle de qualidade:** consiste na definição e aprovação de processos dos quais asseguram que o desenvolvimento do software tenha seguido os padrões de qualidade pela equipe de desenvolvimento do projeto. Desta forma, se houver algum item fora do padrão estabelecido deve-se descartar o mesmo ou refazê-lo dentro das especificações apropriadas.

Para Bezerra (2002) e Rezende (2005) um projeto de qualidade deve possuir as características de desempenho e confiabilidade, atendendo aos padrões de qualidade predefinidos pela equipe de projeto da organização na qual desenvolve o produto.

Seguindo esta concepção, Caetano (2006) afirma que uma abordagem padronizada é de fundamental importância pelo fato de estabelecer uma linguagem comum entre os membros de uma equipe e, conseqüentemente, aperfeiçoar a execução de todas as tarefas. Essencialmente, um padrão estabelece dimensão às atividades tradicionais e, a melhor forma de executá-las, servindo de alicerce ao processo de desenvolvimento do software, garantindo a obtenção de resultados previsíveis e a

redução de custo decorrente da correção de falhas do sistema em fases iniciais do desenvolvimento.

5.2 NORMAS E PADRÕES DE QUALIDADE

O Processo de definição de como a qualidade de software pode ser atingida e como a organização de desenvolvimento avalia o nível de qualidade do software como adequado está diretamente relacionado a definição e utilização de padrões a serem aplicados ao processo de desenvolvimento de software (SOMMERVILLE, 2007).

Ainda na concepção de Sommerville (2007) os tipos de padrões estabelecidos no processo de desenvolvimento de software com garantia de qualidade podem ser classificados como padrões de produto e padrões de processo:

a) **padrões de produto:** aplicados na fase de desenvolvimento de software onde são incluídos os padrões de codificação e documentação;
b) **padrões de processo:** responsável pela definição dos processos a serem seguidos durante o desenvolvimento do software. Nestes padrões são inseridas definições de processos de especificação, projeto e validação.

Para que estes padrões de qualidade interajam de forma eficaz no processo de desenvolvimento do software algumas características devem ser observadas de forma independente ao modelo de qualidade proposto. Cavalcante e Weber (1999) apresentam um conceito sobre estas características:

a) **funcionalidade:** conjunto de funções das quais satisfazem as necessidades específicas;

b) **confiabilidade:** manter o nível de desempenho adequando o tempo e condições estabelecidas;

c) **usabilidade:** o software deve ser de fácil compreensão e utilização para o usuário final;

d) **eficiência:** o software deve apresentar um grau de desempenho adequado independentemente da quantidade dos recursos utilizados;

e) **manutenibilidade:** o software deve ser flexível a mudanças específicas como melhorias, correções e adaptações;

f) **portabilidade:** o software deve estar apto a ser transferido de um ambiente para outro sem esforços excessivos.

"Um problema com a garantia de qualidade baseada em processos é que a equipe de garantia de qualidade pode insistir em que os processos padrões devem ser utilizados independentemente do tipo de software em desenvolvimento" (SOMMERVILLE, 2007, p. 425).

Desta forma, evidencia-se que os padrões de software devem ser adequados de acordo com o tipo de software em questão evitando problemas referentes a elevação de custos e prazos propostos, propiciando a qualidade de desenvolvimento desejada.

5.2.1 Normas ISO e ABNT

A *International Organization for Standardization* (ISO) é uma entidade organizacional internacional representada por membros de diversos países do mundo cujo objetivo é promover o desenvolvimento de padrões industriais unificados estabelecendo uma comunicação de fácil coordenação por meio de especificações de

regras e técnicas com o intuito de garantir a adequação de produtos, serviços ou processos (NERI, 2004).

A ISO regulamenta padrões referentes à Engenharia de Software, tais como as normas: ISO 9000, ISO 9001, ISO 12207 e ISO 9621-1 na qual veio substituir a norma NBR 13596, estabelecendo uma estrutura para processos de desenvolvimento do ciclo de vida do software.

Para a regulamentação da normalização técnica no país, foi fundada em 1940 a Associação Brasileira de Normas Técnicas (ABNT), uma entidade privada, sem fins lucrativos responsável por estabelecer a base necessária ao desenvolvimento tecnológico brasileiro.

A ABNT é um membro fundador da *International Organization for Standardization* (ISO), da Comissão Panamericana de Normas Técnicas (COPANT) e da Associação do Mercosul de Normalização (AMN), sendo a única representante exclusiva no Brasil dessas entidades acrescida da *International Electrotechnical Comission* (IEC) (ABNT, 2006).

5.2.1.1 NBR ISO 9000

A ISO 9000 consiste em um conjunto de padrões dos quais o uso é destinado ao desenvolvimento de sistemas de gerenciamento de qualidade em indústrias dos mais diversificados ramos de atuação que variam de manufatura a indústria de serviço (SOMMERVILLE, 2007).

Pressman (2002) complementa que a ISO 9000 não impõe a maneira na qual a organização deve implementar seu sistema de qualidade e o seu desafio consiste

em desenvolver um sistema no qual atenda as normas especificadas e se adéqüe aos produtos, serviços e cultura da empresa.

Sendo assim, o modelo de garantia de qualidade ISO 9000 traz consigo o conceito de que uma organização é uma rede de processos interligados onde se faz necessário que cada área do projeto esteja dentro dos padrões estabelecidos para existir conformidade com a ISO, sendo documentados e implementados conforme descritos (PRESSMAN, 2002).

No entanto, a definição e utilização de processos não necessariamente resultam em garantia de qualidade do software, ou ainda que uma organização na qual possua a certificação ISO 9000 desenvolva softwares de melhor qualidade quando comparadas a organizações não certificadas.

> O ISO 9000 está simplesmente relacionado com a definição de processos que serão usados em uma empresa e a documentação associada, como processos de controle que podem explicitamente mostrar que esses processos foram seguidos. Isto não está relacionado com a garantia de que os processos refletem as melhores práticas ou com a qualidade de produto (SOMMERVILLE, 2007, p. 428).

5.2.1.2 NBR ISO 9001: 2000

Segundo Valls (2004) a ISO 9001 é a norma de certificação na qual define os requisitos básicos para a implantação de um sistema de gestão de qualidade. Esse padrão foi publicado no Brasil em dezembro de 2000 e constitui um elemento da nova família das normas da série 9000.

O padrão ISO 9001 pode ser caracterizado como o mais geral da série ISO 9000, não voltado especificamente para o desenvolvimento de software. Apesar deste fator, ele estabelece princípios dos quais podem ser aplicados ao software com o apoio

de um documento contendo um conjunto de diretrizes (ISO 9000-3) desenvolvido para interpretar a norma para o uso no processo de software (SOMMERVILLE, 2007).

"O padrão ISO 9001 descreve vários aspectos do processo de qualidade e exibe os padrões e os procedimentos organizacionais que a empresa deve definir" (SOMMERVILLE, 2007, p. 427). Dentre os procedimentos a seguir, pode-se destacar a identificação dos requisitos do cliente necessários para o sistema de gestão da qualidade, estabelecimento de medidas de desempenho para alocação e gerenciamento de recursos de forma coesa e por fim a melhoria do desempenho do processo para assegurar a eficácia do controle dos processos (VALLS, 2004).

Apesar da escolha do processo a ser adotado pela organização ser flexível no padrão ISO 9001, sua definição deve incluir descrições de documentação dos procedimentos voltados a garantia de qualidade dos quais devem estar registrados em um manual da empresa para que possa ser analisado o cumprimento dos processos definidos no decorrer do desenvolvimento do projeto (SOMMERVILLE, 2007).

Segundo Valls (2004) o fato de uma organização possuir o certificado NBR ISO 9001 não necessariamente afirma que a mesma não esta sujeita a ocorrência de falhas ou problemas, mas possibilita uma forma de gerenciamento de recursos, previsibilidade antecipada de riscos e custos mais eficazes e satisfatórios em relação a seus clientes.

5.2.1.3 NBR ISO/IEC 9126-1

A NBR ISO/IEC 9126 é uma norma elaborada pela Associação Brasileira de Norma Técnicas (ABNT), com o título "Engenharia de Software – Qualidade de produto", na qual descreve um modelo de qualidade do produto de software, que veio a substituir a NBR 13596:1996.

A NBR ISO/IEC 9126 pode ser aplicada na definição de requisitos da qualidade e em aspectos voltados a avaliação do produto de software, como a medição e pontuação de forma que as organizações possam estabelecer seus próprios critérios de avaliação e métodos para validação de métricas relacionadas as características encontradas na sua especificação. Desta forma, possibilita-se com o uso na norma segundo (INTHURN, 2001):

a) definição de requisitos necessários referentes a qualidade de um produto de software;

b) avaliação das especificações do produto com o objetivo de garantir a qualidade referente ao controle dos requisitos em todas as etapas de desenvolvimento do software;

c) descrição das características do software desenvolvido por meio de manuais de usuários juntamente com seus atributos absorvidos em cada etapa de desenvolvimento;

d) avaliação do produto final no momento que antecede a entrega do mesmo ao cliente/usuário.

Segundo Campos (2007) esta norma retrata um modelo de qualidade de produto de software na qual é constituída de duas partes: qualidade interna e externa e a qualidade em uso, onde são especificadas as características já mencionadas neste documento e divididas em sub-características manifestadas de forma externa, no momento em que o software tem sua utilização como parte de um sistema computacional e resultam de atributos internos do produto. Dentre as características pode-se citar a tolerância a falhas, tempo de resposta, adaptabilidade e operacionalidade do sistema proposto.

Desta forma, esta norma pode ser considerada de extrema importância e utilidade quando implantada de forma apropriada ao processo de desenvolvimento de software pelo fato de poder estar associada aos riscos de projetos e testes de software garantindo uma maior performance e grau de usabilidade perante o usuário final (CAMPOS, 2007).

5.2.1.4 ISO/IEC 12207

A norma ISO/IEC 12207, aprovada no ano de 1995, recebeu a denominação de Tecnologia da informação – Processos do ciclo de vida do software. É constituída de uma terminologia bem definida, apresentando uma estrutura comum aos processos que envolvem o ciclo de vida do software incluindo desde as fases iniciais referentes a sua concepção até o período onde o software é retirado do mercado (WEBER; NASCIMENTO; MARINHO, 1999).

Gusmão e Moura (2004) complementam que a ISO/IEC 12207 consiste na norma internacional na qual descreve os processos, atividades e tarefas realizadas em todas as fases de desenvolvimento, manutenção e fornecimento do produto de software, com a finalidade de ser utilizada como referência aos demais padrões voltados a qualidade de software.

Para (Jomori; Volpe; Zabeu, 2004) a norma ISO/IEC 12207 consiste em um processo para aquisição e fornecimento de produtos e serviços de software do qual pode ser utilizado para a definição, controle ou ainda melhorias referentes aos processos de vida de software onde suas atividades são constituídas de processos fundamentais, de apoio e organizacionais:

a) **processos fundamentais:** constituem os processos de aquisição, fornecimento, manutenção e operação dos quais dão início aos processos de desenvolvimento ou manutenção do software durante o seu ciclo de vida;

b) **processos de apoio:** incluem os processos de documentação, gerência de configuração, garantia de qualidade e auditoria;

c) **processos organizacionais:** formados pelos processos de gerência, infra-estrutura e treinamento envolvendo os processos de ciclo de vida do software com o objetivo de proporcionar melhoria contínua a estrutura do processo.

Segundo Gusmão e Moura (2004) a norma ISO/IEC 12207 relata como podem ser usadas diferentes formas de processos por organizações distintas por meio de um detalhamento dos processos citados acima. Desta forma, são apresentadas diversas formas de utilização para estes processos, dependendo do ponto de vista da organização, agrupadas de acordo com seus objetivos e necessidades.

5.2.2 Padrões CMM, CMMI e PMBOK

O *Capability Maturity Model for Software* (CMM) ou Modelo de Maturidade da Capacitação para Software, foi desenvolvido pelo *Software Engineering Institute* (SEI - Instituto de Engenharia de Software) criado em *Pittsburgh, Pennsylvania* pela *Carnegie Mellon University* (CMU), nos Estados Unidos, apoiado pelo governo do mencionado país (REZENDE, 2005).

Segundo Couto (2007) o CMM trata-se de um modelo de gestão de qualidade de software no qual descreve os elementos chaves para um processo eficiente

e disciplinado de desenvolvimento de software. Seu objetivo é proporcionar a qualidade do produto por meio da procura da melhoria das atividades elaboradas pela organização.

O CMM pode ser dividido em níveis de maturidade que apresentam prioridades claras das quais definem um conjunto de áreas com o objetivo de proporcionar um caminho de melhoria para a organização, com sua representação baseada em estágios. A seguir são apresentadas as definições de tais níveis segundo Couto (2007):

a) **inicial:** caracteriza-se por processos geralmente caóticos, limitados e com ausência de um ambiente estável e controle de requisitos para a organização;

b) **repetível:** planejamento e gerenciamento de processos do projeto com controle de prazos, custos e funcionalidade bem fundamentados dos quais possibilitam a repetição de práticas eficazes utilizadas em projetos anteriores.

c) **definido:** apresentam processos bem definidos e qualitativamente previsíveis com atividades documentadas e visíveis ao conhecimento de toda a equipe envolvida no nível da organização, no projeto de sistemas.

d) **gerenciado:** medições são realizadas para o controle dos processos por meio de técnicas estatísticas e quantitativas. As variações especiais resultantes de medições podem ser tratadas para prevenir futuras ocorrências.

e) **otimizado:** caracteriza-se pela obtenção de melhorias contínuas de processos baseadas no entendimento de causas comuns de variação inerente aos processos. Desta forma, podem-se prevenir defeitos,

gerenciar mudanças tecnológicas e de processos disseminando-os para toda a organização.

O *Capability Maturity Model Integration* (CMMI) por sua vez, foi criado pelo Instituto de Engenharia de Software (SEI), com o intuito de agregar vários modelos de processos de software. O desenvolvimento do CMMI veio a substituir os CMMs voltados a engenharia de sistemas e de Software, com a integração de outros modelos de engenharia (SOMMERVILLE, 2007).

Couto (2007) complementa que o CMMI é considerado um conjunto de modelos dos quais podem ser utilizados de forma simultânea de acordo com aspectos voltados a estrutura e domínio da aplicação. Seu objetivo é definir uma linha para aprimoramento do processo da organização e sua capacidade de adquirir, gerenciar e manter seus produtos ou serviços, propondo-se a solucionar problemas decorrentes do uso de múltiplos modelos em uma empresa.

Segundo Couto (2007) e Sommerville (2007) o CMMI pode ser abordado por duas formas distintas: por estágio, compatível com o CMM para software na qual a capacitação dos processos de desenvolvimento e gerenciamento são avaliados em cinco níveis de maturidade descritos acima, onde cada nível possui um conjunto de áreas de processos associadas. A abordagem contínua por sua vez, é caracterizada por permitir uma classificação mais detalhada dividida em áreas de processos classificadas em uma escala de 1 a 6 organizadas em quatro categorias conforme a Tabela 1:

Tabela 1. Áreas de processo CMMI

Categoria	Área de processo
Gerenciamento de processo	Definição de processo organizacional Foco no processo organizacional Treinamento organizacional Desempenho de processo organizacional Inovação e implantação organizacional
Gerenciamento de projeto	Planejamento de projeto Monitoração e controle de projeto Gerenciamento de acordo com fornecedores Gerenciamento de projeto integrado Gerenciamento de riscos Integração de equipes Gerenciamento quantitativo de projeto
Engenharia	Gerenciamento de requisitos Desenvolvimento de requisitos Solução técnica Integração de produto Verificação Validação
Apoio	Gerenciamento de configuração Gerenciamento de qualidade de processo de produto Medição e análise Análise de decisão e resolução Ambiente organizacional para integração Análise causal e resolução

Fonte: SOMMERVILLE, I. (2007).

Quando comparadas, a representação por estágio prevê uma seqüência de melhorias por um caminho evolutivo iniciando com práticas básicas de gerência e progredindo por um caminho pré-definido por meio dos níveis de maturidade dos quais devem ser seguidos de forma sucessiva. Em contra partida, a representação contínua apresenta um caminho flexível de implementação de melhorias com liberdade de seleção da seqüência que melhor se adapta aos objetivos da organização (COUTO, 2007).

"O CMMI assim como o CMM não determina as atividades pertinentes a cada área de processo, não sugere possíveis indicadores e artefatos para a realização

das mesmas e tem seu foco nas questões de gerenciamento de software." (COUTO, 2007, p. 95). Desta forma, o modelo necessita de um estudo detalhado para atender os objetivos da organização.

O *Project Management Institute* (PMI) é considerado uma instituição sem fins lucrativos de referência mundial em gerenciamento de projetos. Foi fundada em 1969 na Pensilvânia, Estados Unidos, sendo representado no Brasil por seções regionais formados por voluntários, com o objetivo de reunir as melhores práticas de gerenciamento e promover o profissionalismo e ética em gestão de projetos (GERVAZONI, 2006).

Segundo Bressano (2006) o PMI é responsável pela publicação de um documento do qual serve de guia padrão para a divulgação de boas práticas de gerenciamento de projetos denominado *Guide to the Project Management Body of Knowledge* (PMBOK) no qual está reunida a experiência de profissionais da área de Engenharia de Software.

O PMBOK, publicado pela primeira vez em 1987, é responsável por recomendar os processos a serem executados durante o gerenciamento do projeto em áreas específicas de controle, sugerindo um conjunto de processos para integração dessas áreas servindo como fonte de referência para organizações que buscam a melhoria de seus processos de gerenciamento (GERVAZONI, 2006).

Seguindo este contexto, Bressano (2006) afirma que o PMBOK 2004 é constituído de 44 processos distribuídos em áreas de conhecimento estruturadas classificadas em: escopo, tempo, recursos humanos, qualidade, integração, comunicação, custos, riscos e aquisição. Além disso, os processos também se encontram distribuídos em cinco grupos de processos descritos abaixo (VASCONCELOS, 2004):

a) **iniciação:** define o domínio do projeto com análise de requisitos e recursos disponíveis e libera o projeto ou apenas uma fase do mesmo;

b) **planejamento:** refina os objetivos e realiza um plano da ação necessário para alcance dos objetivos propostos do início do projeto;

c) **monitoramento e controle:** controle e monitoramento constante das atividades do desenvolvimento do projeto, corrigindo desvios dos objetivos esperados;

d) **execução:** executa as atividades referentes ao plano de gerenciamento do projeto com a integração de pessoas e recursos;

e) encerramento: formaliza a entrega e aceitação do produto perante o cliente/usuário.

O guia PMBOK traduz conceitos atuais importantes da prática de gerência de projetos, por meio de uma linguagem de fácil entendimento perante os usuários, servindo de referência básica para os gerentes de projetos que venham a usufruir das áreas de conhecimento disponibilizadas pela metodologia (PMI, 2006).

6 CONTROLE DE ETAPAS DE DEENVOLVIMENTO COM FERRAMENTAS DE AUXÍLIO AO GERENCIAMENTO DO SOFTWARE

Segundo Sotille (2006) a gerência de projetos é caracterizada pela seqüência de decisões das quais envolvem aspectos referentes ao tempo, custo e qualidade, baseados na aplicação de conhecimento, ferramentas e técnicas presentes no desenvolvimento das atividades do projeto de maneira que satisfaçam as exigências e expectativas do cliente/usuário.

