APOIO À VERIFICAÇÃO DE SOFTWARE EM AMBIENTES DE DESENVOLVIMENTO DE SOFTWARE ORIENTADOS À ORGANIZAÇÃO

Andrea Oliveira Soares Barreto

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

Aprovada por:

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

Prof. Ricardo de Almeida Falbo, D. Sc.

Prof. Geraldo Bonorino Xexéo, D. Sc.

RIO DE JANEIRO, RJ - BRASIL

MARÇO DE 2006

BARRETO, ANDREA OLIVEIRA SOARES

Apoio à Verificação de Software em Ambientes de Desenvolvimento de Software Orientados à Organização [Rio de Janeiro] 2006

XIV, 158 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2006)

Dissertação - Universidade Federal do Rio de Janeiro, COPPE

1. Verificação de Software

2. Processo de Software

3. Qualidade de Software

I. COPPE/UFRJ II. Título (série)

Àquele que me deu a vida e que me ama incondicionalmente; Àquele no Qual sempre posso confiar e que é digno de toda honra, toda glória e todo louvor: A Deus, Único Deus, Grande e Eterno Deus.

AGRADECIMENTOS

A Deus, pelo seu infinito amor e cuidado para comigo, por me capacitar e me ajudar a vencer mais esta etapa da minha vida.

Ao meu querido marido, Ahilton, pela grande ajuda durante a realização deste trabalho, muitas vezes como colega de estudo e trabalho, e inúmeras vezes como meu amigo. Por me acompanhar, estando sempre ao meu lado, me compreender, me ajudar nos momentos difíceis e vibrar comigo nos momentos alegres. Pelo grande amor e carinho a mim dedicados e por fazer parte da minha vida.

Aos meus pais, irmãos e demais familiares e amigos por sempre me incentivarem a estudar, por acreditarem em mim e vibrarem comigo a cada vitória alcançada. Pelo grande amor, carinho e compreensão a mim demonstrados e principalmente por entenderem a minha ausência.

A minha orientadora Ana Regina, pela orientação e incentivo ao longo destes anos, pelo grande aprendizado que me proporcionou e por acreditar em mim. Agradeço também pelo cuidado comigo e com o Ahilton desde que nos mudamos para o Rio de Janeiro, se preocupando conosco, nos dando várias oportunidades de trabalho, nos abrindo portas e nos oferecendo sua amizade.

Ao Prof. Ricardo Falbo, por, desde a graduação, me incentivar e me orientar a fazer o mestrado, pelos ensinamentos que foram muito utilizados neste trabalho e por participar da minha banca.

Ao Prof. Geraldo Xexéo, por participar da minha banca.

Aos demais professores do programa, que me proporcionaram um grande aprendizado ao longo das disciplinas ministradas por eles.

A todos os colegas da COPPE com os quais não só estudei, mas também tive a oportunidade de trabalhar, pelo apoio, pelas idéias e sugestões, pela amizade e convivência. Em especial agradeço àqueles com os quais tive mais contato: Ahilton, Ana Cândida, Paula, Gleison, Mariano, Lúcia, Sômulo, Sávio, David, Adriano, Tayana, Rafael, Arilo, Fábio, Jeann, Reinaldo, Catia, Muradas e Estolano.

A todos os colegas com quem trabalhei durante esses anos, na Embratel, Petrobrás, Relacional, TSE e Synapsis, pelos grandes aprendizados que me

proporcionaram e pela amizade. Em especial, agradeço a Carmen Maidantchick, Renata, Cimar, Mariano, Gleison, Káthia Oliveira, Ahilton, Ana Candida, Paula e David.

À Taisa, Solange, Mercedes, Claudia Prata e demais funcionários do PESC pela ajuda, sempre que necessária.

À CAPES, pelo apoio financeiro.

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

APOIO À VERIFICAÇÃO DE SOFTWARE EM AMBIENTES DE DESENVOLVIMENTO DE SOFTWARE ORIENTADOS À ORGANIZAÇÃO

Andrea Oliveira Soares Barreto

Março / 2006

Orientadora: Ana Regina Cavalcanti da Rocha

Programa: Engenharia de Sistemas e Computação

No contexto do desenvolvimento de software, uma das atividades mais importantes é a verificação efetiva da qualidade do software. Uma das formas de se realizar essa verificação buscando obter produtos de melhor qualidade é introduzir atividades de verificação ao longo do processo de desenvolvimento. Verificar software, em linhas gerais, significa avaliar, ao longo do desenvolvimento, se o produto está sendo desenvolvido adequadamente.

Esta dissertação apresenta a definição de um processo de verificação de software, cujo objetivo é apoiar essa verificação durante todo o processo de desenvolvimento e/ou manutenção, possibilitando a melhoria da qualidade dos produtos. Além disso, foi realizado um mapeamento entre o processo de verificação definido e o processo de desenvolvimento de software. Foi, ainda, definido um conjunto inicial contendo características de qualidade, critérios, questões e métricas para avaliar a satisfação das características, de modo a apoiar a execução da verificação. Com o objetivo de fornecer apoio à execução do processo proposto, uma ferramenta foi desenvolvida - *VerificationManager*. Essa ferramenta está inserida no contexto dos Ambientes de Desenvolvimento de Software Orientados à Organização.

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

SUPPORTING SOFTWARE VERIFICATION IN ENTERPRISE ORIENTED SOFTWARE DEVELOPMENT ENVIRONMENTS

Andrea Oliveira Soares Barreto

March / 2006

Advisor: Ana Regina Cavalcanti da Rocha

Department: System and Computing Engineering

In the context of software development, one of the most important activities is the effective verification of software quality. One approach to perform this verification trying to deliver better products is to include verification activities throughout the software development process. Software verification, in general terms, means to evaluate, throughout development, if the product is being developed adequately.

This dissertation presents the definition of a software verification process. The goal of this process is to support the verification throughout all development and/or maintenance process, making it possible to improve product quality. Moreover, an approach to integrate the activities of the verification process and the activities from the development process are presented. It was also defined an initial set with quality characteristics, criteria, questions and metrics to evaluate the characteristic presence. This knowledge is useful during verification activities execution. A tool, named *VerificationManager*, has been developed to support the execution of the process presented. This tool is part of the Enterprise-Oriented Software Development Environments.

ÍNDICE

Índice de Figuras

Índice de Tabelas

CAPÍTULO 1

Introdução

1.1 Motivação

O papel dos computadores na sociedade, o poder de processamento que eles oferecem e a sua variedade de utilizações em domínios diferentes têm crescido drasticamente. Além disso, a capacidade de se colocar produtos no mercado em um intervalo de tempo cada vez menor tem muitas vezes sido fundamental para o sucesso de uma empresa. Desse modo, existe uma constante pressão para que o software seja produzido de forma mais rápida, com alta qualidade e mantendo um custo adequado (BARRETO e ROCHA, 2005).

A demanda e a preocupação com a produção de software de alta qualidade a baixo custo passaram a ser um dos motivos que culminaram na introdução de atividades agregadas sob o nome de garantia da qualidade de software ao longo de todo o processo de desenvolvimento de software. A qualidade é um aspecto que deve ser tratado simultaneamente com o processo de desenvolvimento, pois ela não pode ser imposta depois que o produto está finalizado (ROCHA *et al.*, 2001). No entanto, de acordo com uma pesquisa realizada sobre a qualidade e produtividade no setor de software brasileiro (MINISTÉRIO DA CIÊNCIA E TECNOLOGIA, 2002), em 2002, somente 25,1% das empresas usavam algum sistema de qualidade.

Avaliações para controle da qualidade de software devem estar presentes ao longo de todo o ciclo de vida com o objetivo de (PRESSMAN, 2001):

- Assegurar que os requisitos estabelecidos podem ser alcançados;
- Identificar os requisitos que não podem ser alcançados;
- Garantir que o software é desenvolvido de forma uniforme;
- Descobrir erros para tomar medidas corretivas o mais cedo possível; e,
- Tornar o projeto mais gerenciável.

Segundo BHATTI e KEPLER (2005), uma série de estudos sugere que o esforço gasto com atividades de garantia da qualidade pode chegar a até 80% do esforço necessário para o desenvolvimento de produtos de software.

Dentre as atividades de garantia de qualidade de software está a verificação, que tem o objetivo de minimizar a ocorrência de erros e riscos associados (ROCHA *et al.*, 2001). Verificar software, em linhas gerais, significa avaliar, ao longo do desenvolvimento, se o produto está sendo desenvolvido adequadamente.

A atividade de verificação não é simples, uma vez que várias avaliações devem ser realizadas ao longo do desenvolvimento e cada avaliação requer planejamento, controle e uso de técnicas de avaliação adequadas.

Uma vez que a verificação tem um papel importante na garantia da qualidade do software e não é uma atividade simples, é interessante que exista algum apoio à execução dessa complexa atividade de forma a reduzir o esforço necessário para sua realização e melhorar a qualidade dos produtos de software desenvolvidos.

Um outro aspecto importante no que diz respeito à verificação de software é o conhecimento para realização da atividade. Conhecimento é, indiscutivelmente, um importante ativo de uma empresa e, por isso, sua formalização, captura e reutilização devem ser sempre incentivados (SANTOS, 2003). O conhecimento para planejar e executar a verificação de software é um exemplo de conhecimento presente em uma organização que desenvolve software.

Para executar a verificação em um contexto onde o conhecimento da organização, e, considerando o foco deste trabalho, em que o conhecimento referente à verificação deve ser gerenciado, é importante que as atividades de verificação sejam executadas de forma a garantir que esse conhecimento seja organizado, armazenado e disponibilizado para a organização. Por isso, é de se esperar que realizar a verificação no contexto de um ambiente que proporcione e facilite a gerência de conhecimento traga benefícios para a realização da atividade e para a organização como um todo.

Ambientes de Desenvolvimento de Software Orientados à Organização se propõem a apoiar as atividades de Engenharia de Software, possibilitando a gerência do conhecimento que pode ser útil aos engenheiros de software ao longo dos projetos de uma organização (VILLELA *et al.*, 2000).

O conceito de Ambientes de Desenvolvimento de Software Orientados à Organização (ADSOrg), em cujo contexto este trabalho se insere, surgiu da necessidade de gerenciar o conhecimento organizacional adquirido ao longo de projetos de software. Esses ambientes apóiam as atividades de Engenharia de Software em uma organização, fornecendo o conhecimento acumulado e relevante para essas atividades e dando apoio ao aprendizado organizacional em Engenharia de Software (VILLELA *et al.*, 2001).

1.2 Objetivo da Dissertação

O principal objetivo desta dissertação é definir e desenvolver uma abordagem para apoiar a execução da verificação de software ao longo do processo de desenvolvimento de software.

Para isso, foi definido um processo de verificação de software com base nos processos existentes na literatura e em estudos referentes às técnicas de engenharia de software para a verificação de software. Foi, também, realizado um mapeamento entre o processo de verificação definido e o processo de desenvolvimento, indicando em que pontos do processo de desenvolvimento as atividades do processo de verificação devem ser inseridas.

Além disso, foi proposto um conjunto inicial de critérios, questões e métricas para verificação de alguns artefatos produzidos ao longo do processo de desenvolvimento, baseando-se em dados da literatura.

Foi, ainda, definida e implementada uma ferramenta de apoio à abordagem proposta – a *VerificationManager*, que faz parte do conjunto de ferramentas disponibilizado em um ADSOrg. A ferramenta *VerificationManager* se propõe a apoiar o processo de verificação de software definido neste trabalho. Através desta ferramenta, é disponibilizado conhecimento com o objetivo de auxiliar na execução das atividades de verificação.

1.3 Estrutura do Texto

Esta dissertação está organizada em seis capítulos, além desta introdução, e dois anexos.

No capítulo 2, *Qualidade de Software*, são apresentados conceitos relacionados à qualidade, normas internacionais de qualidade e o planejamento da qualidade em processos de software.

No capítulo 3, *Verificação de Software*, são apresentados definições e conceitos de verificação de software. Métodos e técnicas de verificação, também, são abordados. São apresentados, ainda, diversos estudos que vêm sendo realizados na área, além de definições de processos de verificação de software encontrados na literatura.

No capítulo 4, *A Estação TABA: seus Ambientes e Ferramentas*, são apresentadas as principais características dos Ambientes de Desenvolvimento de Software Orientados à Organização, a estrutura da Estação TABA, e seu estágio atual de implementação.

No capítulo 5, *Uma Abordagem para Verificação em Projetos de Desenvolvimento de Software*, é apresentada a abordagem para verificação de software definida neste trabalho, descrevendo o processo de verificação de software definido e sua integração ao processo de desenvolvimento de software, além de abordar o conjunto de critérios, questões e métricas para verificação.

No capítulo 6, *Apoio à Verificação de Software na Estação TABA: A Ferramenta VerificationManager*, é apresentada a ferramenta de apoio à execução do processo de verificação de software definido neste trabalho: *VerificationManager*.

No capítulo 7 são apresentadas as conclusões, contribuições do trabalho e perspectivas para trabalhos futuros.

O Anexo 1 apresenta a notação utilizada para a modelagem do processo de verificação de software descrito no capítulo 5.

O Anexo 2 apresenta uma lista de características de qualidade, critérios, questões e métricas para apoiar a execução da verificação de software.

O Anexo 3 apresenta o modelo de classes da ferramenta *VerificationManager*.

CAPÍTULO 2

Qualidade de Software

2.1 Introdução

O principal objetivo da engenharia de software é, sem dúvida, melhorar a qualidade do software. A computação, hoje, é parte importante de quase todos os aspectos da vida humana; o software tem sido fundamental para comércio, transportes, medicina, direito, gerenciamento, educação e outros. Os produtos de software são cada vez maiores e mais complexos, e desenvolvê-los requer grandes quantidades de recursos, tais como pessoas, dinheiro e tempo (ROCHA, *et al.*, 2001; HILBURN e TOWHIDNEJAD, 2000). Apesar de alguns sucessos espetaculares e da aceitação geral do software como parte da vida diária, ainda há muito o que aprimorar na qualidade do software que produzimos (PFLEEGER, 2004).

Para produzir um software de boa qualidade é preciso entender o que é qualidade de software. Existem diversos pontos de vista diferentes acerca do significado da qualidade. Uma das mais importantes e referenciadas definições para qualidade é a da norma NBR ISO 9000 (2000) que define qualidade como a totalidade de características de uma entidade que lhe confere a capacidade de satisfazer as necessidades explícitas e implícitas. Para PRESSMAN (2001), qualidade de software é a conformidade com requisitos funcionais e de desempenho explicitamente enunciados, padrões de desenvolvimento documentados e características implícitas que se espera de qualquer software desenvolvido de forma profissional. ROCHA (1987) já sintetizava o conceito de qualidade de software como sendo um conjunto de propriedades a serem satisfeitas em determinado grau, de modo que o software satisfaça às necessidades de seus usuários.

PFLEEGER (2004) descreve a qualidade considerando cinco perspectivas diferentes, com base nas diferentes percepções de qualidade das pessoas:

- A visão transcendental, onde a qualidade é algo que podemos reconhecer, mas não podemos definir;
- A visão do usuário, onde a qualidade é a adequação ao propósito pretendido;
- A visão do fabricante, onde a qualidade é a conformidade com a especificação;

- A visão do produto, onde a qualidade está relacionada às características inerentes ao produto; e,
- A visão do mercado, onde a qualidade depende do valor que os consumidores estão dispostos a pagar pelo produto.

Apesar de muito importante, garantir a qualidade é uma tarefa custosa para o desenvolvimento de software. Estima-se que 50% dos custos do desenvolvimento são gastos com uma atividade específica de garantia da qualidade, o teste (WAGNER e SEIFERT, 2005). A falta de qualidade também pode custar caro; quanto mais tempo um defeito permanece sem ser detectado, mais cara será a sua correção (PFLEEGER, 2004).

Há alguns anos atrás, CROSBY (1988) já dizia que é mais barato construir um produto correto desde o início, sugerindo até que "a qualidade é gratuita" (*Quality is Free*). Considerando o valor investido na qualidade de um produto e os custos para corrigir os problemas detectados, SPEHAR, em concordância com CROSBY, também sugere que a qualidade é gratuita. Ele considera que os custos da qualidade são importantes, pois o valor de cada hora de trabalho não gasta para corrigir problemas (re-trabalho) pode ser usado para desenvolver produtos melhores e em menor tempo ou para melhorar a qualidade dos produtos e processos já existentes (SPEHAR, 2005). Portanto, a qualidade pode se tornar gratuita, pois, ao se construir produtos de má qualidade pode-se gastar o mesmo valor, ou até mais que o valor relativo aos custos da qualidade, para se corrigir os defeitos inseridos no produto, além do prejuízo oriundo da entrega de um produto sem qualidade. Por outro lado, se a qualidade do produto é uma preocupação e um investimento é feito para que o produto tenha qualidade satisfatória, gasta-se com a qualidade, porém o custo da correção de defeitos é bem menor, uma vez que o produto foi desenvolvido com boa qualidade. Estudos mostram que o investimento na qualidade traz um ganho de produtividade, além de uma redução dos custos e do tempo de entrega do produto (GALIN e AVRAHAMI, 2005; PFLEEGER, 2004; QUAQUARELLI, 1997).

Quanto melhor for a qualidade de um produto no sentido de atender as necessidades dos clientes, espera-se ganhar em: maior grau de satisfação dos clientes, aumento das vendas, condições favoráveis para enfrentar a concorrência e aumento da participação da empresa no mercado. Portanto, a qualidade de um produto tem influência direta em sua venda. Quanto menos deficiências o produto apresentar, mais alta será sua

qualidade. Isso permite que se reduzam os índices de erro, diminua-se o re-trabalho e o desperdício, reduzam-se as falhas e os custos no uso e na garantia deste produto, e, aumente-se a satisfação dos clientes (D'OLIVEIRA, 2003). Já em 1994, estes benefícios obtidos através da melhoria da qualidade de um produto eram apresentados como uma reação em cadeia conforme ilustrado na figura 2.1.

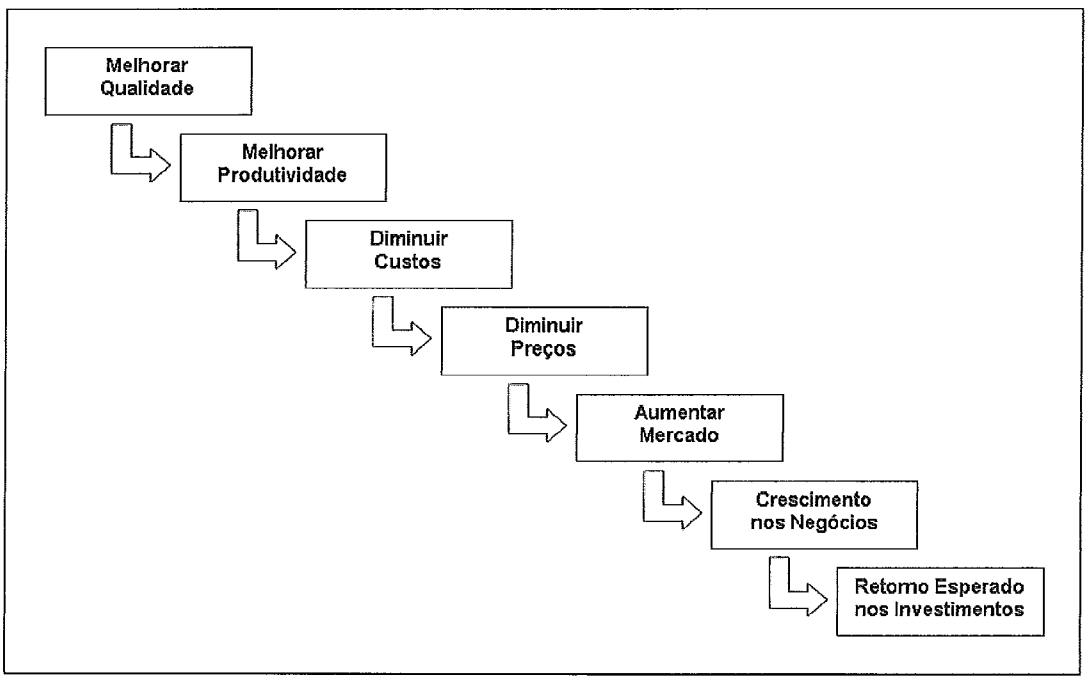


Figura 2.1 – Melhoria da qualidade como uma reação em cadeia (SANDERS e CURRAN, 1994)

A qualidade de produtos de software está fortemente relacionada à qualidade do processo de software (FUGGETTA, 2000). Para muitos engenheiros de software, a qualidade do processo de software é tão importante quanto a qualidade do produto (ROCHA, *et al.*, 2001). Considerando a importância da qualidade do processo para a qualidade do produto final, é de se esperar que melhorar tanto a qualidade do processo quanto a qualidade do produto seja um grande passo para a qualidade de software.

Neste capítulo, são apresentados aspectos relacionados à qualidade de software. A seguir são discutidas a qualidade do processo na seção 2.2 e a qualidade do produto na seção 2.3. A seção 2.4 aborda a medição de software e a seção 2.5 apresenta a garantia da qualidade nas normas e modelos de maturidade. Por fim, na seção 2.6 são apresentadas as considerações finais deste capítulo.

2.2 Qualidade do Processo

Há muitas atividades que afetam a qualidade final do produto. Se alguma dessas atividades não for bem realizada, a qualidade pode sofrer conseqüências (PFLEEGER, 2004). Freqüentemente, os desenvolvedores trabalham de forma desestruturada e estressante, resultando em um nível de qualidade baixo ou desconhecido (SOLINGEN e BERGHOUT, 1999). Melhorar a qualidade de um processo de desenvolvimento de software significa melhorar a forma como os produtos são desenvolvidos, pois, se o desenvolvimento é feito de forma organizada, disciplinada e seguindo as boas práticas de engenharia de software, é de se esperar que os produtos desenvolvidos possuam uma boa qualidade. Além de se produzir produtos de melhor qualidade, ao se adotar um processo é possível repetir a qualidade dos produtos desenvolvidos, aumentando a satisfação dos usuários e clientes que têm a sua disposição produtos de boa qualidade. Portanto, a qualidade do processo influencia a qualidade dos produtos de software, afetando a qualidade em uso percebida pelo cliente (SWEBOK, 2004).

Iniciativas de melhoria de processo de software surgiram como uma das principais abordagens para melhorar a qualidade e a produtividade em organizações de desenvolvimento de software (NGWENYAMA e NIELSEN, 2003). Alguns trabalhos relacionados à melhoria de processo de software resultaram em modelos e normas. Podem ser citados como exemplos, a norma ISO/IEC 12207 (1998), o modelo CMMI (SEI, 2002) e o modelo MPS.BR (MPS.BR, 2005) que serão apresentados nas próximas seções.

2.2.1 A norma ISO/IEC 12207

A norma internacional ISO/IEC 12207 (1998) tem por objetivo auxiliar os envolvidos com a produção de software na definição de seus papéis, através de processos bem definidos, e desta forma proporcionar às organizações que a utilizam um melhor entendimento das atividades a serem executadas nas operações que envolvem, de alguma forma, o software (ROCHA *et al.*, 2001).

A norma ISO/IEC 12207, em sua primeira versão, está dividida em conjuntos de processos. Cada processo é definido em termos de suas próprias atividades e cada atividade é adicionalmente definida em termos de suas tarefas. Uma organização pode selecionar um subconjunto apropriado de processos, de acordo com seus objetivos, pois

a norma foi projetada para ser adaptada para uma organização, projeto ou aplicação específicos (BERGER, 2003). Os processos estão divididos em:

- *Processos Fundamentais*: Aquisição, Fornecimento, Desenvolvimento, Operação e Manutenção;
- *Processos de Apoio*: Documentação, Gerência de Configuração, Garantia da Qualidade, Verificação, Validação, Revisão Conjunta, Auditoria e Resolução de Problema; e,
- *Processos Organizacionais*: Gerência, Infra-estrutura, Melhoria e Treinamento.

Esta norma define, ainda, um processo de adaptação para os processos definidos por ela. Duas evoluções desta norma foram elaboradas até o momento. A primeira em 2002 (ISO/IEC, 2002) e a segunda em 2004 (ISO/IEC, 2004). Essas revisões foram definidas devido à evolução natural da Engenharia de Software ao longo do tempo, ao retorno de seus usuários e, também, à necessidade de adequação à norma ISO/IEC 15504 (ISO/IEC, 2003) que trata da avaliação de processos de software. Como resultado destas duas revisões, novos processos foram definidos (Gerência de Ativos, Gerência de Pedidos de Mudança, Engenharia de Domínio, Recursos Humanos, Avaliação de Produto, Gerência de Programa de Reuso, Usabilidade), outros foram expandidos e para cada processo foram definidos um propósito e uma lista de resultados esperados.

No Brasil, na pesquisa de Qualidade e Produtividade no Setor de Software Brasileiro (MINISTÉRIO DA CIÊNCIA E TECNOLOGIA, 2002), que é realizada normalmente a cada dois anos pelo Ministério da Ciência e Tecnologia, foi constatado que, até 2002, 32,7% das empresas não conheciam a norma ISO/IEC 12207, 8,3% estavam começando a usá-la e apenas 3,9% a usavam sistematicamente. Percebe-se que o uso da norma ainda era bastante pequeno e que ainda havia muito a melhorar.

2.2.2 O Modelo de Maturidade e Capacidade CMMI

Uma outra iniciativa com o objetivo de melhorar os processos de software foi a definição do modelo de maturidade e capacidade CMMI (*Capability Maturity Model Integration*) (SEI, 2002) definido pelo SEI (*Software Engineering Institute*) que consiste de boas práticas que tratam do desenvolvimento e manutenção de produtos e serviços cobrindo o ciclo de vida de um produto desde a concepção até a entrega e a manutenção. O modelo é composto de 24 áreas de processo. Cada área contém práticas

relacionadas que, quando implementadas, coletivamente satisfazem um conjunto de objetivos considerados importantes para fazer melhorias significativas na área em questão (CHRISSIS *et al.*, 2003).

Existem dois tipos de representação no CMMI: contínua e em estágios. A representação contínua usa níveis de capacidade para caracterizar a melhoria relacionada a uma área de processo específica definindo seis níveis de capacidade: Incompleto (0), Desempenhado (1), Gerenciado (2), Definido (3), Gerenciado Quantitativamente (4) e Otimizado (5). Esta representação permite que uma organização selecione uma área de processo específica e melhore com relação a esta área. A representação em estágios define um grupo de áreas de processo para definir uma forma de melhoria para a unidade organizacional, descrito em termos de níveis de maturidade. Nessa representação são definidos os seguintes níveis de maturidade: Inicial (1), Gerenciado (2), Definido (3), Gerenciado Quantitativamente (4) e Otimizado (5). Vale ressaltar que o CMMI define uma área de processo específica para a Garantia da Qualidade do Processo e do Produto, além das áreas Verificação e Validação que também se preocupam com a qualidade do produto de software.

O SEI (SEI, 2002) disponibiliza periodicamente resultados de desempenho relacionados à implantação do CMMI. Esses resultados se baseiam no relato de empresas que adotaram o modelo. Os últimos resultados disponíveis são de dezembro de 2005 e são baseados em informações disponibilizadas por 25 empresas. A tabela 2.1 mostra o percentual médio de melhoria obtida por essas empresas em relação aos seis fatores avaliados na última pesquisa.

Tabela 2.1 – Percentual médio de melhoria obtida por empresas que adotaram o CMMI (SEI, 2002)

Fator	Percentual médio de melhoria
Custos	20%
Tempo do Projeto	37%
Produtividade	62%
Qualidade	50%
Satisfação dos Clientes	14%
Retorno dos Investimentos	4.7 : 1

2.2.3 O Modelo de Referência MPS.BR

Recentemente foi definida uma abordagem para a melhoria da qualidade de software no Brasil: o Modelo de Referência para Melhoria de Processo do Software Brasileiro (MPS.BR, 2005). O MPS.BR tem como objetivo definir um modelo de melhoria e avaliação de processo de software, preferencialmente para as micro, pequenas e médias empresas, de forma a atender as suas necessidades de negócio e a ser reconhecido nacional e internacionalmente como um modelo aplicável à indústria de software. Este é o motivo pelo qual ele está aderente a modelos e normas internacionais. O MPS.BR também define regras para sua implementação e avaliação, dando sustentação e garantia de que está sendo empregado de forma coerente com as suas definições.

O MPS.BR contém os requisitos que as organizações deverão atender e as definições dos processos, dos níveis de maturidade e da capacidade de processos, tendo sido baseado nas normas ISO/IEC 12207, ISO/IEC 15504 (2003) e sendo aderente ao CMMI. Ele define sete níveis de maturidade: Parcialmente Gerenciado (G), Gerenciado (F), Parcialmente Definido(E), Largamente Definido(D), Definido(C), Gerenciado Quantitativamente(B) e Em Otimização(A). A divisão em estágios, embora baseada nos níveis de maturidade do CMMI, tem uma graduação diferente, com o objetivo de possibilitar uma implementação e avaliação mais gradual e adequada às pequenas e médias empresas. A possibilidade de se realizar avaliações considerando mais níveis permite uma visibilidade dos resultados de melhoria de processos com prazos mais curtos. Vale ressaltar que o MPS.BR define um processo específico para a Garantia da Qualidade do Processo e do Produto, além dos processos Verificação e Validação que também se preocupam com a qualidade do produto de software.