Neste contexto, Sommerville (2003) afirma que o gerenciamento de um projeto deve ser iniciado com a elaboração de um planejamento de qualidade, definido por um plano estratégico onde estão estabelecidas as qualidades desejadas ao projeto. Dentre as características a serem observadas, destacam-se a seleção de padrões, métodos e ferramentas a serem utilizadas no desenvolvimento, sujeitos a alterações no decorrer do desenvolvimento.

A elaboração do plano de qualidade resulta na definição da arquitetura do software, fundamental para a organização no desenvolvimento do projeto como um todo, pelo fato de constituir um alicerce pelo qual o mesmo será construído (SOMMERVILLE, 2003).

A arquitetura de software para Garlan e Shaw (1996 apud FACUNTE, 2006) corresponde a um nível de *design* do qual não se limita a algoritmos e estrutura de dados da computação, definindo o conceito de sistema em termos de componentes computacionais e o relacionamento entre esses componentes.

Facunte (2006) afirma que a arquitetura de software proporciona uma visão ampla dos processos e tecnologias envolvidas dos elementos estáticos e dinâmicos, e o modo dos quais estes elementos interagem entre si na organização levando em

consideração as questões referentes ao desempenho, prevenção a mudanças, reutilização de componentes e restrições econômicas ou tecnológicas.

Segundo Facunte (2006) a arquitetura de software tornou-se popular como uma disciplina de engenharia para o desenvolvimento de software a partir de 1996, com a publicação do livro "*Software Architecture*" dos autores Mary Shaw e David Garlan devido à necessidade da análise de várias visões e abstrações de requisitos não documentadas até o presente momento.

Após a definição do plano de qualidade, o gerenciamento de projetos deve seguir um controle de qualidade onde é realizada a monitoração do processo de desenvolvimento do software com o objetivo de garantir a utilização de padrões e procedimentos de garantia de qualidade definidos no início da elaboração do projeto (SOMMERVILLE, 2003).

Ainda Sommerville (2007) afirma que a avaliação automatizada do controle de qualidade por meio do apoio de ferramentas de auxílio, deve ser realizada quando as mesmas estiverem disponíveis, evitando a ocorrência de um esforço adicional causado pelo longo trabalho envolvido na implementação de processos provindos de padrões estabelecidos durante o desenvolvimento do projeto.

O uso de ferramentas beneficia principalmente os processos de software dos quais utilizam algum método de desenvolvimento definido, baseados em uma melhor integração dos requisitos e do projeto. Atualmente, os processos considerados metódicos são apoiados de modo usual por ferramentas *Cases* de análise e projeto. No entanto, os processos podem desfrutar de outros tipos de ferramentas como, por exemplo, ferramentas de testes dentre outras (SOMMERVILLE, 2003).

Dentro deste contexto de busca de técnicas automatizadas de gerenciamento de qualidade de projeto, o presente capítulo tem por objetivo a apresentação de técnicas

e ferramentas referentes ao controle de qualidade de software nas fases de projeto, implementação e testes, com o intuito da obtenção de resultados satisfatórios no que diz respeito ao produto de software final, para a equipe de gerência do projeto e clientes/usuários.

6.1 TÉCNICAS AUTOMATIZADAS PARA CONTROLE DE ATIVIDADES NA FASE DE PROJETO DE SOFTWARE

Conforme visto no capítulo 2.3, uma atividade corresponde a tarefas das quais são realizadas no decorrer do desenvolvimento do software, organizadas de acordo com a complexidade do projeto (BEZERRA, 2002).

Seguindo essa concepção Câmara (2006) afirma que as atividades envolvidas englobam requisitos funcionais e não funcionais como, por exemplo, a definição da linguagem de programação e banco de dados a ser adotado na organização, além de tarefas administrativas, como configuração do repositório do código fonte dentre outros aspectos dos quais necessitam ser gerenciados.

Neste sentido, o uso de ferramentas das quais gerenciam as atividades a serem desenvolvidas pela equipe de desenvolvimento tornaram-se um fator determinante no decorrer do ciclo de vida do software para o controle das tarefas servindo como meio de comunicação eficiente, do qual integra todos os envolvidos no projeto (CÂMARA, 2006).

O objetivo da utilização de ferramentas de controle de atividades é o de possibilitar o acompanhamento, de forma eficiente, das tarefas a serem executadas por cada membro da equipe de desenvolvimento, permitindo a análise do esforço de trabalho de toda a equipe envolvida no desenvolvimento do software proposto (GARCIA, 2007).

A utilização de ferramentas para o controle de atividades deve ser basicamente dividida em dois grupos distintos de usuários: os gerentes do projeto, dos quais são responsáveis pela definição das atividades e os arquitetos, analistas, desenvolvedores e analistas de testes dos quais são responsáveis pela execução das mesmas (CÂMARA, 2006).

Conforme descrito por Câmara (2006) as tarefas definidas pelos gerentes de projeto geralmente possuem o mesmo ciclo de vida, dividido em status, dos quais se podem destacar o status de proposta, ativa, resolvida e finalizada, conforme descritas abaixo:

a) **proposta:** quando o gerente de projetos define uma tarefa pela primeira vez e a encaminha para um membro da equipe;

b) **ativa:** ocorre quando o responsável pela realização da tarefa informa ao projeto que iniciou a execução da mesma;

c) **resolvida:** ocorre quando o membro da equipe responsável pela atividade informa que finalizou a tarefa;

d) **finalizada:** ocorre quando o gerente do projeto analisa o resultado obtido da resolução da tarefa e verifica que o mesmo atendeu as especificações impostas e finaliza o ciclo de vida.

Neste contexto, destacam-se as ferramentas de controle de atividades desenvolvidas das quais permitem associar as tarefas definidas pelos gerentes de projeto ao conjunto de código fonte referente a estas tarefas, justificando a alteração do mesmo de acordo com a necessidade imposta na descrição da atividade como o *Enterprise Architect* e o *Visual Studio Team System*.

6.1.1 Controle de Atividade na ferramenta *Enterprise Architect*

O *Enterprise Architect* é um produto fabricado pela *Sparx Systems*, e distribuído no Brasil pela OAT *Solutions*, uma empresa de consultoria e treinamento em Tecnologia da Informação, na qual atua na definição de fornecimento de ferramentas de auxílio aos processos de desenvolvimento e manutenção de software e gerência de projetos (OAT Solutins, 2007).

O *Enterprise Architect* é uma ferramenta de gerência de projetos utilizada em todo o ciclo de desenvolvimento do software, não somente para o controle de atividades do projeto, como também para manutenção, realização de testes e controle de mudanças de requisitos. Possui como característica o fácil manuseio para produção de documentação com resultados eficientes possibilitando a criação de diagramas UML citados no capítulo 3.4 deste trabalho e modelagem de sites da web (LIMA, 2007).

Neste sentido, o objetivo da ferramenta é modelar o processo de desenvolvimento do software, por meio da descrição das atividades envolvidas no projeto da empresa com etapas bem definidas e documentá-las juntamente com aspectos referentes aos resultados obtidos da implementação da tarefa em questão, como esforços realizados e seu tempo de execução (LIMA, 2007).

Neste contexto, a OAT *Solutions* (2007) afirma que o *Enterprise Architect* disponibiliza uma serie de recursos de gerenciamento referente a modelagem de processos e sistemas na notação UML, para a equipe de desenvolvimento, destacados abaixo:

a) proporciona a utilização de técnicas de levantamento de requisitos referente a usuários;

b) permite a automatização na engenharia de código contemplando múltiplas linguagens de programação;

c) pode ser integrado a ferramentas de gerenciamento de versão de código fonte como o *Subversion*;

d) garante o uso compartilhado e seguro pelos membros da equipe de desenvolvimento.

Neste contexto, Lima (2007) concorda com a OAT *Solutions* (2007) e complementa que o EA possibilita o desenvolvimento de diagramas utilizando os conceitos da UML 2.0, e a geração e engenharia reversa de classes escritas em diferentes linguagens de programação como, por exemplo, VB.net, Java, Delphi e PHP, em diferentes plataformas, exigindo um esforço mínimo de hardware do qual sofre variação de acordo com o sistema operacional adotado.

6.1.2 Controle de Atividades na ferramenta *Visual Studio Team System*

O *Microsoft Visual Studio Team System* (VSTS) é uma ferramenta desenvolvida pela Microsoft para gerenciar e automatizar o desenvolvimento de software. Sua utilização envolve todo o ciclo de vida de desenvolvimento de um software (SEMENIUK, 2007).

"Um work item pode ser considerado como uma atividade em seu projeto. Work items são tarefas de trabalho que servem como base ao gerenciamento e rastreabilidade dos projetos dentro da estrutura do Visual Studio Team System" (GARCIA, 2007 p. 55).

Os *work items* foram desenvolvidos com o objetivo de atender todas as funções voltadas aos processos baseados no *Microsoft Solution Framework*,

possibilitando a implementação conforme critérios estabelecidos para cada processo de desenvolvimento proposto (GARCIA, 2007).

Os *work items* são divididos em tipos (WIT) específicos de acordo com a metodologia utilizada no desenvolvimento. Os WIT disponíveis no *template MSF for CMMI Process Improvement*, fornecido pela ferramenta, serão exemplificados a seguir (CÂMARA, 2006):

a) ***requirement:*** consiste na solicitação de uma necessidade do sistema. Pode ser considerado um caso de uso ou ainda um requisito funcional que precisa ser implementado.

b) ***task:*** é considerada uma atividade propriamente dita e deve sempre estar relacionada a um produto de trabalho.

c) ***bug:*** serve para reportar defeitos ou incidentes dos quais necessitam ser resolvidos no projeto em desenvolvimento.

d) ***risk:*** definidos como uma expectativa de perda da qual pode ocorrer em determinado momento do sistema e precisa de um plano de correção.

e) ***change request:*** é um pedido de alteração formal no qual irá modificar o funcionamento alguma funcionalidade já finalizada.

f) ***review:*** consiste no pedido formal da revisão de um produto de trabalho por algum membro da equipe de desenvolvimento.

g) ***issue:*** relatar assuntos dos quais não se sabe onde classificar. Depois de esclarecidos podem se tornar *bugs* ou requisições.

Os *work items* fazem parte de uma ação determinada no projeto, constituindo parte de um processo de *workflow* de uma organização, sendo desta forma, considerado como uma unidade fundamental para o controle do desenvolvimento das

quais se podem relacionar ou não entre si, ou ainda servir de ponte para outras ações específicas (GARCIA, 2007).

Câmara (2006) concorda com Garcia (2007) e relata que os *work items* são registros em banco de dados sem um grau de hierarquia padrão definido. Entretanto, podem-se definir hierarquias virtuais das quais permitem melhor percepção das atividades de trabalho, dividindo os processos em áreas, caracterizadas como forma de organizar os grupos, e iterações, como sendo a versão de seus códigos fontes.

6.2 TÉCNICAS AUTOMATIZADAS PARA CONTROLE DE VERSÃO DE CÓDIGO FONTE NA FASE DE IMPLEMENTAÇAO DE SOFTWARE

No contexto de engenharia de software "uma versão do sistema é caracterizada como uma instância de um sistema que difere de alguma maneira, de outras instâncias". As versões de sistema podem apresentar características distintas no quesito funcionalidade, ou ainda serem semelhantes, porém projetadas para serem utilizadas em configurações diferentes de hardware ou software (SOMMERVILLE, 2007, p. 462).

O gerenciamento de versões e o controle de acesso ao código fonte constituem a *Software Configuration Management* (SCM), práticas das quais representam um processo fundamental para desenvolvimento do ciclo de vida do software. A SCM deve ser abordada independentemente da metodologia a ser adotada pela equipe de desenvolvimento, diferenciando-se apenas no grau de profundidade de sua utilização (CÂMARA; LEITE, 2006).

O gerenciamento de versões envolve o planejamento de processos com a finalidade de facilitarem a identificação e rastreabilidade das versões de um

determinado software. Tais procedimentos são responsáveis por permitirem a reconstituição de uma versão específica do sistema da qual foi alterada indevidamente pelos desenvolvedores do sistema (SOMMERVILLE, 2007).

Seguindo este conceito, Câmara e Igor (2006) afirmam que os sistemas de controle de versões somente possuem funcionalidade quando possibilitam a consulta e restauração de versões anteriores de arquivos, comparação de diferentes versões e demonstração das alterações sofridas entre os arquivos associadas a suas respectivas datas e autores.

Ainda Sommerville (2007) complementa que para a criação de uma versão específica do sistema se faz necessário a identificação dos componentes de software a serem inclusos na nova versão, utilizando uma forma única de identificação, pois cada componente pode apresentar diversas versões diferentes, destacando três técnicas básicas: numeração de versão, identificação baseada em atributos e a identificação orientada a objetos.

a) **numeração de versões:** constitui uma forma simples, mais comumente utilizada para a identificação do componente, onde o mesmo recebe um numero único determinado de versão;
b) **identificação baseada em atributos:** cada componente é especificado por um nome e grupo e atributos associados a cada versão dos quais permitem sua identificação;
c) **identificação orientada a objetos:** assim como na identificação anterior, cada componente é especificado por um nome e grupo e atributos, porém associados a solicitações de mudanças aplicadas a um componente.

O uso de ferramentas é um artifício importante, geralmente utilizado, para o auxílio no gerenciamento de versões. Embora as ferramentas disponíveis no mercado atual diferenciem-se por possuírem características próprias na forma de gerenciamento e *interfaces* com os usuários, todas se baseiam no princípio geral de gerenciamento, armazenamento e controle do acesso aos componentes de sistema. Para cada nova versão criada, um identificador único é definido pelo sistema de gerenciamento por meio do *check-in* do componente (SOMMERVILLE, 2007).

Dentre as ferramentas de gerenciamento de versões pode-se destacar o uso do *Subversion* (SVN), uma ferramenta *open source* e o *Team Foundation Version Control* (TFVC), uma ferramenta desenvolvida pela Microsoft integrada a todo o ciclo de desenvolvimento do sistema.

6.2.1 Subversion (SVN)

O *Subversion* é uma ferramenta de controle de versão *open source*, com projeto de desenvolvimento iniciado no ano de 2000, com o objetivo de suprir limitações e corrigir falhas do CVS, um controle de versão base para a construção do SVN. Seu código é aberto e sua funcionalidade não se limita apenas ao controle de versão de arquivos, como também a consulta de versões e segurança na manipulação de códigos fontes de arquivos (DIAS, 2006).

O *Subversion* possui as características de um sistema cliente/servidor, onde o servidor corresponde a parte centralizada da aplicação, responsável pelo armazenamento dos dados, e o cliente por sua vez, máquinas das quais acessam os dados armazenados. Sua estrutura permite o número de conexão ilimitado de clientes ao repositório, permitindo a leitura e escrita de arquivos, gravando as informações

referentes a cada alteração realizada (COLLINS-SUSSMAN; FITZPATRICK; PILATO, 2002).

Esta ferramenta utiliza um modelo denominado *copy-modify-merge* (copiar-modificar-combinar) para a realização do controle de versão. Este modelo permite que os usuários acessem o repositório e utilizem arquivos do projeto simultaneamente, criando cópias dos arquivos e diretórios existentes para sua maquina local. Depois de modificados, os arquivos locais são combinados *(merge)* e uma nova versão final é gerada (COLLINS-SUSSMAN; FITZPATRICK; PILATO, 2002).

Seguindo este contexto, Dias (2006) complementa que o protocolo de rede utilizado para o *Subversion* faz uso de uma largura de banda eficiente enviando diferenças entre cliente/servidor e vice-versa sempre que possível, trabalhando com operações atômicas de *commit* das quais impossibilitam a efetivação de apenas uma parte das alterações realizadas no código fonte quando enviadas ao repositório.

Pode-se concluir que o modelo *copy-modify-merge* possibilita o trabalho em paralelo no desenvolvimento, editando os arquivos localmente sem a intervenção de outro membro da equipe até o momento da efetivação das alterações. Depois de finalizadas as alterações o *Subversion* permite que o arquivo seja publicado no repositório, ocorrendo a combinação das alterações quando efetivadas por outros usuários, evitando conflitos freqüentes desnecessários.

6.2.2 Team Foundation Version Control

O *Team Foundation Version Control* (TFVC) é uma ferramenta, desenvolvida pela Microsoft, responsável pelo controle de versões de um sistema. O controle de código fonte é totalmente integrado ao ciclo de desenvolvimento do

software, armazenado no banco de dados SQL Server, possibilitando a utilização de regras eficazes para efetivação da alteração do arquivo proposto (DURÃES, 2007).

Os métodos utilizados pela ferramenta para o controle de versão são semelhantes aos disponíveis no Subversion, como o modelo copy-modify-merge, no qual permite os usuários trabalharem de forma simultânea com o mesmo arquivo evitando o conflito de informações alteradas. O *Team Foundation Server* (TFS), parte servidor da ferramenta, também trabalha com *commit* atômico mantendo a integridade de todas as alterações realizadas (GARCIA, 2007).

O *Team Foundation Version Control* possui a vantagem de ser integrado ao desenvolvimento de todo o ciclo de vida do software obrigando, por exemplo, um membro da equipe a informar um *work item*, tarefa criada pelo gerente de projetos, que solicitou o desenvolvimento/manutenção de um determinado código fonte, caso contrário não efetivando a atualização do mesmo no repositório (CÂMARA; LEITE, 2006).

O *Team Foundation Server* elaborou um protocolo *web services* (HTTP) exclusivo para todas as comunicações efetuadas entre cliente e servidor, possibilitando um fácil acesso de um membro da equipe com o repositório no qual estão armazenadas as informações referentes ao projeto. Alem desses recursos, destacam-se certas funcionalidades da ferramenta essenciais para a obtenção do controle do ciclo de desenvolvimento de um software, conforme descritas abaixo (CÂMARA; LEITE, 2006):

a) ***check-out:*** constitui a operação na qual ocorre a indicação de que um arquivo localizado no repositório será alterado;

b) ***check-in:*** utilizadas para indicar a finalização de uma alteração de um arquivo previamente obtido por um *check-out*. Neste caso, as modificações são disponibilizadas no servidor em uma nova versão;

c) ***changesets:*** representam todas as modificações realizadas em um determinado arquivo por uma única opção de *check-in*;

d) ***check-in policy:*** corresponde ao estabelecimento de regras das quais devem ser respeitadas para que a efetivação das alterações no repositório;

e) ***branching:*** é um recurso do qual permite o uso de arquivos por diferentes usuários em paralelo sem que interfiram no trabalho uns dos outros;

f) ***merging:*** é responsável pela combinação das alterações realizadas em paralelo por diferentes membros da equipe em um mesmo arquivo.

Pode-se concluir que a ferramenta *Team Foundation Version Control* possui características comuns a outras ferramentas de controle de código fonte, mas destaca-se por abranger todo o ciclo de vida de desenvolvimento do software com recursos adicionais como a política da utilização de regras para efetivar as alterações realizadas em um determinado arquivo proporcionando um controle mais efetivo das versões do sistema proposto.