Apesar de a primeira avaliação MPS.BR ter sido realizada em setembro de 2005, até janeiro de 2006, cinco empresas já haviam sido avaliadas com sucesso pelo MPS.BR. Dentre as cinco empresas, uma foi avaliada no nível G, em Pernambuco, três no nível F, sendo uma no estado do Rio de Janeiro e duas no estado de São Paulo e uma no nível E também no estado do Rio de Janeiro (MPS.BR, 2005). Além destas avaliações já concluídas, outras avaliações estão em andamento (inclusive aspirando ao nível D) ou serão iniciadas em breve.

Portanto, há várias abordagens que visam a melhorar a qualidade do processo de software e que, além de proporcionar melhor qualidade aos produtos desenvolvidos, podem trazer muitos outros benefícios às organizações. Algumas empresas,

visualizando tais benefícios, têm investido na melhoria da qualidade de seus processos, mas ainda há muito a ser feito para que tenhamos uma indústria de software de alta qualidade no Brasil.

2.3 Qualidade do Produto

Enquanto a qualidade do processo se preocupa em melhorar a forma como um produto é desenvolvido, a qualidade do produto visa a acompanhar o produto em desenvolvimento, avaliando a qualidade dos produtos intermediários gerados desde o início do desenvolvimento até a conclusão do produto final.

A qualidade desejável em um produto de software está relacionada aos requisitos identificados pelo cliente que podem ser tanto os requisitos funcionais quanto os requisitos não funcionais. Os requisitos funcionais são aqueles relacionados às funcionalidades que o software deve oferecer, enquanto os não-funcionais dizem respeito ao comportamento que o software deve apresentar em relação a um conjunto de características estabelecidas para o produto, tais como, facilidade de uso, desempenho, adequação a normas e padrões, interação com outros produtos, dentre outros. Portanto, a qualidade necessária a um produto de software pode ser especificada através de requisitos, os requisitos de qualidade.

Ao se desenvolver um produto com a qualidade desejada pelo usuário, é de esperar que a satisfação do usuário seja obtida. Tal satisfação freqüentemente é considerada um resultado fundamental da busca pela qualidade e alguns estudos mostram que ela tem um impacto positivo nos custos, lucros e crescimento de vendas na organização (MADU *et al.*, 1996). Além disso, em algumas organizações, a qualidade de um produto de software tem sido usada como base para muitas decisões importantes, incluindo a melhoria da qualidade do produto, a realização de grandes aquisições e a monitoração de contratos (JUNG *et al.*, 2004).

No entanto, identificar os requisitos de qualidade de um produto não é uma tarefa trivial. Para auxiliar nesta tarefa, pode-se descrever a qualidade de um produto através de um conjunto de características que devem ser alcançadas em um determinado grau para que o produto atenda às necessidades de seus usuários. Cada uma das características de qualidade pode ser detalhada em vários níveis de sub-características, chegando-se a um amplo conjunto de atributos que descrevem a qualidade de um produto de software.

Com o objetivo de facilitar a definição e avaliação da qualidade de um produto através de características de qualidade, vários modelos de qualidade foram definidos e diferenciam-se entre si com relação às características modeladas, ao relacionamento e à dependência entre elas. Os primeiros trabalhos nessa área são os de BOEHM (1978) e McCALL (1977), cujos modelos apresentam uma hierarquia de características, em que cada uma delas contribui para a qualidade como um todo. Diversos outros modelos foram definidos ao longo dos anos, cada um com suas particularidades (STYLIANOU e KUMAR, 2000; STRONG *et al.*, 1997; DROMEY, 1996; BLASCHEK, 1995; CAMPOS, 1994; BELCHIOR, 1992; BAHIA, 1992; ROCHA e PALERMO, 1989; ROCHA, 1983).

Para consolidar as diferentes visões da qualidade em um modelo, a norma ISO/IEC 9126 foi definida. Antes desta norma, em 1998 já havia sido definida a norma internacional ISO/IEC 14598 (ISO/IEC, 1998) que trata da avaliação de produtos de software. A seção 2.3.1 apresenta a norma ISO/IEC 9126 enquanto a seção 2.3.2 descreve a norma ISO/IEC 14598.

2.3.1 A norma ISO/IEC 9126

A norma internacional ISO/IEC 9126 (ISO/IEC, 2001) define um modelo de qualidade que organiza as características e sub-características de qualidade desejáveis para um produto de software. Ela pode ser usada para especificar e avaliar a qualidade do produto, permitindo validar a completeza da definição dos requisitos, identificar os requisitos de software, identificar os objetivos do projeto de software, identificar os objetivos do teste de software, identificar os critérios de garantia da qualidade e identificar os critérios de aceitação para um produto de software completo.

Essa norma visa a apoiar a avaliação dos produtos de software de forma que o produto avaliado atenda às necessidades específicas em um determinado contexto de uso, conforme mostra a figura 2.2.

O modelo de qualidade definido na norma ISO/IEC 9126 (2001) está subdivido em:

- Modelo de qualidade para características internas e externas, e,
- Modelo de qualidade para qualidade em uso.

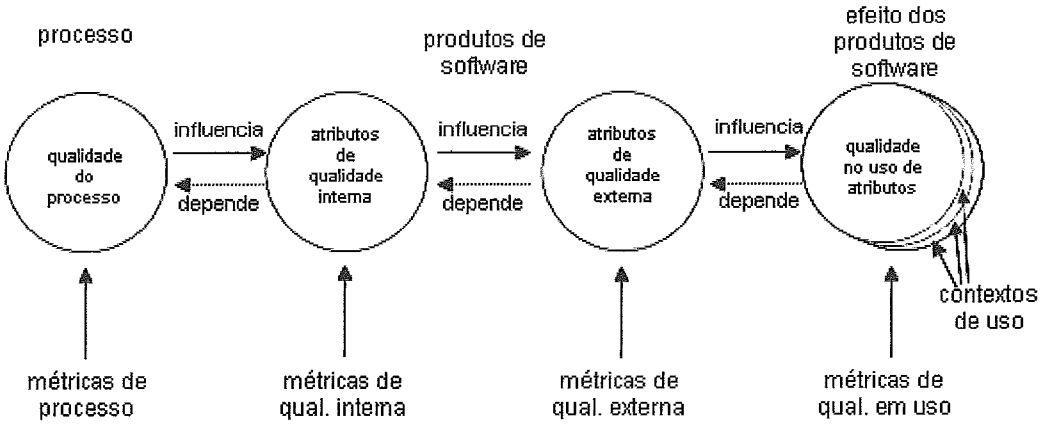


Figura 2.2 – Qualidade no ciclo de vida do software (ISO/IEC 9126, 2001)

A qualidade interna é a totalidade de características do produto de software vista por uma perspectiva interna (dos desenvolvedores). Já a qualidade externa é a totalidade de características observadas por uma perspectiva externa (dos avaliadores ou usuários) e pode ser vista quando o produto é executado. Por fim, a qualidade em uso é a visão de qualidade pela perspectiva de uso (dos usuários) do produto de software. Essa última mede o nível de sucesso na realização de tarefas pelos usuários em um ambiente particular de operação, ao invés de medir o produto de software em si.

O modelo de qualidade para características internas e externas classifica os atributos de qualidade em seis características:

- *Funcionalidade*: refere-se à existência de um conjunto de funções que satisfazem às necessidades implícitas ou explícitas e suas propriedades específicas;
- *Confiabilidade*: refere-se à capacidade de o software manter seu nível de desempenho, sob condições estabelecidas, por um período de tempo;
- *Usabilidade*: refere-se ao esforço necessário para usar um produto de software, bem como o julgamento individual de tal uso por um conjunto explícito ou implícito de usuários;
- *Eficiência*: refere-se ao relacionamento entre o nível de desempenho do software e a quantidade dos recursos utilizados sob as condições estabelecidas;
- *Manutenibilidade*: refere-se ao esforço necessário para fazer modificações específicas no software; e,

- *Portabilidade*: refere-se à capacidade de o software ser transferido de um ambiente para outro.

Para cada uma dessas características são definidas algumas sub-características que estão descritas no Anexo II.

A qualidade em uso também pode ser vista como a capacidade de o produto de software permitir que determinados usuários alcancem metas especificadas como efetividade, produtividade, segurança e satisfação em um contexto especificado. No modelo de qualidade em uso, os atributos são classificados em quatro características:

- *Efetividade*: refere-se à capacidade de o produto de software possibilitar aos usuários atingir metas especificadas com acurácia e completeza em um contexto de uso especificado;
- *Produtividade*: refere-se à capacidade de o produto de software possibilitar aos usuários utilizar uma quantidade adequada de recursos em relação à efetividade alcançada em um contexto de uso especificado;
- *Segurança*: refere-se à capacidade de o produto de software oferecer níveis aceitáveis de risco de danos a pessoas, negócios, software, propriedades ou ao ambiente em um contexto de uso especificado; e,
- *Satisfação*: refere-se à capacidade de o produto de software satisfazer aos usuários em um contexto de uso especificado.

Segundo pesquisa de Qualidade e Produtividade no Setor de Software Brasileiro (MINISTÉRIO DA CIÊNCIA E TECNOLOGIA, 2002), até 2002, no Brasil, 34,2% das empresas não conheciam a norma ISO/IEC 9126, 54,4% conheciam a norma mas não a usavam, 7,5% estavam começando a usá-la e apenas 3,9% a usavam sistematicamente.

2.3.2 A norma ISO/IEC 14598

A norma internacional ISO/IEC 14598 (ISO/IEC, 1998) define uma estratégia para avaliação de produtos de software e deve ser usada em conjunto com a norma ISO/IEC 9126. Ela está dividida em seis partes: 1 - Visão Geral, 2 - Planejamento e Gestão, 3 - Processo para Desenvolvedores, 4 - Processo para Adquirentes, 5 - Processo para Avaliadores e 6 - Documentação de Módulos de avaliação. Dentre essas partes, a parte 5 - Processo para Avaliadores - merece destaque, pois fornece requisitos e recomendações

para implementação da avaliação do produto de software, quando as partes envolvidas precisam entender, aceitar e confiar nos resultados da avaliação. O principal objetivo desse processo de avaliação é promover as seguintes características:

- *Repetição*: a avaliação do mesmo produto realizada pelo mesmo avaliador sob a mesma especificação de avaliação deve gerar resultados considerados idênticos;
- *Reprodução*: a avaliação do mesmo produto sob a mesma especificação de avaliação, porém realizada por outro avaliador, deve gerar resultados considerados idênticos;
- *Imparcialidade*: a avaliação não deve se basear em resultados particulares; e,
- *Objetividade*: os resultados da avaliação devem ser factuais, não influenciados pelos sentimentos ou opiniões do avaliador.

Até 2002, no Brasil, 39,8% das empresas não conheciam a norma ISO/IEC 14598, 55,6% conheciam a norma mas não a usavam, 3,4% estavam começando a usá-la e apenas 1,2% a usavam sistematicamente, conforme constatou a pesquisa de Qualidade e Produtividade no Setor de Software Brasileiro (MINISTÉRIO DA CIÊNCIA E TECNOLOGIA, 2002).

Para avaliar a qualidade de um produto de software, é importante que, além de se ter um processo de avaliação definido (como descreve esta norma), sejam definidos também critérios para a avaliação. A norma internacional ISO/IEC 12207 (1998) e o modelo CMMI (SEI, 2002) definem que para avaliar um determinado artefato ou produto intermediário do desenvolvimento é preciso definir um conjunto de critérios, em relação aos quais, o artefato será avaliado. A seção a seguir apresenta alguns critérios para avaliação de produtos de software.

2.3.3 Critérios para Avaliação da Qualidade de Software

A avaliação da qualidade de um produto de software deve ser baseada nos requisitos de qualidade especificados para o produto. No entanto, avaliar a qualidade a partir dos requisitos pode não ser tão simples. Com o objetivo de auxiliar nesta avaliação, alguns critérios que atendam aos requisitos especificados podem ser identificados. Tais critérios devem ser avaliados e, se satisfeitos, indicam o atendimento aos requisitos especificados.

A norma internacional ISO/IEC 12207 (1998) define um conjunto de critérios para avaliação de determinados artefatos. A tabela 2.3 exemplifica critérios definidos nessa norma.

Tabela 2.2 – Exemplo de critérios para avaliação definidos norma ISO/IEC 12207 (1998)

Artefato	Critério
Requisitos do software	Rastreabilidade para os requisitos do sistema e o projeto do sistema
	Consistência externa com os requisitos do sistema
	Consistência interna
	Viabilidade de testes
	Viabilidade do projeto do software
	Viabilidade da operação e da manutenção
Código do software e os resultados dos testes	Rastreabilidade para os requisitos e projeto do item de software
	Consistência externa com os requisitos e projeto do item de software
	Consistência interna entre os requisitos da unidade
	Cobertura de teste das unidades
	Adequação dos métodos e padrões de codificação utilizados
	Viabilidade de integração e testes do software
	Viabilidade da operação e da manutenção

Devido à importância dos critérios em uma avaliação e aos benefícios advindos do uso destes, ao longo dos anos vários trabalhos foram realizados com o intuito de definir critérios adequados à avaliação de artefatos produzidos ao longo do desenvolvimento de software:

- ACKERMAN (1989) apresenta os critérios *completeza*, *ambigüidade*, e *consistência* como relevantes para avaliação de requisitos.
- O SPAWAR (*Space and Naval Warfare Systems Center*) (1997) define um conjunto de critérios para avaliação de diversos artefatos, dentre eles:
    - *Plano de projeto:* clareza, completeza, nível de detalhe e correção, manutenibilidade;
    - *Requisitos do sistema*: clareza, completeza, conformidade, consistência, funcionalidade, manutenibilidade, desempenho, confiabilidade, viabilidade de testes e rastreabilidade;
    - *Requisitos do software*: clareza, completeza, conformidade, consistência, uso de dados, funcionalidade, nível de detalhe, manutenibilidade, desempenho, confiabilidade, viabilidade de testes e rastreabilidade; e,
    - *Projeto de alto nível do software e Projeto detalhado do software*: clareza, completeza, conformidade, consistência, correção, uso de

dados, funcionalidade, nível de detalhes, manutenibilidade, desempenho, confiabilidade, viabilidade de testes e rastreabilidade.

- BEAUFOND *et al.* (1997) apresentam uma lista extensa de critérios para avaliação de especificações orientadas a objetos. Os critérios estão organizados segundo os objetivos de qualidade *Utilizabilidade, Confiabilidade Conceitual* e *Confiabilidade da Representação* e são associados a diversos fatores e subfatores de qualidade.
- O CMMI (SEI, 2002) apresenta alguns critérios para avaliação de projeto de software e código. Para o projeto são apresentados os seguintes critérios: *modularidade, clareza, simplicidade, manutenibilidade, testabilidade, portabilidade, confiabilidade, precisão, segurança, escalabilidade* e *usabilidade*. Para o código são apresentados os seguintes critérios: *modularidade, clareza, simplicidade* e *manutenibilidade*.

Para auxiliar na determinação da satisfação dos critérios, alguns autores definem questões (*checklist*) que devem ser respondidas, ajudando na avaliação. *Checklists* são desenvolvidos e revisados com base na experiência acumulada pela equipe (KAN, 2003). Em (BEAUFOND *et al.*, 1997) são apresentadas algumas questões para cada critério definido. A NASA (*National Aeronautics and Space Administration*) (1993) apresenta um *checklist* extenso para avaliação de diversos artefatos entre eles requisitos, projeto, código em várias linguagens de programação e casos de teste. O SPAWAR (*Space and Naval Warfare Systems Center*) (1997) define várias questões para avaliação de cada critério definido por ele.

Outros *checklists* definidos para a avaliação de qualidade de software podem ser encontrados em (McCONNELL, 2004; MYERS, 2004; RAKITIN, 2001; HUMPHREY, 1997; PORTER *et al.*, 1995, MAGUIRE, 1993; ACKERMAN, 1989; HUMPHREY, 1989; DUNN, 1984; FREEDMAN e WEINBERG, 1982). A tabela 2.4 apresenta um exemplo de *checklist* definido em (NASA, 1993).

Tabela 2.3 – Exemplo de *checklist* para avaliação segundo determinados critérios (NASA, 1993)

Artefato	Critério	Questões
Requisitos do software	Completeza	Todos os requisitos, restrições e suposições do software estão identificados?
		Todos os requisitos e restrições estão priorizados?
	Consistência Externa	Os requisitos do software são consistentes com os requisitos do cliente?
		Os requisitos do software são consistentes com os requisitos do sistema?

2.4 Medição e Métricas de Software

Um dos principais objetivos da engenharia de software é aprimorar nossos processos, recursos e métodos de modo a melhor produzir e manter produtos. No entanto, algumas vezes, expressamos os objetivos de aprimoramento de forma genérica, sem uma descrição quantitativa de em que ponto estamos e aonde queremos chegar. Por essa razão, a medição de software tem se tornado um aspecto fundamental para a boa prática da engenharia de software. Quantificando onde e o que podemos aprimorar, descrevemos nossas ações e suas conseqüências em uma linguagem matemática, o que nos permite avaliar nosso progresso (PFLEEGER, 2004).

Para FENTON e PFLEEGER (1997), a medição é o processo no qual símbolos ou números são atribuídos aos atributos de uma entidade do mundo real de forma a descrevê-la de acordo com regras claras e bem definidas. Neste contexto, uma entidade é um objeto ou um evento do mundo real e um atributo é uma característica ou propriedade desta entidade. Segundo a norma de processo de medição de software, ISO/IEC 15939 (2002), o objetivo da medição em software é coletar, analisar e apresentar dados relacionados aos produtos desenvolvidos e processos implementados na organização e em seus projetos, para apoiar a gestão efetiva dos processos, e demonstrar objetivamente a qualidade dos produtos.

As medições podem ser aplicadas a processos de software com o objetivo de melhorá-los continuamente. Podem, também, ser usadas ao longo de um projeto de software para auxiliar nas estimativas, controle da qualidade e medição da produtividade. Finalmente, medições podem ser usadas por engenheiros de software para auxiliar na avaliação da qualidade de produtos de trabalho e na tomada de decisão ao longo do projeto (PRESSMAN, 2001).

Para serem efetivas, as medições devem concentrar-se em objetivos específicos; serem realizadas sobre todos os produtos, processos e recursos do ciclo de vida e terem seus resultados interpretados com base em características do contexto organizacional e do ambiente (ROCHA *et al.*, 2001). Além disso, a medição deve ser integrada ao processo que está sendo realmente utilizado na empresa e deve ser o máximo possível automatizada, pois diminui custos, não interfere nas atividades diárias da equipe de desenvolvimento e ainda aumenta a correção e freqüência da coleta dos dados (BESSA, 2005).

As medições são realizadas a partir da coleta de dados (medidas) relativos a indicadores de medição definidos (métricas). De acordo com CHRISTENSEN e THAYER (2001), uma medida é uma unidade de medição e uma métrica é definida como um indicador calculado ou composto baseado em duas ou mais medidas. Portanto, para se obter uma medição efetiva, é necessária a definição de métricas que possam medir o alcance dos objetivos de medição.

Existem diversas classificações para as métricas de software. Para KAN (2003) as métricas de software podem ser classificadas em três categorias:

- *Métricas de produto*: descrevem as características do produto tais como tamanho, complexidade, características de projeto, desempenho e nível de qualidade.
- *Métricas de processo*: podem ser usadas para melhorar o desenvolvimento e manutenção de software. Um exemplo de métrica de processo é a efetividade da correção de defeitos durante o desenvolvimento.
- *Métricas de projeto*: descrevem as características do projeto e sua execução. Como exemplo podemos citar, custo, cronograma e produtividade.

As métricas de qualidade de software são um subconjunto das métricas de software que focam na qualidade do produto, processo e projeto. Normalmente, as métricas de qualidade de software estão mais fortemente relacionadas às métricas de processo e de produto do que às métricas de projeto (KAN, 2003).

Além dessas classificações, as métricas ainda podem ser classificadas como subjetivas ou objetivas. Métricas subjetivas são aquelas que dependem do julgamento humano e seus resultados podem variar de pessoa para pessoa refletindo o julgamento de quem realizou a medição. Utilizando uma métrica subjetiva, se uma mesma pessoa medir um atributo em momentos diferentes poderá encontrar respostas distintas, mesmo que o atributo não tenha se modificado. Métricas objetivas, ao contrário, pressupõem que o processo de quantificação seja baseado em regras numéricas bem definidas, permitindo que um mesmo resultado seja obtido independentemente da pessoa que realizou a medição, do momento ou do ambiente em que isto foi feito. Para isso, basta que a condição em que se encontrava o atributo não tenha se modificado (ESTOLANO, 2005).

A definição de métricas deve considerar objetivos específicos para a organização. O ideal é que se tenha um planejamento da medição, isto é, uma metodologia para decidir

o que e como deve ser medido, e como interpretar os dados coletados. Uma das abordagens de planejamento da medição é através da derivação métricas a partir de objetivos e necessidades de informação, e a interpretação dos dados no contexto dos objetivos (BASILI *et al.*, 1994).

A "medição orientada a objetivos" é a definição para um programa de medição baseado em objetivos definidos de maneira precisa e explícita, que estabeleçam como as métricas deverão ser utilizadas, provendo suporte à adequação, consistência e completude do planejamento da medição, e conseqüentemente à coleta de dados (ESTOLANO, 2005).

Uma das abordagens de medição que parte dos objetivos é o GQM (*Goal, Question, Metric*). O Paradigma GQM (BASILI *et al.*, 1994) é uma abordagem para medição de produtos e processos de Engenharia de Software. Baseia-se na hipótese de que para uma organização medir de forma objetiva, ela deve identificar, explicitar e especificar precisamente os objetivos de medição da organização e também aqueles relativos a cada projeto; deve relacionar esses objetivos aos dados necessários, para defini-los operacionalmente; e também deve fornecer um *framework* para análise e interpretação dos dados com respeito aos objetivos definidos.

Em um programa de medição baseado em GQM, a atividade de análise é definida de maneira precisa e explícita através de um objetivo de medição. Os objetivos são definidos de modo que sejam operacionalmente tratáveis, por meio do refinamento para um conjunto de questões quantificáveis que são utilizadas para extrair a informação apropriada dos modelos, representando as dimensões dos objetivos. As métricas são derivadas com base nas questões, formalizando o processo e levando à escolha e/ou definição de métricas relevantes. Esse refinamento é documentado minuciosamente em um plano GQM, registrando todo o raciocínio utilizado na escolha das métricas. Os dados coletados são interpretados no contexto dos objetivos e questões definidos, considerando as limitações e suposições relativas a cada métrica (SOLINGEN e BERGHOUT, 1999).

Dessa forma, o GQM ajuda na realização do refinamento apropriado de questões abstratas de qualidade do domínio de Engenharia de Software e na derivação de mapeamentos de medição adequados entre o mundo empírico e um modelo formal (ESTOLANO, 2005).

Com o objetivo de auxiliar a medição de software durante o desenvolvimento de produtos de software, a norma internacional ISO/IEC 15939 (2002) define um processo

de medição de software, descrito através de um modelo que especifica as atividades e tarefas necessárias para se identificar, definir, selecionar e melhorar a medição de software dentro da estrutura de um projeto ou organização, auxiliando a especificar, de forma adequada, quais são as necessidades de medição, como as métricas e a análise devem ser aplicadas e como determinar se os resultados da análise são válidos. Ela também fornece definições para termos de medição que são comumente utilizados na indústria de software.

Como resultados da implementação bem sucedida do processo de medição definido na norma ISO/IEC 15939 espera-se que:

- O compromisso organizacional para medição seja estabelecido e mantido;
- As necessidades de informação dos processos técnicos e gerenciais sejam identificadas;
- Um conjunto apropriado de métricas, guiado pelas necessidades de informação, seja identificado ou desenvolvido;
- As atividades de medição sejam identificadas;
- As atividades de medição identificadas sejam planejadas;
- Os dados necessários sejam coletados, armazenados, analisados e que os resultados sejam interpretados;
- Os produtos de informação sejam utilizados para dar suporte à decisão e para fornecer base objetiva para comunicação;
- O processo de medição e as métricas sejam avaliadas; e
- Melhorias sejam comunicadas ao responsável pelo processo de medição.

Uma outra abordagem para medição de software é o PSM (*Practical Software Measurement*) (McGARRY *et al.*, 2001). O PSM apresenta um processo de medição e análise definido pelo Departamento de Defesa dos Estados Unidos com base nas melhores práticas do Departamento de Defesa e indústrias que praticam a medição de software. Ele fornece informações que visam a alcançar os objetivos técnicos, cronograma e custos planejados para a medição em um projeto. O PSM é compatível com a norma de medição de software, ISO/IEC 15939 (2002).

Também com o objetivo de auxiliar na definição de métricas para avaliação da qualidade, a norma internacional ISO/IEC 9126 (2001) define um conjunto de métricas

para avaliação de cada característica de qualidade nela definida. As métricas estão divididas em:

- *Métricas internas*: podem ser aplicadas a um produto de software não executável, durante estágios de seu desenvolvimento (como um pedido de proposta, especificação de requisitos, especificação de projeto ou código fonte). Métricas internas fornecem aos usuários a capacidade de medir a qualidade de produtos intermediários e assim prever a qualidade do produto final. Isso permite ao usuário identificar problemas relativos à qualidade e iniciar ações corretivas o quanto antes no ciclo de desenvolvimento de software. Exemplos de métricas internas definidas nesta norma são a remoção de defeitos e o impacto de mudanças.
- *Métricas externas*: podem ser usadas para medir a qualidade do produto de software através da medição do comportamento do sistema do qual ele faz parte. As métricas externas podem ser usadas apenas durante o estágio de testes no ciclo de vida e durante qualquer estágio operacional. A medição é realizada durante a execução do produto de software no ambiente de sistema no qual ele deve operar. Como exemplo de métricas externas definidas nesta norma temos adequação funcional e tempo médio entre falhas.
- *Métricas de qualidade em uso*: medem se um produto satisfaz às necessidades especificadas por usuários no que diz respeito a atingir níveis definidos de eficiência, produtividade, segurança e satisfação em um contexto de uso especificado. Isso só pode ser atingido em um ambiente de sistema real. Freqüência de erro e escala de satisfação são exemplos de métricas de qualidade em uso definidas nesta norma.

Para cada métrica proposta na norma ISO/IEC 9126 são apresentados o nome e o propósito da métrica, o método para sua aplicação, as fórmulas de cálculo da métrica e os elementos computacionais usados na composição da métrica, a interpretação do valor medido, o tipo de escala da métrica, o tipo de medida, os dados de entrada para a medição, as referências à ISO/IEC 12207 e o os usuários interessados nos resultados da métrica.

Portanto, a medição de software é uma infra-estrutura bastante útil e eficiente para se caracterizar a qualidade, e então, se definir estratégias para acompanhamento e melhoria da qualidade dos processos e produtos de software.

2.5 Garantia da Qualidade nas Normas e Modelos de Maturidade

Para viabilizar o controle e a garantia da qualidade de forma disciplinada e organizada, algumas abordagens para garantia da qualidade foram definidas na literatura.

A norma internacional ISO 12207 (ISO/IEC, 1998) define o processo de garantia da qualidade como um dos processos de apoio de ciclo de vida. Esse processo visa a fornecer garantia adequada de que os processos e produtos de software, no ciclo de vida do projeto, estejam em conformidade com seus requisitos especificados e sejam aderentes aos planos estabelecidos. Esse processo apresenta as seguintes atividades:

- *Implementação do processo*: consiste em definir um processo de qualidade adequado a cada projeto de desenvolvimento, determinando os objetivos do processo, e documentar o processo definido em um plano de garantia da qualidade que deve especificar também padrões de qualidade, recursos, cronograma e responsabilidades para conduzir as atividades de garantia da qualidade, entre outros;
- *Garantia do produto*: consiste em garantir que todos os planos exigidos pelo contrato estejam de acordo com o mesmo, os produtos de software e sua documentação estejam de acordo com o contrato e os planos e que os produtos de software satisfaçam totalmente os requisitos contratuais;
- *Garantia do processo*: consiste em garantir que os processos de ciclo de vida do software utilizados no projeto estejam de acordo com o contrato e com os planos, que as práticas de engenharia de software e os ambientes necessários ao desenvolvimento do produto estejam de acordo com o contrato, com as negociações e com os planos. Deve-se garantir, também, que as medições do processo e do produto estejam de acordo com os padrões e procedimentos estabelecidos e que a equipe tenha a qualificação e o conhecimento necessários para o projeto; e,
- *Sistemas de garantia da qualidade*: consiste em garantir a execução de atividades adicionais de gerência da qualidade de acordo com a norma ISO 9000 (2000) e conforme especificado no contrato.

O CMMI (SEI, 2002) define a área de processo Garantia da Qualidade do Processo e do Produto como uma área do Nível de maturidade 2 – Gerenciado. Uma área de processo do CMMI é dividida em objetivos. Os objetivos são componentes requeridos do modelo que descrevem as características que devem ser implementadas para satisfazer a área de processo. As práticas são componentes não obrigatórios, mas esperados, que são considerados importantes para se atingir o objetivo a que se referem.