6.3 TÉCNICAS AUTOMATIZADAS PARA TESTES DE SOFTWARES

O teste de software consiste em uma área da Engenharia de Software de extrema relevância, cujo objetivo é prover evidencias de confiabilidade e qualidade de

software na detecção de falhas encontradas durante o processo de desenvolvimento, precavendo custos financeiros desnecessários. (INTHURN, 2001).

Neste sentido, Caetano (2006) concorda com Inthurn (2001) e afirma que o teste de software consiste em uma atividade de fundamental importância a ser realizada no ciclo de vida de desenvolvimento de um software. Os testes permitem a implementação de um software capaz de satisfazer as necessidades dos clientes, apto a rodar nas plataformas específicas no projeto e executar suas funcionalidades fundamentais independentemente da tecnologia utilizada no desenvolvimento.

No contexto de testes de software, destacam-se os responsáveis pela realização dos mesmos, denominados testadores que possuem a tarefa de auxiliar os usuários e o terminal de informática a convergir sobre os pontos de vista no que diz respeito à qualidade, no sentido de analisar se o software satisfaz os requerimentos especificados (CAETANO, 2006).

Neste sentido, Sanches (2006) afirma que a função do testador durante o ciclo de desenvolvimento do software é planejar casos de testes, definindo seu escopo, identificação das áreas e as funcionalidades a serem validadas, executar os testes e ainda analisar os resultados obtidos com o objetivo de forçar determinadas funções do sistema das quais podem gerar um problema. Caso identifique algo de anormal, o testador ainda deve relatar o mesmo a equipe responsável pelo desenvolvimento para que o problema seja corrigido.

Durante a etapa de desenvolvimento do software, os testadores devem escrever e configurar os testes sempre focados na qualidade da aplicação para que possam contribuir de forma eficaz no resultado do produto final. O auxílio de ferramentas será útil para o desenvolvimento do projeto, desde que se adéqüem a

realidade do software e aos tipos de testes dos quais devem ser realizados em determinadas funções do sistema (SANCHES, 2006).

Seguindo essa concepção, Wazlawick (2004) relata dentre os diversos tipos de testes existentes, os testes de unidade, cujo objetivo é analisar se o resultado obtido dos componentes do sistema disponível atende de forma eficaz a especificação do projetista e aos testes de caso de uso, geralmente efetivados pelos analistas de sistemas experientes, dos quais visam se o sistema proposto atende aos requisitos levantados anteriormente na fase inicial do projeto.

Os testes unitários realizados para identificação de problemas referentes ao desenvolvimento de software não é uma técnica recente no mercado. Seu conceito pode ser definido como sendo códigos dos quais testam códigos, ou seja, são partes de códigos gerados para a aplicação de testes de determinadas partes do sistema, como os componentes específicos do processo e persistência dos dados obtidos de acordo com requisitos inicialmente levantados (SANCHES, 2006).

Dentre os recursos disponíveis na realização de testes unitários se encontra a possibilidade da criação de cenários de testes e simulação de situações das quais a aplicação pode apresentar defeitos que comprometem seu desempenho. Tais procedimentos trazem consigo a garantia de que o método ou função testada está de acordo com as especificações impostas, proporcionando uma segurança adicional ao programador (SANCHES, 2006).

Segundo Inthurn (2001) no desenvolvimento de software orientado a objetos é necessário testar cada unidade, não necessariamente de forma seqüencial, analisando alguns fatores determinantes para a funcionalidade do modulo testado dos quais se destacam:

a) ***interface* com o módulo:** verificar a consistência das informações das quais entram e saem;

b) **estrutura de dados local:** analisar se os dados armazenados temporariamente permanecem íntegros durante todos os passos de execução dos códigos;

c) **caminhos básicos:** a execução de todas as instruções deve ser executada ao menos uma vez, a fim de verificar se os resultados obtidos estão de acordo com o esperado;

d) **caminhos de tratamento de erros:** os caminhos para execução dos erros devem ser executados com o intuito de validar os valores não verdadeiros.

Sanches (2006) complementa Inthurn (2001) afirmando que os testes unitários podem simular os testes de interfaces realizados durante o desenvolvimento de alguma funcionalidade do sistema, auxiliando e substituindo o processo padrão de execução da *interface* com o usuário para testar pontos específicos do projeto, executando apenas uma classe do qual fará o teste de funcionalidade em questão.

Dentre as ferramentas de teste de software pode-se destacar o uso do *JUNIT*, uma ferramenta da família *XUnit*, e o *Visual Studio Team System for Testers*, uma ferramenta desenvolvida pela Microsoft integrada a todo o ciclo de desenvolvimento do sistema.

6.3.1 XUNIT

O *XUnit* é um mecanismo eficiente de auxílio aos desenvolvedores de software para adição de testes unitários automáticos, estruturados e eficientes nas

atividades de desenvolvimento. Este *framework* foi desenvolvido em 1988, e desde então evoluiu na prática para se transformar em um padrão para *frameworks* automáticos de testes unitários (CARDINAL, 2007).

Segundo Paula Neto (2006) o *XUnit* é um nome genérico do qual representa qualquer estrutura de testes automáticos unitários de fundamental importância a engenharia de software pelo fato de ser uma ferramenta da qual se podem efetuar testes de maior qualidade quando comparados aos convencionais pelo fato de expor mais facilmente os erros encontrados durante a análise dos testes.

6.3.1.1 JUNIT

O *JUNIT* é uma ferramenta *open source*, na qual pertence à família do *XUnit*, desenvolvida por Kent Beck e Erich Gamma, com o objetivo de proporcionar ao seu utilizador um método sólido e confiável na detecção de problemas encontrados no sistema dos quais se propõem a resolver, por meio da criação de testes automatizados na linguagem de programação Java (PAULA NETO, 2006).

O *JUNIT* é um *framework* desenvolvido para facilitar a criação rápida de códigos para automação de testes unitários com a apresentação de resultados bem definidos dos quais possibilitam um grau eficiente de controle da qualidade do software, por meio da validação de todos os métodos das classes existentes, destacando as possíveis falhas ou erros (PAULA NETO, 2006).

Neste sentido, Oliveira (2005) concorda com Paula Neto (2006) e complementa que os testes unitários criados por meio da ferramenta *JUNIT,* contém classes formadas por um ou mais métodos dos quais são realizados os testes, organizadas de forma hierárquica, separadas, integradas ou ainda de forma que o

sistema seja testado em partes separadas, sendo reutilizáveis, mantendo seu valor ao longo do desenvolvimento do projeto.

Segundo Paula Neto (2006), o *JUNIT* é uma ferramenta *open source*, orientada a objetos, amplamente utilizada pelos desenvolvedores da comunidade de código aberto, cujo principal característica é a execução e análise dos testes realizados de forma rápida e eficiente pela ferramenta, sem a necessidade da interrupção do processo de desenvolvimento proposto, evitando atrasos inesperados nos prazos estabelecidos.

6.3.1.2 VISUAL STUDIO TEAM SYSTEM FOR TESTERS

O *Visual Studio Team System 2005 for testers* é uma ferramenta de testes desenvolvida pela Microsoft na qual integra os testes aos códigos do projeto, possibilitando a aplicação de testes íntegros e com garantia de que as mudanças realizadas não interferem a qualidade do código desenvolvido (SANCHES, 2006).

Segundo Brito (2008) o VSTS para testes de software disponibiliza um conjunto eficiente de ferramentas de testes integradas em sua totalidade ao ambiente de desenvolvimento, cujo objetivo é minimizar os riscos a falhas e defeitos dos quais podem causar problemas referentes a custos desnecessários e melhorar significativamente a qualidade do software depois de disponibilizado para utilização no ambiente de produção do cliente.

O VSTS para *testers* possui o recurso de utilizar, na criação de testes unitários em projetos, uma fonte de dados dinâmica para a realização de determinados testes, denominado *Data-driven*. Essa característica é determinante para não somente garantir uma cobertura de testes de maior amplitude como impedir que os mesmos se

tornem viciados, ou seja, “pode ser que nunca seja testada certa massa de dados que, quando executados em produção, podem causar algum dano ao projeto e, conseqüentemente, um desconforto por parte do cliente” (SANCHES, 2006, p. 73).

Ainda Sanches (2006) afirma que o VSTS para *testers* garante a qualidade contínua da aplicação por possibilitar o desenvolvimento de cenários de navegação dos quais facilitam o processo de testes em etapas das quais necessitam este tipo de aplicação, pelo fato de permitir a criação de um caminho de execução, gravação deste caminho e execução do numero de vezes necessárias para a realização dos testes da aplicação. O objetivo principal da criação destes cenários é o ganho de tempo na execução dos testes sem a necessidade de abrir o *browser,* navegar pela URL da aplicação e testar a funcionalidade desenvolvida.

7 TRABALHOS CORRELATOS

Os trabalhos descritos a seguir, utilizam técnicas existentes na Engenharia de Software, das quais devem estar presentes em todo o ciclo de vida do projeto para o desenvolvimento de sistemas de qualidade, funcionais e menos propícios a falhas que resultam em custos não previstos no decorrer do projeto.

Foram encontrados trabalhos diversos dos quais relatam diferentes técnicas de gerenciamento de projeto, onde suas utilidades diferem-se de acordo com as características do software proposto, como por exemplo, sua estrutura e grau de necessidade de documentação.

Neste sentido, optou-se para o desenvolvimento desta pesquisa a seleção de trabalhos dos quais trazem consigo as técnicas referentes a modelos, processos, normas e padrões de maior conceito perante a Engenharia de Software, com o auxílio de ferramentas de controle, utilizadas em todas as fases de desenvolvimento do software.

Portanto, para a apresentação dos trabalhos correlatos foi adotada uma estrutura dividida em: trabalhos que apresentam conceitos de modelos, processos, normas e padrões de desenvolvimento de software e trabalhos que apresentam ferramentas de gerenciamento de software.

7.1 TRABALHOS QUE APRESENTAM CONCEITOS DE MODELOS E PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE

Os trabalhos apresentados a seguir relatam técnicas de gerenciamento de software, aplicadas à área de atuação desta pesquisa, com o objetivo de proporcionar

um produto final com qualidade perante aos clientes/usuários. O primeiro se propõe a evidenciar conceitos referentes a metodologias de desenvolvimentos existentes com suas características e a criação do protótipo de uma ferramenta para avaliação das aquisições, enquanto o segundo relata os impactos causados pela aplicação da metodologia XP nas organizações das quais desenvolvem softwares.

7.1.1 Estudo e prototipação de ferramenta para gerência de projetos, enfocando a gerência de aquisições

Este trabalho de conclusão de curso de Tecnologia em Informática foi desenvolvido no ano de 2003, na Universidade Luterana do Brasil, em Canoas, Rio Grande do Sul (GOMES, 2003).

O objetivo proposto neste trabalho foi a realização do estudo de técnicas de gerência de projetos baseadas nos conceitos das metodologias como o PMBOK, CMM, ISO 12207/1997 e a implementação de um protótipo cuja funcionalidade é o auxílio na avaliação das aquisições necessárias pelo gerente de projetos.

Dentre as funcionalidades encontradas no protótipo destacam-se o cadastro do projeto, de fornecedores aptos ou não a proverem o serviço ao projeto, e o cadastro da documentação referente ao processo de cotação e contratação, auxiliando o gerente de projetos a determinar qual o fornecedor é mais adequado a atender as solicitações e descrições exigidas pelo projeto por meio de funções encontradas no banco de dados da aplicação.

O protótipo foi modelado em UML, na linguagem *Visual Basic* pelo fato de possuir sua estrutura orientada a eventos da qual segundo o autor apresenta recursos de fácil manuseio para a criação de *interfaces*.

Neste sentido os resultados obtidos na elaboração deste trabalho foram satisfatórios segundo o autor desta pesquisa, por demonstrarem que se podem obter qualidade e produtividade eficaz quando os padrões de desenvolvimento de software estudados são adotados e automatizados.

7.1.2 Impacto da Aplicação da Metodologia XP nas organizações de desenvolvimento de Software

Esta pesquisa refere-se a um Trabalho de Conclusão do Curso de Sistemas de Informação da Universidade do Vale do Sapucaí, em Pouso Alegre, aplicada no ano de 2007 (BORBOREMA, 2007).

O objetivo deste trabalho é apresentar o impacto do uso da metodologia ágil XP, baseado em estudos dos quais relatam falhas referentes ao não cumprimento de orçamentos, cronogramas e funcionalidades adequadas, no desenvolvimento de softwares, ocasionando a insatisfação do produto final perante o cliente.

Neste contexto, foram analisados conceitos e aplicações voltados ao uso da metodologia XP, seguidos de um estudo de caso realizado com empresas das quais utilizaram a metodologia proposta e demonstraram seus impactos por meio de um questionário com perguntas direcionadas a aplicação da XP.

Segundo os autores desta pesquisa, o objetivo proposto de levantar dados para comprovar o uso e efeitos da utilização da XP pelas empresas foi alcançado concluindo, dentre outros, que o motivo pelo qual as organizações optaram pela metodologia descrita para o desenvolvimento de software foi em primeiro lugar a busca de qualidade, seguida da agilidade, produtividade, confiabilidade e adaptação da equipe ao processo analisado.

7.2 TRABALHOS QUE APRESENTAM A AUTOMATIZAÇÃO DE ETAPAS DE DESENVOLVIMENTO DE SOFTWARE POR MEIO DE FERRAMENTAS

Os trabalhos a seguir apresentam a importância e a utilização de ferramentas no controle de gerenciamento de software com qualidade, um fator relacionado a realização desta pesquisa, de fundamental importância para o gerenciamento de sistemas. O primeiro tem o objetivo de apresentar os diferentes tipos de testes existentes, com a utilização de ferramentas de auxílio no desenvolvimento de software, enquanto o segundo se propõe a demonstrar características de uma ferramenta de controle de versão de um sistema.

7.2.1 Automação de Testes Utilizando Ferramentas *Open Source*

A pesquisa proposta refere-se a uma Monografia apresentada como requisito parcial para conclusão do Bacharelado em Ciência da Computação, realizada na Universidade de Brasília (UnB), no ano 2006 (LOUSA; NUNES, 2006).

O objetivo deste trabalho é a realização de testes automatizados com a utilização de ferramentas de auxílio *open source*, por meio de estudos dos conceitos referentes a qualidade de software presentes na Engenharia de Software, destacando a importância da aplicação de testes no desenvolvimento do projeto.

Com este intuito, foram analisados diferentes tipos de testes e elaborado uma estrutura para aplicação dos mesmos em um software produzido na Universidade de Brasília. Dentre o planejamento, foram especificados quais os testes a serem executados e quais as ferramentas propicias para o determinado tipo de atividade selecionada.

Dentre as ferramentas de testes utilizadas destacam-se a *Checkstyle* para a verificação de aderência ao padrão de codificação, e o *JUnit* para a realização dos testes estruturais de unidade. Para o calculo da porcentagem de código acessado pelos testes estruturais foram utilizadas as ferramentas Cobertura, versão 1.8 e o *plugin* da *IDE Eclipse* denominado *Coverlipse*, versão 0.9.5.2.

Segundo os autores da pesquisa, os resultados obtidos foram satisfatórios por apresentarem os testes de software automatizados como uma etapa fundamental no desenvolvimento de um projeto pelo fato de simplificar e tornar a execução dos testes mais rápidos e eficazes quando comparadas a teste manuais.

7.2.2 Uma Ferramenta de apoio ao desenvolvimento de Software baseado em Componentes

O presente trabalho é resultado de um artigo desenvolvido na Universidade de Santa Cruz do Sul, no Rio Grande do Sul publicado no ano de 2001 no XV Simpósio Brasileiro de Engenharia de Software (KROTH; PFAFFENSELLER, MATHEUS; PFAFFENSELLER, MOISÉS, 2001).

O objetivo deste trabalho é proporcionar o gerenciamento de componentes de sistema e a otimização do processo de construção de software de forma organizada por meio de técnicas confiáveis para o desenvolvimento realizado em grupos de trabalho, com o auxílio de ferramentas das quais possibilitam o reuso do software e facilitem a visualização das funções de cada artefato gerado.

Neste contexto, foi desenvolvida uma ferramenta de apoio ao desenvolvimento de software baseada em componentes, cuja *interface* é mostrada na Figura 6, da qual possibilita a realização de um gerenciamento de versões dos

componentes, implementação de técnicas de armazenamento, e a definição de uma política de manutenção, exercendo o controle ao acesso aos componentes em níveis de permissão com o armazenamento do histórico sobre as alterações efetuadas.

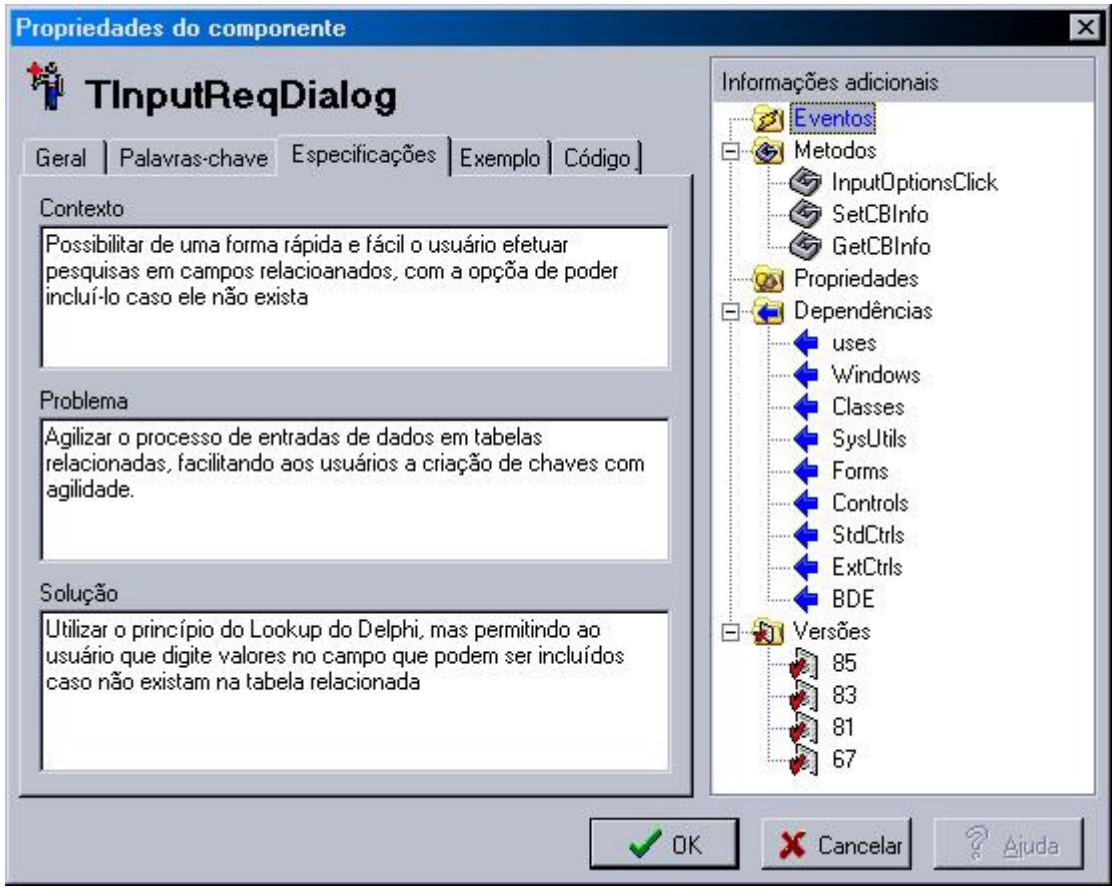

Figura 6. Interface do software (propriedades do componente)
Fonte: KROTH, E; PFAFFENSELLER, Matheus; FAFFENSELLER, Moisés, (2001)

Segundo os autores deste trabalho, os resultados obtidos corresponderam as expectativas por apresentarem soluções para o controle de versões, políticas de manutenção com restrições de acesso divididos em níveis de usuários dentre outras funcionalidades, por meio da criação de uma ferramenta de gerenciamento capaz de suprir deficiências técnicas de desenvolvimento existentes, agilizando o processo de construção de software.

8 TECNOLOGIAS DE GERENCIAMENTO UTILIZADAS NO PROCESSO DE DESENVOLVIMENTO DO CICLO DE VIDA DO SOFTWARE

A presente pesquisa consiste no estudo de conceitos da Engenharia de Software referentes ao ciclo de vida do software. Dentre os fatores analisados encontram-se os modelos, processos, normas e padrões de desenvolvimento de software encontrados na atualidade, com a utilização e aplicação de ferramentas de auxílio nas etapas de gerenciamento de atividades, controle de código fonte e realização de testes unitários visando a qualidade do produto final.

O trabalho se propõe a evidenciar a importância da qualidade no gerenciamento de softwares, fator do qual tem levado as organizações a buscarem técnicas eficientes de desenvolvimento visando a melhoria contínua dos processos por meio das seguintes etapas: levantamento bibliográfico, estudo referentes ao ciclo de vida do software, levantamento dos trabalhos correlatos, análise de pontos passíveis de gerenciamento automatizado em etapas específicas de aplicação, e por fim, a utilização de ferramentas de auxílio ao processo de software.

As etapas descritas acima serão detalhadas no decorrer deste capítulo.

8.1 METODOLOGIA DO TRABALHO DESENVOLVIDO

A metodologia abordada neste trabalho é resultado de um estudo de fatores dos quais resultaram na identificação de etapas passíveis de gerenciamento automatizado de um software, por meio de ferramentas de auxílio seguidas de sua implementação com a utilização dos métodos do MSF, processo do qual reuni práticas eficientes de desenvolvimento de sistemas.

Dentre os pontos levantados, podem-se destacarem as etapas seguintes deste capítulo.

8.1.1 Levantamento Bibliográfico

O levantamento bibliográfico baseou-se na procura de temas referentes ao ciclo de vida de um software com o objetivo de analisar as técnicas de desenvolvimento eficazes para um produto final de qualidade. Desta forma, pode-se compreender os conceitos de processos dos quais envolvem o desenvolvimento de software, com suas principais características e funcionalidades, seguidas da análise de ferramentas de auxílio para automatização de etapas de desenvolvimento, e por fim, o levantamento de trabalhos correlatos referentes a pesquisa realizada.

No decorrer desta etapa, observaram-se diversificadas bibliografias no idioma português, referentes ao ciclo de vida de software por se tratar de um assunto indispensável na busca pela qualidade no processo de desenvolvimento de software exigida na atualidade pelas organizações.

As referências foram encontradas de forma impressa, por meio de livros da Engenharia de Software ou meio eletrônico, referentes a trabalhos de conclusão de curso, monografias, dissertações relatórios e periódicos científicos.

Outra forma de referência da qual contribuiu significativamente para o desenvolvimento desta pesquisa foram as publicações em artigos científicos acessados por meio eletrônico, referentes a metodologias e ferramentas de auxílio no controle automatizado de etapas de desenvolvimento de software com qualidade.

A seguir, serão descritos de forma detalhada os conceitos referentes ao ciclo de vida de desenvolvimento do software realizados nesta pesquisa.

8.1.2 Estudo referente a Modelos, Processos, Normas e Padrões de desenvolvimento de Software

Durante o levantamento bibliográfico realizado nesta pesquisa, procurou-se compreender todo o processo do qual envolve o ciclo de vida de um software, com o intuito de selecionar as metodologias que melhor se adéquam para a realização do controle automatizado de etapas de desenvolvimento do software de busca de produtos em uma loja virtual, um agente *e-commerce*, desenvolvido pelo acadêmico Jaison Niehues.

O estudo teve início com o levantamento de informações referente ao histórico da Engenharia de Software, um conceito criado para solucionar problemas de softwares encontrados na época como, por exemplo, atrasos no prazo de entrega e difícil manutenção de sistemas complexos com o objetivo de melhorar a qualidade do produto final por meio de metodologias estabelecidas.

A etapa seguinte foi constituída da análise das metodologias disponibilizadas na Engenharia de Software, um fator fundamental para a seqüência da elaboração deste trabalho. Dentre os recursos estudados, foram compreendidos os modelos e processos de desenvolvimento de software associados a normas e padrões de qualidade.

Baseando-se nas teorias abordadas, observou-se o modelo iterativo o mais propício a ser utilizado no contexto da aplicação por possibilitar a realização do trabalho em ciclos de desenvolvimentos compostos pelas etapas de análise, projeto, implementação e testes realizados a cada iteração facilitando a identificação de falhas em etapas anteriores ao término do desenvolvimento de todo o sistema.

No contexto de processos de desenvolvimento, optou-se pela adoção do processo MSF, pelo fato de utilizar métodos flexíveis e com ausência de detalhes, associadas a práticas do padrão CMMI, sugerido para processos formais. Ambas as técnicas possuem o objetivo de proporcionar o aumento da qualidade, eficiência e produtividade da equipe por meio de processos bem definidos e organizados.

A seguir, serão descritos os trabalhos correlatos referentes a esta pesquisa.

8.1.3 Levantamento dos Trabalhos Correlatos

O levantamento dos trabalhos correlatos ocorreu durante todo o período compreendido entre a elaboração desta pesquisa, baseando-se nos temas referentes aos modelos de processos de desenvolvimento, e principalmente voltados a ferramentas de auxílio para o controle automatizado de etapas específicas do projeto com o objetivo de proporcionar um gerenciamento de qualidade.

Por meio deste levantamento pretende-se colaborar de forma significativa para o desenvolvimento de pesquisas futuras nesta área de conhecimento por se tratar de um assunto de fundamental importância na Engenharia de Software e por se tratar de um tema não abordado até o presente momento em trabalhos de conclusão de Curso na UNESC.

Na seqüência deste capítulo, serão descritas detalhadamente as etapas selecionadas para o controle automatizado por meio de ferramentas de controle.

8.1.4 Identificar etapas passíveis de gerenciamento automatizado no processo de desenvolvimento do Software

O estudo dos conceitos das metodologias referentes ao desenvolvimento de software teve por objetivo a identificação e compreensão das etapas das quais envolvem o ciclo de vida de um software em busca da qualidade, uma abordagem na qual vêm obtendo ênfase nas organizações da atualidade.

Dentre as metodologias analisadas, pode-se verificar que os processos de software existentes apresentam-se divididas em etapas de desenvolvimento semelhantes, caracterizadas pelo levantamento de requisitos, análise de requisitos, projeto, implementação, testes, e por fim a implantação, diferenciando-se entre si pela forma na qual são organizadas.

Todas as etapas citadas são de fundamental importância na busca da qualidade de software e conseqüentemente devem ser elaboradas de acordo com o processo adotado pela organização, sempre que possíveis automatizadas, e em forma de ciclos iterativos e incrementais, permitindo a identificação de não conformidades antes da implantação do software no cliente.

No entanto, a presente pesquisa optou por evidenciar o gerenciamento automatizado das etapas de projeto, implementação e testes, mais especificamente nas atividades referentes ao controle de atividades, código fonte e testes unitários por constituírem etapas posteriores a análise de requisitos, já realizado no trabalho desenvolvido pelo acadêmico Jaison Niehues, por meio da diagramação em UML.

As atividades, por sua vez, foram selecionadas perante as fases de desenvolvimento por possibilitarem, quando implementadas de forma apropriada, o

gerenciamento eficiente pelo gerente do projeto, no progresso do software, resultando em um produto final de qualidade.

Enquanto o controle das tarefas a serem efetuadas possibilita o acompanhamento por parte do gerente, das atividades das quais estão sendo executadas por determinado usuário, o controle de código fonte permite que versões de arquivos possam ser rastreadas e recuperadas caso sejam resultado de alterações indesejáveis.

Por fim, os testes unitários, possuem o objetivo de testar os códigos implementados referentes a alguma tarefa proposta com o intuito de minimizar a ocorrência de falhas em determinados módulos do sistema.

8.1.5 Compreender e utilizar Ferramentas de Auxílio ao Processo de desenvolvimento de Software

Conforme citado anteriormente, foi adotado o modelo de processo MSF utilizado em todo o gerenciamento do ciclo de vida do software, por apresentar uma metodologia propícia ao sistema, associada a padrões do CMMI, um processo formal de gerenciamento.

Definidas as atividades para a realização do gerenciamento de forma automatizada, deu-se seqüência a este trabalho a pesquisa relativa a ferramentas de auxílio encontradas no mercado atual para as funções especificadas.

Como resultado da pesquisa efetuada, foram encontradas diversificadas ferramentas cujo objetivo é promover a automatização de atividades propostas no desenvolvimento do ciclo de vida de um software avaliadas de acordo com as características explicitadas. Dentre as ferramentas selecionadas, pode-se destacar o

Visual Studio Team System, o *Enterprise Architect*, o *Subversion* e o *JUnit,* descritas no capítulo 6 do presente trabalho.

Depois de especificadas as ferramentas, baseando-se na compreensão de suas funcionalidades e necessidades impostas pelas etapas de desenvolvimento selecionadas iniciou-se o processo prático de automatização do software, exemplificadas a seguir.

8.1.5.1 GERENCIAMENTO DE ATIVIDADES NO VSTS

A utilização do ambiente de desenvolvimento do VSTS com todos os seus recursos disponíveis exige um planejamento de sua instalação de forma adequada sendo necessário levar em consideração alguns fatores pré-determinantes como pré-requisitos de software e o tipo de instalação a ser adotada, descritos no Apêndice A deste documento.

Dentre os tipos de instalação disponibilizados pela ferramenta, optou-se pela escolha do modelo *MSF for CMMI Process Improvement*, por apresentar as características de gerenciamento selecionadas para a elaboração desta pesquisa.

As atividades informadas nesta etapa de gerenciamento, denominadas pela ferramenta de *work items*, condizem com as tarefas realizadas neste projeto, seguindo conceitos do processo MSF, conforme citadas na Figura 7.

All Work Items [Results]* | Start Page | Test Manager | Source Control Explorer

ID	Work Item...	State	Assigned To	Title
31	Task	Closed		Definir Permissões ao menbro da equipe
32	Task	Closed		Reposição do Código fonte
33	Task	Closed		Migração das Tarefas
34	Task	Proposed	Administrador	Definir Politicas de Check-in
35	Task	Proposed	desenvolvedor_01	Desenvolvimento do Fonte Referente a Solicitações de usuários
36	Task	Closed		Estrutura do Projeto
37	Task	Proposed	gerente_01	Criando a Versão Build 1
38	Task	Proposed	gerente_01	Definir Configuração do Plano de Manutenção
39	**Task**	**Closed**		**Conversão do Banco de Dados**
40	Task	Proposed	desenvolvedor_02	Documentar os sistema desenvolvido
41	Task	Proposed	analista_teste_01	Cenários de Testes Unitários
42	Task	Proposed	Administrador	Criação do Plano de Projeto
43	Task	Proposed	desenvolvedor_01	Migrar a Base de dados
44	Task	Closed		Criar Plano de Iteração (para niveis de versão)
45	Task	Proposed	Administrador	Reposição do Código fonte
46	Task	Active	desenvolvedor_02	Criar manual de procedimentos para Formulário
47	Task	Proposed	gerente_01	controle codigo fonte
48	Review	Active	Administrador	Revisão a estrutura do Banco de Dados Atual
49	Review	Active	gerente_01	Revisão Acessos dos Membros da Equipe do Projeto
50	Review	Active	analista_teste_02	Revisão dos testes de Unidade

Figura 7. Lista de *work items* na ferramenta VSTS.
Fonte: MICROSOFT, (2008)

As tarefas foram armazenadas no banco de dados da aplicação permitindo ao gerente do projeto um domínio eficiente do andamento do projeto. As atividades foram divididas em duas categorias, referentes a tarefas e revisões, associadas aos usuários cadastrados para a representação dos membros da equipe, responsáveis pelo desenvolvimento da aplicação.

O controle efetivado possibilitou o acesso das informações provenientes do desenvolvimento do sistema por meio do acompanhamento do status determinado pela atividade em questão, descritos na Figura 7, podendo ser acessadas diretamente do *Microsoft Excel*, recurso disponível pela ferramenta.

Em determinadas situações foram incluídos links entre *os work items* e as *changesets*[3], recurso disponível para indicar a seqüência na qual as atividades devem

[3] Representam o conjunto das alterações introduzidas por uma única operação de *check-in*

ser seguidas e qual o código fonte foi modificado. Para exemplificar, pode-se citar o link criado no *work item* com identificador 48, demonstrado na Figura 8.

Figura 8. *Links* associados a *work items*.
Fonte: MICROSOFT (2008)

Neste sentido, foi realizada a análise individual de cada *work item*, a medida na qual os mesmos sofreram algum tipo de alteração comprovando a eficiência do histórico na identificação de modificações sofridas, conforme descrito no *work item* 39, referente a conversão do banco de dados efetuada no trabalho, conforme visualizado na Figura 9.

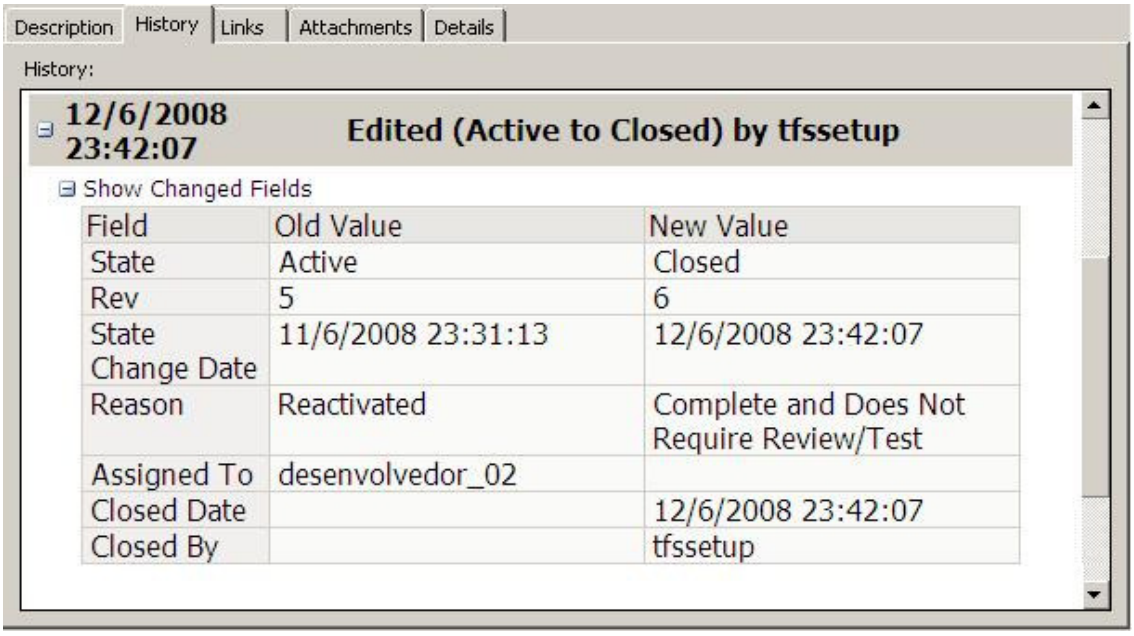

Figura 9. Histórico da alteração de um *work item*.
Fonte: MICROSOFT (2008).

Possibilitou também com o controle automatizado das atividades, uma previsão dos cronogramas e prazos de maneira eficiente por meio de relatórios que retratam o andamento do projeto, criados automaticamente pela ferramenta, acessado diretamente de um portal no qual ficam armazenadas diversas informações do processo, conforme mostrado na Figura 10.

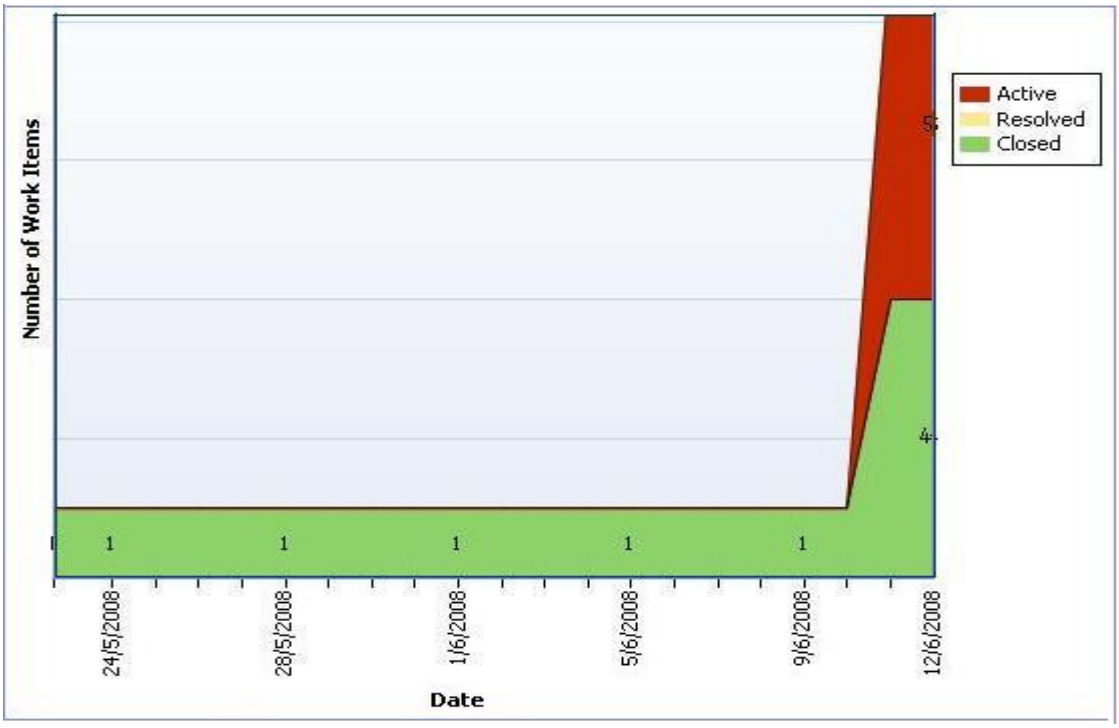

Figura 10. Relatório do andamento das atividades.
Fonte: MICROSOFT (2008).