Essa área de processo possui os seguintes objetivos:

(i) *Avaliar Objetivamente Processos e Produtos do Trabalho*: consiste em avaliar objetivamente a aderência dos processos executados, produtos do trabalho e serviços, às descrições do processo, padrões e procedimentos aplicáveis. É composto das práticas "Avaliar Objetivamente Processos" e "Avaliar Objetivamente Produtos do Trabalho e Serviços"; e,

(ii) *Prover Visão Objetiva*: consiste em acompanhar de forma objetiva e comunicar aspectos de não-conformidades e assegurar a resolução das não-conformidades identificadas. É composto das práticas "Comunicar e Assegurar Resolução das Questões de Não-Conformidade" e "Estabelecer Registros".

O MPS.BR (MPS.BR, 2005) também define um processo Garantia da Qualidade do Nível F – Gerenciado - cujo propósito é garantir que os produtos de trabalho e processos estão em conformidade com os planos e recursos pré-definidos. O modelo espera, como resultado da implantação deste processo, que:

(i) A aderência dos produtos e processos aos padrões, procedimentos e requisitos aplicáveis seja avaliada objetivamente;

(ii) Os produtos de trabalho sejam avaliados antes de serem entregues ao cliente e em marcos pré-definidos ao longo do ciclo de vida de desenvolvimento;

(iii) Os problemas e as não-conformidades sejam identificados, registrados e comunicados; e,

(iv) Ações corretivas para não-conformidades sejam estabelecidas e acompanhadas até as suas efetivas conclusões.

2.6 Conclusão

A qualidade pode ser crítica para a sobrevivência e o sucesso das empresas e, neste contexto, as empresas que desenvolvem software estão adotando procedimentos cuidadosos ao longo do processo de desenvolvimento de seus produtos para atingir uma qualidade adequada do produto.

Neste capítulo foram apresentados os aspectos relacionados à qualidade de software, considerando tanto a qualidade do processo quanto a qualidade do produto. Foram apresentados vários modelos e normas relacionados à qualidade de software, bem como características de qualidade e alguns modelos que as relacionam disponíveis na literatura.

Foram também apresentados critérios e métricas para a avaliação da qualidade de um produto de software e algumas abordagens para garantia da qualidade disponíveis na literatura.

No próximo capítulo, discute-se a verificação de software que é uma das atividades de garantia da qualidade de um produto de software.

CAPÍTULO 3

Verificação de Software

3.1 Introdução

No contexto atual do desenvolvimento de software, é cada vez maior a necessidade de verificar efetivamente a qualidade do software (GOLUBIC, 2003). Verificar software, em linhas gerais, significa avaliar, ao longo do desenvolvimento, se o produto está sendo desenvolvido adequadamente.

Uma das formas de realizar essa verificação, buscando obter produtos de melhor qualidade, é introduzir atividades de verificação ao longo do processo de desenvolvimento. Estima-se que 30% a 50% dos custos totais de um projeto são gastos nas tarefas de detecção e remoção de defeitos. Um importante princípio para reduzir esse custo, é encontrar e corrigir defeitos o mais cedo possível (LAITENBERGER *et al.*, 2002).

Uma questão importante no contexto da verificação de software é definir o que é *erro*, *defeito* e *falha*. Para PFLEEGER (2004), um *defeito* ocorre quando uma pessoa comete um engano, chamado *erro*, na realização de alguma atividade relacionada a um software. Já uma *falha* é uma divergência entre o comportamento requerido para o sistema e o comportamento real. Ela pode ser descoberta antes ou depois de o sistema ser entregue ao cliente, durante os testes ou durante a operação e a manutenção. Portanto, um defeito é uma visão interna do sistema, observada pelo ponto de vista dos desenvolvedores, enquanto uma falha é vista a partir de um referencial externo ao sistema: um problema que o usuário vê. Por conveniência, o termo *defeito* será usado de maneira genérica, podendo se incluir a ocorrência de uma falha.

Defeitos podem ser inseridos no produto de software durante todas as fases do processo de desenvolvimento. A Tabela 3.1 apresenta a relação entre as atividades do processo de desenvolvimento com as atividades nas quais os defeitos são inseridos e removidos. Idealmente, os defeitos de um produto de software devem ser detectados o mais cedo possível, para evitar re-trabalho. Segundo BOEHM e BASILI (2001), 40% a 50% do esforço de um projeto é gasto com re-trabalho que poderia ser evitado. Além disso, o custo para detectar e corrigir defeitos cresce bastante à medida que eles são propagados para fases posteriores do processo de desenvolvimento. Estudos mostram

que o custo de corrigir um defeito de projeto ou de codificação na própria fase é entre 10 a 100 vezes menor do que o custo de corrigi-lo na fase de testes (ANDERSSON, 2003). Essa é uma das principais motivações para o uso da verificação de software, que procura detectar os defeitos o quanto antes.

Tabela 3.1 – Relação entre as atividades do processo de desenvolvimento com a inserção e a remoção de defeitos (KAN, 2003)

Fase do Desenvolvimento	Inserção de Defeito	Remoção de Defeito
Requisitos	Processo de levantamento de requisitos e desenvolvimento das especificações	Verificação e validação dos requisitos
Projeto	Atividades de projeto	Verificação do projeto
Implementação	Codificação	Verificação do código
Integração	Processo de integração	Testes de integração
Teste de Unidade	Correções inadequadas	Testes
Teste de Componente	Correções inadequadas	Testes
Teste do Sistema	Correções inadequadas	Testes

Além de possibilitar a detecção de defeitos mais cedo, uma verificação efetiva pode aumentar a produtividade em projetos de software. QUAQUARELLI (1997) acompanhou, em 1997, a implantação de um processo de verificação na Cad.Lab S.p.A., uma empresa italiana com aproximadamente 250 empregados e 15 anos de experiência em desenvolvimento de software, e pôde constatar que a produtividade nos projetos aumentou cerca de 25% após a implantação do processo de verificação.

Apesar da importância da verificação no desenvolvimento de software e dos benefícios que essa prática pode trazer, até 2002, no Brasil, apenas 24,7% das empresas entrevistadas realizavam atividades de verificação, e ainda assim, de forma não documentada, enquanto 19,2% sequer adotavam a verificação em seus projetos, conforme constatou uma pesquisa realizada pelo MINISTÉRIO DA CIÊNCIA E TECNOLOGIA (2002).

Neste capítulo, é apresentado o estudo realizado na literatura sobre verificação de software. Na próxima seção são discutidas as definições e conceitos de verificação de produtos de software. Na seção 3.3 são discutidos os métodos e técnicas para realizar essa verificação. A seção 3.4 apresenta a verificação de software nas normas e modelos de maturidade. Por fim, na seção 3.5 são apresentadas as considerações finais deste capítulo.

3.2 Definições e Conceitos de Verificação de Software

O termo "verificação" foi definido de diversas formas na literatura. A norma internacional ISO/IEC 12207 (1998) define verificação como sendo a confirmação, por exame e fornecimento de evidência objetiva, do atendimento aos requisitos especificados. O objetivo da verificação é determinar se os produtos de software de uma atividade atendem completamente aos requisitos ou condições impostas a eles nas atividades anteriores (ISO/IEC, 1998). De acordo com PFLEEGER (2004), a verificação assegura que cada função opera corretamente. Já para o CMMI (SEI, 2002), a verificação deve garantir que os artefatos verificados atendam aos requisitos especificados para eles. Segundo SCHULMEYER e MACKENZIE (1999), verificação é a garantia de que produtos de uma fase particular do processo estão consistentes com os requisitos desta fase e da fase anterior.

A verificação de software está fortemente associada à validação de software sendo que, normalmente, elas são executadas em conjunto, pois muitas vezes é difícil determinar onde uma atividade começa e onde a outra termina. De uma maneira geral, pode-se dizer que a verificação se preocupa em avaliar se o produto está sendo desenvolvido corretamente, enquanto a validação visa a assegurar que se está desenvolvendo o produto correto, isto é, o produto que o cliente deseja.

Também, para a validação, existem diversas definições. Para PFLEEGER (2004) a validação assegura que o sistema implementou todos os requisitos, de maneira que cada função do sistema possa ser relacionada com um requisito particular na especificação. A norma internacional ISO/IEC 12207 (1998) define validação como a confirmação, por exame e fornecimento de evidência objetiva, de que os requisitos específicos, para um determinado uso pretendido, são atendidos. O objetivo da validação é assegurar que o software que está sendo desenvolvido é o software correto de acordo com os requisitos do usuário (ROCHA *et al.*, 2001). A norma internacional ISO/IEC 12207 (1998) define, ainda, que nas atividades de desenvolvimento, a verificação refere-se ao processo de examinar o resultado de uma atividade para determinar sua conformidade com os requisitos estabelecidos para a mesma atividade, enquanto a validação se refere ao processo de examinar um produto para determinar sua conformidade com as necessidades do usuário. Esta norma define que a validação é feita normalmente no produto final sob condições de operação definidas, podendo, contudo, tornar-se necessária em fases anteriores. Por exemplo, em projetos cujos requisitos não são bem

conhecidos, pode-se tornar necessária uma validação dos requisitos identificados inicialmente através de um protótipo. O CMMI (SEI, 2002) define que, durante o desenvolvimento dos requisitos, eles devem ser validados para garantir que o produto a ser desenvolvido se comportará adequadamente no ambiente de uso pretendido para ele.

Atividades de verificação e validação devem ser executadas ao longo do desenvolvimento de um produto, começando geralmente com a verificação e/ou a validação dos requisitos, seguindo com a verificação dos produtos intermediários (projeto, código, dentre outros) e concluindo com a validação do produto final no ambiente operacional. Essa seqüência de execução das atividades de verificação e validação ao longo do desenvolvimento, relacionando tais atividades às fases do desenvolvimento do produto, também é conhecida como o "modelo V" (TIAN, 2005) e está ilustrada na figura 3.1.

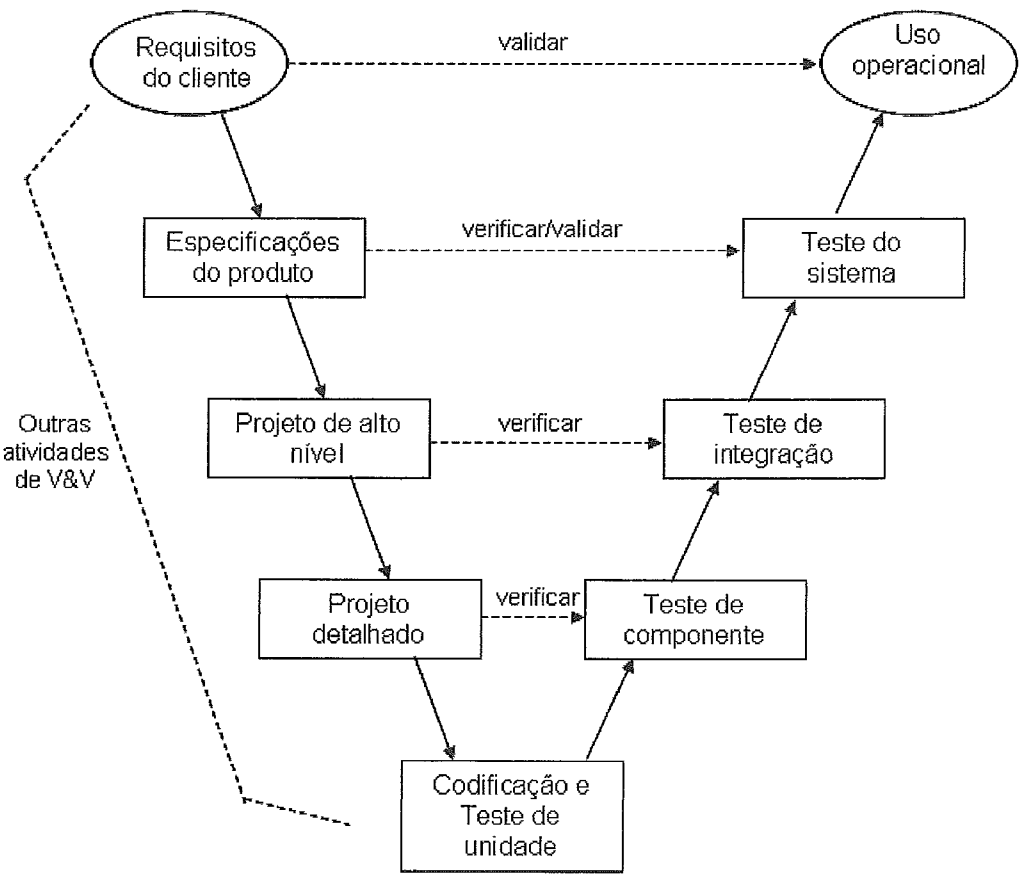


Figura 3.1 – Atividades de verificação e validação associadas no "modelo V" (TIAN, 2005)

Neste trabalho serão usadas as definições mais tradicionais para esses termos. Consideramos que verificação consiste em garantir que os produtos de software de uma

atividade atendem completamente aos requisitos para eles especificados. Validação consiste em garantir que o software desenvolvido é o software correto de acordo com os requisitos do usuário. A verificação avalia se o produto está sendo desenvolvido corretamente enquanto a validação avalia se está sendo construído o produto correto (PRESSMAN, 2001).

3.3 Métodos e Técnicas de Verificação de Software

A verificação de produtos de software pode ser realizada através da aplicação de métodos e técnicas específicos ao longo do desenvolvimento do produto. Segundo PFLEEGER (2004), um método ou técnica é um procedimento formal para produzir algum resultado. Com o objetivo de guiar a execução da verificação, diversos métodos e técnicas para verificação de software são descritos na literatura. O uso desses métodos e técnicas deve ser adequadamente planejado e controlado para que se alcancem os objetivos propostos para a verificação.

Dois métodos de verificação são destacados na literatura (THELIN, 2002; SEI, 2002): a revisão por pares e os testes de software. Revisão por pares é um método estático de verificação no qual um artefato é examinado por qualquer integrante da equipe do projeto (inclusive a equipe de garantia da qualidade), exceto o autor do artefato, com o propósito de detectar defeitos (LAITENBERGER *et al.*, 2002). É conduzida de acordo com processos bem definidos que incluem participantes com vários papéis e responsabilidades específicas. Por ser um método estático, isto é, que não depende da execução do código, a revisão por pares pode ser aplicada desde o início do projeto, ajudando a detectar defeitos mais cedo.

A aplicação de revisão por pares em um projeto pode trazer vários benefícios (SCHULMEYER e MACKENZIE ,1999):

- Diminuição do tempo de depuração por ocasião dos testes;
- Detecção dos defeitos de análise e projeto mais cedo, reduzindo os custos de correção destes defeitos;
- Identificação de ineficiências no projeto ou código;
- Garantia da aderência a padrões da organização;
- Aumento da legibilidade do código;
- Aumento da satisfação do usuário; e,

- Aumento da manutenibilidade.

Além desses benefícios, BOEHM e BASILI (2001) destacam que a revisão por pares é um método bastante eficiente, pois detecta de 31% a 93% dos defeitos de um produto, sendo que, na média são detectados 60% dos defeitos.

Alguns métodos de verificação podem ser classificados como sendo tipos particulares do método revisão por pares e diferem entre si pelo grau de formalidade e pelos papéis e responsabilidades dos participantes. Dentre estes, destacam-se os métodos *Walkthrough* e Inspeção (SEI, 2002; LAITENBERGER *et al.*, 2002; SCHULMEYER e MACKENZIE, 1999) que serão descritos em detalhes nas seções 3.3.1 e 3.3.2, respectivamente.

Um outro método de verificação que merece destaque no contexto deste trabalho é o teste, cujo objetivo é examinar o comportamento do software através de sua execução (JURISTO *et al.*, 2003). Como o teste é baseado na execução do código, ele só pode ser usado depois que partes do software tenham sido implementadas, e, portanto, em uma verificação baseada somente em testes, os defeitos provavelmente serão detectados tardiamente. Para possibilitar a detecção de defeitos tão cedo quanto possível, a proposta é associar os testes às revisões por pares. Por isso, revisões e testes devem ser vistos como métodos complementares. As informações obtidas durante as revisões são extremamente úteis para os testes, por permitirem a identificação dos módulos críticos e propensos a erros (ROCHA *et al.*, 2001). O ideal é que as revisões por pares e os testes sejam usados em conjunto, sempre que possível. (BARRETO e ROCHA, 2005).

Tanto a inspeção e o *walkthrough*, que são tipos particulares de revisão por pares, quanto os testes devem ser aplicados ao longo do desenvolvimento do produto, nos momentos adequados. SCHULMEYER e MACKENZIE (1999) propõem um mapeamento entre alguns métodos de verificação e as fases do desenvolvimento de um produto, considerando o momento em que cada um dos métodos deveria ser aplicado. Este mapeamento está descrito na tabela 3.2.

Tabela 3.2 –Mapeamento entre métodos de verificação e fases do desenvolvimento (SCHULMEYER e MACKENZIE, 1999)

Método	Requisitos	Projeto	Unidade de Código	Código Integrado	Sistema
Inspeção	X	X	X		
Walkthrough	X	X	X		
Testes			X	X	X

Além da revisão por pares e do teste, outros métodos e técnicas de verificação podem ser encontrados na literatura tais como prova formal (PFLEEGER, 2004; RAKITIN, 2001), auditoria (SWEBOK, 2004; SCHULMEYER e MACKENZIE, 1999), *cleanroom* (PFLEEGER, 2004; KAN, 2003; PRESSMAN, 2001), análise de algoritmo (SCHULMEYER e MACKENZIE, 1999), que não serão detalhados neste capítulo por não estarem no escopo deste trabalho.

3.3.1 Walkthrough

Este método, que é um tipo particular de revisão por pares, tem por objetivo é avaliar um ou mais artefatos para identificar defeitos e considerar possíveis soluções alternativas. Ele consiste de uma preparação e uma reunião de avaliação de no máximo duas horas e da qual, segundo SCHULMEYER e MACKENZIE (1999), participam no máximo sete pessoas. Os participantes assumem papéis específicos, tais como moderador, autor, secretário e membros. Ao moderador cabe, por exemplo, definir os demais participantes, organizar o *walkthrough* e definir local, data e agenda da reunião. O autor é o responsável pela produção dos artefatos avaliados além de apresentá-los durante a reunião. O secretário deve auxiliar o moderador durante a reunião enquanto os demais membros devem avaliar os artefatos, registrar os defeitos encontrados e sugerir soluções.

A preparação se inicia quando o moderador ou o autor distribui aos participantes os artefatos a serem revisados assim que eles forem concluídos. Então os participantes analisam artefatos antes da reunião e registram os problemas encontrados e possíveis soluções identificadas (ESA, 1995).

A reunião, por sua vez, é iniciada pelo moderador no local e data planejados. O autor guia os participantes através da leitura dos artefatos ou da execução dos mesmos, se pertinente, enquanto cada participante faz os comentários e sugestões que achar adequado. Os problemas encontrados e as soluções propostas são registrados, e a equipe, com base nos problemas encontrados, decide pela aceitação ou não dos artefatos avaliados. Caso julguem adequado, por exemplo, no caso de encontrem problemas muito graves, os participantes podem decidir pela rejeição dos artefatos. Finalmente, o autor corrige os problemas encontrados e, caso a decisão tenha sido a rejeição dos artefatos, uma outra avaliação dos mesmos deve ser realizada (SCHULMEYER e MACKENZIE, 1999).

Em um *walkthrough*, a atitude dos participantes é fundamental, pois os defeitos encontrados não devem ser vistos como fraquezas do autor, mas como inerentes à dificuldade de desenvolver o produto (MYERS, 2004). Vale destacar que as reuniões de *walkthrough* focam não somente a detecção de defeitos nos artefatos, mas também visam a fornecer implementações alternativas (IEEE, 1998).

De acordo com McCONNELL (2004), no que diz respeito a código, um *walkthrough*, se usado de forma inteligente e disciplinada, pode encontrar entre 30% e 70% dos defeitos em um programa.

O *walkthrough* é uma boa escolha se o grupo de revisores é muito grande, pois com muitos participantes, haverá vários pontos de vista diferentes. Uma vez que este método discute soluções alternativas, várias soluções serão apresentadas e, se todos os envolvidos forem convencidos de que uma determinada solução é boa, ela provavelmente não apresenta falhas graves.

3.3.2 Inspeção

Desde que a inspeção de software foi proposta por FAGAN (1976) há 30 anos atrás, ela tem sido adotada pela indústria devido ao seu impacto na qualidade dos produtos desenvolvidos (WINKLER *et al.*, 2005; LANUBILE *et al.*, 2003). A inspeção, um tipo de revisão por pares, é realizada de acordo com um processo bem definido, através do qual os participantes analisam um ou mais artefatos com o propósito de detectar defeitos (LAITENBERGER *et al.*, 2002).

As principais características da inspeção são a preparação para a reunião realizada individualmente por cada participante e o uso de *checklists* para facilitar a detecção de defeitos. Neste método, os participantes devem avaliar os artefatos com base nos critérios definidos nos *checklists* antes da reunião (SCHULMEYER e MACKENZIE, 1999).

Uma inspeção é composta por um conjunto de atividades realizadas por uma equipe tecnicamente qualificada para verificar se os artefatos possuem qualidade satisfatória (REIS, 2005). FAGAN (1976) propôs um processo de inspeção e diversos estudos foram realizados com base nesta proposta gerando novas abordagens para o processo (KALINOWSKI *et al.*, 2004; RAKITIN, 2001; SAUER *et al.*, 2000; PORTER e JOHNSON, 1997; VOTTA, 1993).

Em (RAKITIN, 2001) é proposto um processo de inspeção composto das seguintes atividades:

- Planejamento: Consiste em definir critérios de avaliação para os artefatos, métricas a serem coletadas, participantes, data, hora e local para reunião de inspeção, preparar o material para a inspeção, definir se haverá reunião preliminar, e distribuir o material aos participantes;
- Reunião Preliminar: Essa atividade é opcional e deve ser realizada somente se houver necessidade de esclarecimentos a respeito dos artefatos a serem inspecionados;
- Preparação: Nesta atividade cada inspetor deve revisar os artefatos com base nos critérios definidos no planejamento e documentar os problemas encontrados;
- Reunião de Inspeção: Nesta atividade os artefatos revisados são lidos e durante a leitura os participantes discutem cada problema encontrado individualmente durante a atividade de preparação. Os defeitos, identificados em consenso por todos os participantes, são registrados para posterior correção. Neste momento deve ser decidido se há necessidade de outra reunião de inspeção; e,
- Continuação: Esta atividade consiste em definir um cronograma para correção dos defeitos encontrados e na correção efetiva destes defeitos de acordo com o cronograma definido.

Desde a primeira definição do processo de inspeção, FAGAN (1976) já afirmava que, para que uma inspeção fosse eficaz, era necessária uma boa equipe de inspeção, na qual cada um de seus membros assumisse um papel específico. Com o objetivo de aumentar a eficácia de uma inspeção, TRAVASSOS *et al.* (2001) apresentam os seguintes papéis que devem ser assumidos pela equipe:

- Organizador: planeja todas as atividades da inspeção;
- Inspetor: responsável por identificar os defeitos nos artefatos;
- Moderador: é o elemento chave para o sucesso de uma inspeção. Ele deve gerenciar e liderar a equipe com o objetivo de garantir que os procedimentos estão sendo seguidos e que os participantes estão assumindo os seus papéis adequadamente; e,

- Autor: produziu os artefatos que estão sendo revisados e é o responsável por, posteriormente, corrigir os defeitos acordados em consenso.

Para auxiliar na identificação de defeitos, uma das tarefas mais importantes da inspeção, independentemente do processo de inspeção adotado, podem-se utilizar diferentes técnicas de leitura, sendo que a técnica escolhida deve identificar as informações a serem verificadas e apoiar a localização de defeitos nestas informações (SAYÃO e LEITE, 2005). Técnicas de leitura são métodos sistemáticos e estruturados que visam a auxiliar o inspetor a realizar a inspeção segundo um foco específico (REIS, 2005). Como exemplos de técnicas de leitura têm-se:

- PBR - *Perspective-Based Reading*: é uma técnica para revisão de requisitos que considera as diferentes perspectivas (visões) dos atores do processo e utiliza diferentes *checklists* para guiar a leitura e análise do artefato. No processo de inspeção segundo a técnica PBR, os diferentes atores envolvidos no processo de desenvolvimento vêem os artefatos sob diferentes pontos de vista e, portanto, são selecionados de acordo com a utilização que farão do artefato. Numa inspeção do documento de requisitos, podem ser selecionados, por exemplo: usuário final, projetista, programador, representante da equipe de manutenção, executor de testes e outros. O documento de requisitos será visto através de diferentes visões: o usuário final deseja ver ali refletido o conjunto de funcionalidades a ser atendido pelo software; o projetista irá utilizá-lo como base para a criação da estrutura do software, considerando as funcionalidades e restrições descritas, e o testador verá o documento de requisitos como fonte para a geração dos testes, que deverão assegurar que o mesmo atende às necessidades especificadas (SAYÃO e LEITE, 2005).
- CBR - *Checklist-Based Reading*: é uma técnica na qual os revisores usam uma lista de questões (*checklist*) que os orienta sobre quais tipos de defeitos devem procurar. Os inspetores devem revisar o documento até obterem uma cobertura completa do documento em relação ao *checklist* (THELIN, 2002); e,
- OORTs – *Object-Oriented Reading Techniques*: são técnicas para leitura de projetos orientados a objetos que foram desenvolvidas e direcionadas à identificação de defeitos (TRAVASSOS *et al.*, 2002). Diferente da PBR, o processo de leitura utilizando OORTs precisa ser em duas vias. Na revisão dos

requisitos, a eficiência e a consistência dos diagramas de projeto precisam ser verificadas (por meio de leituras horizontais) para assegurar que todos os documentos sejam consistentes. Além disso, é necessária a verificação da consistência entre os artefatos de projeto e os requisitos do sistema (por meio de leitura vertical), para assegurar que o projeto do sistema está correto com relação aos requisitos funcionais (ROCHA *et al.*, 2001).

Assim como no *walkthrough*, os inspetores, com base nos problemas encontrados, decidem pela aceitação ou não dos artefatos avaliados. Caso julguem adequado, por exemplo, no caso de encontrarem problemas muito graves, os participantes podem decidir pela rejeição dos artefatos e, neste caso, uma outra reunião de inspeção dos mesmos deve ser realizada (SCHULMEYER e MACKENZIE, 1999).

Muitos estudos têm sido realizados com o objetivo de identificar os benefícios do uso da inspeção. KIRNER *et al.* (2005) relatam que a inspeção aplicada ao código reduz os custos do projeto em até 39%, enquanto a inspeção aplicada ao projeto reduz esses custos em até 44%. Já a produtividade tem um acréscimo de 30% a 50% e o tempo de desenvolvimento tem uma redução de 10% a 30%, quando as inspeções são usadas no projeto. Segundo McCONNELL (2004), em projetos que usam inspeções de projeto e de código, apesar de a inspeção ser responsável por 10% a 15% do orçamento dos projetos, o uso das inspeções reduz o custo total do projeto. Além disso, o uso combinado das inspeções de projeto e de código normalmente detecta de 70% a 85% dos defeitos.

De acordo com ESA (1995), as inspeções de software são eficientes porque detectam mais que 50% dos defeitos introduzidos ao longo do desenvolvimento do produto, além de serem econômicas, pois não só trazem uma redução do número de defeitos, como também reduzem o custo, uma vez que se economizam os recursos que seriam gastos para removê-los.

A aplicação de inspeções de forma bem planejada pode trazer aprendizados significativos para a equipe. Com uma boa metodologia para aplicação de inspeções se consegue identificar quando e onde os defeitos geralmente estão ocorrendo (REIS, 2005). Os desenvolvedores que participam de inspeções têm a oportunidade de aprender, o que traz uma melhoria do trabalho por eles realizado e, com isso, pode-se obter um aumento de produtividade de cerca de 20%, conforme constatou McCONNELL (2004).

Apesar dos vários benefícios que o uso da inspeção e do *walkthrough* pode trazer, até 2002, conforme constatou uma pesquisa de Qualidade e Produtividade no Setor de Software Brasileiro (MINISTÉRIO DA CIÊNCIA E TECNOLOGIA, 2002), no Brasil, somente 16,3% das empresas entrevistadas usavam inspeção ou *walkthrough* em seus projetos.

Desde que os primeiros trabalhos sobre inspeção foram realizados na década de 70, muitos outros trabalhos têm sido realizados com o objetivo de explorar este método e identificar melhorias possíveis. Em (KALINOWSKI, 2004) é apresentada uma infra-estrutura computacional de apoio ao processo de inspeção de software chamada ISPIS. Essa infra-estrutura apóia a inspeção de diferentes artefatos produzidos ao longo do processo de desenvolvimento de software de forma sistemática e utilizando dados históricos e informações experimentalmente avaliadas sobre inspeções de software. ISPIS pode ser instanciada para realização de inspeções de diferentes artefatos com equipes geograficamente distribuídas. A infra-estrutura apóia o processo de inspeção no planejamento da inspeção, na detecção de defeitos, na coleção de defeitos (descarte de alguns defeitos e eliminação de duplicatas), na discriminação de defeitos e na continuação (onde se decide se uma nova inspeção deve ou não ocorrer). Um estudo experimental foi realizado para avaliar o apoio de ISPIS para a atividade de planejamento e seus resultados foram positivos. Em complemento, um estudo de caso mostrou a viabilidade de se utilizar ISPIS em inspeções reais. Além disso, a infra-estrutura tem sido efetivamente utilizada em inspeções realizadas na COPPE/UFRJ (KALINOWSKY e TRAVASSOS, 2004).

Em (LANUBILE *et al.*, 2003) é descrita uma ferramenta chamada IBIS (*Internet-Based Inspections System*) que tem por objetivo apoiar a verificação do documento de especificação de requisitos de software segundo a técnica PBR, considerando que a equipe de inspeção está dispersa geograficamente. Para isso a ferramenta adota um processo de inspeção adaptado que minimiza as atividades síncronas e os problemas de coordenação. A ferramenta armazena o documento a ser inspecionado, possibilita o cadastro e a seleção dos inspetores, fornece *checklists* e formulários aos inspetores e registra os problemas detectados por cada um deles. Após o registro dos problemas, os relatórios são analisados e os problemas são agrupados; a consolidação é feita posteriormente pelo coordenador do processo.

Alguns trabalhos foram realizados na COPPE/UFRJ com o objetivo de fornecer apoio automatizado à aplicação de técnicas de leitura. REIS (2005) apresenta uma

ferramenta de apoio à aplicação de técnicas de leitura orientadas a objetos. A ferramenta, denominada *Orion Tool*, se propõe a fornecer um apoio automatizado consistente para aplicação de OORTs, padronizar os resultados da inspeção, com a finalidade de facilitar seu emprego em estudos experimentais e na indústria, e aprimorar as análises dos resultados e do comportamento dos inspetores durante a inspeção. A ferramenta ainda coleta métricas relacionadas ao desempenho do inspetor e ao seu comportamento durante a inspeção, avaliando se ele seguiu a ordem proposta pela técnica. (SILVA *et al.*, 2004) descrevem uma ferramenta de apoio à aplicação da técnica de leitura PBR para a perspectiva do usuário. A ferramenta fornece uma ajuda direcionada e um guia para a execução das tarefas especificadas pela técnica, facilitar o relato das discrepâncias identificadas e verificar a conformidade da aplicação da técnica.

3.3.3 Teste

O teste de software é uma das atividades mais importantes para a garantia da qualidade do produto e freqüentemente a mais usada (TIAN, 2005). Realizar testes consiste em verificar dinamicamente o comportamento de um programa, usando um conjunto de casos de teste adequadamente selecionados, em relação ao comportamento esperado para o mesmo (SWEBOK, 2004). O principal objetivo para a realização de testes é encontrar defeitos no software, com a finalidade de corrigi-los antes que estes se manifestem quando o usuário estiver utilizando o sistema. Caso os testes não consigam provocar falhas nem evidenciar defeitos, isso deve significar um aumento de confiança no sistema e indicar que o mesmo esteja correto (MALDONADO e FABRI, 2001). Para isso, deve-se realizar um conjunto de testes eficiente e com cobertura adequada.

Com o objetivo de auxiliar a realização dos testes, têm-se quatro etapas de teste que devem ser conduzidas ao longo de todo o processo de desenvolvimento do software (ROCHA *et al.*, 2001):

- Planejamento: Consiste em definir um plano de testes que determine os critérios a serem avaliados no produto a ser testado, os recursos necessários e o cronograma para execução dos testes;
- Projeto de casos de teste: Consiste em selecionar as técnicas de testes que serão usadas, especificar os casos de teste e definir os procedimentos de teste;

- Execução: Consiste em executar os casos de teste especificados seguindo os procedimentos definidos; e,
- Avaliação dos resultados: Consiste em analisar os resultados dos testes confrontando-os com os resultados esperados.

Além das diversas etapas de teste definidas ao longo do desenvolvimento de um produto, o teste, geralmente, envolve várias fases. Primeiro, cada unidade[1] do programa é testada, isolada das demais unidades do sistema. Esse teste, conhecido como *teste de unidade*, verifica se a unidade funciona de forma adequada aos tipos de entrada esperadas, a partir do estudo do projeto da unidade. Quando todas as unidades já tiverem sido testadas, a próxima fase é realizar o *teste de integração* para assegurar que as interfaces entre as unidades foram definidas e tratadas adequadamente. Após assegurar que as informações são passadas entre as unidades de acordo com o que foi projetado, é o momento de realizar o *teste do sistema* para assegurar que ele tem a funcionalidade desejada (PFLEEGER, 2004).

O teste de unidade tem como objetivo encontrar problemas em módulos individuais e são aplicados nos menores componentes de código criados. Cada componente ou unidade é executado isoladamente, possibilitando verificar se a informação entra e sai de forma adequada (PRESSMAN, 2001). Os testes de unidade geralmente são realizados pelo próprio programador, pois exigem um ambiente de produção adequado e especializado (PRICE e HERBERT, 2000). Uma das vantagens do teste de unidade é permitir a introdução de algum paralelismo nos testes, uma vez que as unidades são testadas isoladamente e por isso várias unidades podem ser testadas ao mesmo tempo (MYERS, 2004). Segundo McCONNELL (2004) o teste de unidade detecta de 15% a 50% dos defeitos.

Após realizar o teste de unidade, o teste de integração deve ser realizado. O teste de integração pode ter sua execução iniciada assim que alguns componentes ficarem prontos. No contexto do ciclo de desenvolvimento, um componente pronto significa um componente implementado e com os testes de unidade aprovados, ou seja, componentes individuais funcionando corretamente e atingindo os seus objetivos (PFLEEGER, 2004). O *teste de integração* trata da junção de vários componentes para funcionarem de forma integrada, com o foco nas interfaces e nos problemas de interação entre esses componentes. Portanto, no teste de integração, cada componente é tratado como uma

[1] Uma unidade é um componente de software que não pode ser subdividido (IEEE, 1990).

unidade atômica, enquanto as interconexões entre eles são modeladas e examinadas para testar as interfaces e interações entre tais componentes (TIAN, 2005). O teste de integração começa com a integração de algumas unidades e culmina com o sistema integrado, após todas as unidades terem sido integradas e testadas. Em muitas organizações, os desenvolvedores são responsáveis por realizarem esses testes, porém, frequentemente eles primeiro integram todas as unidades e depois começam os testes de integração, ao invés de integrar e testar as unidades interativamente. Esta estratégia é desaconselhável por trazer vários problemas, uma vez que dessa forma é muito mais difícil detectar os defeitos e corrigi-los (RAKITIN, 2001). Segundo McCONNELL (2004) o teste de integração detecta de 25% a 40% dos defeitos.

Após a integração das unidades devidamente testada, deve-se realizar o *teste do sistema* que tem como objetivo determinar se o software e demais elementos que compõem o sistema, tais como hardware e banco de dados, atendem aos requisitos especificados (RAKITIN, 2001). Esse teste deve ser realizado, preferencialmente, por pessoas que não estejam ligadas ao desenvolvimento, podendo utilizar usuários, assim como analistas de outros projetos (PRICE e HERBERT, 2000). Conforme constatou McCONNELL (2004) o teste do sistema detecta de 25% a 55% dos defeitos. A tabela 3.3 resume a taxa de detecção de defeitos em cada fase de teste.

Tabela 3.3 –Taxa de detecção de defeitos em cada fase de teste (McCONNELL, 2004)

Fase de Teste	Menor Taxa	Maior Taxa
Teste de Unidade	15%	50%
Teste de Integração	25%	40%
Teste do Sistema	25%	55%

O teste do sistema é formado pelas seguintes fases: *teste funcional*, *teste de desempenho*, *teste de aceitação* e *teste de instalação* (PFLEEGER, 2004). A figura 3.2 ilustra tais fases juntamente com os testes de unidade e de integração.

O *teste funcional* verifica se o sistema integrado realiza as funções especificadas nos requisitos (PFLEEGER, 2004). Após garantir que as funcionalidades do sistema são executadas conforme especificado, deve-se realizar o *teste de desempenho* que tem como objetivo avaliar como o sistema se comporta em relação aos requisitos não-funcionais especificados tais como tempo de resposta, uso do processador, segurança, dentre outros (WALLACE *et al.*, 1996).

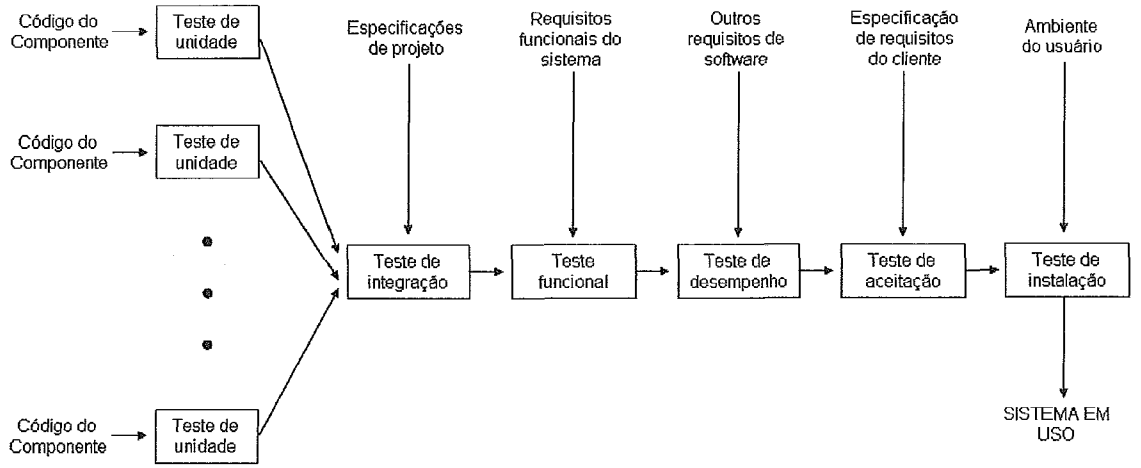


Figura 3.2 – Fases de Teste (PFLEEGER, 2004)

Após testar o sistema e garantir que ele se comporta conforme especificado, tanto para os requisitos funcionais quanto para os não-funcionais, o cliente deve avaliar o sistema para determinar se o sistema construído é o que ele deseja. Neste momento, o teste de aceitação deve ser realizado. O *teste de aceitação* avalia o comportamento do sistema em relação aos requisitos do cliente, que define tarefas específicas para determinar se seus requisitos foram atendidos. O principal participante deste teste é o cliente, porém os desenvolvedores também podem participar (SWEBOK, 2004). Após a aceitação do sistema por parte do cliente, ele deve ser avaliado quanto à sua instalação no ambiente de uso. Essa avaliação é feita através do *teste de instalação* que tem por objetivo permitir que os usuários executem as funcionalidades do sistema e documentem problemas adicionais específicos do ambiente de operação (PFLEEGER, 2004).

Para guiar os desenvolvedores e testadores durante a realização dos testes, diversas técnicas de teste foram definidas na literatura (TIAN, 2005; SWEBOK, 2004; RAKITIN, 2001; ROCHA *et al.*, 2001). As técnicas de teste de software podem ser classificadas de acordo com a origem das informações utilizadas para estabelecer os requisitos de teste. Estas técnicas podem ser classificadas como (MALDONADO E FABRI, 2001):

- Funcional: aborda o software de um ponto de vista macroscópico e estabelece os requisitos de teste a partir da especificação do produto. Esta técnica também é chamada de "caixa-preta" ou "comportamental".

- Estrutural: estabelece os requisitos de teste com base na implementação do código. Esta técnica também é chamada de "caixa-branca" ou "caixa-de-vidro";
- Com base em erros: estabelece os requisitos explorando os erros típicos e comuns cometidos durante o desenvolvimento do *software*; e,
- Com base em máquina de estado finito: utiliza a estrutura de máquinas de estado finito e o conhecimento subjacente para determinar os requisitos de teste.

A maior diferença entre teste funcional e estrutural está na perspectiva e no foco: Teste funcional foca no comportamento externo de um sistema de software ou de alguns de seus componentes, considerando o objeto a ser testado como uma caixa preta que nos impede de ver seu interior. Por outro lado, teste estrutural foca na implementação interna, considerando o objeto a ser testado como uma caixa branca que nos permite ver seu interior. A tabela 3.4 apresenta uma comparação entre essas duas técnicas de teste.

Tabela 3.4 – Comparação entre teste funcional e estrutural (TIAN, 2005)

Característica	Teste Funcional	Teste Estrutural
Perspectiva	Trata os objetos de testes como uma caixa preta, focando em testar as relações de entrada e saída ou comportamento funcional externo	Trata os objetos como uma caixa de vidro onde detalhes internos de implementação são visíveis e testados
Objetos	Geralmente mais adequados para sistemas de software grandes ou partes substanciais destes	Normalmente usado para testar objetos pequenos, tais como produtos de software pequenos ou pequenas unidades de produtos de software maiores
Momento de execução	Normalmente utilizado em fases posteriores, como no teste de sistemas ou de aceitação	Mais usado em fases iniciais dos testes de sistemas de software maiores, como no teste de unidades ou de componentes
Foco de defeito	Falhas relacionadas a funções externas específicas podem ser observadas, levando a detecção e remoção de defeitos correspondentes	Falhas relacionadas à implementação interna podem ser observadas, levando a detecção e remoção de defeitos relacionados
Detecção e remoção de defeitos	Eficiente na detecção e remoção de problemas de interfaces e interações	Eficiente em problemas localizados dentro de uma unidade pequena
Testador	Tipicamente executado por testadores profissionais especificamente dedicados a testes ou terceiros	Normalmente realizado pelos próprios desenvolvedores

Considerando os objetos a serem testados, programas ou códigos-fonte existem em várias formas e são escritos em diferentes linguagens de programação. Eles podem ser vistos tanto como partes individuais ou como um todo integrado. Consequentemente, há

diferentes níveis de testes correspondentes a diferentes visões do código, e diferentes níveis de abstração. Em cada nível de abstração, pode-se escolher por focar tanto no comportamento geral ou nos elementos individuais que fazem parte dos objetos de testes, resultando na diferença entre teste funcional e teste estrutural. A tendência é de que em níveis mais altos de abstração, o teste funcional seja mais usado, enquanto que em níveis mais baixos, seja mais usado o teste estrutural (TIAN, 2005).

Com o objetivo de explorar o uso dos testes e identificar melhorias possíveis, vários trabalhos têm sido realizados. DONEGAN *et al.* (2005) descrevem experiências em automação de testes sistêmicos funcionais em uma organização e uma análise da aplicação de testes funcionais automatizados no contexto de projetos de desenvolvimento de software. A conclusão foi que, de forma geral, a automação de testes sistêmicos funcionais gerou bons resultados para a organização. Verificou-se que o melhor resultado foi a execução híbrida dos testes, usando a codificação de scripts para os casos de uso apenas nas primeiras iterações e executando testes manuais para aqueles implementados nas últimas.

CRESPO *et al.* (2004) apresentam uma metodologia para implantação ou melhoria do processo de teste em empresas desenvolvedoras de software que foi desenvolvida pelo CenPRA (*Centro de Pesquisas Renato Archer*) com o objetivo de viabilizar a utilização das práticas de teste pelas empresas. A metodologia de teste desenvolvida engloba técnicas, procedimentos e ferramentas, capacitando as empresas a desenvolverem produtos de melhor qualidade. Nesta metodologia, a implantação do processo de teste envolve um conjunto de atividades que vai desde o levantamento das necessidades da empresa, passa pela realização de treinamentos da equipe técnica e vai até ao acompanhamento dos trabalhos realizados, constituindo assim, um completo ciclo de implantação da atividade de teste dentro da empresa. Para experimentação e validação desta metodologia, ela foi aplicada em uma micro-empresa de software como parte de um projeto de melhoria de processo e foram observados, dentre outros, os seguintes resultados: maior controle do projeto, melhor qualidade do produto final gerado e maior disposição dos clientes em esperar mais pelo produto final. Foi constatado, também que o retorno dos investimentos é de médio e longo prazo.

Em (FANTINATO *et al.*, 2004) é apresentado um *framework* para a automação da execução de teste funcional de software. O *framework*, denominado *AutoTest*, é composto por um conjunto de ferramentas de software, scripts de teste, modelos de planilhas e regras e procedimentos de utilização e permite automatizar a execução de

casos de teste com pouca necessidade de codificação, dependendo do projeto em questão. Com o uso da ferramenta é possível obter um conjunto de casos de teste automatizados que pode ser facilmente executado, re-executado e atualizado.

Em (VEGAS, 2002) foi realizada uma pesquisa procurando determinar como seria possível auxiliar os desenvolvedores a melhorar a seleção de técnicas de testes. Para isso, foi proposto um esquema de caracterização de técnicas de testes. O esquema pode ser usado para se construir um repositório de informações sobre essas técnicas. Esse esquema descreve sistematicamente todas as técnicas de testes, priorizando aspectos pragmáticos das técnicas, possibilitando seleções mais objetivas. O esquema é composto por um conjunto de atributos, agrupados a elementos do processo de teste a que se referem. Esses elementos são então agrupados a diferentes estágios do processo de testes. Nesse trabalho foi proposta uma ferramenta conceitual que pode ser utilizada para a seleção das técnicas de testes a serem utilizadas em um projeto.

3.4 Verificação de Software nas Normas e Modelos de Maturidade

Para que a verificação de software seja realizada de forma organizada, disciplinada e seguindo um conjunto de atividades bem definidas, algumas abordagens de verificação de software foram definidas na literatura.

A norma internacional *ISO/IEC 12207* (1998) define o processo de verificação como um processo de apoio cujo objetivo é determinar se os produtos de software desenvolvidos em uma determinada atividade atendem completamente os requisitos ou condições impostas a eles nas atividades anteriores. A norma considera que, para a eficácia de custo e desempenho, a verificação deve ser integrada, o quanto antes, ao processo que a utiliza. O processo definido nesta norma é composto de duas atividades:

1. Implementação do processo: refere-se a aspectos relativos ao plano de verificação e ao seu cumprimento; e,
2. Verificação: refere-se à execução da verificação.

A atividade de implementação do processo consiste das seguintes tarefas: (i) determinar se o projeto justifica um esforço de verificação; (ii) estabelecer um processo de verificação; (iii) determinar as atividades do ciclo de vida e os produtos de software que requerem verificação; e (iv) desenvolver e documentar um plano de verificação.

A segunda atividade do processo de verificação consiste das seguintes tarefas:

(i) Verificação do contrato: verifica se o fornecedor possui capacidade para satisfazer os requisitos, se esses são coerentes e abrangem as necessidades do usuário, se existem procedimentos adequados para manipular as mudanças nos requisitos, critérios de aceitação, dentre outros;

(ii) Verificação do processo: verifica se o planejamento do projeto e a atribuição de tempos estão adequados, se o que foi determinado no projeto está sendo seguido e está de acordo com o contrato, se a equipe possui treinamento, dentre outros;

(iii) Verificação dos requisitos: verifica se os requisitos do sistema e do software são coerentes, são factíveis e testáveis e se os requisitos do software refletem com precisão os requisitos do sistema, dentre outros;

(iv) Verificação de projeto: verifica se o projeto está correto e coerente com os requisitos, se ele implementa apropriadamente as seqüências de eventos, entradas, saídas, requisitos de segurança com métodos rigorosos, dentre outros;

(v) Verificação do código: verifica se o código está de acordo com os requisitos, se é testável e correto, se está de acordo com os padrões de codificação, se implementa requisitos críticos e de segurança com métodos rigorosos, dentre outros;

(vi) Verificação da integração: verifica se os componentes e unidades de código, bem como itens de hardware e software foram integrados completa e corretamente de acordo com o plano de integração, dentre outros; e,

(vii) Verificação da documentação: verifica se a documentação está adequada, completa, coerente, se está sendo desenvolvida no prazo e se o gerenciamento de configuração dos documentos segue os procedimentos especificados.

Duas evoluções desta norma foram elaboradas até o momento. A primeira em 2002 (ISO/IEC, 2002) e a segunda em 2004 (ISO/IEC, 2004). A primeira evolução descreve os seguintes resultados esperados para uma implementação bem sucedida do processo de verificação: uma estratégia de verificação é desenvolvida e implementada; critérios para verificação de todos os produtos de trabalho requeridos são identificados; atividades de verificação requeridas são executadas; defeitos são identificados e registrados; e resultados de atividades de verificação são disponibilizados para o cliente

e outras partes envolvidas. A segunda evolução desta norma não trouxe qualquer alteração para o processo de verificação.

O *CMMI* (SEI, 2002) define a área de processo Verificação como uma área do nível de maturidade 3 – Definido. Essa área define que a verificação é um processo incremental, porque ocorre ao longo do desenvolvimento do produto e dos produtos de trabalho (artefatos), começando com a verificação dos requisitos, avançando com a verificação dos demais produtos de trabalho envolvidos, e finalizando com a verificação do produto final.

Uma área de processo do CMMI é dividida em objetivos e práticas. Os objetivos são componentes requeridos do modelo que descrevem as características que devem ser implementadas para satisfazer a área de processo. As práticas são componentes não obrigatórios, mas esperados, que são considerados importantes para se atingir o objetivo a que se referem. Essa área de processo possui os seguintes objetivos: (i) Preparar para a verificação; (ii) Realizar revisões por pares; e (iii) Verificar os produtos de trabalho selecionados. A figura 3.3 mostra os três objetivos da área de processo verificação e como estes se relacionam.

O primeiro objetivo - Preparar para a verificação - é composto de três práticas:

- *Selecionar Produtos de Trabalho para Verificação*: envolve identificar os produtos de trabalho para verificação, os requisitos a serem satisfeitos por cada produto de trabalho identificado, os métodos de verificação a serem usados para cada produto de trabalho identificado e integrar ao plano do projeto estas informações.
- *Estabelecer o Ambiente de Verificação*: envolve identificar os requisitos do ambiente de verificação, os recursos que estão disponíveis para reuso e modificação, os equipamentos e ferramentas de verificação, e adquirir o ambiente de apoio à verificação, o que pode incluir software e equipamentos de teste.
- *Estabelecer Procedimentos e Critérios de Verificação*: envolve identificar um conjunto abrangente e integrado de procedimentos de verificação para os produtos de trabalho, desenvolver e refinar, quando necessário, os critérios de verificação, e identificar os resultados esperados, tolerâncias permitidas na observação e outros critérios para satisfazer os requisitos.

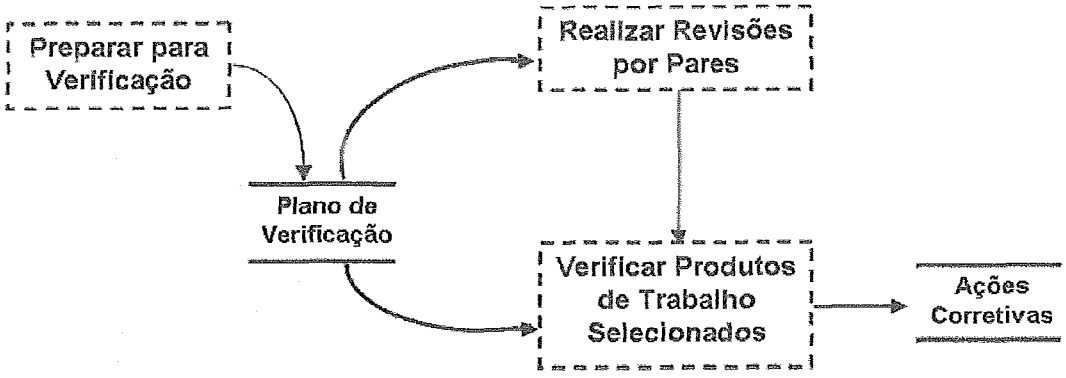


Figura 3.3 – Área de Processo Verificação do CMMI (SEI, 2002)

Três práticas compõem o objetivo Realizar revisões por pares:

- *Preparar Revisões por Pares*: envolve determinar qual o tipo de revisão será conduzida, definir requisitos para coleta de dados durante a revisão, estabelecer e manter critérios de entrada e saída para a revisão, critérios para requisitar nova revisão e *checklists* para garantir que os produtos de trabalho sejam revisados de forma consistente, desenvolver um cronograma de revisões, garantir que os produtos de trabalho satisfazem os critérios de entrada das revisões antes de sua distribuição aos revisores, definir papéis para os participantes das revisões e preparar a revisão por pares revisando os produtos de trabalho.
- *Conduzir Revisões por Pares*: envolve atribuir os papéis aos participantes da revisão por pares, identificar e documentar defeitos e outras questões pertinentes nos produtos de trabalho, registrar os resultados da revisão, incluindo os itens de ação, coletar dados da revisão por pares, conduzir uma revisão por pares adicional, se necessário, e garantir que os critérios de saída da revisão estão satisfeitos.
- *Analisar Dados das Revisões por Pares*: envolve registrar dados relacionados à preparação, condução e resultados das revisões por pares, armazenar os dados para referências e análises futuras, proteger os dados contra o uso indevido e analisar os dados da revisão por pares.

Por fim, o terceiro objetivo, Verificar os produtos de trabalho selecionados, é composto de duas práticas. São elas:

- *Realizar Verificação*: envolve realizar a verificação dos produtos de trabalho selecionados, armazenar os resultados da verificação, identificar itens de ação

resultantes da verificação dos produtos e documentar a execução dos métodos de verificação.

- *Analisar Resultados da Verificação e Identificar Ações Corretivas*: envolve comparar os resultados reais obtidos com os resultados esperados, identificar produtos que não atendam aos requisitos estabelecidos, identificar problemas com os métodos, critérios e ambiente de verificação, analisar os dados da verificação procurando defeitos, armazenar os resultados da análise em um relatório, disponibilizar informação sobre como os defeitos podem ser corrigidos e formalizá-los em um plano.

O modelo de referência *MPS.BR* (MPS.BR, 2005) define o processo de verificação como um dos processos do nível de maturidade D. Este processo tem como propósito confirmar que cada serviço e/ou produto de trabalho do processo ou do projeto reflete apropriadamente os requisitos especificados. Os resultados esperados da implementação bem sucedida deste processo são:

- VER 1 - Uma estratégia de verificação é desenvolvida e implementada, incluindo o ambiente necessário e o método para verificação;
- VER 2 - Critérios para verificação de todos os produtos de trabalho requeridos são identificados e um ambiente para verificação é estabelecido;
- VER 3 - Atividades de verificação requeridas, incluindo revisões por pares, são executadas e os produtos são verificados em relação aos requisitos especificados;
- VER 4 - Resultados de atividades de verificação são analisados e disponibilizados para o cliente e outras partes envolvidas; e,
- VER 5 - Defeitos são identificados e registrados e ações corretivas são determinadas.

3.5 Conclusão

A garantia da qualidade tem se tornado cada vez mais fundamental no mercado atual, em que se busca altos níveis de satisfação do cliente, além de altos níveis de produtividade e de qualidade. Neste contexto, uma das abordagens mais utilizadas e promissoras para a avaliação da qualidade de software é a verificação. A verificação tem se mostrado efetiva não só para melhorar a qualidade dos produtos de software,

como também, tem trazido um aumento de produtividade e satisfação do cliente, como se pôde constatar neste capítulo.

Foi apresentada uma revisão da literatura sobre verificação de software, que buscou caracterizá-la e ressaltar sua importância nos projetos de software. Foram, também, descritos os principais métodos e técnicas a serem utilizados na verificação de software, bem como o tratamento dado à verificação de software nos principais modelos de maturidade e normas.

No próximo capítulo, descreve-se a Estação TABA e os ambientes de desenvolvimento de software orientados à organização, onde a ferramenta de apoio à verificação de software, um dos resultados deste trabalho, está integrada.

CAPÍTULO 4

A Estação TABA: seus Ambientes e Ferramentas

4.1 Introdução

Ambientes de Desenvolvimento de Software (ADS) são sistemas computacionais que apóiam o desenvolvimento, reparo e melhoria de produtos de software, e o controle e gerenciamento destas atividades (MOURA, 1992). Esses ambientes têm evoluído ao longo do tempo, buscando fornecer apoio mais amplo e efetivo aos desenvolvedores de software, de forma que metas como aumento da produtividade, melhoria da qualidade, diminuição de custos e diminuição do tempo para introdução no mercado possam ser alcançadas (VILLELA, 2004).

O Projeto TABA, iniciado em 1990, teve como objetivo inicial construir ambientes de desenvolvimento de software (ADS) centrados em processos e adequados a projetos com diferentes características. Em 1997, iniciou-se uma importante evolução da Estação TABA para torná-la um ambiente capaz de instanciar Ambientes de Desenvolvimento de Software Orientados a Domínio (ADSOD), considerando não apenas as características específicas dos projetos, mas também o domínio da aplicação. A motivação para essa evolução foi a identificação de que um dos principais problemas no desenvolvimento de software era o desconhecimento do domínio por parte dos desenvolvedores de software (OLIVEIRA, 1999).

Uma outra evolução dos ambientes TABA definiu e implementou o modelo para construção de Ambientes de Desenvolvimento de Software Orientados a Organização (ADSOrgs), que consideram, além das características do projeto e do domínio da aplicação, também as particularidades de organizações específicas e seu conhecimento organizacional (VILLELA, 2004).