Os *work items* foram utilizados em todas as etapas de desenvolvimento propostas, pelo fato da ferramenta VSTS abranger todo o ciclo de vida do software.

8.1.5.2 GERENCIAMENTO DE ATIVIDADES NO *ENTERPRISE ARCHITECT*

A instalação do EA não exigiu um esforço adicional, pois diferentemente da ferramenta *Visual Studio Team System*, a configuração do ambiente ocorre após o momento da instalação.

O *Enterprise Architect*, apesar de proporcionar a fase inicial da modelagem estrutural do sistema em UML, não define o processo a ser utilizado no desenvolvimento do software, apresentando funcionalidades bem definidas, documentadas e com ausência de detalhes, características encontradas no processo da

Microsoft Solution Framework, modelo adotado para a elaboração do presente trabalho.

As tarefas adicionadas ao EA representam as mesmas atividades informadas na Ferramenta VSTS, e conseqüentemente seguem os mesmos critérios estabelecidos, relatados anteriormente, sendo descritas conforme Figura 11.

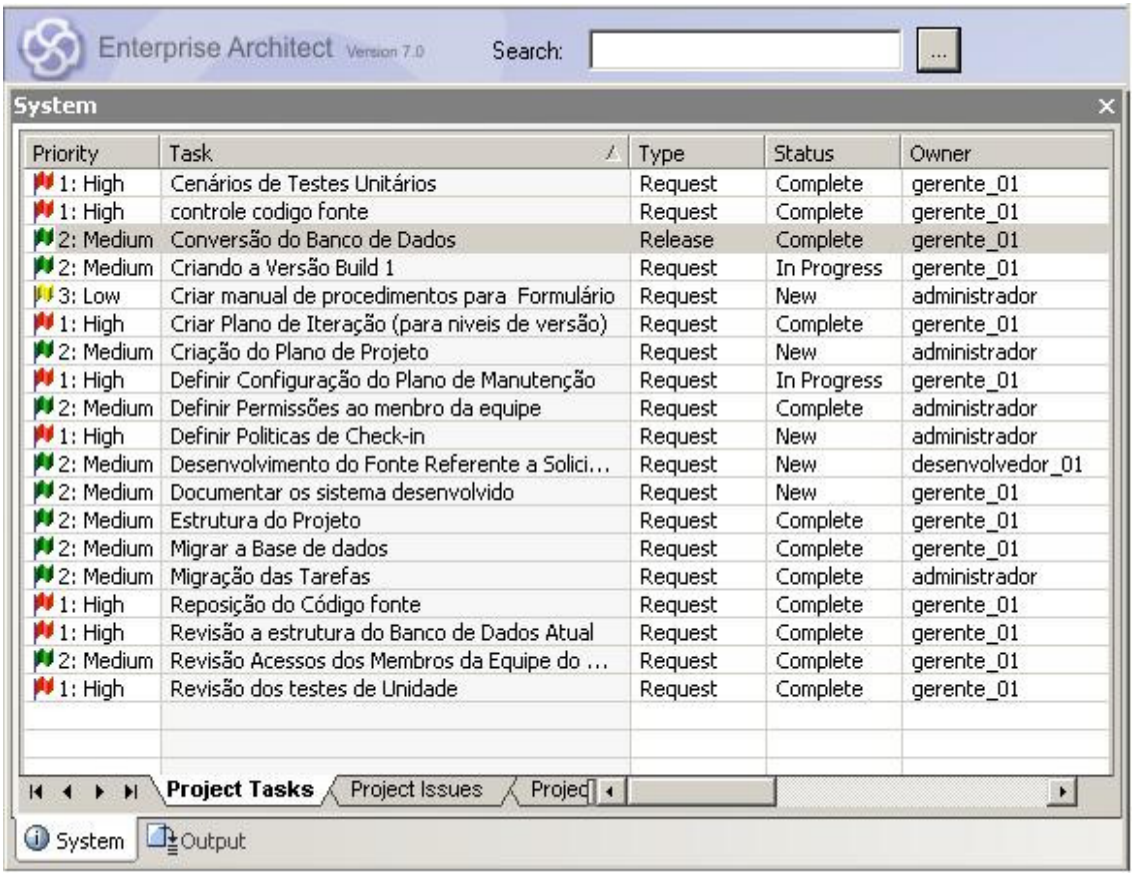

Figura 11. Lista de Atividades na ferramenta *Enterprise Architect*.
Fonte: *SPARX SYSTEMS* (2008).

O gerenciamento das atividades de forma automatizada no EA possibilitou o gerenciamento da aplicação de forma simples e eficaz, porém com a ausência da integração com as demais etapas de desenvolvimento, propícios para sistemas dos quais exigem um menor nível de controle.

Apesar de não integrar todas as fases de desenvolvimento do software, o *Enterprise Architect* consiste em uma ferramenta de fácil entendimento da qual pode ser associada a outras ferramentas de auxílio como, por exemplo, o *Subversion* para efetuar o controle de código fonte do sistema.

8.1.5.3 CONTROLE DE VERSÃO NO *TEAM FOUNDATION VERSION CONTROL*

O *Team Foundation Version Control* é uma ferramenta que por *default*, é ativada no momento da instalação do TFS, sendo necessárias apenas configurações para a adaptação ao projeto, referente a atualização dos repositórios dos arquivos e permissões de acesso.

Neste sentido, foram criados um repositório de código fonte no servidor, e outros dois locais, denominados *workspaces* pelo TFVC, em máquinas diferentes, representando sua utilização por dois membros da equipe de desenvolvimento.

O próximo passo foi a criação de uma política de *check-in* voltada aos *work items*, conforme Figura 12, um recurso disponibilizado pela ferramenta, cujo objetivo é manter o controle do acesso aos códigos fonte no servidor, obrigando o desenvolvedor associar uma atividade na qual justifica a alteração do fonte alterado.

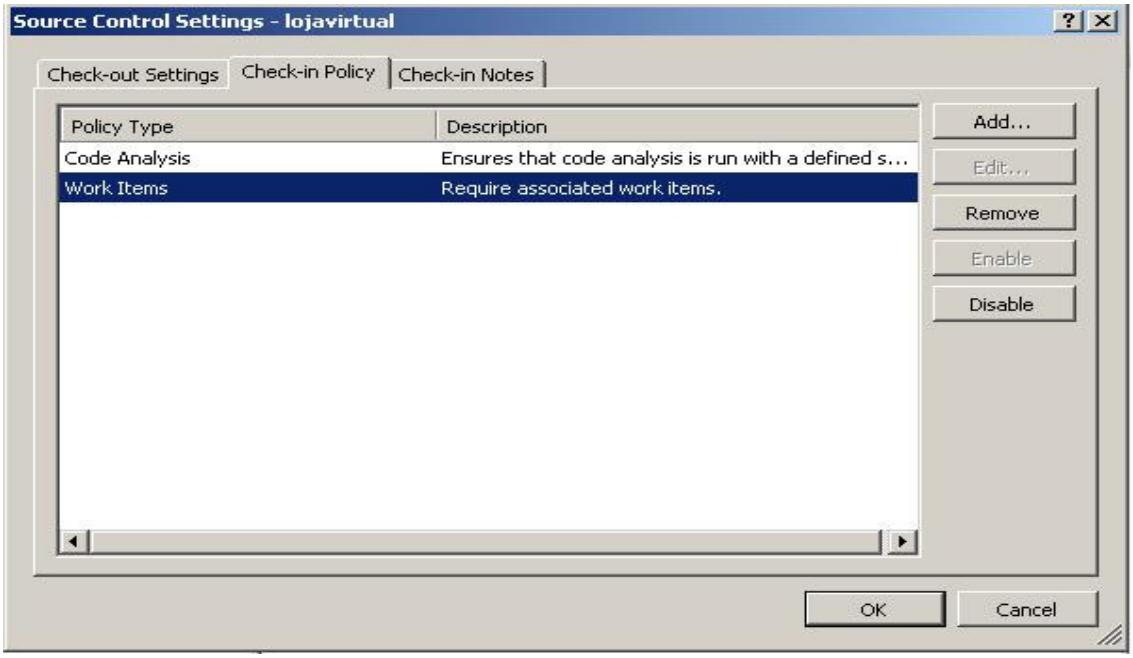

Figura 12. Política de *check-in* na ferramenta VSTS.
Fonte: MICROSOFT (2008).

Após definidas as configurações apropriadas ao projeto, foram efetuadas alterações em determinados códigos fontes do sistema para comprovar a eficiência do controle de código fonte automatizado por meio das funcionalidades encontradas no TFVC. Dentre as modificações efetivadas, pode-se citar a inclusão do campo CEP, do fonte __contato.php, conforme figura 13.

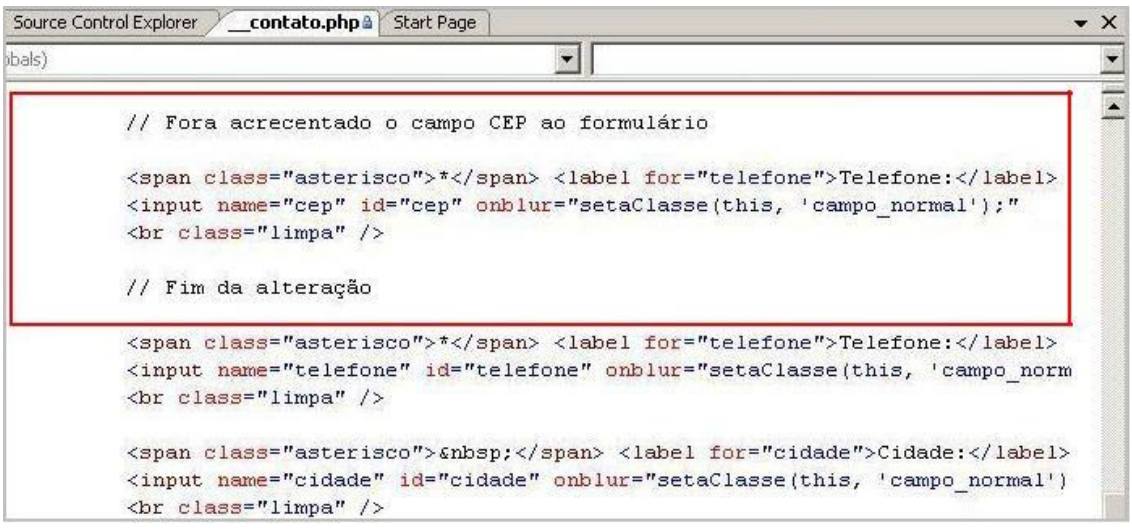

Figura 13. Implementação do campo CEP no arquivo __contato.php.

Depois de realizada a alteração, observou-se que o fonte fora salvo apenas no *workspace* local com seu nome associado a uma versão especificada pela ferramenta, uma característica importante para impedir conflitos e perdas de informações. No momento em que foi realizada a atualização, entrou em cena a política de *check-in* aplicada no projeto, obrigando o desenvolvedor a selecionar uma tarefa, conforme demonstrado abaixo na Figura 14.

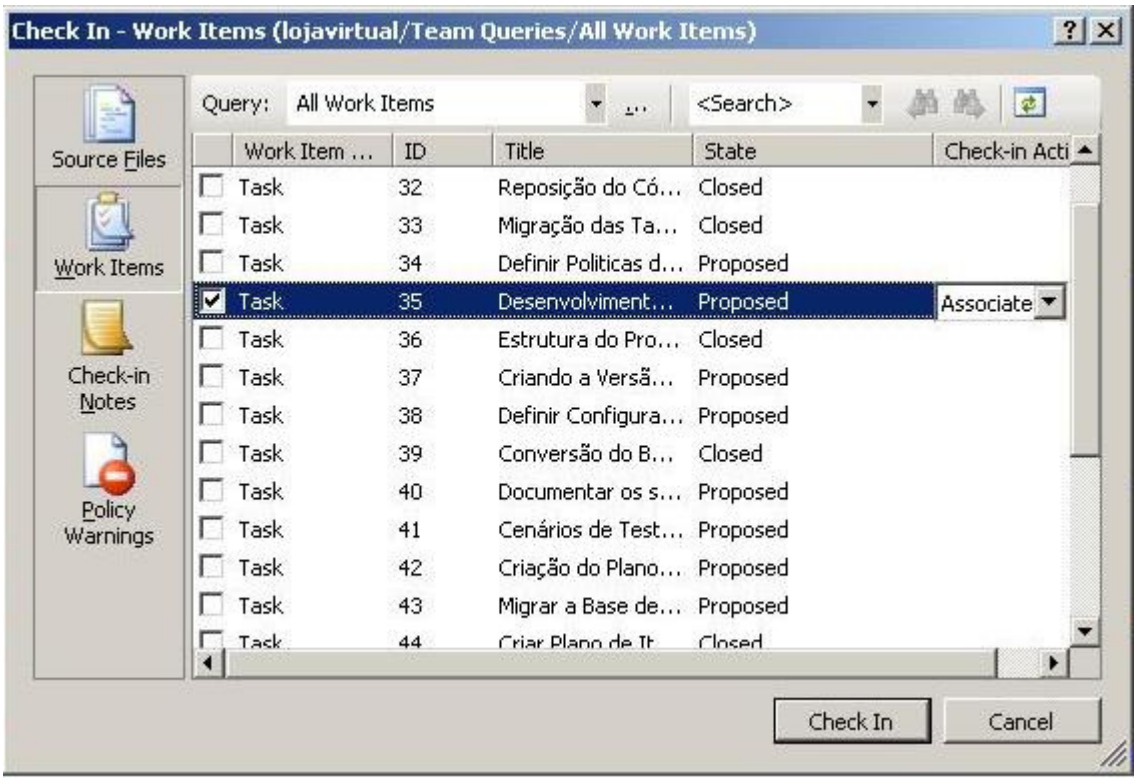

Figura 14. Lista para seleção de tarefas.

Finalizado o processo de atualização do código fonte no servidor, procurou-se evidenciar as formas propostas para a recuperação e restauração de versões anteriores do sistema, um método de fundamental importância para o desenvolvimento de uma aplicação. Dentre as possibilidades disponíveis no TFVC destacam-se a maneira de recuperação de versões por meio de *changesets*, por data, versão do

repositório local, ou ainda ultima versão, implementada conforme descrito na Figura 15:

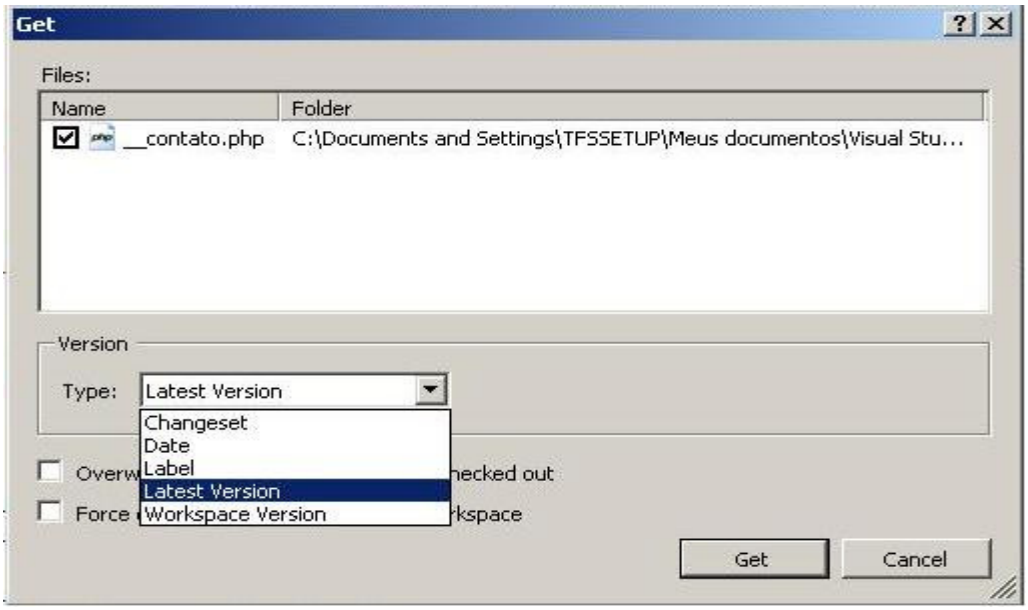

Figura 15. Formas de Recuperação de Versão do arquivo.

Neste contexto, o gerenciamento de versão automatizado possibilitou o acompanhamento preciso das alterações realizadas nos arquivos, com um histórico bem documentado com informações referentes a datas e usuários dos quais praticaram as modificações, e ainda a recuperação de versões anteriores de diversas maneiras, impedindo um atraso no desenvolvimento do projeto por perdas indesejáveis de arquivos.

8.1.5.4 CONTROLE DE VERSÃO NO *SUBVERSION*

O *Subversion* consiste em uma ferramenta open source para o controle de versão de código fonte na qual armazena documentos de diferentes naturezas. A sua utilização pode ser facilitada pela instalação do *TortoiseSVN*, um ambiente gráfico

utilizado neste trabalho voltado ao sistema Operacional Windows que permite o acesso aos repositórios diretamente no *Windows Explorer*.

Finalizada a instalação do SVN, foram criados os repositórios no servidor e em diretórios locais da mesma forma na qual fora realizada no TFVC, seguido da migração dos códigos fontes do servidor para as pastas locais, por meio da opção *check-out*[4] encontrada no SVN, armazenando as informações desta operação no histórico do projeto.

A partir da configuração do ambiente de gerenciamento, foram realizadas modificações em determinados códigos fontes para relatar o controle automatizado dos arquivos do sistema por meio da ferramenta *Subversion.*

Assim como no TFVC, o *Subversion* não atualiza o código fonte diretamente no servidor, fazendo uma cópia no repositório da máquina local. Ao término da modificação desejada, deve-se então aplicar o comando SVN *commit*, para que o arquivo possa ser direcionado ao repositório principal. Ao optar pela efetivação da alteração, o SVN apresenta uma lista contendo os arquivos da pasta local com a opção para documentar o motivo da alteração, conforme descrita na figura 16.

[4] Operação da qual indicamos a realização da alteração de um arquivo no qual se encontra sob controle de versão.