O objetivo deste capítulo é descrever a Estação TABA, sua estrutura e as ferramentas atualmente disponíveis. A seção 4.2 descreve a Estação TABA e seus ambientes. A seção 4.3 descreve o estágio atual da Estação TABA e seus ambientes, enquanto a seção 4.4 descreve algumas ferramentas disponíveis nos ambientes TABA e que merecem destaque no contexto deste trabalho. Por fim, a seção 4.5 apresenta as conclusões e considerações finais deste capítulo.

4.2 A Estação TABA e seus Ambientes

Desde a primeira definição da Estação TABA, suas funções têm sido definidas, revistas e ampliadas (ROCHA *et al.*, 1990; TRAVASSOS, 1994; FALBO, 1998; OLIVEIRA, 1999). Atualmente, as principais funções da Estação TABA são (VILLELA, 2004; BIANCHI, 2005; BESSA, 2005):

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

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

(iii) Apoiar, através dos ADSOrg, o desenvolvimento e a manutenção de software, bem como a gerência destas atividades.

(iv) Apoiar a avaliação e melhoria dos Ambientes Configurados.

Essas funções são realizadas através dos seguintes ambientes (figura 4.1):

- **Meta-Ambiente**: ambiente que apóia a configuração de ambientes para organizações específicas;
- **Ambiente Configurado:** ambiente configurado a partir do meta-ambiente e que apóia a instanciação de ADSOrgs para projetos específicos; e,
- **Ambiente Instanciado (ADSOrg)**: ambiente de desenvolvimento de software instanciado a partir do Ambiente Configurado, que é utilizado durante o desenvolvimento de um projeto específico e que atende às características deste projeto.

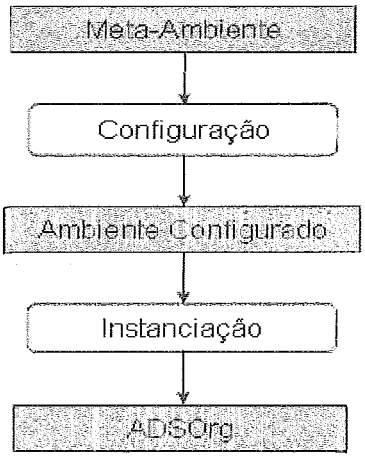


Figura 4.3 - Ambientes da Estação TABA (VILLELA, 2004)

Os ambientes da Estação TABA são utilizados em diferentes contextos e por profissionais de perfis distintos. O Meta-ambiente é utilizado em uma entidade externa à organização desenvolvedora de software por profissionais especializados em engenharia de software, que fazem a manutenção e evolução dos ativos de processos e demais recursos disponíveis neste ambiente. O principal processo que é executado neste ambiente é o de Configuração de ambientes, que gera os ambientes que serão utilizados nas organizações, os chamados Ambientes Configurados. Portanto a principal função do Meta-ambiente é produzir ambientes configurados para as organizações, conforme suas características individuais e de acordo com os processos que esta executa.

O Ambiente Configurado, gerado pelo meta-ambiente, é utilizado por profissionais de nível gerencial, pelo grupo de processos e pelo grupo de métricas da organização. O principal processo que ocorre neste ambiente é o de Instanciação, que resulta na geração dos ambientes que serão utilizados pelos projetos. Este processo leva em consideração as características individuais de cada projeto, portanto para cada projeto que é iniciado na organização é gerado um ADSOrg pelo ambiente configurado.

O Ambiente Instanciado (ADSOrg) para cada projeto é utilizado pela equipe do projeto. Este é o ambiente que dá suporte às várias atividades dos processos do ciclo de vida de um software.

4.2.1 Definição de Processos em Níveis

Como a Estação TABA é baseada em processos, os ambientes que a compõem são resultantes de uma estratégia de definição de processos em níveis (ROCHA *et al.*, 2001). De acordo com este modelo (figura 4.2), a definição de processos de software se realiza em três etapas: definição do processo padrão da organização, especialização do processo padrão e instanciação para projetos específicos. Como resultado, têm-se processos de software em diferentes níveis de abstração.

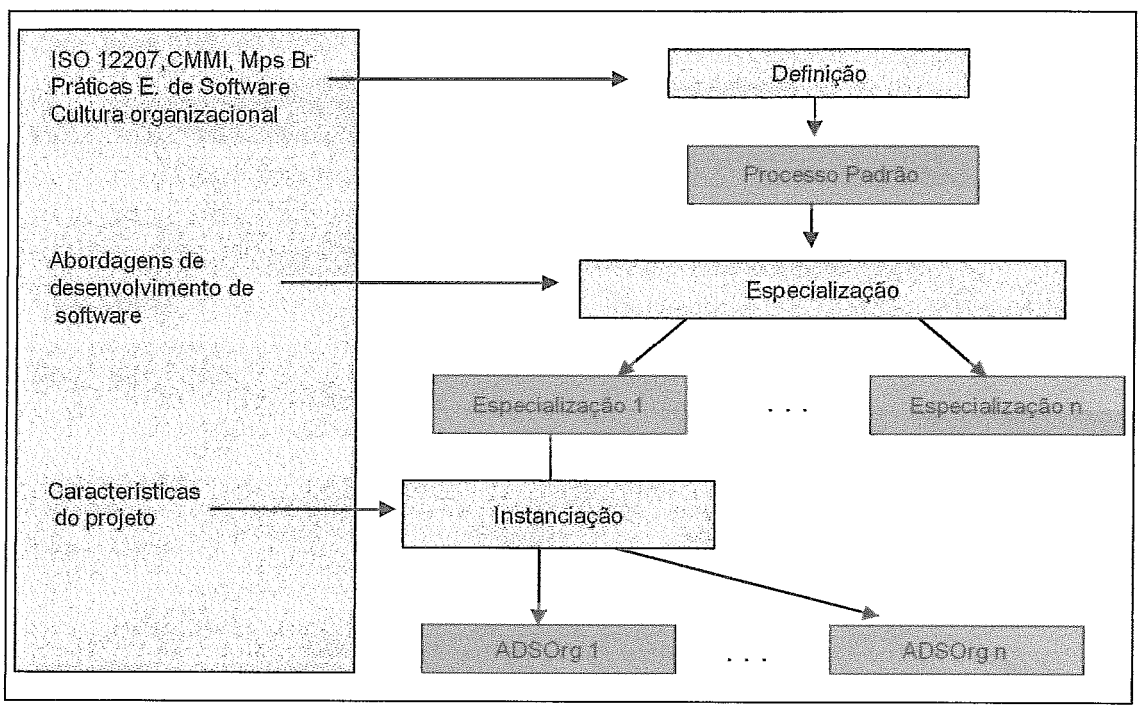


Figura 4.2 - Modelo para a definição de processos de software

A definição de um processo padrão estabelece uma estrutura comum a ser utilizada pela organização nos seus projetos de software e constitui a base para a definição de todos os processos. Dessa forma, estabelece-se um processo básico que servirá como ponto de partida para a posterior definição dos processos de software adequados às diferentes características de cada projeto, permitindo economia de tempo e esforço na definição de processos. O processo padrão é definido com o apoio da ferramenta *Config* no processo de Configuração de Ambientes (VILLELA, 2004), que ocorre no meta-ambiente.

Tendo em vista os diferentes paradigmas de desenvolvimento que possuem características distintas e requerem diferentes abordagens de desenvolvimento, e os

diversos tipos de software que podem ser desenvolvidos, o processo de software padrão da organização deve ser adaptado (especializado), considerando-se as características relacionadas ao paradigma de desenvolvimento utilizado (por exemplo: orientação a objetos). Assim, durante a etapa de especialização do processo padrão, atividades poderão ser adicionadas ou modificadas, de acordo com o contexto para o qual se está realizando a especialização. Os processos especializados também são definidos utilizando-se o processo de Configuração de Ambientes e a ferramenta *Config*.

Desta forma, os processos padrão e especializados são definidos no meta-ambiente, e ficam disponíveis para uso no ambiente configurado.

A instanciação para projetos específicos consiste na adaptação de um processo especializado para um determinado projeto, considerando-se as suas peculiaridades. Nesta etapa, são selecionados o modelo de ciclo de vida, os métodos e as ferramentas que serão utilizados no projeto, os recursos humanos e suas responsabilidades ao longo do processo e os artefatos consumidos e produzidos. As atividades do processo especializado são mapeadas para o modelo de ciclo de vida escolhido para o projeto, e novas atividades necessárias ao projeto podem ser inseridas no processo. O processo de instanciação ocorre no Ambiente Configurado com o apoio da ferramenta *AdaptPro* (BERGER, 2003). O processo instanciado é definido no ambiente configurado, e é executado em um projeto a partir dos ambientes instanciados.

4.3 Estágio Atual da Estação TABA

De 1990 a 2003, o projeto TABA teve objetivos puramente acadêmicos e de pesquisa. A partir de 2003, os ambientes TABA passaram a ser utilizados por empresas brasileiras desenvolvedoras de software o que vem fornecendo *feedback* ao projeto e criando novos desafios (MONTONI *et al.*, 2005; SANTOS, *et al.*, 2005). Os ambientes configurados e instanciados TABA estão em uso em várias empresas privadas e órgãos do governo, em diversos lugares do país, apoiando a implantação do CMMI e do MPS.BR.

Já é possível observar bons resultados do uso dos ambientes TABA na indústria. Alguns trabalhos destacam o uso desses ambientes como ponto forte para uma avaliação bem sucedida, sendo esses ambientes reconhecidos como facilitadores tanto pelas empresas quanto pela própria equipe de avaliação (ROCHA *et al.*, 2005a; ROCHA *et al.*, 2005b; FERREIRA *et al.*, 2005; NUNES *et al.* 2005). Mesmo estando em uso há

pouco tempo na indústria, empresas já alcançaram níveis de avaliação importantes com o apoio dos ambientes TABA e outras estão se preparando para obter avaliações bem sucedidas com o mesmo apoio. A tabela 4.1 apresenta esses resultados.

Tabela 4.1 – Número de empresas que já alcançaram níveis de avaliação ou estão se preparando para alcançá-los com o apoio dos ambientes TABA.

Avaliação	Empresas já avaliadas	Empresas se preparando para avaliação
CMMI Nível 2	3	1
CMMI Nível 3	-	3
MPS.BR Nível G	-	5
MPS.BR Nível F	1	-
MPS.BR Nível E	1	1
MPS.BR Nível D	-	1
MPS.BR Nível C	-	1

Neste contexto, a Estação TABA e seus ambientes estão em constante evolução. No estágio atual, a Estação TABA possui um conjunto de requisitos organizado em seis grupos (VILLELA, 2004; BIANCHI, 2005; BESSA, 2005):

(i) *Requisitos gerais, comuns a todos os ambientes*: possuir interface consistente, possuir um modelo comum de armazenamento de dados, apoiar a reutilização de conhecimento, o controle de versões e a gerência de configuração, possuir mecanismo de integração de ferramentas externas e possuir apoio para a avaliação do processo e para a melhoria contínua do processo.

(ii) *Requisitos específicos do meta-ambiente*: possuir mecanismo de integração de ferramentas internas, apoiar a definição de processos, gerar ambientes configurados para organizações específicas, permitir novas configurações sem perda do conhecimento organizacional e a incorporação de novos conhecimentos e experiências.

(iii) *Requisitos comuns ao meta-ambiente e aos ambientes configurados*: possuir conhecimento sobre processo de software e abordagens de desenvolvimento, permitir a descrição de tarefas, apoiar a definição de Teorias de Domínio e permitir a evolução das Teorias de Domínio que fazem parte do ambiente.

(iv) *Requisitos específicos dos ambientes configurados*: permitir a descrição da estrutura organizacional, dos profissionais da organização e dos processos organizacionais, apoiar a definição de processos para projetos específicos a

partir dos processos especializados, gerar ADSOrg para projetos específicos e apoiar a produção de conhecimento organizacional.

(v) *Requisitos comuns aos ambientes configurados e aos ambientes instanciados*: fornecer acesso ao conhecimento sobre o domínio da aplicação, apoiar o entendimento dos processos organizacionais, fornecer acesso ao conhecimento organizacional, permitir a evolução do modelo das organizações clientes envolvidas.

(vi) *Requisitos específicos dos ambientes instanciados (ADSOrg)*: apoiar a definição de arquiteturas de referência, o trabalho cooperativo e a execução do processo e a sua gerência, possuir suporte para a avaliação do produto, permitir a instanciação da Teoria do Domínio referente à aplicação, apoiar a localização de profissionais e a aquisição de conhecimento para a organização.

4.3.1 Conjunto Atual de Ferramentas da Estação TABA

O conjunto de ferramentas disponíveis na Estação TABA evolui constantemente buscando oferecer suporte mais efetivo às atividades relacionadas ao desenvolvimento e/ou manutenção de software e à gerência de processos. A tabela 4.2 relaciona as ferramentas atualmente disponíveis e passíveis de integração aos ambientes TABA.

Tabela 4.2 – Ferramentas disponíveis da Estação TABA

Ferramenta	Descrição
Acknowledge	Ferramenta para apoiar o processo de captura de conhecimento ao longo dos processos de software, englobando registro, filtragem e empacotamento de conhecimento (MONTONI, 2003; MONTONI *et al.*, 2003; MONTONI *et al*, 2004).
ActionPlanManager	Ferramenta para apoiar a elaboração de planos de ação ao longo do projeto.
AdaptPro	Ferramenta para apoiar a instanciação de processos de software para projetos específicos a partir dos processos especializados da organização (BERGER, 2003).
AvalPro	Ferramenta de apoio à Avaliação de Processo Instanciado compatível com o CMMI Nível 3 (ANDRADE, 2005).
Base de Métricas	Ferramenta que fornece uma estrutura capaz de comportar métricas, questões e objetivos previamente definidos para serem utilizadas durante a construção do plano de medição da organização ou de um projeto (ESTOLANO, 2005).

Tabela 4.2 (Continuação) – Ferramentas disponíveis da Estação TABA

Ferramenta	Descrição
Biblioteca de Ativos	Ferramenta de apoio à definição de uma Biblioteca de Ativos de Processo para organização, integrada à ferramenta de Gerência de Configuração.
Config	Ferramenta para apoiar a configuração de ambientes para as organizações (VILLELA, 2004).
ControlManager	Ferramenta para apoiar o planejamento do acompanhamento e controle do projeto.
DocPlan	Ferramenta para apoiar o planejamento da documentação a ser produzida em um projeto de software (MARTINS, 2004).
GeraDoc	Ferramenta para gerar documentos a partir da agregação de outros documentos (MARTINS, 2004).
Editar	Ferramenta para apoiar a definição de teorias de domínio e tarefas (OLIVEIRA, 1999; ZLOT, 2002; ZLOT *et al.*, 2002).
Edited	Ferramenta para apoiar a definição de teorias de domínio e tarefas (OLIVEIRA *et al.*, 2000).
GConf	Ferramenta para apoio à gerência de configuração dos artefatos produzidos em um projeto de software (FIGUEIREDO, 2004).
Gênesis	Ferramenta para apoiar a atividade de investigação do domínio (GALOTTA, 2000).
MBR	Ferramenta de apoio à escolha de fazer, comprar ou reutilizar um determinado componente durante o processo de desenvolvimento de software.
MedPlan	Ferramenta para apoiar o planejamento das medições no contexto da organização e do projeto (SCHNAIDER *et al.*, 2004).
Metrics	Ferramenta para apoiar a coleta de métricas baseada no plano de medição (SCHNAIDER *et al.*, 2004).
Navegue	Ferramenta para apoiar a atividade de investigação do domínio (GALOTTA, 2000).
OrgPlan	Ferramenta para elaboração do plano de organização para um projeto.
Pilot	Ferramenta de apoio à realização de projetos-piloto para avaliar melhorias propostas nos processos da organização.
Planilha de Atividades	Ferramenta para registro das atividades do desenvolvedor no projeto.
ProcKnow	Ferramenta de apoio à descrição dos processos organizacionais, sejam de software ou não (VILLELA, 2004).
ProjectStatus	Ferramenta que informa a situação dos projetos sendo conduzidos pela organização.
QFuzzy	Ferramenta para apoiar a identificação dos requisitos de qualidade de produtos (OLIVEIRA, 1999).
Regcon	Ferramenta para apoiar a atividade de investigação do domínio (GALOTTA, 2000).

Tabela 4.2 (Continuação) – Ferramentas disponíveis da Estação TABA

Ferramenta	Descrição
ReqManager	Ferramenta para apoiar a gerência de requisitos com a construção da matriz de rastreabilidade.
RHPlan e RHManager	Ferramentas para apoiar o planejamento, monitoração e avaliação da alocação de profissionais aos projetos de software, que inclui a solicitação de contratação e capacitação, o acompanhamento das horas dedicadas a cada atividade, além da atualização, ao final do projeto, das competências por eles possuídas (SCHNAIDER, 2003).
RiscManager	Ferramenta para apoiar o planejamento e a monitoração de riscos em projetos de software que baseia-se na reutilização do conhecimento organizacional sobre riscos (FARIAS, 2002; FARIAS *et al.*, 2003).
Sapiens	Ferramenta que permite a descrição e visualização de estruturas organizacionais, englobando os profissionais alocados e as competências requeridas e possuídas ao longo dessas estruturas (SANTOS, 2003; SANTOS *et al.*, 2003; SANTOS *et al.*, 2004).
TechSolution	Ferramenta de apoio a seleção de alternativas de solução na fase referente a Solução Técnica de um processo de desenvolvimento de software (FIGUEIREDO *et al.*, 2005).
TempPlan, TempManager, CustPlan e CustManager	Ferramentas para apoiar o planejamento e o controle de tempo e custo em projetos de software, baseadas na reutilização do conhecimento organizacional e nos modelos paramétricos COCOMO II e Análise de Pontos de Função (BARCELLOS, 2003; BARCELLOS *et al.*, 2003).
ValidationManager	Ferramenta para apoiar a validação de software ao longo do processo de desenvolvimento de software.
VerificationManager	Ferramenta para apoiar a verificação de software ao longo do processo de desenvolvimento de software (BARRETO e ROCHA, 2005).

Dentre essas ferramentas, algumas merecem destaque no contexto deste trabalho e serão detalhadas na próxima seção. A ferramenta *VerificationManager*, um dos resultados desta dissertação, será descrita no capítulo 6.

4.4 Ferramentas da Estação TABA Relacionadas a este Trabalho

Algumas ferramentas da Estação TABA possuem funcionalidades que estão fortemente associadas ao contexto deste trabalho e que, por isso, são descritas com mais detalhes a seguir.

O objetivo da ferramenta *AdaptPro* é apoiar a institucionalização dos processos padrão, facilitando a adoção destes processos em todos os projetos da organização. Ela auxilia na caracterização de um projeto e no planejamento do processo específico para o projeto guiando a adaptação do processo padrão da organização para o projeto com base em suas características. Esta ferramenta apóia todo o planejamento do processo para o projeto, incluindo a escolha do modelo de ciclo de vida para o projeto, o mapeamento das atividades do processo padrão para o modelo de ciclo de vida escolhido, a inclusão de atividades necessárias ao projeto, a exclusão de atividades não adequadas ao projeto e a seleção das ferramentas que irão apoiar a execução do processo planejado para o projeto. Após o planejamento, esta ferramenta instancia um ambiente específico para o projeto (ADSOrg) com base no processo planejado. Através desta ferramenta, o processo de verificação a ser adotado em um projeto pode ser adaptado às particularidades deste, uma vez que o processo de verificação está integrado ao processo de desenvolvimento padrão. A figura 4.3 apresenta a tela da *AdaptPro* que apóia a inclusão e exclusão de atividades no momento da adaptação do processo padrão para um projeto específico.

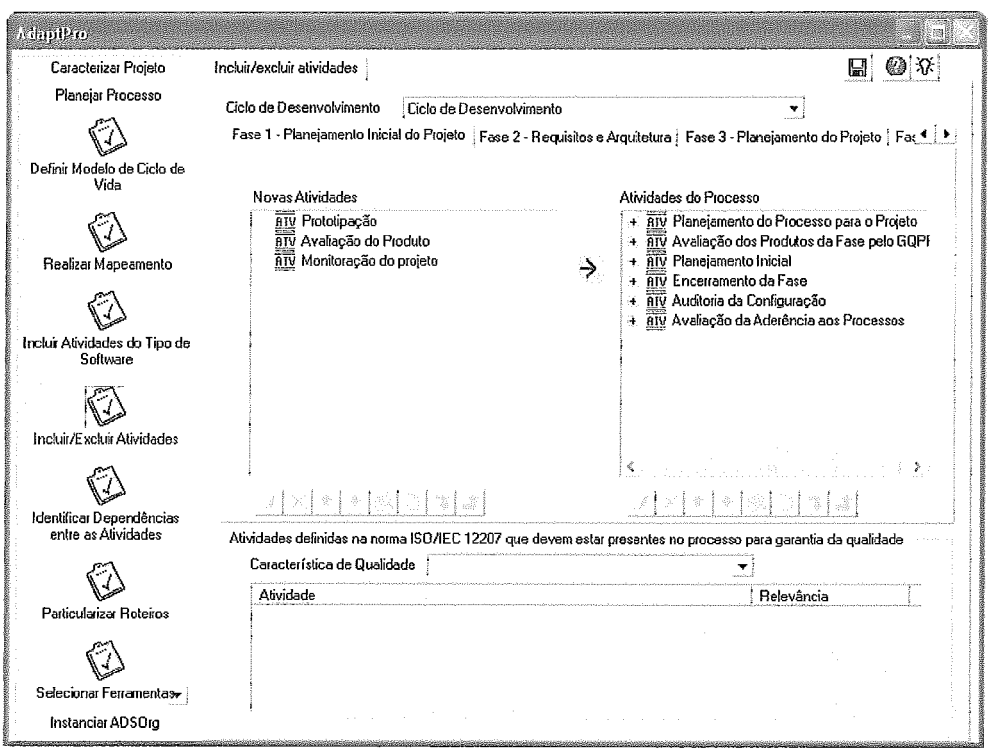


Figura 4.3 – AdaptPro – Uma ferramenta de apoio à adaptação do processo para projetos específicos

A ferramenta *QFuzzy* tem como objetivo apoiar a identificação dos requisitos de qualidade de produtos de software a serem desenvolvidos em um projeto. Para isso, a ferramenta disponibiliza o conjunto de características ou sub-características de qualidade definidas na norma ISO/IEC 9126 que serão avaliadas por um grupo de avaliadores escolhidos para o projeto específico. A cada projeto, considerando as particularidades do projeto em questão, os avaliadores determinam o grau de importância de cada característica para o produto a ser desenvolvido. A definição do perfil dos avaliadores é fundamental, pois se procura definir um peso para cada avaliador através de um Questionário de Identificação do Perfil de Especialista (QIPE). Após os avaliadores analisarem as características de qualidade e determinarem o grau de importância desejado para cada uma delas, a ferramenta consolida as diferentes opiniões e determina o grau de importância de cada característica de qualidade para o produto específico, utilizando o modelo *fuzzy* (BELCHIOR, 1997) que considera o grau de importância dado por cada avaliador e o peso do respectivo avaliador. A figura 4.4 exibe a tela de seleção dos avaliadores escolhidos para um dado projeto.

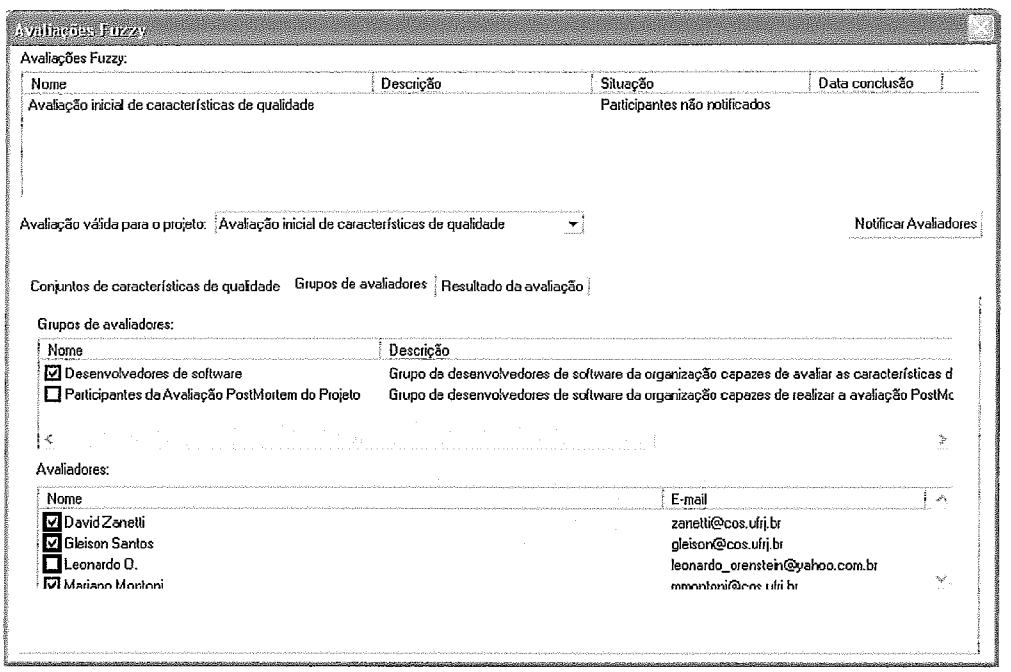


Figura 4.4 – QFuzzy – Uma ferramenta de apoio à identificação dos requisitos de qualidade para produtos de software.

As características de qualidade identificadas a partir da ferramenta *QFuzzy*, bem como o grau de importância dado a cada uma delas, auxiliam na determinação de quais

características de qualidade e critérios de avaliação devem ser usados na verificação de um determinado artefato.

Um dos requisitos da Estação TABA é apoiar a reutilização de conhecimento. Com esse objetivo a ferramenta *Acknowledge* foi desenvolvida. Ela apóia o processo de captura de conhecimento ao longo dos processos de software, englobando registro, filtragem e empacotamento de conhecimento. Através dessa ferramenta, o conhecimento adquirido por um usuário ao executar uma verificação pode ser registrado e posteriormente ser reutilizado, através da mesma ferramenta, por outros usuários ao executar a mesma atividade. O uso dessa ferramenta durante a verificação possibilita que o conhecimento sobre verificação de software acumulado pela organização e relevante ao desenvolvimento de software seja consultado e que novos conhecimentos adquiridos sejam armazenados. A ferramenta permite, também, o registro de idéias e lições aprendidas ao longo do processo de desenvolvimento. A consulta de conhecimento é acessível através do ícone enquanto o registro de conhecimento é acessível através do ícone . Ambos os ícones estão disponíveis em todas as ferramentas dos ADSOrgs no canto superior direito da tela. A figura 4.5 apresenta uma tela da ferramenta de apoio à verificação com os ícones para a consulta e o registro de conhecimento disponíveis.

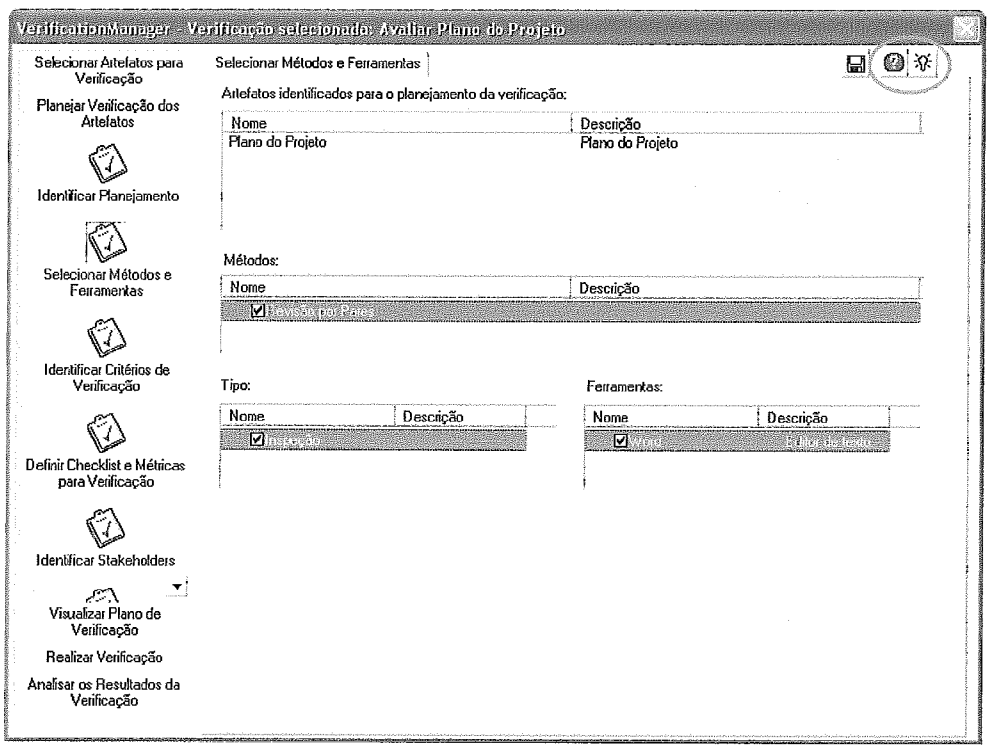


Figura 4.5 – Interface com a Acknowledge – Uma ferramenta de apoio à captura e disseminação do conhecimento.

Ao se realizar a verificação ao longo do projeto, alguns problemas podem ser encontrados e ações corretivas devem ser executadas com o objetivo de resolvê-los. As ações corretivas devem ser gerenciadas, ou seja, criadas, acompanhadas e executadas, e podem ser agrupadas em planos de ação. Para apoiar a gerência das ações corretivas, a Estação TABA dispõe da ferramenta *ActionPlanManager* que apóia a elaboração de planos de ação ao longo do projeto. Essa ferramenta é responsável por gerenciar os planos de ação definidos ao longo de todo o desenvolvimento do produto, permitindo a sua criação, a atribuição de responsabilidades para a sua execução, o acompanhamento e o registro de execução dos planos de ação definidos. A figura 4.6 exibe a tela desta ferramenta que permite a definição de planos de ação.

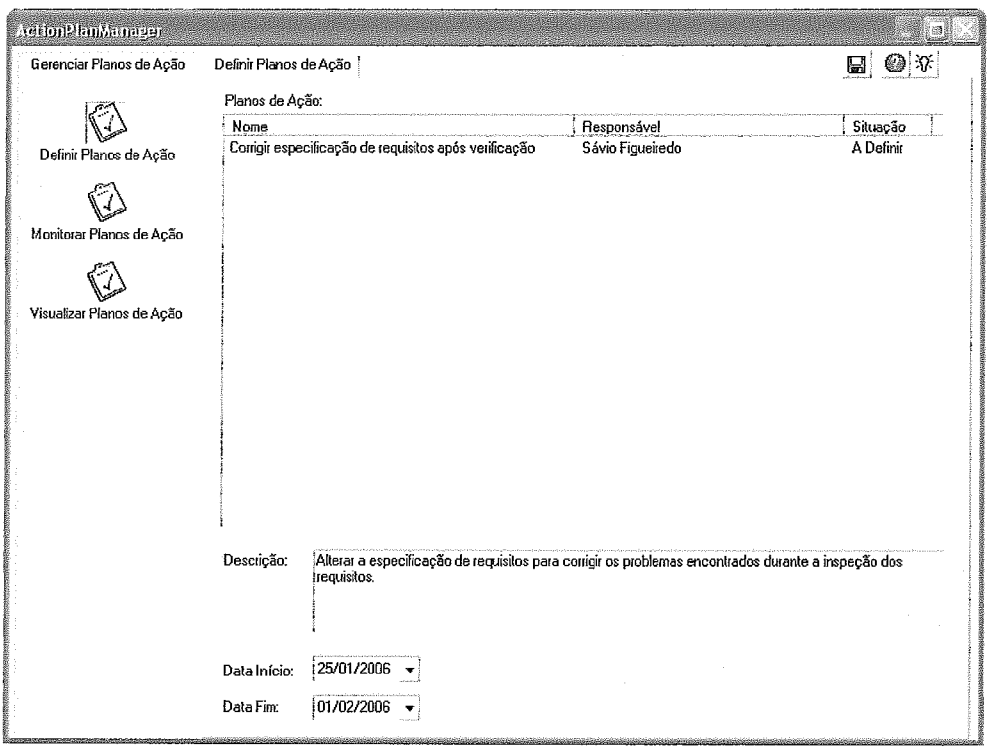


Figura 4.6 –ActionPlanManager – Uma ferramenta de apoio à definição e acompanhamento de planos de ação.

A verificação de software está fortemente relacionada à validação de software. Com o objetivo de apoiar também a validação de software, um outro trabalho está sendo desenvolvido. O trabalho apresenta uma abordagem para apoiar a validação de software, definindo um processo de validação, reunindo o conhecimento necessário à execução dessa atividade (NATALI, *et al.*, 2005). A ferramenta *ValidationManager* foi definida e

implementada como um apoio ferramental para a execução dessa atividade na Estação Taba. Essa ferramenta apóia o planejamento da validação, desde a identificação de quais artefatos devem ser validados até a definição de um plano completo para a realização da validação no projeto. A figura 4.7 exibe a tela desta ferramenta que apóia a identificação das características de qualidade e critérios a serem avaliados na validação do software, uma das etapas do planejamento da validação.

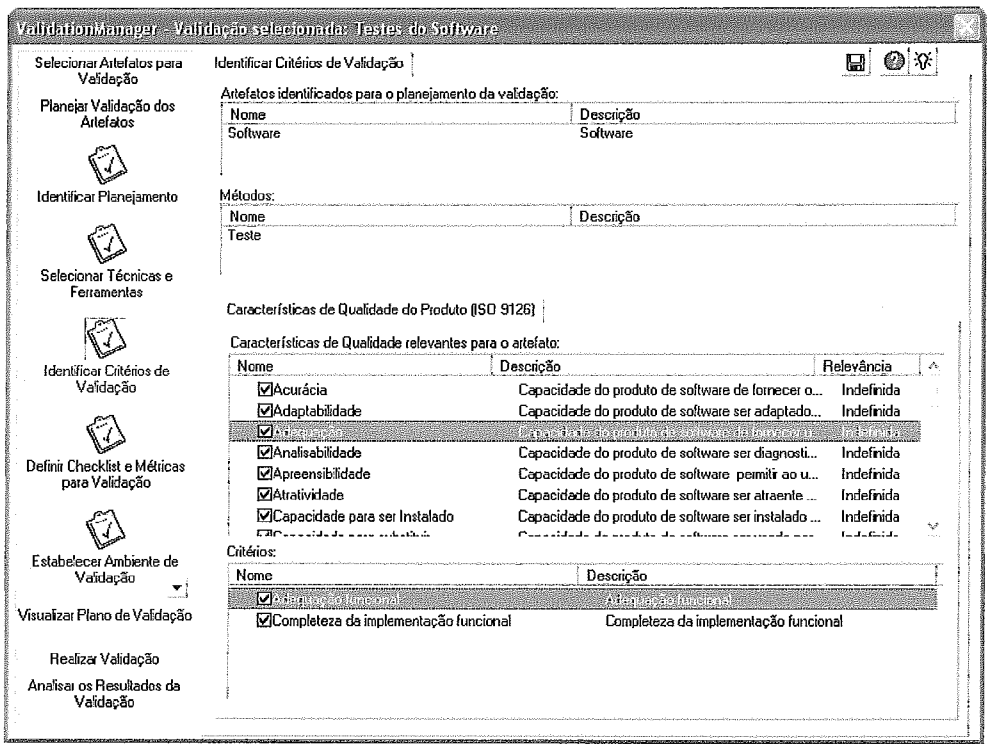


Figura 4.7 –ValidationManager – Uma ferramenta de apoio à validação de software.

4.5 Conclusão

Neste capítulo foi descrita a estação TABA em seu estágio atual. Foram descritos com mais detalhes as ferramentas atualmente disponíveis e que interagem com a ferramenta de apoio à verificação de produtos de software desenvolvida neste trabalho.

No próximo capítulo será apresentado o trabalho realizado com o objetivo de fornecer apoio às atividades relacionadas à verificação de software na Estação TABA e no capítulo 6 será descrita, em detalhes, a ferramenta *VerificationManager*.

CAPÍTULO 5

Uma Abordagem para Verificação em Projetos de Desenvolvimento de Software

5.1 Introdução

A verificação de software não é simples, uma vez que várias avaliações devem ser realizadas ao longo do processo de desenvolvimento e cada avaliação requer planejamento, controle e uso de técnicas de verificação adequadas. O planejamento da verificação deve ser realizado pelo gerente de projetos, sendo iniciado no planejamento do projeto e refinado ao longo do projeto. A execução da verificação deve ser realizada pela equipe do projeto baseada no planejamento realizado.

Tanto o planejamento quanto a execução da verificação são fortemente centrados na experiência e conhecimento adquiridos em projetos anteriores. Quanto maior a experiência do gerente e da equipe do projeto, mais eles serão capazes de planejar e executar a verificação para o projeto corrente. Para que a organização evolua aprendendo com seus próprios erros e acertos, é necessário que esse conhecimento seja gerenciado de forma a tornar possível sua captura, recuperação e futura utilização.

Baseado nos conceitos de verificação de software, este trabalho apresenta uma abordagem para verificação de software. Essa abordagem define um processo para modelar as atividades de verificação que devem ser realizadas ao longo do projeto e um mapeamento entre esse processo e o processo de desenvolvimento, indicando em que pontos do processo de desenvolvimento as atividades de verificação devem ser inseridas.

Com o objetivo de auxiliar na execução das atividades de verificação, a abordagem aqui apresentada sugere um conjunto de critérios, questões e métricas para verificação de artefatos produzidos ao longo do processo de desenvolvimento, baseando-se em dados da literatura.

A seção a seguir apresenta o processo de verificação definido e o conhecimento sugerido como apoio à execução de algumas atividades desse processo. A seção 5.3 apresenta a integração do processo de verificação definido ao processo de desenvolvimento. Por fim, a seção 5.4 apresenta as considerações finais deste capitulo.

5.2 O Processo de Verificação de Software Definido

O processo de verificação de software é composto de um conjunto de atividades e tarefas relacionadas ao planejamento da verificação e à execução da verificação de acordo com o planejamento realizado. Esse processo trata sistematicamente a verificação de artefatos produzidos ao longo de todo o ciclo de vida de um projeto de desenvolvimento de software.

O processo apresentado neste trabalho foi definido tendo como base a literatura de verificação de software. Dentre as principais referências podemos citar:

- A norma internacional ISO/IEC 12207 (1998), que define vários processos de ciclo de vida de software, dentre eles um processo para verificação de software;
- O CMMI (SEI, 2002) que define uma área de processo para verificação.

Desta forma, o processo foi definido de modo a ser aderente tanto à norma internacional ISO/IEC 12207 quanto ao CMMI. Além disso, como o Modelo Brasileiro de Melhoria de Processo de Software – MPS.BR (MPS.BR, 2005) é fortemente baseado na ISO/IEC 12207, na norma ISO/IEC 15504 e no CMMI, este processo também é aderente a esse modelo.

A descrição do processo definido apresenta os aspectos que devem ser considerados, os artefatos consumidos e gerados, os critérios de entrada e saída para cada atividade e os responsáveis pela realização de cada atividade. Deve-se observar que este processo poderá, ainda, ser adaptado à realidade de uma organização específica, considerando os documentos a serem gerados, a realização de sub-atividades e possíveis limitações ou restrições impostas.

Para descrever graficamente o processo foi utilizada uma linguagem para modelagem de processos organizacionais definida em (VILLELA, 2004). Uma simplificação dessa linguagem descrevendo os elementos gráficos utilizados é apresentada no Anexo I.

O processo de verificação de software definido envolve a execução de duas macro-atividades, conforme ilustrado na figura 5.1:

- Planejar a Verificação: o objetivo dessa macro-atividade é definir o plano de verificação para o projeto, identificando as atividades de verificação e os artefatos sujeitos à verificação; e,

- Executar a Verificação: o objetivo dessa macro-atividade é verificar os artefatos selecionados na macro-atividade anterior de acordo com o plano de verificação definido. A verificação de um artefato deve ser executada através de pelo menos uma revisão por pares ou um teste, de acordo com o que foi planejado.

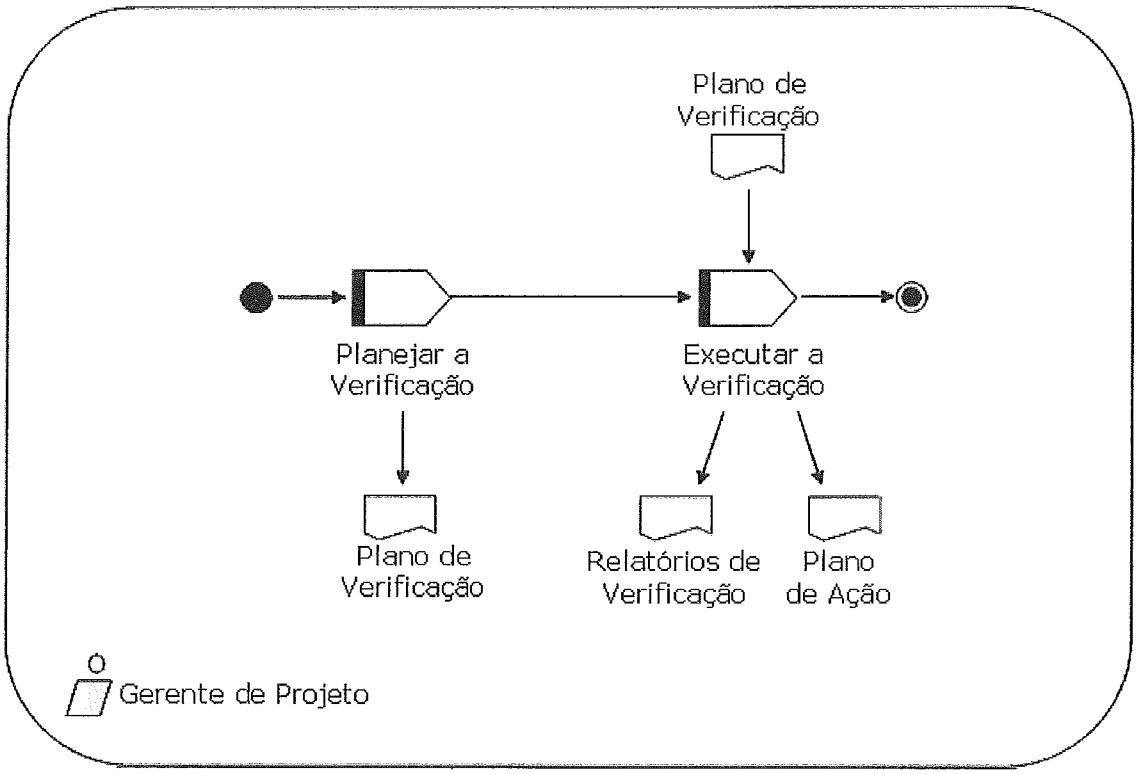


Figura 5.4 – Macro-atividades do Processo de Verificação (BARRETO e ROCHA, 2005)

A primeira macro-atividade do processo, **Planejar a Verificação**, pode ser dividida em duas atividades:

(i) Selecionar Artefatos para Verificação; e,

(ii) Planejar a Verificação do Artefato.

A figura 5.2 ilustra as atividades que compõem o planejamento da verificação.

(i) Selecionar Artefatos para Verificação

Descrição: Esta atividade consiste em analisar os artefatos que serão produzidos ao longo do desenvolvimento e selecionar aqueles que deverão ser verificados. Os artefatos devem ser selecionados com base em suas contribuições para o alcance dos objetivos e requisitos do projeto, considerando também os riscos do projeto.

Artefatos Requeridos: Lista de artefatos sujeitos à verificação.

Artefatos Produzidos: Plano de Verificação.

Critérios de Entrada: Não há.

Critérios de Saída: Artefatos selecionados para verificação.

Responsável: Gerente do Projeto.

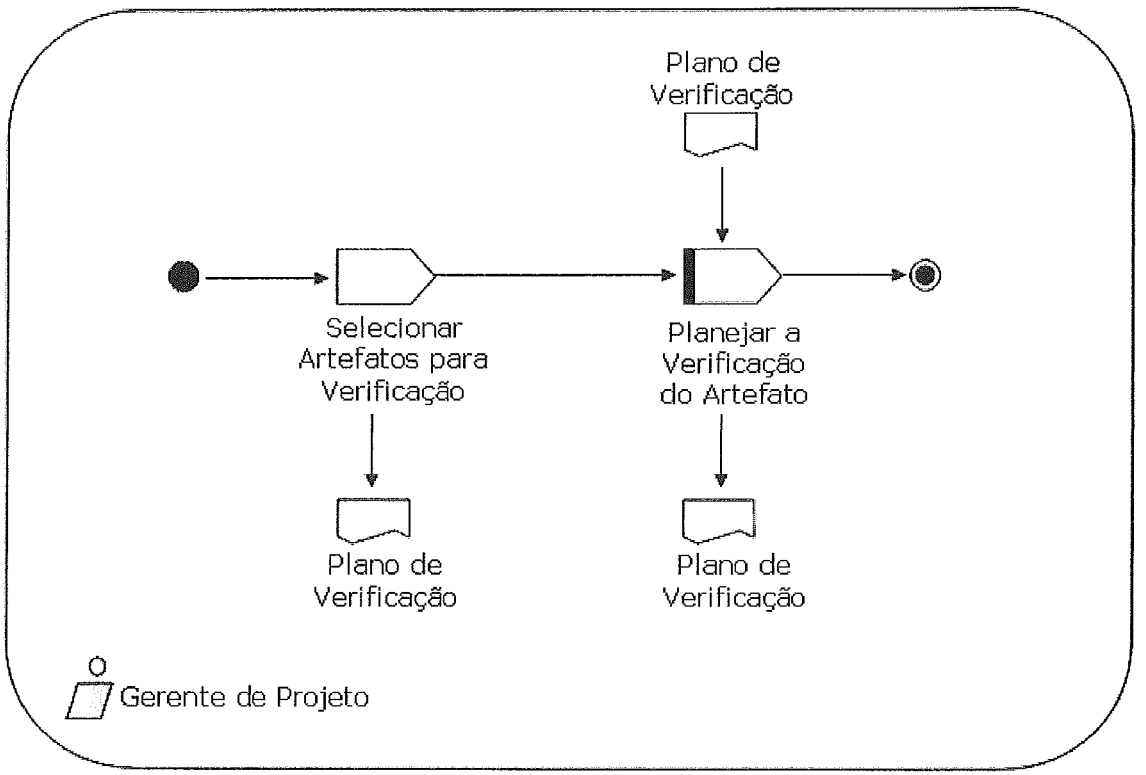


Figura 5.2 – Detalhamento da Macro-atividade Planejar a Verificação (BARRETO e ROCHA, 2005)

Para cada artefato selecionado na atividade (i), a atividade **Planejar a Verificação do Artefato** deve ser executada. O objetivo dessa atividade é planejar a verificação do artefato, identificando os métodos e as ferramentas a serem usados na verificação, o responsável e os participantes, os critérios para verificação do artefato, o modelo dos relatórios de verificação do artefato e os destinatários desses relatórios. Como mostra a figura 5.3, essa atividade pode ser dividida em cinco sub-atividades:

a. Selecionar Métodos e Ferramentas;
b. Identificar Critérios de Verificação;
c. Identificar Responsáveis;
d. Definir Destinatários dos Relatórios de Verificação; e,
e. Adaptar Modelo dos Relatórios de Verificação.

a. Selecionar Métodos e Ferramentas

Descrição: Esta atividade consiste em identificar os métodos de verificação que serão aplicados ao artefato, bem como as ferramentas que serão usadas para apoiar a verificação. De acordo com os métodos selecionados, a verificação do artefato será realizada através de revisão por pares e/ou testes.

Artefatos Requeridos: Lista de métodos e ferramentas disponíveis.

Artefatos Produzidos: Plano de Verificação.

Critérios de Entrada: Seleção dos artefatos para verificação concluída.

Critérios de Saída: Métodos e ferramentas selecionados.

Responsável: Gerente do Projeto.

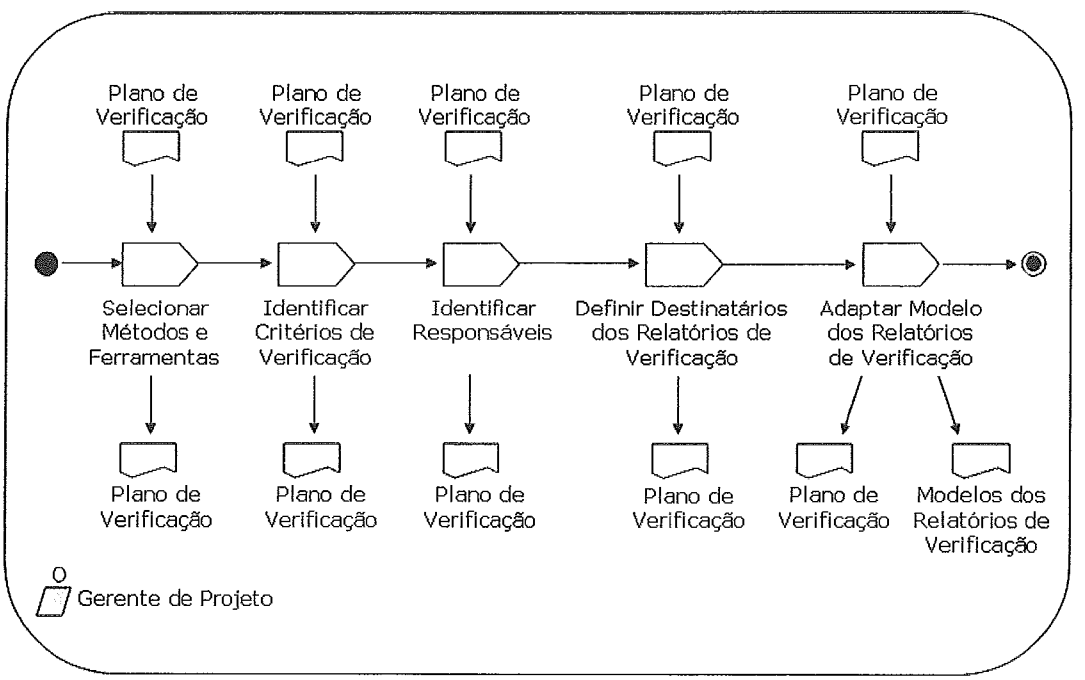


Figura 5.3 – Atividade Planejar a Verificação do Artefato (BARRETO e ROCHA, 2005)

b. Identificar Critérios de Verificação

Descrição: Esta atividade consiste em selecionar os critérios que serão usados para avaliar o artefato. Critérios de verificação são definidos para garantir que o artefato está de acordo com os requisitos estabelecidos para o mesmo.

Artefatos Requeridos: Lista de critérios disponíveis para a verificação.

Artefatos Produzidos: Plano de Verificação.

Critérios de Entrada: Seleção dos artefatos para verificação concluída.

Critérios de Saída: Critérios para verificação do artefato identificados.

Responsável: Gerente do Projeto.

c. **Identificar Responsáveis**

Descrição: *Esta atividade consiste em definir a pessoa que será responsável pela* verificação do artefato, bem como aquelas que participarão da avaliação.

Artefatos Requeridos: Lista de pessoas que participam do projeto.

Artefatos Produzidos: *Plano de Verificação.*

Critérios de Entrada: Seleção dos artefatos para verificação concluída.

Critérios de Saída: Responsável e participantes identificados.

Responsável: *Gerente do Projeto.*

d. **Definir Destinatários dos Relatórios de Verificação**

Descrição: *Esta atividade consiste em identificar as pessoas que, ao final da* verificação, receberão os relatórios de verificação do artefato.

Artefatos Requeridos: Lista de pessoas que participam do projeto.

Artefatos Produzidos: *Plano de Verificação.*

Critérios de Entrada: Seleção dos artefatos para verificação concluída.

Critérios de Saída: Destinatários do relatório identificados.

Responsável: *Gerente do Projeto.*

e. **Adaptar Modelo dos Relatórios de Verificação**

Descrição: *Esta atividade consiste em adaptar o modelo dos laudos de verificação* e/ou do relatório de testes que serão produzidos durante verificação do artefato de acordo com os critérios selecionados, com o objetivo de torná-los mais adequados ao projeto.

Artefatos Requeridos: Não há.

Artefatos Produzidos: Plano de Verificação.

Critérios de Entrada: *Critérios para verificação do artefato identificados.*

Critérios de Saída: Modelo dos relatórios adaptado.

Responsável: Gerente do Projeto.

Ao longo do planejamento da verificação, o plano de verificação contendo as informações do planejamento é produzido e atualizado. Esse plano é o principal guia para a execução da verificação.

Ao longo do projeto, os artefatos que foram selecionados para verificação devem ser verificados conforme planejado. Para isso a macro-atividade **Executar a Verificação**

deve ser realizada. A verificação de um artefato deve ser executada através de pelo menos uma revisão por pares ou um teste, de acordo com o que foi planejado. Essa macro-atividade se divide em quatro atividades como mostra a figura 5.4. São elas:

(i) Realizar Revisões por Pares;

(ii) Realizar Testes;

(iii) Enviar Relatórios de Verificação aos Destinatários; e,

(iv) Analisar os Resultados da Verificação.

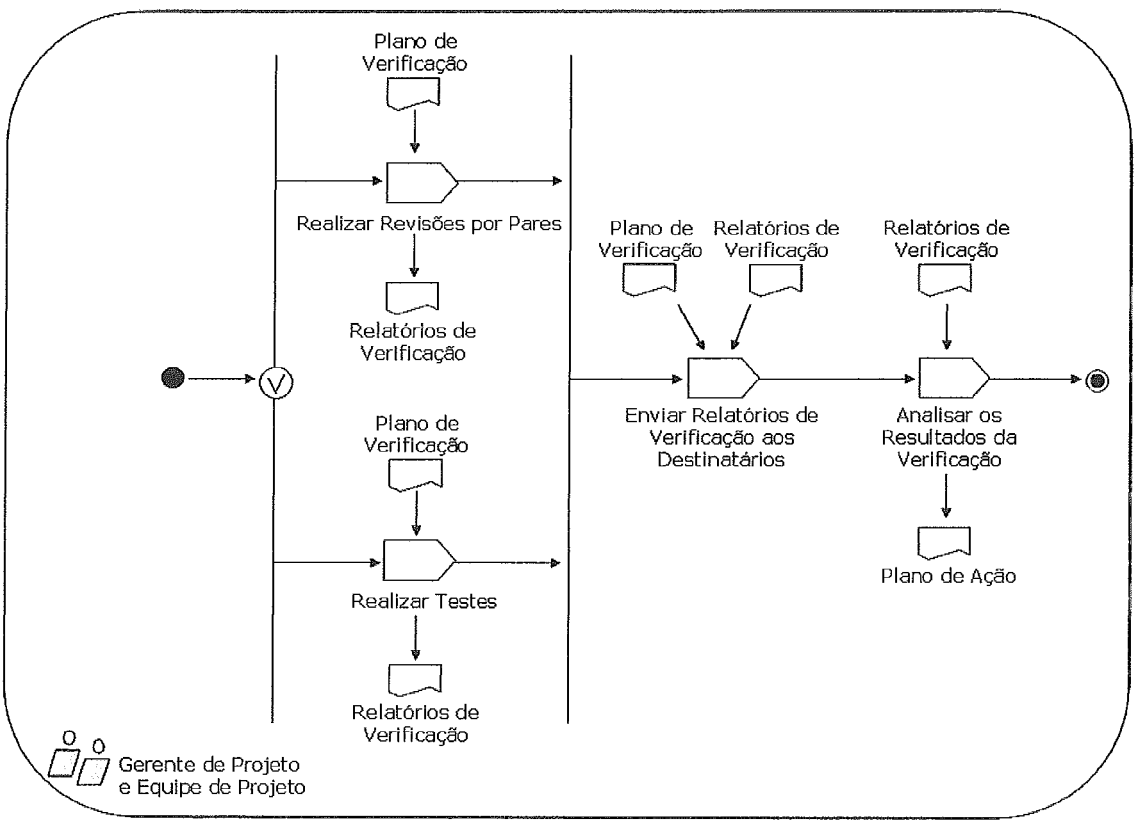


Figura 5.4 – Detalhamento da Macro-atividade Executar Verificação (BARRETO e ROCHA, 2005)

(iii) Realizar Revisões por Pares

Descrição: Esta atividade consiste em realizar as revisões por pares conforme descrito no plano de verificação em relação ao artefato em questão.

Artefatos Requeridos: Plano de Verificação, Artefato a ser verificado.

Artefatos Produzidos: Laudo de Preparação Individual para Avaliação do Artefato, Laudo Final da Avaliação do Artefato.

Critérios de Entrada: Artefato finalizado.

Critérios de Saída: Artefato verificado.

Responsável: Equipe do Projeto.

(iv) Realizar Testes

Descrição: *Esta atividade consiste em realizar os testes adequados para verificação* do artefato em questão conforme descrito no plano de verificação. Podem ser realizados: o teste unitário, o teste de integração do software, o teste do software, o teste do sistema, o teste de aceitação, dentre outros.

Artefatos Requeridos: Plano de Verificação, Artefato a ser verificado.

Artefatos Produzidos: Relatório de Testes.

Critérios de Entrada: Artefato finalizado.

Critérios de Saída: Artefato verificado.

Responsável: Equipe do Projeto.

(v) Enviar Relatórios de Verificação aos Destinatários

Descrição: Esta atividade consiste em encaminhar os relatórios de verificação produzidos nas duas atividades anteriores aos destinatários definidos no planejamento da verificação do artefato em questão.

Artefatos Requeridos: Plano de Verificação, Relatórios de Verificação.

Artefatos Produzidos: Não há.

Critérios de Entrada: Verificação do artefato concluída.

Critérios de Saída: Relatórios de verificação enviados aos destinatários.

Responsável: Equipe do Projeto.

(vi) Analisar os Resultados da Verificação

Descrição: Esta atividade consiste em analisar os resultados da verificação, documentados nos relatórios de verificação e, a partir dessa análise, identificar os itens de ação necessários e registrá-los para posterior execução.

Artefatos Requeridos: Relatórios de Verificação.

Artefatos Produzidos: Lista dos itens de ação.

Critérios de Entrada: Verificação do artefato concluída.

Critérios de Saída: Resultados da verificação analisados e itens de ação identificados.