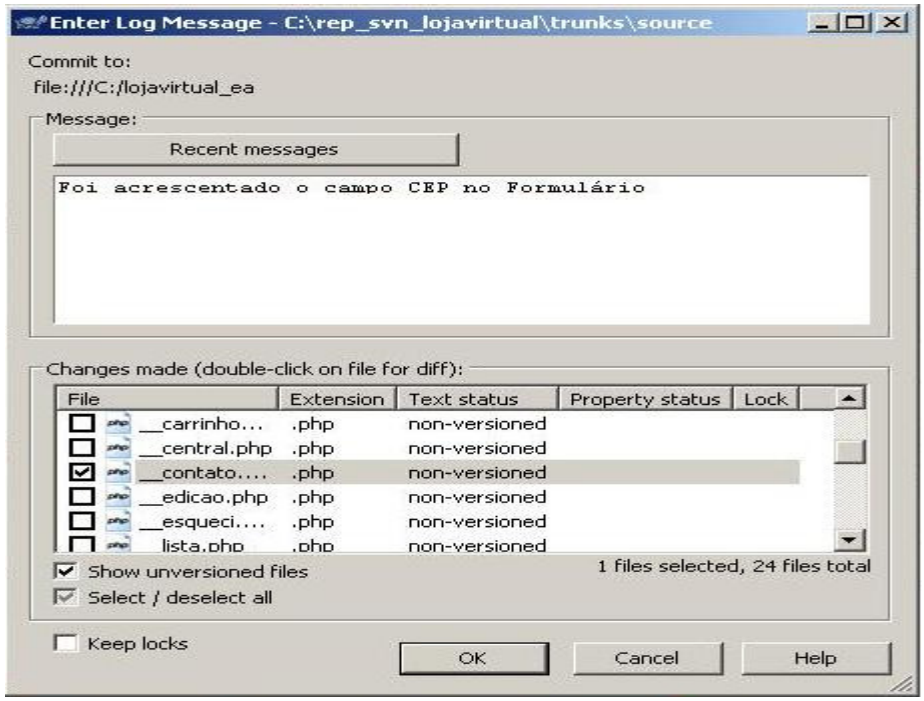

Figura 16. Seleção dos arquivos para efetuar o *commit*.

Desta forma, observou-se que depois de selecionado, o arquivo fora atualizado no repositório do servidor, proporcionando o controle satisfatório da versão do sistema com um histórico da ação realizada e ainda relatando a respeito do sucesso ou não da transação, conforme mostrado na Figura 17.

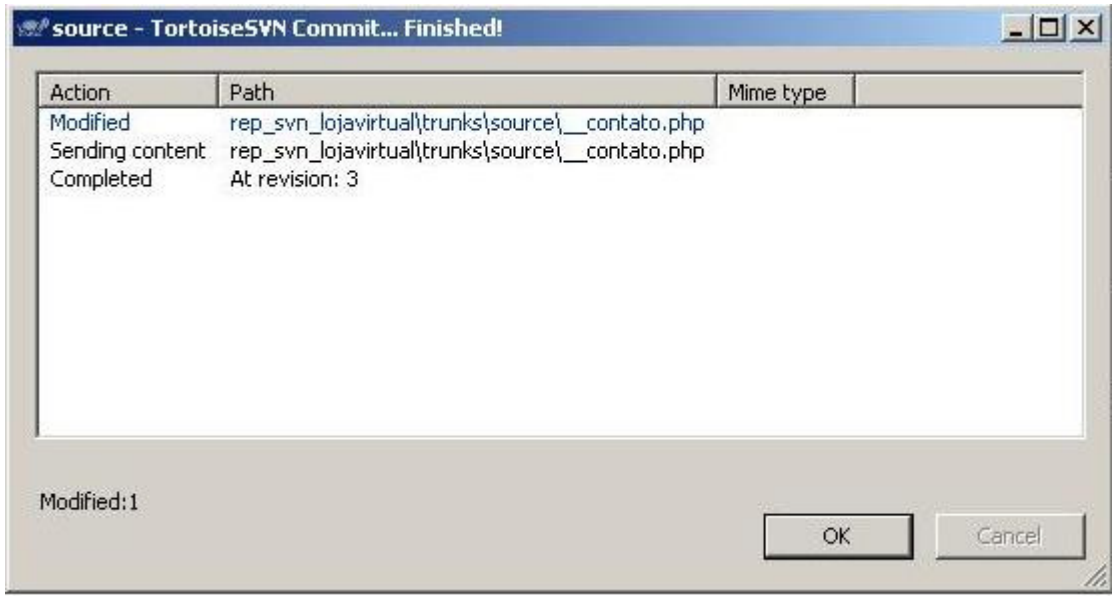

Figura 17. Histórico do resultado da operação de *commit*.

Neste contexto, pode-se concluir que o gerenciamento automatizado do controle de versão no *Subversion* foi satisfatório, pois possibilitou a alteração dos arquivos sem causar problemas encontrados em empresas das quais não utilizam nenhuma ferramenta de auxílio para esta etapa de desenvolvimento, referentes a desenvolvedores sobrescreverem arquivos modificados por outro usuário.

Alem do mais, o *Subversion* possibilita sua integração com outras ferramentas como, por exemplo, o *Enterprise Architect* utilizado neste trabalho, contribuindo de forma significativa para o gerenciamento eficiente do gerente do projeto na organização.

8.1.5.5 TESTE DE UNIDADE NO *VSTS FOR TESTERS*

A criação e execução de testes unitários é uma função da qual se faz presente no processo *Microsoft Solution Framework* de desenvolvimento de software e por este motivo devem ser aplicados em todos os projetos dos quais visam a qualidade como resultado final.

Neste contexto, fora criado para o presente trabalho um projeto de testes denominado TestLojaVirtual e inserido dentro deste, artefatos necessários para a elaboração desta etapa de controle automatizado.

Na seqüência, foram adicionados a este projeto os formulários de testes unitários, já previamente estruturados pela ferramenta, modificados de acordo com a necessidade do projeto. Dentre os testes criados, pode-se citar a realização do teste denominado Numero_Acessos, com o objetivo de verificar o resultado proveniente de um cálculo de divisão entre dois algarismos.

Para a criação do teste proposto, foram desenvolvidos dois arquivos de código fonte, o primeiro com o nome de Verifica_Media_Acesso.cs no qual realiza a divisão de duas variáveis nomeadas acessos e dias, dividendo e divisor respectivamente, recebidas por parâmetro do arquivo Numero_Acessos.cs, desenvolvido conforme descrito na Figura 18.

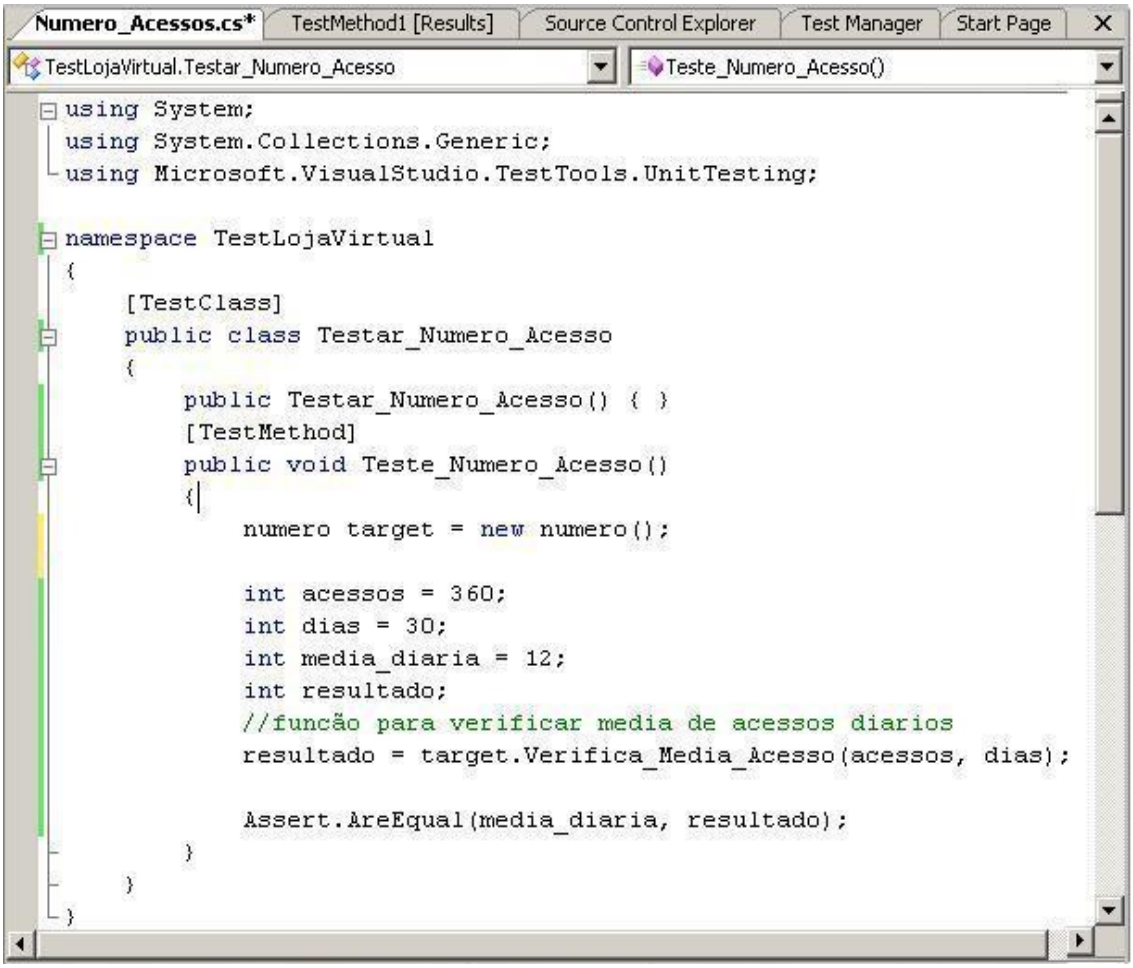

Figura 18. Código fonte do arquivo Numero_Acessos.cs

Neste contexto, o arquivo Numero_Acessos.cs refere-se a um teste elaborado para verificar se o resultado da operação condiz com o valor retornado pela variável de nome resultado pelo primeiro arquivo que representa o código a ser testado.

Desta forma, foram passados por parâmetro os valores 360 e 30 respectivamente para o arquivo Verifica_Media_Acesso.cs, e esperou-se que o mesmo

retornasse como resultado do cálculo efetuado, o valor 12 referente a média de acessos diários ao projeto.

Após a criação do teste, tratou-se de executá-lo e efetuar a análise das informações obtidas, armazenando-as para a realização de futuras comparações. O resultado final deste procedimento apresentou-se de forma satisfatória, conforme mostra a Figura 19, pois não relatou divergência entre valores resultantes.

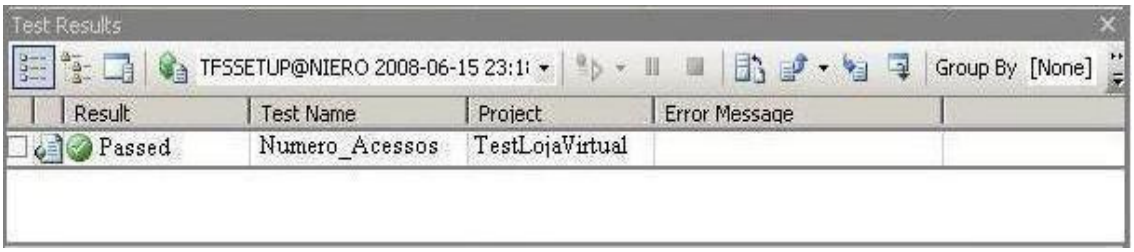

Figura 19. Resultado do teste efetuado na ferramenta VSTS

Neste sentido, pode-se observar que o controle automatizado por meio dos testes unitários consiste na implementação de códigos dos quais testam outros códigos, evidenciando pontos passíveis de falhas antes da implantação do sistema no cliente, proporcionando benefícios indispensáveis ao ciclo de vida do software.

8.1.6 Resultados Obtidos

A presente pesquisa resultou na compreensão de metodologias voltadas ao desenvolvimento de softwares, disponibilizadas na Engenharia de Software seguido pelo gerenciamento automatizado de etapas referentes ao controle de código fonte, controle das atividades a serem executadas no projeto, e por fim, testes de unidade, por meio do auxílio de ferramentas.

O estudo dos processos e modelos analisados descreveu detalhadamente as fases fundamentais de desenvolvimento a serem seguidas em um projeto, juntamente com a seqüência na qual devem ser elaboradas no ciclo de vida do software para se obter um produto final de qualidade e satisfação perante o cliente/usuário.

O gerenciamento das atividades a serem realizadas pelos membros da equipe possibilitou o controle eficiente do estágio no qual se apresenta o software. O *status* no qual se encontram as atividades registradas permite a análise do esforço dos responsáveis pela sua execução, sem que o gerente tenha que consultar individualmente o fluxo dessas tarefas.

Na seqüência, o controle de versão permitiu o trabalho de diferentes programadores em um determinado código fonte de forma simultânea, sem que fossem ocasionados conflitos desnecessários. Além do mais, possibilitou a restauração de versões anteriores de arquivos, e demonstração das alterações efetuadas entre os arquivos associados aos seus respectivos usuários.

Por fim, os testes unitários realizados em partes especificas do sistema apresentaram-se de maneira satisfatória por demonstrarem que a funcionalidade da implementação atendeu as exigências do projetista quando comparados com os requisitos levantados.

Pode-se concluir que o gerenciamento automatizado das etapas descritas acima resulta, de forma geral, em softwares de qualidade, menos propícios a falhas e com orçamentos e prazos bem definidos.

CONCLUSÃO

O presente trabalho evidenciou a importância da Engenharia de Software no desenvolvimento de sistemas de forma eficiente com qualidade para atender as necessidades das organizações da atualidade. Seu principal objetivo é fornecer técnicas das quais possibilitem ao gerente do projeto um controle eficaz dos processos durante a execução das etapas do ciclo de vida do software.

Neste contexto, a pesquisa alcançou seu objetivo, baseando-se no estudo de metodologias referentes a modelos, processos, normas e padrões de desenvolvimento voltados ao controle de qualidade de software associados a ferramentas de gerenciamento, fator indispensável para as organizações da atualidade.

No entanto, a definição de uma metodologia padrão e ferramentas em busca do software com qualidade é dificultada em determinadas situações pela diversidade de técnicas existentes na atualidade associadas à falta de conhecimento das mesmas. Para definir o processo ideal, deve-se utilizar os métodos cujos propósitos melhor condizem com a aplicação a ser desenvolvida e não adaptar o sistema a uma metodologia sugerida.

Em complemento aos temas analisados foram aplicadas técnicas de gerenciamento automatizadas no desenvolvimento do software de busca de produtos em uma loja virtual, desenvolvido pelo acadêmico Jaison Niehues, por meio de ferramentas de auxílio nas atividades de testes, controle de tarefas a serem realizadas e ainda no controle de versão do sistema.

Em relação às metodologias abordadas nesta pesquisa, pôde-se observar que os processos apresentados diferenciam-se entre si pelo grau de documentação exigido e na forma da qual as etapas de desenvolvimento são encadeadas. Desta forma, as

empresas devem adotar o método do qual melhor se adéqüe as características do sistema, sendo auxiliadas sempre que possível por ferramentas de controle, independentemente do procedimento adotado.

Seguindo a concepção adotada nesta pesquisa, sugere-se como trabalhos futuros a implementação de novos recursos de gerenciamento automatizado referentes as etapas aplicadas neste projeto com a realização do estudo de diferentes tipos de testes de software, e ainda promover o controle nas fases de análise de requisitos e arquitetura de software.

Outra recomendação sugerida trata-se da análise de diferentes ferramentas de auxílio, existentes no mercado, para o gerenciamento de projetos com qualidade. Dentre elas, destacam-se as ferramentas das quais abrangem, preferencialmente, todo o ciclo de desenvolvimento do sistema, uma característica pouco encontrada na atualidade.

Como o trabalho faz uso de determinados processos para o controle de gerenciamento, seria importante também ressaltar o gerenciamento de um software em diferentes metodologias não abordadas nesta pesquisa, relatando seus principais conceitos e impactos causados perante a organização.

REFERÊNCIAS

Associação Brasileira de Normas e Técnicas. **ABNT,** São Paulo, 2006. Disponível em: < http://www.abnt.com.br/default.asp?resolucao=1024X768>. Acesso em: 07 set. 2007.

BEZERRA, Eduardo. **Princípios de analise e projeto de sistemas com UML.** Rio de Janeiro: Campus, 2002.

BORBOREMA, Thiago. **Impacto da aplicação da Metodologia XP nas organizações de desenvolvimento de software.** 2007. Trabalho de Conclusão de Curso – Sistemas de Informação, Universidade do Vale do Sapucaí, Pouso Alegre, 2007.

BRAUDE, Eric. **Projeto de Software da programação a arquitetura: uma abordagem baseada em Java.** Porto Alegre: Bookman, 2005.

BRESSANO, Alércio; CÂMARA, Fábio; SANCHES, Andrey et al. **Visual Studio Team System Rocks.** Florianópolis: Visual Books, 2006.

BRITO, Alessandro Antonio de. Testes no Team System – Parte 1, 2008. Disponível em: < http://www.devmedia.com.br/articles/viewcomp.asp?comp=7958>. Acesso em: 16 mai. 2008.

CAETANO, Cristiano. Cargos e salários: Quanto ganha o profissional de teste e qualidade de software. **Linha de Código**, set. 2007. Disponível em: < http://www.linhadecodigo.com.br/artigos.asp?id_ac=1299 > Acesso em: 09 nov. 2007.

_____. Processos e qualidade de software. Linha de Código, mar. 2006. Disponível em: < > Acesso em 29 jan. 2008.

CAMPOS, Fábio Martinho. **Quais as Reais Características da Qualidade da NBR ISO/IEC 9126-1?**, Taubaté, 2007. Disponível em: < http://www.linhadecodigo.com.br/ArtigoImpressao.aspx?id=1444>. Acesso em: 13 fev. 2008.

CAPRA, Roberto Neto. **A culpa é desse tal de CMMI**, ago. 2007. Disponível em: < http://www.linhadecodigo.com.br/Artigo.aspx?id=1382 >. Acesso em: 12 set. 2007.

CARDINAL, Mário. Infra-estruturas dirigidas a teste, jul. 07. Disponível em: <http://www.microsoft.com/brasil/msdn/arquitetura/Journal/Test_Driven_Infrastructures.mspx> Acesso em: 15 mai. 2008.

COLLINS-SUSSMAN, Ben; FITZPATRICK, Brian W.; PILATO, C. Michael. **Version Control with Subversion,** 2002. Disponível em: < http://svnbook.red-bean.com/en/1.4/svn-book.pdf >. Acesso em: 03 maio 2008.

COUTO, Ana Brasil. **CMMI – Integração dos Modelos de Capacitação e Maturidade de Sistemas.** Rio de Janeiro:Ciência Moderna, 2007.

DIAS, André Felipe. **Subversion**, set. 2006. Disponível em: < http://www.pronus.eng.br/artigos_tutoriais/gerencia_configuracao/subversion.php?pagNum=1>. Acesso em: 02 abr. 2008.

DURÃES, Ramon. **Visual Studio Team System**, mai. 2007. Disponível em: <http://blogs.2pc.com.br/2pc/archive/2007/05/23/visual-studio-team-system.aspx>. Acesso em: 13 abri. 2008.

_____. **Explorando o Visual Studio Team System**, jan. 2008. Disponível em: <http://www.linhadecodigo.com.br/ArtigoImpressao.aspx?id=1666>. Acesso em: 15 abr. 2008.

GARCIA, Marcus. Visual **Studio Team System: Team Foundation.** Rio de Janeiro: Brasport Livros e Multimídia Ltda, 2007.

GERVAZONI, Thiago Prastorello. **XP Extreme Programming: parte 1.** Linha de Código, Jul. 2005. Disponível em: < http://www.linhadecodigo.com.br/artigos.asp?id_ac=764 >. Acesso em: 09 nov. 2007.