Responsável: Equipe do Projeto.

Conforme descrito no Capítulo 3, a aplicação dos métodos de verificação requer um *planejamento específico para cada método. Esse planejamento* é uma tarefa suficientemente complexa que tem sido investigada por diversos trabalhos (LIMA, 2005; KALINOWSKI *et al.*, 2004; LANUBILE *et al.*, 2003; D'OLIVEIRA, 2003; BINDER, 2001). Por esse motivo, o planejamento específico para a aplicação dos métodos selecionados para a verificação de um determinado artefato não será tratado neste trabalho.

Um fator importante para a execução de um processo é a definição das responsabilidades por cada atividade a ser executada. A tabela 5.1 apresenta um resumo das responsabilidades descritas no processo de verificação definido.

Tabela 5.1. Responsáveis pelas atividades do processo de verificação definido (BARRETO e ROCHA, 2005)

Atividade do Processo de Verificação	Gerente do Projeto	Equipe do Projeto
Selecionar Artefatos para Verificação	✓	
Planejar a Verificação do Artefato	✓	
Realizar Revisões por Pares		✓
Realizar Testes		✓
Enviar Relatórios de Verificação aos Destinatários		✓
Analisar os Resultados da Verificação		✓

Uma das vantagens do processo definido neste trabalho é a sua aderência à área de processo Verificação definida no modelo CMMI (SEI, 2002) e ao processo de Verificação definido no MPS.BR (MPS.BR, 2005). A tabela 5.2 relaciona as atividades do processo definido com as práticas específicas da área de processo verificação definida no modelo, ilustrando a cobertura do processo definido com relação às práticas da área de processo Verificação do CMMI. Já a tabela 5.3 relaciona as atividades do processo definido com os resultados esperados pelo processo de verificação definido no MPS.BR.

Tabela 5.2. Relacionamento entre o processo definido e a área de processo verificação do CMMI (BARRETO e ROCHA, 2005)

Atividade do Processo / Prática Especifica	Selecionar Artefatos para Verificação	Planejar Verificação do Artefato	Realizar Revisões por Pares	Realizar Testes	Enviar Relatórios de Verificação aos Destinatários	Analisar os Resultados da Verificação
SP 1.1 - Selecionar Produtos de Trabalho para Verificação	✓					
SP 1.2 - Estabelecer o Ambiente para Verificação		✓				
SP 1.3 - Estabelecer Procedimentos e Critérios para Verificação		✓	✓			
SP 2.1 - Preparar para Revisões por Pares			✓			
SP 2.2 - Conduzir Revisões por Pares			✓			
SP 2.3 – Analisar Dados das Revisões por Pares					✓	✓
SP 3.1 – Realizar a Verificação			✓	✓		
SP 3.2 - Analisar os Resultados da Verificação e Identificar Ações Corretivas					✓	✓

Tabela 5.3. Relacionamento entre o processo definido e o processo de verificação do MPS.BR

Atividade do Processo / Resultados Esperados	Selecionar Artefatos para Verificação	Planejar Verificação do Artefato	Realizar Revisões por Pares	Realizar Testes	Enviar Relatórios de Verificação aos Destinatários	Analisar os Resultados da Verificação
VER 1. Uma estratégia de verificação é desenvolvida e implementada, incluindo o ambiente necessário e o método para verificação	✓	✓				
VER 2. Critérios para verificação de todos os produtos de trabalho requeridos são identificados e um ambiente para verificação é estabelecido		✓				
VER 3. Atividades de verificação requeridas, incluindo revisões por pares, são executadas e os produtos são verificados em relação aos requisitos especificados			✓	✓		
VER 4. Resultados de atividades de verificação são analisados e disponibilizados para o cliente e outras partes envolvidas					✓	✓
VER 5. Defeitos são identificados e registrados e ações corretivas são determinadas						✓

5.3 Um Conjunto de Critérios para Verificação

A verificação de software é realizada através de avaliações de artefatos produzidos ao longo do processo de desenvolvimento. Para avaliar cada um desses artefatos é preciso definir um conjunto de critérios, em relação aos quais os artefatos serão avaliados. Devido a essa necessidade, este trabalho apresenta, como uma sugestão, um conjunto de critérios para a verificação de alguns artefatos. Este conjunto foi definido a partir de uma seleção da literatura existente, baseada no conhecimento de especialistas e buscando reunir o conhecimento que poderia ser mais facilmente aplicável à indústria. Vale salientar que estes critérios representam um conjunto que, apesar de ser baseado na literatura, ainda não foi avaliado através de experimentos formais. Apesar disso, ele já está em uso na indústria, o que possibilitará a sua avaliação formal e identificação de possíveis adequações e melhorias.

O conjunto definido apresenta os critérios para verificação divididos em dois níveis: o primeiro nível define uma lista de características de qualidade que devem estar presentes no artefato avaliado ou no produto final. Em linhas gerais, características de qualidade são atributos que determinam a qualidade do produto sob o ponto de vista de seus diferentes usuários. O segundo nível consiste de uma lista de critérios definidos para cada característica de qualidade, que auxiliam na determinação da presença da característica.

Com o objetivo de ajudar a determinar se um critério foi ou não atendido, foram definidas questões e métricas para os critérios. Esse conjunto foi definido a partir de uma pesquisa na literatura que buscou relacionar características de qualidade, critérios, questões e métricas para avaliação de características de qualidade (McCONNELL, 2004; MYERS, 2004; RAKITIN, 2001; ISO/IEC, 2001; OLIVEIRA, 1999; ISO/IEC, 1998; SPAWAR, 1997; BEAUFOND *et al.*, 1997; HUMPHREY, 1997; PORTER *et al.*, 1995; NASA, 1993; MAGUIRE, 1993; ACKERMAN *et al.*, 1989; HUMPHREY, 1989; DUNN, 1984; FREEDMAN e WEINBERG, 1982).

As características de qualidade presentes no conjunto definido foram organizadas em dois grupos:

- Características de Qualidade presentes na norma internacional ISO/IEC 9126 (ISO/IEC, 2001); e,

- Características Gerais de Qualidade. Neste grupo estão as características comuns a *qualquer software, que não estão presentes na ISO/IEC 9126, mas que foram* definidas a partir dos diversos trabalhos analisados.

As tabelas 5.4 e 5.5 apresentam exemplos de características de qualidade da ISO/IEC 9126 e de *características de qualidade gerais, respectivamente,* usadas neste trabalho. Mais detalhes sobre essas características são descritos no Anexo II.

Tabela 5.4. Exemplo de características de qualidade da ISO/IEC 9126

Característica de Qualidade	Sub-Característica de Qualidade
Funcionalidade	Adequação
	Acurácia
	Interoperabilidade
	Segurança
	Conformidade de funcionalidade
Manutenibilidade	Analisabilidade
	Modificabilidade
	Estabilidade
	Testabilidade

Tabela 5.5. Exemplo de características de qualidade gerais

Característica de Qualidade	Sub-Característica de Qualidade
Fidedignidade	Completeza
	Consistência
Legibilidade	Clareza
	Correção da representação

O conjunto definido para este trabalho é apresentado detalhadamente no Anexo II. Para exemplificar, as tabelas 5.6 e 5.7 apresentam uma parte desse conjunto para a especificação de requisitos do software e para o código de unidades, respectivamente.

No contexto deste trabalho, os seguintes artefatos foram considerados:

- Plano do Projeto;
- Especificação de Requisitos do Cliente;
- Especificação de Requisitos do Sistema;
- Projeto da Arquitetura do Sistema;
- Especificação de Requisitos do Software;
- Modelo de Análise;
- Modelo de Projeto;
- Modelo de Análise & Projeto: Análise e projeto do software juntos em um único artefato;

- Plano de Testes de Unidades;
- Código de Unidades;
- Relatório do Teste de Unidades;
- Plano de Testes de Integração do Software;
- Relatório do Teste de Integração do Software;
- Relatório do Teste de Integração do Sistema;
- Plano de Testes do Software;
- Software;
- Relatório do Teste do Software;
- Plano de Testes de Integração do Sistema;
- Plano de Testes do Sistema;
- Sistema; e,
- Relatório do Teste do Sistema.

Tabela 5.6. Exemplo do conjunto para verificação da especificação de requisitos do software

Artefato	Método	Característica de Qualidade	Critério	Checklist
Especificação de Requisitos do Software	Revisão por Pares	Clareza	Clareza	O significado de cada requisito é compreensível?
				Os requisitos estão descritos com um nível de detalhes suficiente para o entendimento?
				Os requisitos podem ser entendidos e desenvolvidos por um grupo independente?
		Consistência	Consistência Interna	Os requisitos são consistentes entre si?
			Consistência Externa	Os requisitos do software são consistentes com os requisitos do cliente?
				Os requisitos do software são consistentes com os requisitos do sistema?
		Testabilidade	Viabilidade de testes	Cada requisito é testável?
		Adequação	Adequação às necessidades do cliente	Os requisitos descritos são adequados às *necessidades do cliente*?
		Segurança	Controle de acesso	Todos os usuários do software estão identificados?
				Os requisitos de segurança estão especificados?

Além de avaliar se um determinado critério foi satisfeito, é importante também, garantir que todas as características de qualidade desejáveis ao produto final foram avaliadas em algum momento ao longo do desenvolvimento do produto. Para facilitar a análise dessa cobertura das características de qualidade desejáveis, foi definido também um mapeamento entre as características de qualidade aqui consideradas e os artefatos

nos quais cada uma poderia ser avaliada. Esse mapeamento também é apresentado no Anexo II. *Como exemplo temos que a característica de qualidade Adequação*, definida na ISO/IEC 9126 como uma sub-característica de *Funcionalidade*, poderia ser avaliada nos seguintes artefatos:

- Especificação de Requisitos do Cliente;
- Especificação de Requisitos do Sistema;
- Especificação de Requisitos do Software;
- Código de Unidades;
- Plano de Testes do Software; e,
- Software.

Tabela 5.7. Exemplo do conjunto para verificação do código de unidades

Artefato	Método	Característica de Qualidade	Critério	Checklist	Métrica
Código de unidades	Revisão por Pares	Clareza	Clareza	O código é fácil de ler e entender?	
				O código possui comentários adequados e suficientes para o entendimento?	
				As mensagens de erro e códigos de retorno são claras e precisas?	
		Tolerância a defeitos	Impedimento de defeitos	Todas as prováveis condições de erro foram tratadas?	
	Teste	Maturidade	Maturidade dos testes		Maturidade do teste: X=A/B A= Número de casos de teste aprovados B= Número de casos de teste executados
		Comportamento em relação ao tempo	Tempo de resposta		

5.4 Uma Abordagem para Integração do Processo de Verificação Definido ao Processo de Desenvolvimento

O processo de verificação descrito na seção 5.2 não é executado de forma seqüencial, pois ele é dependente do processo de desenvolvimento e, por isso, sua execução deve ser integrada à execução do processo de desenvolvimento. A figura 5.5

apresenta um exemplo de execução do processo de verificação integrado ao processo de desenvolvimento.

A primeira atividade (*Selecionar Artefatos para Verificação*) deve ser executada durante o planejamento do projeto. As demais atividades devem ser executadas várias vezes ao longo do desenvolvimento do projeto, uma para cada artefato selecionado.

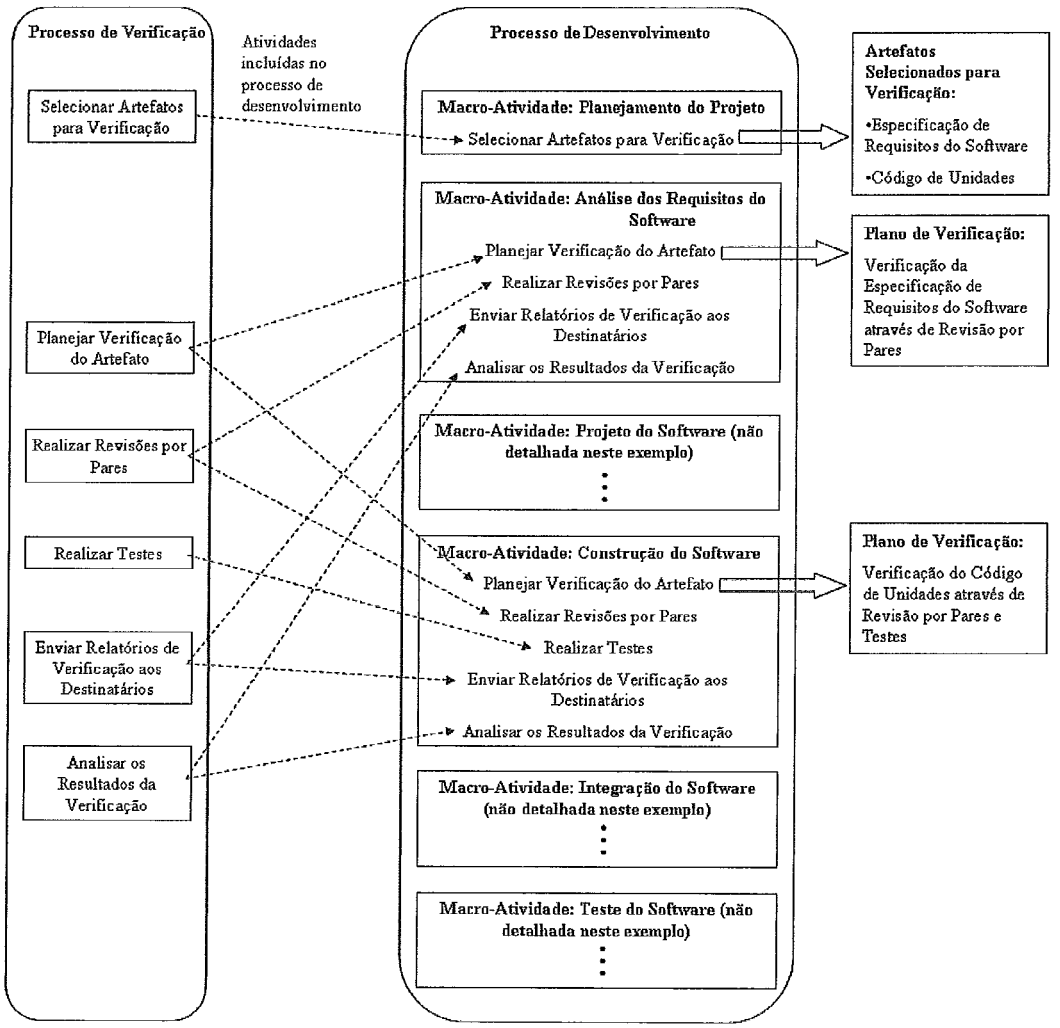


Figura 5.5 – Exemplo de Processo de Verificação integrado ao Processo de Desenvolvimento (BARRETO e ROCHA, 2005)

Ao longo do processo de desenvolvimento, os artefatos que foram selecionados para verificação serão produzidos e, para cada um desses artefatos, a atividade *Planejar a Verificação do Artefato* deve ser executada antes da produção do artefato. Após a produção do mesmo, as atividades *Realizar Revisões por Pares* e/ou *Realizar Testes* devem ser executadas conforme o planejamento da verificação. As atividades *Enviar*

Relatórios de Verificação aos Destinatários e *Analisar os Resultados da Verificação* também devem ser executadas após a realização da verificação, seja através de revisão por pares ou através de teste.

Uma abordagem para guiar o usuário na execução dessas atividades é incluí-las no processo de desenvolvimento. Desta forma o processo de desenvolvimento pode ser alterado dependendo da seleção dos artefatos para verificação e do planejamento da verificação de cada artefato.

5.5 Conclusão

Este capítulo apresentou uma abordagem para a verificação em projetos de software fundamentada nos conceitos de verificação de software presentes na literatura. A abordagem propõe a utilização de um processo que organiza as atividades de verificação a serem executadas e a integração desse processo ao processo de desenvolvimento de software. Além disso, foi sugerido um conjunto com informações sobre características de qualidade, critérios, questões e métricas que podem ser usados para auxiliar na execução das atividades de verificação.

Vale ressaltar que, apesar de este trabalho estar inserido no contexto dos Ambientes de Desenvolvimento de Software Orientados à Organização, disponíveis na Estação TABA, é de se esperar que o processo definido, a sua integração com o processo de desenvolvimento e o conjunto de critérios aqui sugeridos possam ser utilizados em um contexto mais geral, não ficando restritos apenas aos ADSOrgs.

No capítulo a seguir, serão descritos o uso e a implementação da abordagem apresentada neste capítulo nos Ambientes de Desenvolvimento de Software Orientados à Organização, disponíveis na Estação TABA.

CAPÍTULO 6
Apoio à Verificação de Software na Estação TABA: A Ferramenta *VerificationManager*

6.1 Introdução

A Estação TABA tem como objetivo instanciar Ambientes de Desenvolvimento de Software destinados a apoiar as atividades realizadas ao longo de um projeto. Para isso, deve-se, inicialmente, definir o processo de desenvolvimento a ser utilizado no projeto e, em seguida, instanciar o ambiente para apoiar a execução do processo. A abordagem para verificação de software proposta neste trabalho foi implementada na Estação TABA com o objetivo de introduzir nela a verificação de software, mais especificamente no ADSOrg, uma vez que, até então, não existiam ferramentas que fornecessem esse apoio nesses ambientes.

A verificação de software nos ADSOrgs visa a apoiar o planejamento e a execução da verificação ao longo do desenvolvimento. Como requisitos básicos a serem atendidos têm-se:

(i) Apoiar a identificação dos artefatos que devem ser verificados ao longo do projeto;

(ii) Apoiar o planejamento específico para a verificação de cada artefato identificado;

(iii) Apoiar o planejamento para a realização de revisões por pares;

(iv) Apoiar o planejamento para a realização de testes;

(v) Apoiar a realização de revisões por pares;

(vi) Apoiar a realização testes;

(vii) Apoiar o registro da execução da verificação realizada tanto através de revisão por pares quanto através de teste; e,

(viii) Apoiar o registro da análise dos resultados da verificação.

Estes requisitos foram definidos tendo como base as atividades de verificação que são realizadas ao longo de um projeto de software. No contexto deste trabalho, o planejamento específico para a aplicação da revisão por pares e do teste, e a execução destes dois métodos não serão tratados. Desta forma, dentre os requisitos básicos

enunciados acima, este trabalho se propõe a implementar os requisitos (i) *Apoiar a identificação dos artefatos que devem ser verificados ao longo do projeto*; (ii) *Apoiar o planejamento específico para a verificação de cada artefato identificado*; (vii) *Apoiar o registro da execução da verificação realizada tanto através de revisão por pares quanto através de teste* e (viii) *Apoiar o registro da análise dos resultados da verificação*.

A verificação no contexto de Ambientes de Desenvolvimento de Software Orientados à Organização pretende, além de atender os requisitos acima, disponibilizar ao gerente de projeto todo o conhecimento existente na organização sobre métodos e critérios para verificação de cada artefato. Além disso, as idéias, diretrizes, problemas, dúvidas e lições aprendidas relacionados à verificação de software e registradas pelos gerentes e equipes de projetos anteriores podem apoiar no planejamento e execução da verificação em novos projetos.

Este capítulo apresenta a verificação de software na Estação TABA, bem como o apoio automatizado à execução dessas atividades. A seção a seguir apresenta a implementação da abordagem proposta na Estação TABA, descrevendo como as atividades relacionadas à verificação serão realizadas nos ADSOrgs. Na seção 6.3, a ferramenta *VerificationManager*, definida e implementada para apoiar a execução dessas atividades, é apresentada. Por fim, na seção 6.4 apresentamos as considerações finais deste capítulo.

6.2 Implementação da Abordagem Proposta na Estação TABA

A abordagem proposta neste trabalho foi implementada na Estação TABA, considerando uma parte dos requisitos definidos para apoiar a verificação de software. Algumas atividades de verificação foram inseridas no processo de desenvolvimento e a partir dessas atividades é possível executar a ferramenta definida e implementada neste trabalho. A figura 6.1 apresenta uma parte de um processo de desenvolvimento usado na Estação TABA, onde se podem identificar atividades de verificação em negrito.

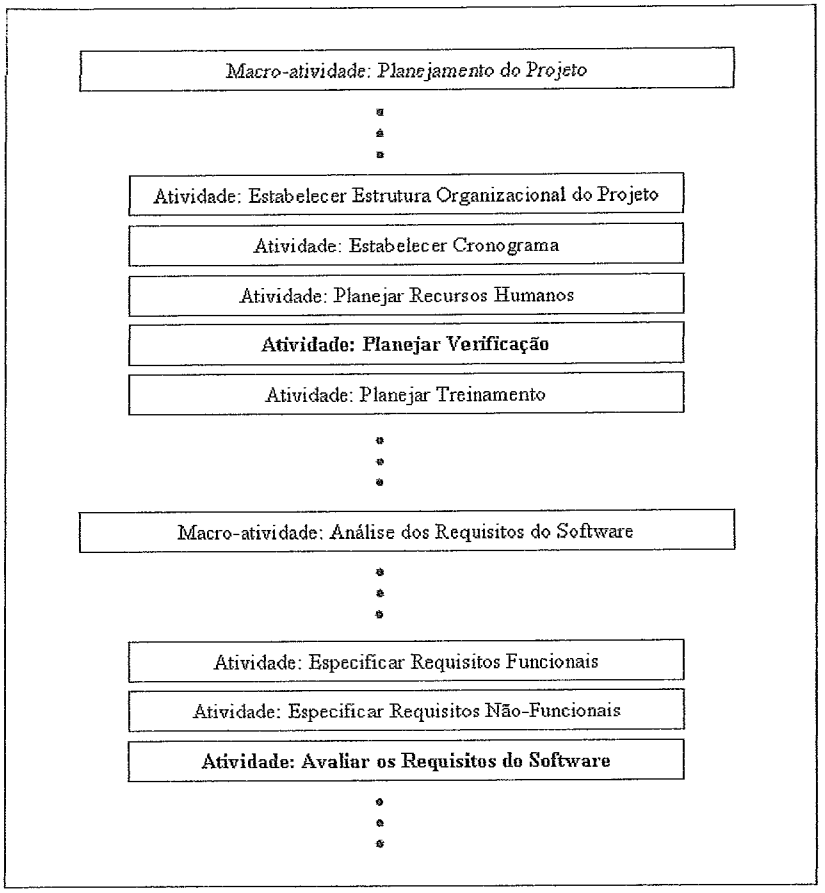


Figura 6.5 – Exemplo de atividades de verificação inseridas no processo de desenvolvimento

A implementação da abordagem proposta para o processo de verificação em um ADSOrg envolve a execução de quatro atividades:

(i) Selecionar Artefatos para Verificação;

(ii) Planejar Verificação de Artefatos;

(iii) Realizar Verificação; e,

(iv) Analisar os Resultados da Verificação.

As atividades (i), (ii) e (iv) foram descritas anteriormente, na definição do processo de verificação, no capítulo 5. A ferramenta *VerificationManager* se preocupa em apoiar a execução dessas três atividades. A atividade (iii) envolve a execução das atividades *Realizar Revisões por Pares, Realizar Testes* e *Enviar Relatórios de Verificação aos Destinatários* definidas anteriormente, também no capítulo 5. A ferramenta *VerificationManager* apóia somente o registro da execução da verificação realizada tanto através de revisão por pares quanto através de teste e o envio dos relatórios de

verificação aos destinatários, ou seja, a ferramenta desenvolvida não apóia o planejamento nem a execução específicos da revisão por pares ou do teste.

A aquisição do conhecimento durante a execução do processo é feita através da integração da ferramenta *VerificationManager*, objeto deste trabalho, com a ferramenta de Aquisição de Conhecimento, descrita no Capítulo 4, que foi desenvolvida com o objetivo de permitir o registro do conhecimento adquirido pelo usuário do ADSOrg durante as várias atividades do processo. Através dessa ferramenta é permitido o registro de idéias, dúvidas, problemas encontrados, diretrizes e lições aprendidas pelo gerente e equipe do projeto, durante a execução das atividades do processo de verificação. Após serem registradas, essas informações são, então, filtradas e empacotadas para armazenamento no repositório da organização e vínculo ao processo e atividade adequados. A partir desse momento, o novo conhecimento estará acessível aos usuários da ferramenta *VerificationManager*.

Vale ressaltar que toda a infra-estrutura aqui definida e implementada na Estação TABA para apoiar a verificação, isto é, o processo de verificação definido neste trabalho, o conjunto sugerido para auxiliar a verificação de alguns artefatos e apoio ferramental proposto através dos ADSOrgs e da ferramenta *VerificationManager*, já está disponível nos ambientes configurados e em uso por organizações na indústria.

Além disso, a implementação da abordagem proposta na Estação TABA contribui para que a mesma ofereça um apoio explícito à área de processo Verificação do modelo CMMI (SEI, 2002), bem como ao processo de verificação do modelo de referência MPS.BR (MPS.BR, 2005). Nesse contexto espera-se que a Estação TABA possa contribuir para avaliações bem sucedidas em qualquer uma dessas abordagens para as empresas que hoje a utilizam.

A seguir são apresentadas as quatro atividades de verificação que devem ser executadas nos ADSOrgs com o apoio da ferramenta *VerificationManager*.

6.2.1 Selecionar Artefatos para Verificação

Esta atividade faz parte do processo de verificação definido e está descrita em detalhes no capítulo 5. Ela consiste basicamente em identificar quais artefatos serão verificados ao longo do projeto. Para executar essa atividade, com base nas particularidades do projeto, o gerente deve identificar, dentre os artefatos que estão sujeitos à verificação, aqueles que serão efetivamente verificados no projeto em

questão. O conhecimento existente na organização sobre quais artefatos são sujeitos à verificação deve ser gerenciado na Estação TABA e disponibilizado ao gerente no momento da execução desta atividade.

6.2.2 Planejar Verificação de Artefatos

Esta atividade também faz parte do processo de verificação definido e está descrita em detalhes no capítulo 5, sendo dividida em cinco sub-atividades. Ela consiste em definir os detalhes para a verificação de cada artefato identificado na atividade anterior. Ao executar essa atividade, o gerente deve escolher, dentre os métodos disponíveis para verificação do artefato em questão, aqueles que serão usados no projeto. Nesse momento devem ser escolhidas também as ferramentas que serão usadas para apoiar a verificação do artefato, por exemplo, apoiando a aplicação de um método específico. O conhecimento existente na organização sobre quais métodos devem ser aplicados para a verificação de um dado artefato e quais ferramentas estão disponíveis para apoiar a verificação de um artefato usando um determinado método deve ser gerenciado na Estação TABA e disponibilizado ao gerente durante a execução dessa atividade.

Os critérios para verificação do artefato em questão também devem ser identificados nesta atividade. Com base no conjunto de critérios pré-definidos para a organização, o gerente pode fazer algumas modificações nesse conjunto, adaptando-o para atender a necessidades específicas do projeto. Esse conjunto é composto de características de qualidade que devem estar presentes no artefato e critérios, questões e métricas que auxiliam na determinação da satisfação das características de qualidade. O conhecimento existente na organização sobre quais características de qualidade devem ser avaliadas em um dado artefato e quais critérios, questões e métricas podem ser usados para auxiliar essa avaliação deve ser gerenciado na Estação TABA e disponibilizado ao gerente durante a execução dessa atividade.

Nesta atividade devem ser identificadas, ainda, a pessoa responsável pela verificação do artefato em questão e aquelas que participarão da avaliação. Por fim, devem ser definidos, os modelos para os relatórios de verificação que serão gerados de acordo com os métodos escolhidos, isto é, o modelo para os laudos usados nas revisões por pares, caso esse método tenha sido escolhido, e o modelo para o relatório de teste, caso o método teste tenha sido selecionado.

A qualquer momento deve ser possível visualizar o plano de verificação gerado com as informações do planejamento de cada artefato.

6.2.3 Realizar Verificação

Esta atividade envolve a execução das atividades *Realizar Revisões por Pares*, *Realizar Testes* e *Enviar Relatórios de Verificação aos Destinatários* que fazem parte do processo de verificação definido e estão descritas em detalhes no capítulo 5. Ela consiste em executar a verificação de acordo com o(s) método(s) escolhido(s) seguindo o que foi planejado na atividade anterior. Neste momento, os critérios definidos no plano de verificação devem ser analisados para determinar a satisfação ou não das características de qualidade que foram definidas como relevantes para a avaliação do artefato. Além disso, os relatórios de verificação definidos no plano de verificação devem ser produzidos, bem como as informações adicionais obtidas durante a realização da verificação devem ser coletadas e armazenadas. Pode-se, por exemplo, coletar medidas relacionadas à aplicação dos métodos, como a duração de uma reunião de revisão por pares, ou informações que caracterizem a qualidade do artefato, como o número de defeitos encontrados durante a avaliação.

Ao final da execução da verificação, os relatórios produzidos devem ser enviados aos destinatários definidos no plano de verificação para análise dos resultados.

6.2.4 Analisar os Resultados da Verificação

Esta atividade faz parte do processo de verificação definido e está descrita em detalhes no capítulo 5. Ela consiste basicamente em realizar uma análise dos resultados da verificação e identificar ações corretivas, caso necessário, e a possibilidade de melhorias ao longo das atividades de verificação realizadas. As ações corretivas necessárias devem ser documentadas em planos de ação possibilitando a gerência e execução dessas correções.