_____. **Introdução ao MSF – Microsoft Solutions Framework**. Linha de Código, jul. 2005. Disponível em: < http://www.linhadecodigo.com.br/Artigo.aspx?id_ac=771 >. Acesso em: 11 nov. 2007.

_____. **Iniciação ao PMBOK no Gerenciamento de Projetos.** Linha de Código, mar. 2006. Disponível em: < http://www.linhadecodigo.com.br/Artigo.aspx?id=974>. Acesso em: 07 fev. 2008.

GOMES, Marco Antônio Pereira. **Estudo e Prototipação de Ferramenta para gerência de Projetos, enfocando a gerência de aquisições. 2003.** Trabalho de

Conclusão de Curso – Tecnologia em Informática, Universidade Luterana do Brasil, Canoas, Rio Grande do Sul, 2003.

GUSMÃO, Cristine Martins Gomes de; MOURA, Hermano Perrelli de. **Gerência de Risco em Processos de Qualidade de Software: uma Análise Comparativa,** Recife, 2004. Disponível em: < www.sbc.org.br/bibliotecadigital/download.php?paper=235> Acesso em: 26 fev. 2008.

GUSTAFSON, David A. **Teoria e problemas de Engenharia de Software**. Porto Alegre: Bookman, 2003.

INTHURN, Cândida. **Qualidade & Teste de Software.** Florianópolis: Visual Books, 2001.

JOMORI, Sergio Massao. VOLPE, Renato Luiz Della. ZABEU, Ana Cecília Peixoto. **Qualidade de software**. IETEC Instituto de educação Tecnológica, 2004. Disponível em:<http://www.ietec.com.br/ietec/techoje/techoje/tecnologiadainformacao/2004/07/01/2004_07_01_0001.2xt/-template_interna>. Acesso em: 21 fev. 2008.

KROTH, Eduardo; PFAFFENSELLER, Moisés; PFAFFENSELLER, Matheus. **Uma ferramenta de apoio ao desenvolvimento de software baseado em componentes.** 2001. Artigo - Universidade de Santa Cruz do Sul, Cruz do Sul, Rio Grande do Sul, 2001.

LEME FILHO, Trajano. **Metodologia de desenvolvimento de sistema.** Rio de Janeiro: Axcel Books do Brasil Editora, 2003.

LIMA, Adilson da Silva. **UML 2.0 Do Requisito a Solução.** 2 ed. São Paulo: Érica Ltda, 2007.

LOUSA,Hugo Antônio de Azevedo; NUNES,Carla de Tunes. **Automação de Testes Utilizando Ferramentas Open Source.** 2006. Trabalho de Conclusão de Curso – Ciência da Computação, Universidade de Brasília, Brasília, 2006.

MARTINS, José Carlos Cordeiro. **Gerenciando Projetos de desenvolvimento de software com PMI, RUP e UML.** 3 ed. Rio de Janeiro: Brasport, 2006.

MARTINS, Paula Ventura; SILVA, Alberto Rodrigues. **Comparação de metamodelos de processos de desenvolvimento de software**, 2004. Disponível em:

< http://berlin.inesc.pt/alb/static/papers/2004/pv-quatic2004.pdf >. Acesso em: 06 nov. 2007.

MICROSOFT. **Guia da Equipe de Recursos de Segurança: Visão Geral,** nov. 2006. Disponível em: < http://www.microsoft.com/brasil/technet/desktopdeployment/bdd/2007/secfea/secfea_3.mspx > Acesso em: 01 nov. 2007.

Microsoft Developer Network (MSDN). **Visual Studio Team System**. Dez. 2007. Disponível em: <http://www.microsoft.com/brasil/msdn/teamsystem/default.mspx> Acesso em: dez. 2007.

NERI, Hilmer Rodrigues.. **Técnicas de engenharia de software,** Goiânia, 2004. 36 f. Material de aula. Disponível em: <http://dns.redes.unb.br/material/ESOO/T%E9cnicas%20da%20Engenharia%20de%20Software%202.pdf> Acesso em: 13/02/2008.

NIEHUES, Jaison. **Mecanismo de busca de produtos em uma loja virtual por meio da utilização de um agente inteligente e conceitos *E-commerce***. 2007. 109 f. Trabalho de Conclusão de Curso (Especialização) – Faculdade de Ciência da Computação, Universidade do Extremo Sul Catarinense, Criciúma 2007.

SOLUTION, OAT. **Enterprise Architect**, 2007. Disponível em: < http://www.oatsolutions.com.br/ferramentas_EA.htm>. Acesso em: 29 abril 2008.

OLIVEIRA, Eric C. M. Java: **Testes Unitários e JUnit,** jan. 2005. Disponível em: < http://www.linhadecodigo.com.br/Artigo.aspx?id=576>. Acesso em: 17 mar. 2008.

PAULA NETO, Aristides Vicente de. **Criando testes com JUnit,** 2006. Disponível em: <http://www.javafree.org/dependencias/tutoriais/testes_junit.pdf>. Acesso em: 15 abri. 2008.

PETERS, James F. **ENGENHARIA DE SOFTWARE**. 3 ed. Rio de Janeiro: Elsevier, 2001.

PRESSMAN, Roger S. **Engenharia de software**. São Paulo: Afiliada, 1995.

_____. **Engenharia de software**. 5 ed. São Paulo: McGraw-Hill, 2002.

Programa Brasileiro da Qualidade e Produtividade em Software (PBQP Software). **Treze anos acompanhando e disseminando a cultura da qualidade.** Set 2006**.** Disponível em: < http://www.mct.gov.br/upd_blob/0006/6446.pdf > Acesso em: 08 set. 2007.

Project Management Institute, **PMI. Um Guia do Conjunto de Conhecimentos em Gerenciamento de Projetos Pmi,** 2006. Disponível em: <http://www.pmisp.org.br/exe/educacao/pmbok.asp>. Acesso em: 03 fev. 2008.

REZENDE, Denis Alcides. **Engenharia de software e sistemas de informação.** 3 ed. Rio de Janeiro: Brasport, 2005.

SCOTT, Kendall. **O Processo Unificado Explicado**. **Porto** Alegre: Bookman, 2003.

SEMENIUK, Joel. **Visão geral do Microsoft Visual Studio Team System 2005**, fev. 2007. Disponível em: http://www.microsoft.com/brasil/msdn/tecnologias/teamsystem/VS05TeamSysInt.mspx. Acesso em: 13 abr. 2008.

SOMERVILLE, Ian. **Engenharia de Software.** 6 ed. São Paulo: Addison Pearson, 2003.

_____. **Engenharia de Software.** 8 ed. São Paulo: Addison Pearson, 2007.

SOTILLE, Mauro. Gerenciamento de projetos na engenharia de software**. Pmtech Capacitação em projetos**. abr. 2006. Disponível em: **<** http://www.pmtech.com.br/artigos/Gerenciamento_Projetos_Software.pdf >. Acesso em: 06 abr. 2008.

QUADROS, Moacir. **Gerência de Projetos de Software: Técnicas e Ferramentas.** Florianópolis: Visual Books, 2002.

VALLS, Valéria Martin. **O enfoque por processos da NBR ISO 9001 sua aplicação nos serviços de informação, xxx 2004. Disponível em: <**http://www.scielo.br/pdf/ci/v33n2/a18v33n2.pdf>. Acesso em: 19 fev. 2008.

VASCONCELOS, Daniel Teófilo. **RUP e XP: uma visão geral.** Linha de código, out. 2005. Disponível em: < http://www.linhadecodigo.com.br/artigos.asp?id_ac=826 >. Acesso em: 10 nov. 2007.

VASCONCELOS, Ivo M. Michalick. **Guia PMBOK, 3ª. Edição: Mudando para Melhor**, nov. 2004. Disponível em: < http://www.pmimg.org.br/downloads/PMBOKThirdEdition_Ivo_04112004.PDF> Acesso em: 03 mar. 2008.

VIANNA, Mauro. **Conheça o Microsoft Solutions Framework (MSF),** Linha de Código, dez. 2001. Disponível em: < http://www.linhadecodigo.com.br/artigos.asp?id_ac=78 >. Acesso em: 11 nov. 2007.

_____. **Conheça o Rational Unified Process (RUP),** Linha de Código, jan. 2002. Disponível em: < http://www.linhadecodigo.com.br/Artigo.aspx?id=79 >. Acesso em: 12 nov. 2007.

WAZLAWICK, Raul Sidnei. **Análise e projeto de sistemas de informação orientados a objetos.** Rio de Janeiro: Elsevier, 2004.

WEBER, Kival Chaves; da Rocha, Ana Regina Cavalcante. **Qualidade e produtividade em software.** 3 ed. São Paulo: Makron Books, 1999.

WEBER,Kival Chaves ; NASCIMENTO, Célia Joseli do; MARINHO, Diva da Silva. **A História da ABNT, em detalhes**, 2006. Disponível em: <http://www.abnt.org.br/default.asp?resolucao=1280X800>. Acesso em: 18 fev. 2008.

APÊNDICE A – Instalação e configuração da Ferramenta VSTS

CONFIGURAÇÃO DO AMBIENTE DE DESENVOLVIMENTO

Pré Requisitos

A instalação do *Team Foundation Server* requer softwares da plataforma *Microsoft* considerados de ultima geração dos quais são responsáveis pela integração, segurança e comunicação e *workflow*[5] do sistema (GARCIA, 2007).

Pré Requisitos de Software

Os requisitos de software a serem seguidos são:

a) *Windows Server 2003 com SP1*;

b) *SQL Server 2005 (Sql Server +Analysis Server + Integration Services + Reporting Services);*

c) *IIS 6.0* com *ASP.NET*;

d) *Windows Sharepoint Services* com SP2;

e) .NET Framework 2.0 atualizado.

---

[5] Workflow no contexto de software é uma seqüência de tarefas encadeadas e relacionadas a um aspecto importante do projeto.

Pré Requisitos de Hardware

A configuração de *hardware* varia de acordo com a quantidade de projetos e usuários simultâneos onde para a utilização de 1-2 projetos, e 5-20 usuários são necessários no mínimo:

a) processador único de 600 GHz;

b) HD de 1 GB;

c) Memória RAM de 256 MB.

Instalação e configuração do Ambiente de desenvolvimento (Servidor)

O *Team Foundation Server* apresenta dois tipos de instalação: a *Single Server*, na qual possui apenas um único Servidor responsável por administrar a camada de dados e aplicações do TFS e a *Dual Server*, onde existe a complexidade de diversos outros servidores dos quais administram os separadamente os serviços instalados do TFS (GARCIA, 2007).

Para cada tipo de instalação devem-se seguir os pré-requisitos de hardware estabelecidos para evitar problemas durante a execução das tarefas e conseqüentemente evitar desperdício de tempo. A seguir serão descritos os passos da instalação seguindo os critérios estabelecidos para um *Single Server* (GARCIA, 2007).

Passo 1: Primeiramente deve-se analisar se requerimentos de hardware estão de acordo com o tipo de instalação desejado. Em seguida, após a instalação do *Windows Server 2003* é necessário que se realizem todos os *updates* solicitados pelo *Windows Update*.

Passo 2: Instalar na seqüência o *Internet Information Services* (IIS) 6.0 habilitando o *ASP.NET* não habilitando as extensões do *Front Page Server Extensions*. Por *default*, o serviço não é instado automaticamente no *Windows Server 2003* por motivo de segurança.

Passo 3: Instalar a atualização *WindowsServer2003-KB914961-SP2-x86-PTB* seguido do *Microsoft.NET Framework 2.0*.

Passo 4: Na seqüência deve ser instalado o *SQL Server 2005 Standard* ou *Enterprise Edition* com uma instância padrão neste computador. A instalação do mesmo não requer grandes cuidados sendo selecionados todos os serviços disponíveis no banco de dados de acordo com a Figura 20.

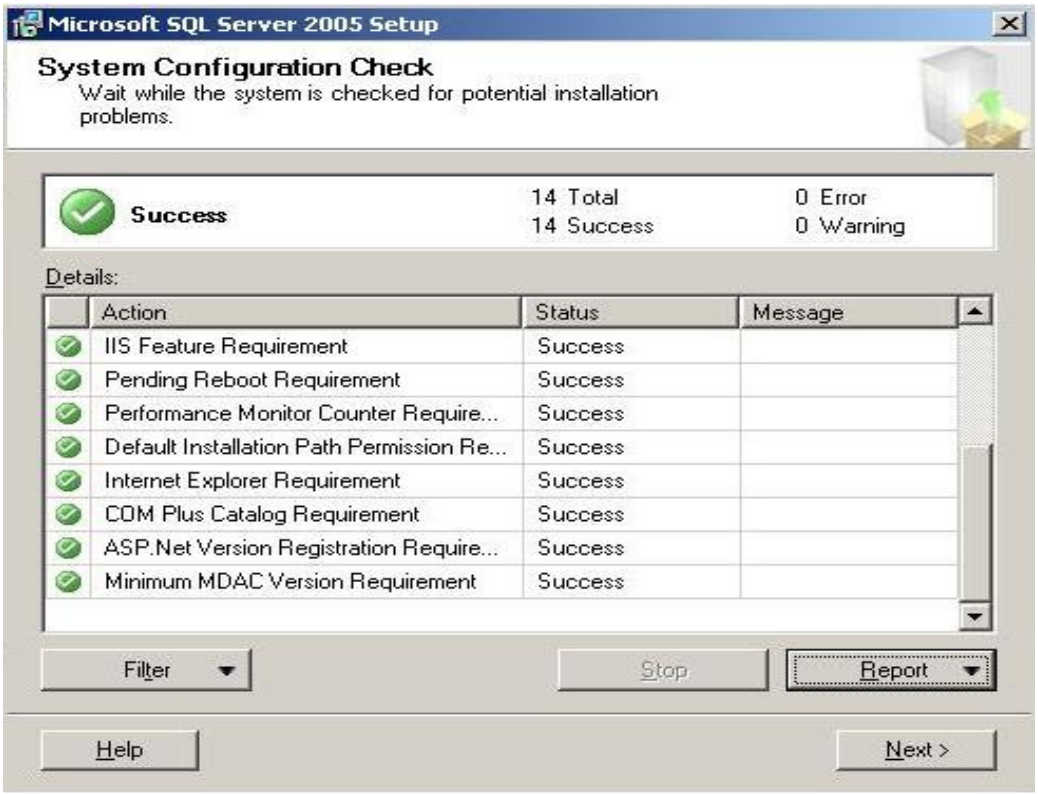

Figura 20. Instalação do *SQL Server 2005*.
Fonte*:* MICROSOFT (2008)

Passo 5: depois de instalado o *SQL Server 2005*, deve-se procurar pelo *hotfix AS2005-KB914595-x86-ENU* referente a plataforma utilizada disponível na pasta *SQLServerKB* do cd de instalação do TFS.

Passo 6: o próximo passo é verificar a funcionalidade do *SQL Server 2005* tomando o cuidado para não executar o *Reporting Services Tool* que será configurado de acordo com as exigências do TFS.

Passo 7: em seguida, deve-se instalar o *hotfix NDP20-KB913393-X86* também disponível na mídia de instalação do TFS.

Passo 8: o próximo passo é a instalação do *Windows SharePoint Services* com *Service Pack 2* mostrada na figura 21, sem configurá-lo, pois o TFS irá se encarregar disso.

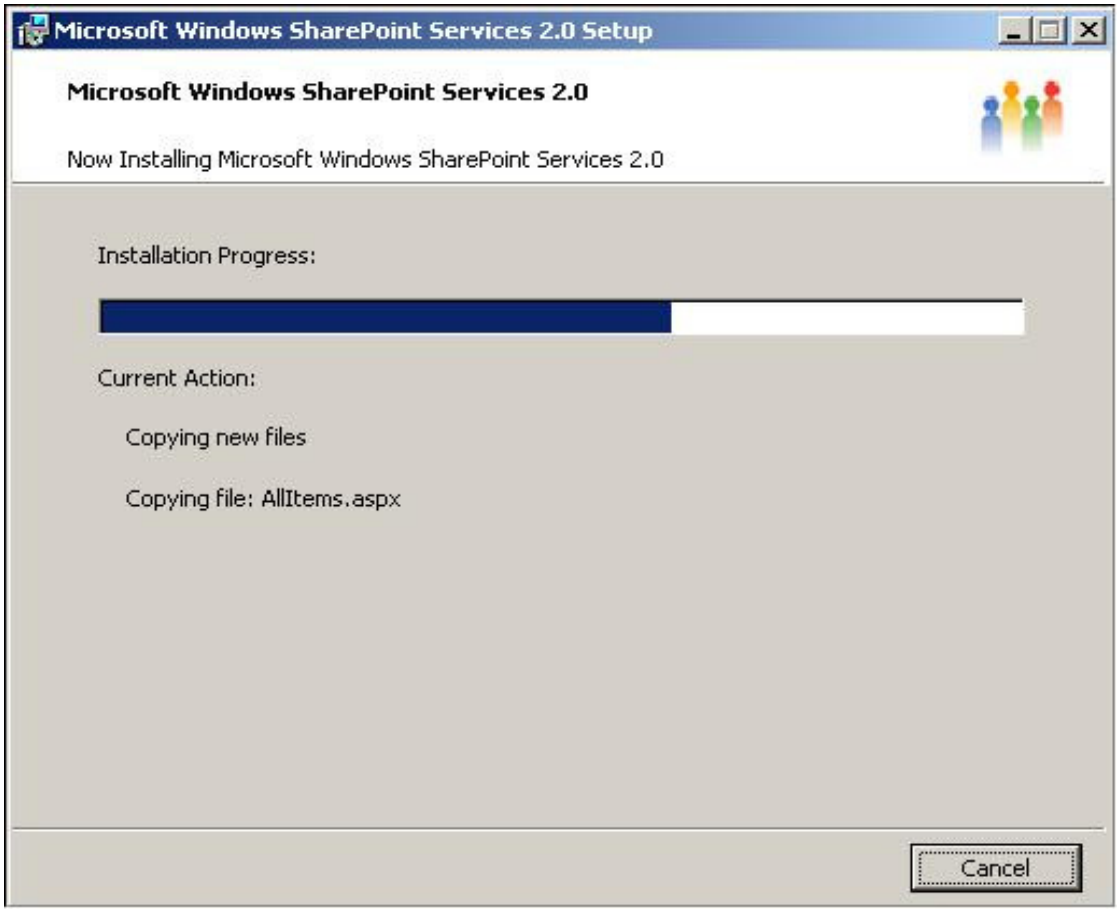

Figura 21. Instalação do *Microsoft SharePoint Services.*
Fonte: MICROSOFT (2008).

O mesmo deve ser instalado como "*Farm*" um detalhe importante que caso não seguido irá impedir a instalação do TFS.

Passo 9: Depois de realizados estes procedimentos deve-se novamente verificar as atualizações do *Windows Update* e verificar se todas as portas de redes necessárias estão abertas para a utilização dos serviços.

Passo 10: O próximo passo é a criação e identificação das contas que serão utilizadas. Devem ser criadas três contas especiais: *TFSSETUP* para instalação do TFS, o *TFSSERVICE* para instalar os serviços e o *TFSREPORTS* para instalar o *SQL Reporting Services*.

Passo 11: O próximo passo é a instalação do *Team Foundation Server*.

Procure no CD do TFS o arquivo instalador ou *autorun.exe*. Após executar o procedimento abrirá uma tela conforme figura 22.

Figura 22. Instalação do *Team Foundation Server*
Fonte: *MICROSOFT* (2008)

Escolha a opção *Install Team Foundation Server* e em seguida opte por *Single-Server Installation*, conforme ilustra a figura 23.

Figura 23. Tipo de Instalação do TFS *Single Server*
Fonte: MICROSOFT (2008)

Na seqüência é realizada a verificação por parte do TFS, resultando um relatório com avisos e problemas encontrados desde a configuração de pré-requisitos. Geralmente os problemas da instalação iniciam neste momento quando os processos não foram seguidos de forma correta. Caso ocorram erros, uma lista de detalhes é disponibilizada para que sejam realizadas as correções, demonstradas na figura 24, durante o processo de instalação realizado.

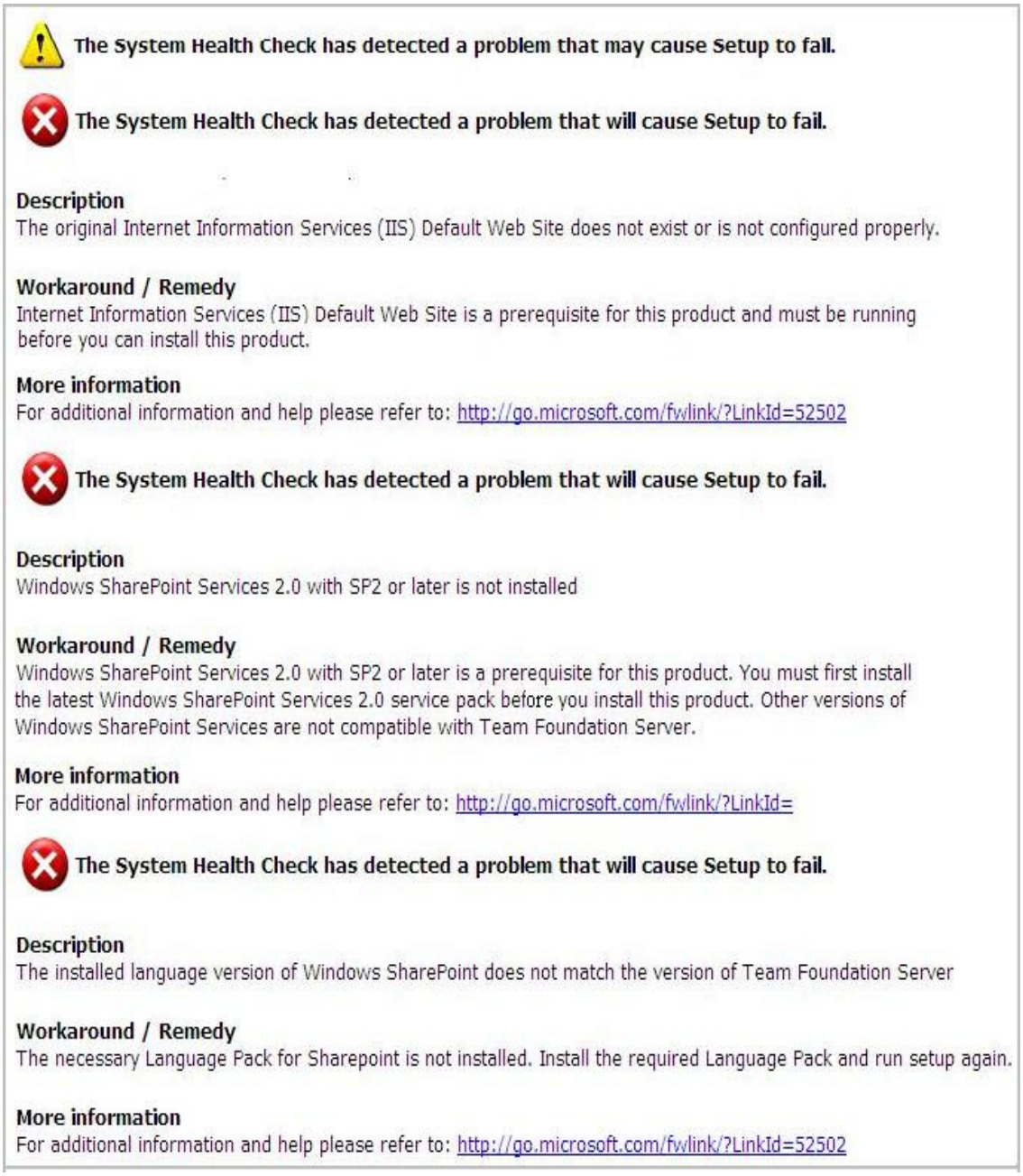

Figura 24. Relatório de erros ocorridos durante a instalação do TFS.
Fonte: MICROSOFT (2008).

Depois de corrigidos os erros, serão solicitados a conta de *logon* para relatórios na qual deve ser informada a conta *TFSREPORTS*. A próxima etapa pode-se caso exista um servidor de SMTP disponível, configurar o "*Team Foundation Alerts*" para que sejam enviados e-mails que fazem parte do *workflow* do TFS.

Para finalizar, após aguardar o final da instalação execute o *GetRegistrationEntries* por meio do seguinte endereço: *http://localhost:8080/services/v1.0/Registration.asmx.*

"A instalação do *Team Foundation Server* é encarada por muitos como o grande problema do TFS" (GARCIA, 2007 p.37). Caso ocorra algum problema durante a instalação do mesmo é porque certamente algumas das etapas ou pré-requisitos não foram cumpridas de forma satisfatória.

Instalação e configuração do Ambiente de desenvolvimento (Cliente)

A instalação do *Visual Studio Team Suíte* requer menos atenção nos pré-requisitos de software quando comparado ao *Team Foundation Server* pelo fato de possuir tais requisitos na mídia de instalação conforme descrita abaixo:

Passo 1: Antes de iniciar a instalação da ferramenta é necessário que se realizem todos os *updates* solicitados pelo *Windows Update* do Sistema Operacional, no caso utilizado o *Windows XP Professional*.

Passo 2: Procure no DVD do *Team Suíte* o arquivo instalador ou *autorun.exe*. Após executar o procedimento abrirá uma tela, descrita na Figura 25.

Figura 25. Instalação do *Visual Team Suíte*
Fonte: MICROSOFT (2008)

Passo 3: Logo após surge uma tela onde é questionado sobre a utilização da ajuda da Microsoft na detecção de *bugs* e possíveis problemas encontrados.

Passo 4: Na tela seguinte ocorre a inspeção dos pré-requisitos necessários antes da instalação. A chave de validação *Trial* do produto é introduzida automaticamente sem a necessidade de informar este campo.

Passo 5: A próxima etapa é a escolha do diretório e tipo da instalação desejada. Conforme a Figura 26 existem três opções de instalação: *Default, Full* e *Custom*. No projeto proposto foi selecionada a opção *Full*, onde são instalados todos os recursos disponíveis.

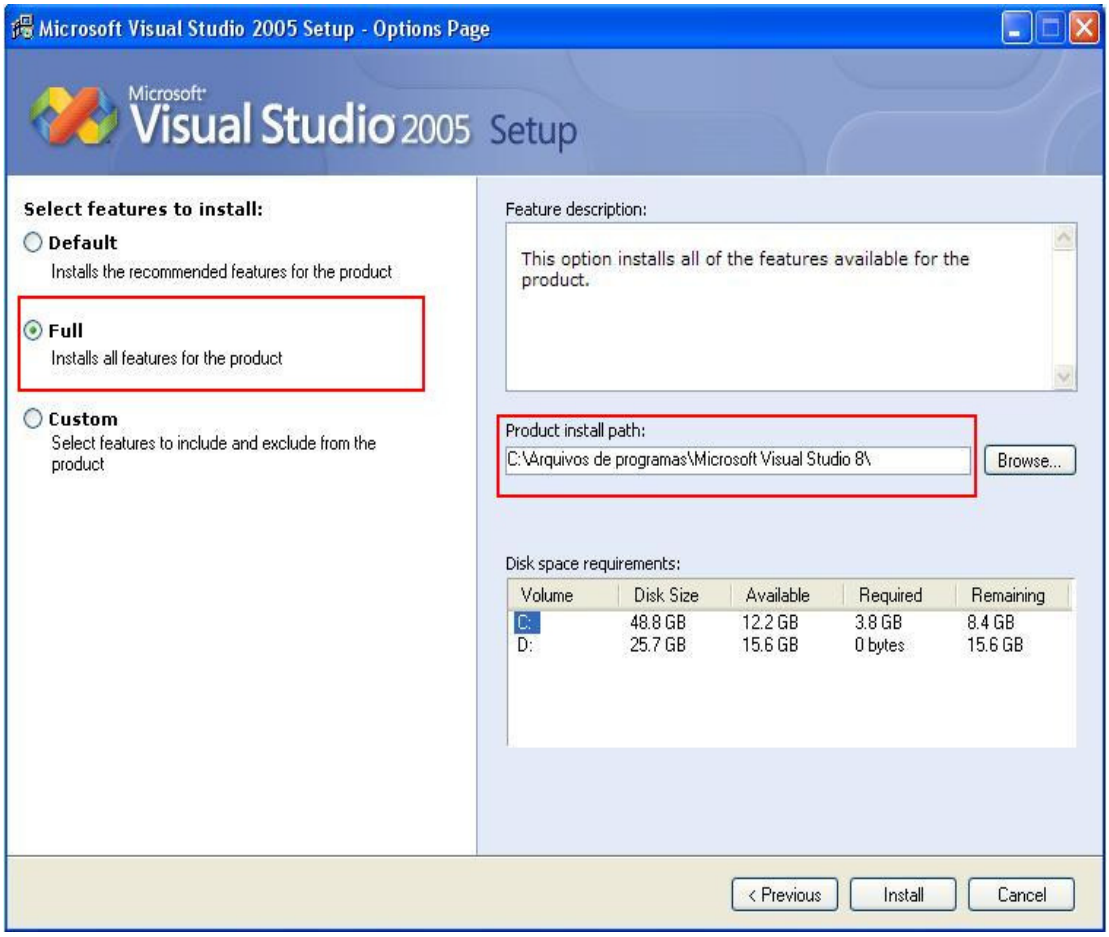

Figura 26. Tipo de instalação do *Visual Team Suíte*
Fonte: MICROSOFT (2008)

Passo 6: após clicar em "*install*" é iniciada a instalação dos componentes do *Visual Studio Team Suíte*. O próximo passo é aguardar o final da instalação. Quando a barra de progresso chegar ao fim deverá aparecer uma tela mostrando o sucesso da instalação, representado na figura 27.

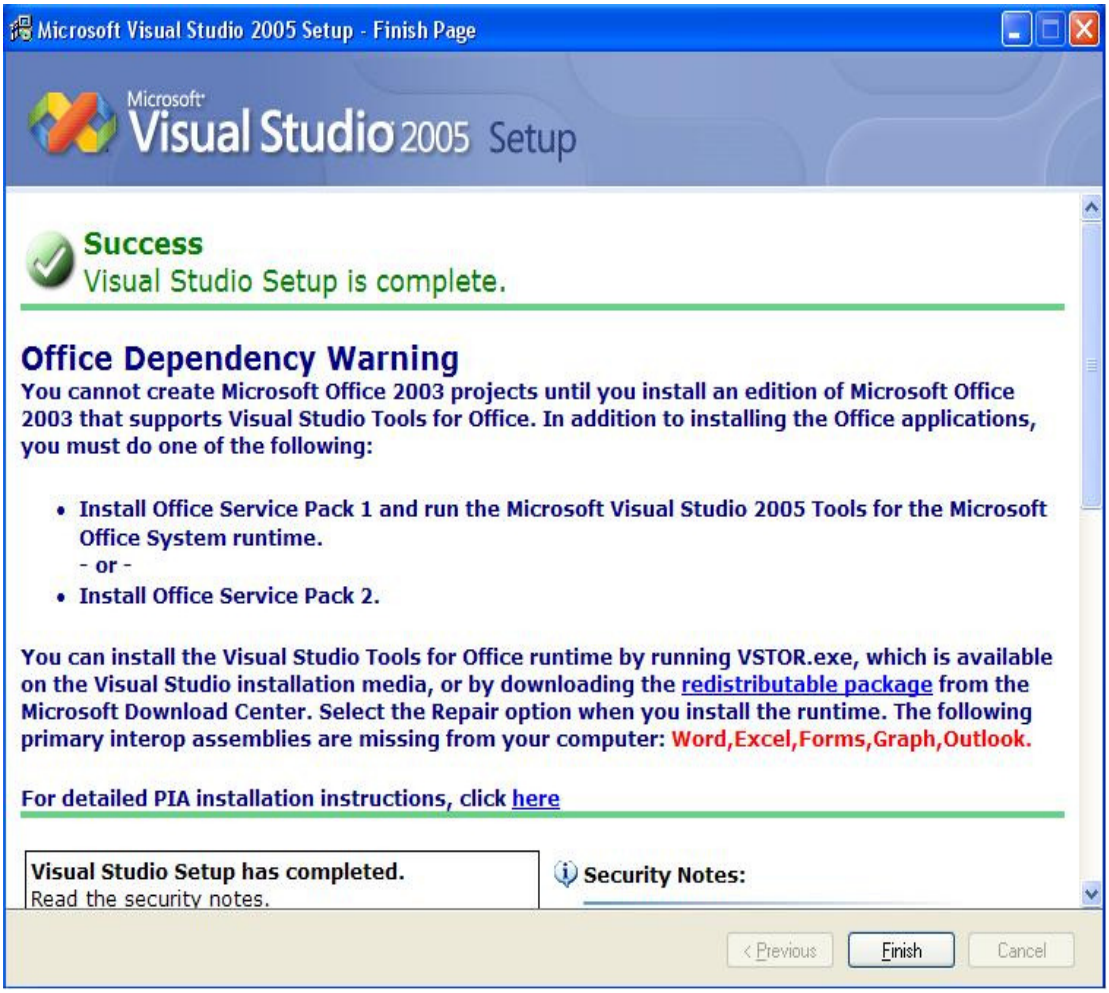

Figura 27. Final da instalação do *Visual Team Suíte*
Fonte: MICROSOFT (2008)

Depois de finalizada a instalação do *Visual Studio Team System* deve-se instalar o *Team Explorer*, ferramenta da qual possibilita a utilização dos recursos oferecidos pelo VSTS. A instalação deve ser realizada em todos os computadores que precisarem de acesso ao TFS, segue os seguintes passos.

Passo 1: acesse o instalador ou *autorun.exe* disponível no CD do TFS.

Passo 2: selecione a opção *Install* do *Team Explorer*.

Passo 3: para finalizar, informe o local de destino da instalação e espere a conclusão do processo.

Para a utilização de uma linguagem de programação ou plataforma da qual não fornece suporte nativo ao TFS pode-se contar com mecanismos de integração. No desenvolvimento desse projeto implementado na linguagem PHP foi instalado o *VS.PHP*, um editor PHP para o *Visual Studio* 2005 desenvolvido pela *Jcx.Software*, uma parceira da Microsoft.

ANEXO A – Estrutura da Ferramenta VSTS

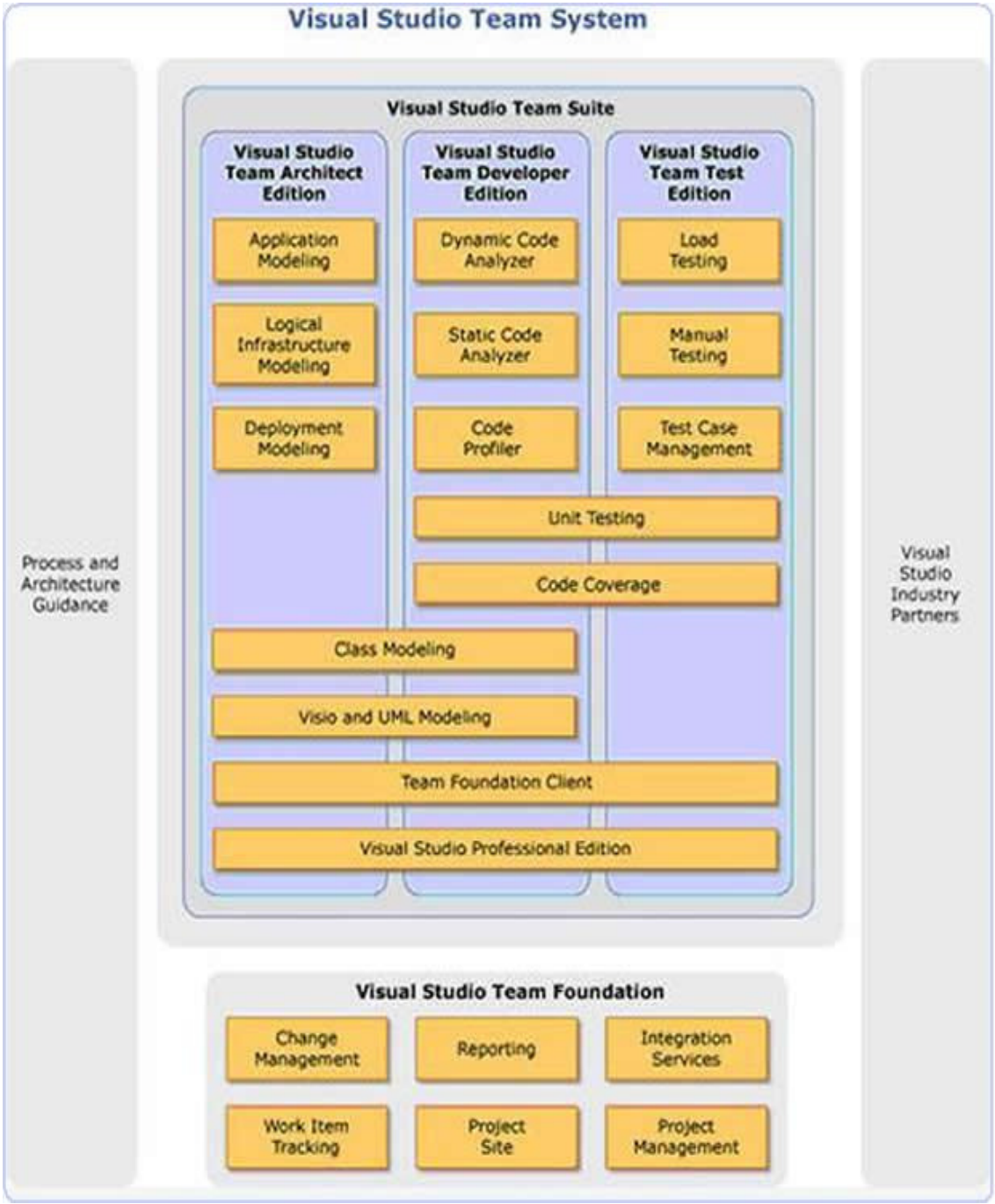

Figura 28. Diagrama da expansão do *Visual Studio*
Fonte: GARCIA, M. (2007).