6.3 A Ferramenta VerificationManager

Buscando-se apoiar a abordagem de verificação de software descrita neste trabalho, a ferramenta *VerificationManager* foi definida e implementada. Essa ferramenta apóia as atividades do planejamento da verificação e o registro da execução da verificação e

da análise dos resultados obtidos durante a execução. A ferramenta é disponibilizada nos ambientes configurados e pode ser usada nos ambientes instanciados.

VerificationManager baseia-se fundamentalmente no processo definido no Capítulo 5 e guia o usuário durante a realização das atividades definidas nesse processo. Essa ferramenta possui dois módulos de execução de acordo com o usuário que a está executando. Isso ocorre porque algumas atividades só podem ser executadas por determinados usuários (como pode ser visto na tabela 5.1). Os módulos são apresentados abaixo:

(i) *Módulo do Gerente do Projeto*: Selecionar Artefatos para Verificação, Planejar Verificação do Artefato.

(ii) *Módulo da Equipe do Projeto*: Realizar Verificação e Analisar os Resultados da Verificação.

Todas as telas da ferramenta apresentam um mesmo padrão de interface. No lado esquerdo pode-se visualizar o processo de verificação que guia a ferramenta e no lado direito da tela identifica-se a atividade que está sendo realizada pelo usuário. Além disso, é possível realizar a busca e o registro de conhecimento no que diz respeito às atividades do processo através da interação com uma ferramenta de Aquisição de Conhecimento – *Acknowledge* descrita no capítulo 4.

Para apresentar a ferramenta, será descrito um exemplo de seu uso. Consideremos uma organização que tenha definido os seguintes artefatos como sujeitos à verificação: Plano do Projeto, Especificação de Requisitos do Cliente, Especificação de Requisitos do Sistema, Projeto da Arquitetura do Sistema, Especificação de Requisitos do Software, Modelo de Análise, Modelo de Projeto, Código de Unidades, Software e Sistema. Esse conhecimento deve ser registrado para uso na organização através dos ambientes configurados da Estação TABA, juntamente com a informação de quais métodos podem ser usados para a verificação de cada um desses artefatos. Ao se configurar um ambiente para uma organização, o conjunto de artefatos sujeitos a verificação e os métodos aplicáveis a cada um deles é disponibilizado, podendo ser modificado de acordo com as necessidades da organização. A figura 6.2 apresenta a tela para cadastro desse conhecimento no ambiente configurado.

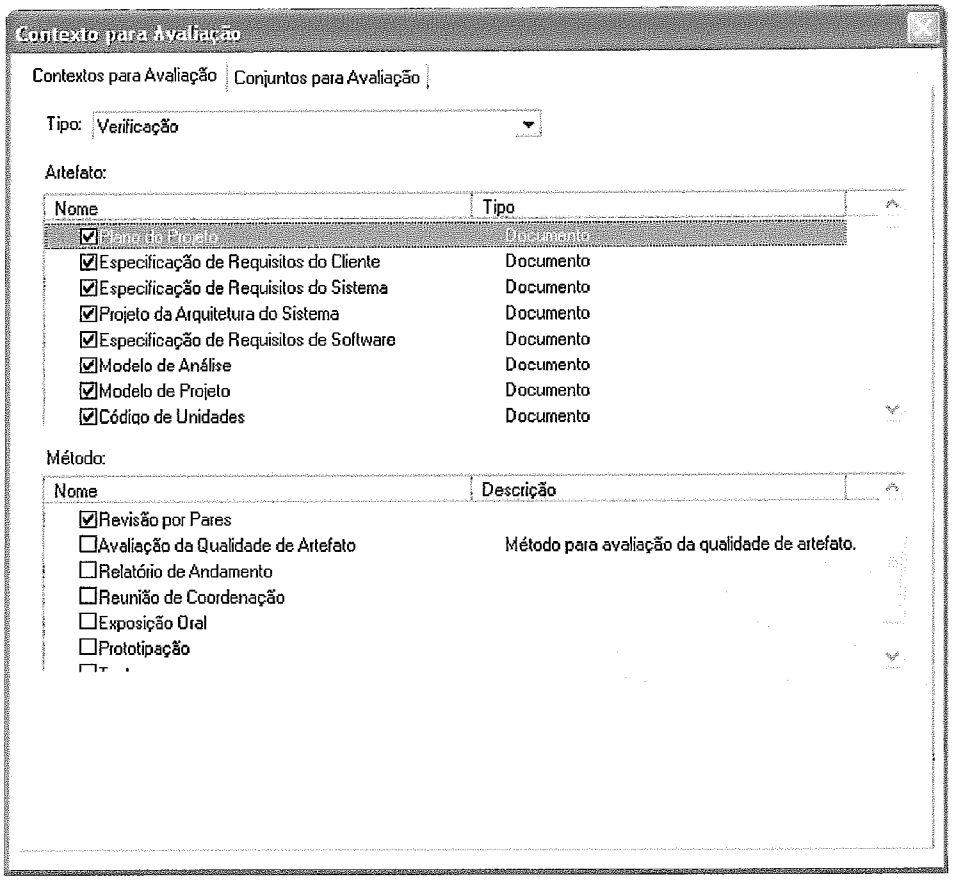


Figura 6.2 – Tela de cadastro dos artefatos sujeitos à verificação definidos para a organização

O conhecimento relativo às características de qualidade que devem ser avaliadas em cada artefato sujeito à verificação e aos critérios, questões e métricas que podem ser usados para auxiliar nessa avaliação também deve ser registrado para uso na organização através dos ambientes configurados da Estação TABA. Novamente é disponibilizado o conjunto definido neste trabalho ao se configurar um ambiente para uma organização, podendo ser modificado de acordo com as necessidades da organização. A figura 6.3 apresenta a tela para cadastro desse conhecimento no ambiente configurado.

As duas seções seguintes descrevem a execução do exemplo de uso na ferramenta *VerificationManager* divididas em planejamento e execução da verificação.

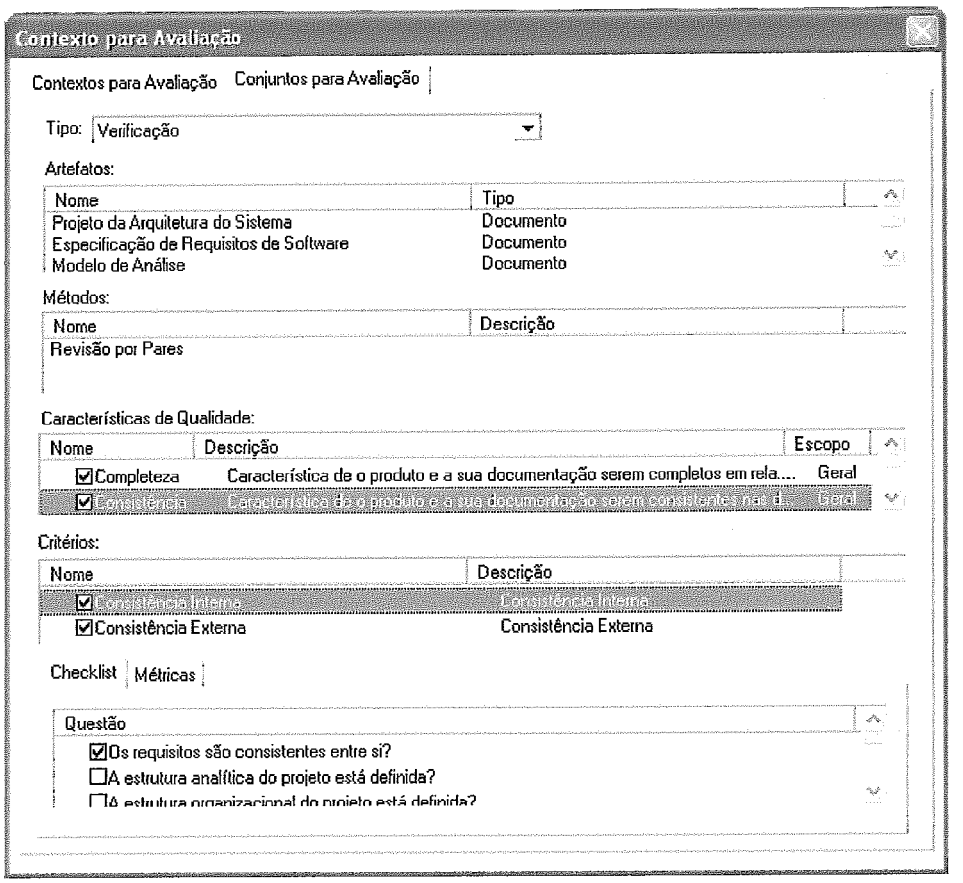


Figura 6.3 – Tela de cadastro dos conjuntos de critérios para verificação para cada artefato definidos para a organização

6.3.1 Planejamento da Verificação na Ferramenta *VerificationManager*

Para planejar a verificação ao longo do projeto, a primeira atividade a ser realizada é a seleção dos artefatos que serão verificados ao longo do projeto. Durante o planejamento do projeto, o gerente deve identificar, dentre os artefatos sujeitos à verificação, aqueles que serão verificados no projeto em questão, conforme dito na seção 6.2.1. Para isso, a ferramenta apresenta a lista dos artefatos sujeitos à verificação definida para a organização e permite que o gerente modifique essa lista considerando as particularidades do projeto. A figura 6.4 apresenta a tela de apoio a essa atividade. Considerando o exemplo de uso da ferramenta, não foi necessário modificar a lista de artefatos definida para a organização.

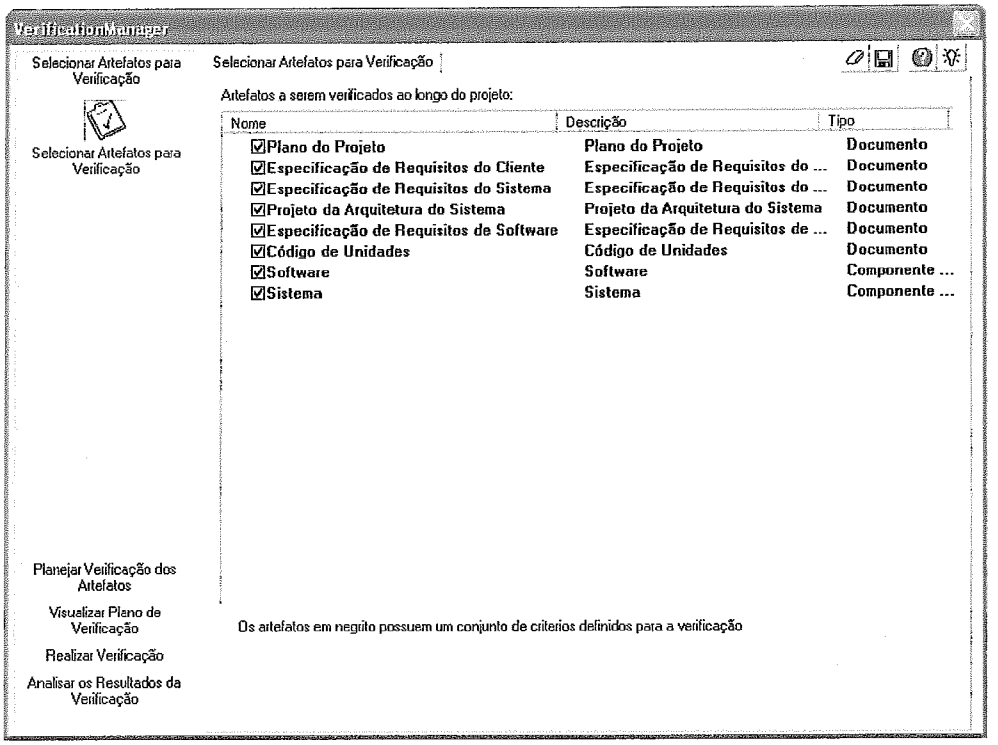


Figura 6.4 – Tela de seleção dos artefatos que serão verificados ao longo do projeto

Após identificar os artefatos que serão verificados ao longo do projeto, o gerente deve planejar a verificação para cada artefato selecionado. À medida que o projeto evolui, os artefatos que foram selecionados são produzidos. No início de cada fase do projeto, o planejamento da verificação para os artefatos que serão produzidos na fase em questão e que foram selecionados no planejamento do projeto pode ser realizado. Se a organização julgar mais adequado, esse planejamento pode ser realizado no início do projeto para todos os artefatos e, caso necessário, revisado ao longo do projeto.

Para exemplificar, realizaremos o planejamento da verificação do artefato *Especificação de Requisitos do Software*. O planejamento da verificação dos demais artefatos pode ser realizado de forma similar.

A figura 6.5 apresenta a tela para identificação do planejamento a ser realizado. A identificação de um planejamento envolve determinar em qual atividade do processo de desenvolvimento a verificação a ser planejada será executada e qual artefato será verificado. Uma verificação pode ser identificada por um nome e possui uma data de planejamento, uma data de execução e uma situação. Uma verificação pode estar em planejamento, planejada, em execução, executada, aprovada ou insuficiente.

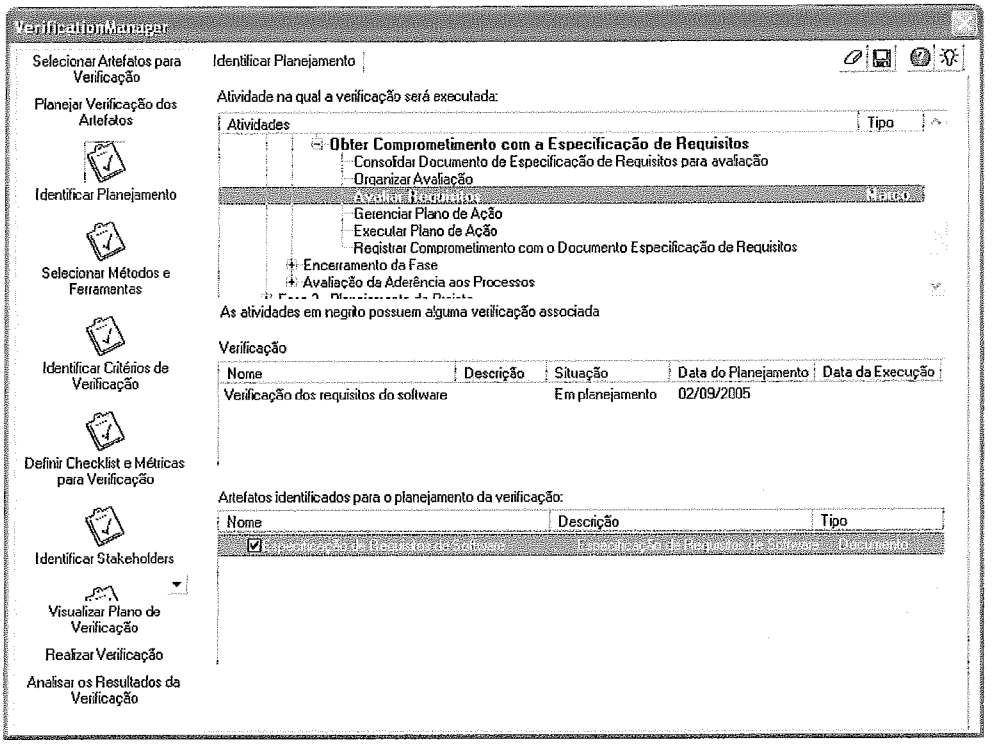


Figura 6.5 – Tela de identificação do planejamento realizado

Após identificar o planejamento que deseja realizar, o gerente deve efetivamente iniciar o planejamento identificando os métodos e ferramentas que serão usados na verificação do artefato. Para isso, conforme dito na seção 6.4.2, ele deve selecionar os métodos que serão usados, dentre os métodos que foram definidos pela organização para a verificação do artefato em questão. Além disso, o gerente deve determinar quais ferramentas serão usadas para auxiliar na verificação planejada. A figura 6.6 apresenta a tela que apóia a seleção dos métodos e ferramentas para verificação.

Após identificar os métodos que serão usados na verificação planejada, o gerente deve continuar o planejamento, adaptando para o projeto o conjunto de características e critérios definido para a organização. Para auxiliar nessa adaptação, a ferramenta apresenta as características de qualidade agrupadas em características de qualidade gerais e características de qualidade do produto, de acordo com a divisão descrita no capítulo 5, e permite que o gerente desconsidere algumas características e critérios.

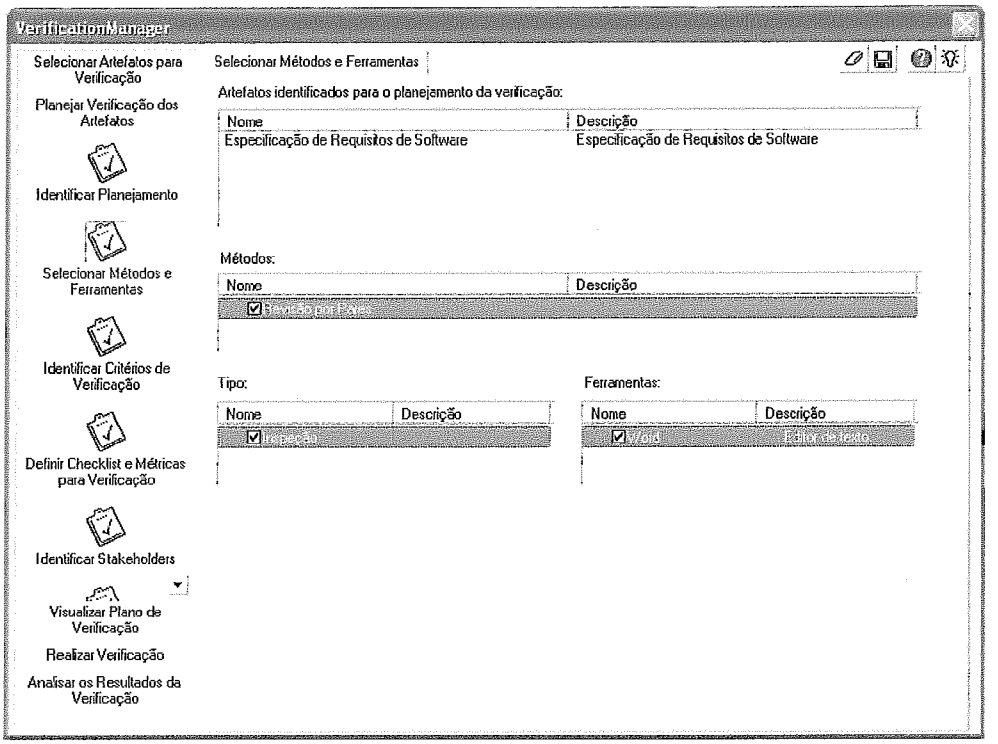


Figura 6.6 – Tela de seleção de métodos e ferramentas

Neste momento a ferramenta de verificação apresenta uma integração com a ferramenta *QFuzzy*, descrita no capítulo 4, que está disponível nos ADSOrgs e é utilizada para avaliação da relevância das características de qualidade do produto em um projeto específico. A relevância de cada característica de qualidade do produto, avaliada através da *QFuzzy*, é exibida na ferramenta *VerificationManager*, com o objetivo de ajudar o gerente na adaptação do conjunto de características a serem verificadas no projeto específico. Caso não seja realizada nenhuma avaliação da relevância das características de qualidade do produto através da ferramenta *QFuzzy*, a ferramenta *VerificationManager* apresentará a relevância dessas características como *Indefinida*.

As figuras 6.7 e 6.8 apresentam as telas que apóiam a identificação das características de qualidade gerais e das características de qualidade do produto, respectivamente, e a seleção dos critérios que irão auxiliar na determinação da presença das características de qualidade no artefato a ser verificado. Considerando o exemplo de uso da ferramenta, a avaliação das características de qualidade do produto não foi realizada e, portanto, a relevância dessas características está indefinida. Ao avaliar as necessidades do projeto e o produto a ser desenvolvido, o gerente do projeto julgou que a característica de qualidade *Segurança* não era relevante e, por isso, ela foi desmarcada.

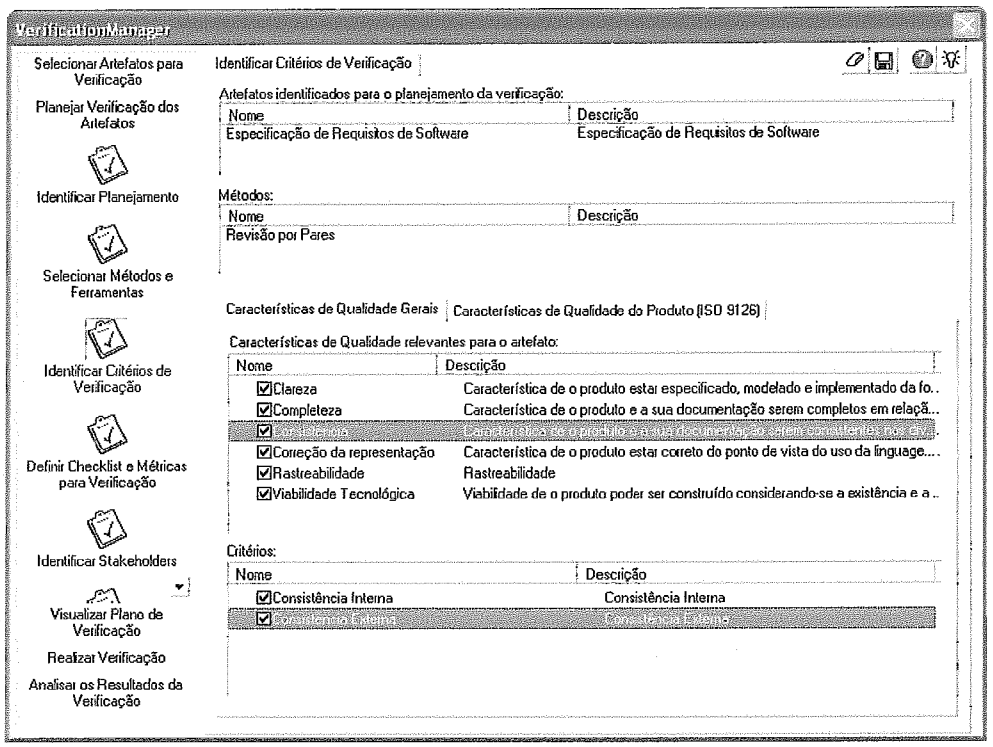


Figura 6.7 – Tela de identificação das características de qualidade gerais e critérios associados

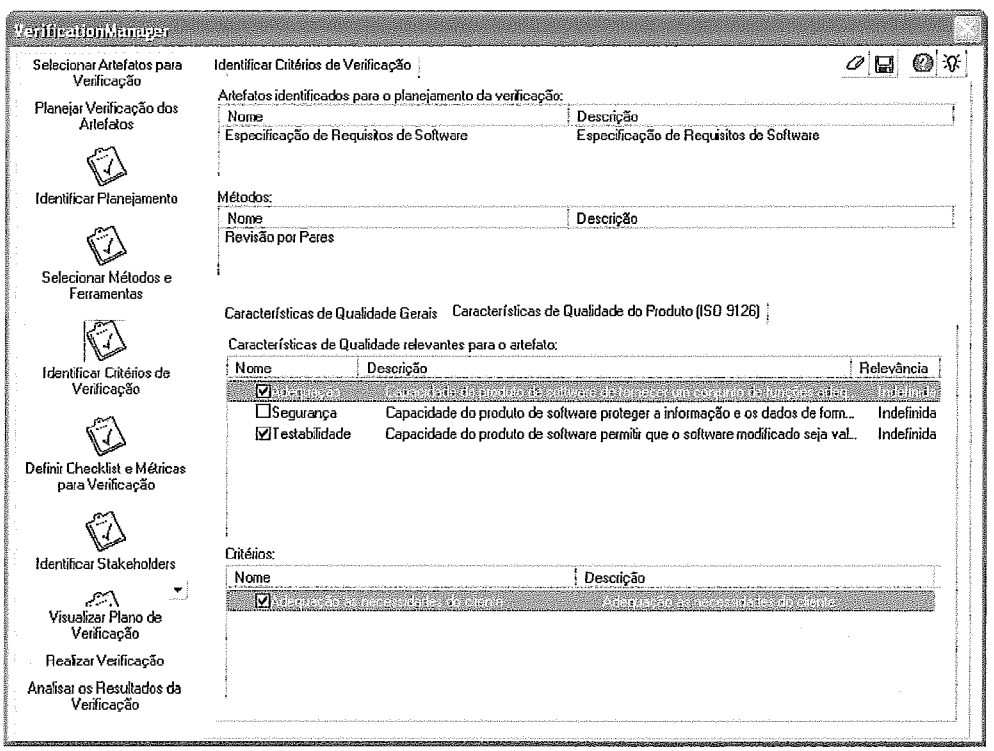


Figura 6.8 – Tela de identificação das características de qualidade do produto e critérios associados

Após identificar características de qualidade que deverão ser avaliadas no artefato e os critérios que serão usados para essa avaliação, o gerente deve continuar o planejamento, adaptando para o projeto o conjunto de questões e métricas definido para a organização. Para auxiliar nessa adaptação, a ferramenta apresenta as questões e métricas definidas pela organização para a verificação do artefato em questão e permite que o gerente desconsidere algumas questões e métricas. A figura 6.9 apresenta a tela que apóia a adaptação do conjunto de questões e métricas para verificação. Considerando o exemplo de uso da ferramenta, não foi necessário adaptar o conjunto de questões e métricas definidas para a organização.

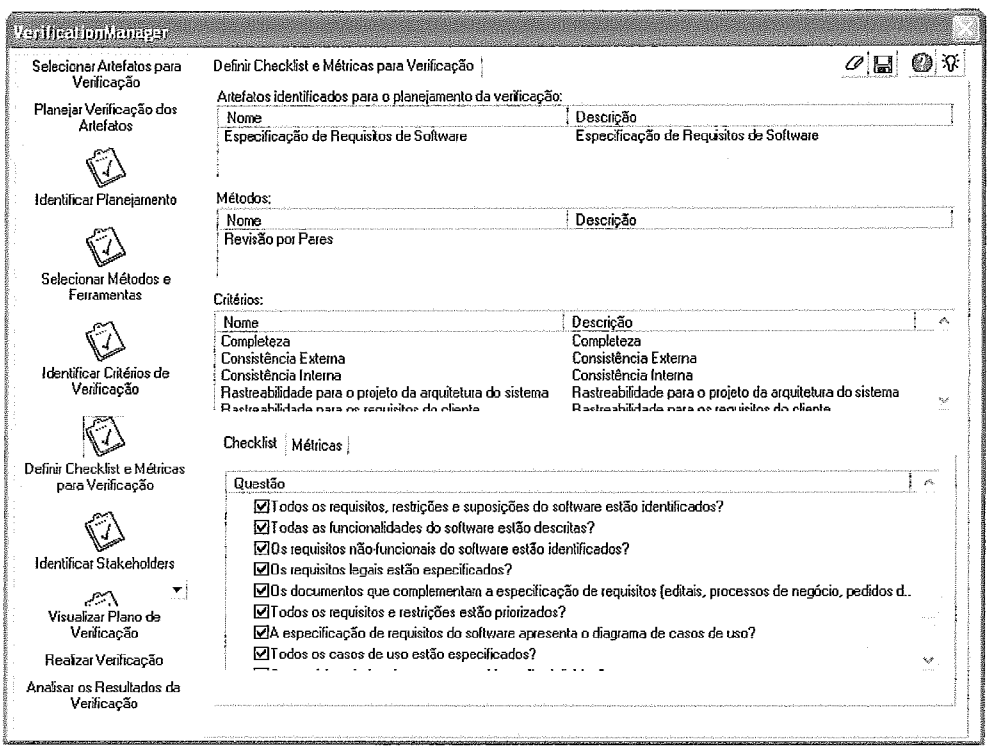


Figura 6.9 – Tela de adaptação das questões e métricas para o projeto específico

Após adaptar o conjunto de características de qualidade, critérios, questões e métricas que será usado para a verificação do artefato, o gerente deve dar continuidade ao planejamento, identificando o responsável pela verificação, os participantes e os destinatários dos relatórios de verificação. Para isso, a ferramenta apresenta uma lista das pessoas que participam do projeto (tanto como organização desenvolvedora quanto

como organização cliente) e permite que o gerente identifique aquelas que exercerão os papéis citados acima. A figura 6.10 apresenta a tela que apóia a identificação das pessoas envolvidas na verificação planejada.

Para finalizar o planejamento da verificação, o gerente deve adaptar os modelos dos relatórios de verificação a serem gerados de acordo com os métodos escolhidos. Para isso a ferramenta gera esses modelos com as informações referentes aos critérios que serão usados na verificação, conforme o planejamento realizado pelo gerente. Além de gerar esses modelos, a ferramenta os associa à atividade de execução da verificação definida na identificação do planejamento, facilitando a execução da verificação nos ADSOrgs. A figura 6.11 apresenta a tela para geração dos modelos dos relatórios. Considerando o exemplo de uso e o método Revisão por Pares selecionado na figura 6.6, a ferramenta gera o modelo para o laudo de preparação individual e o laudo final para avaliação da especificação de requisitos do software.

A qualquer momento é possível visualizar o plano de verificação gerado com as informações do planejamento de cada artefato, como mostra a figura 6.12.

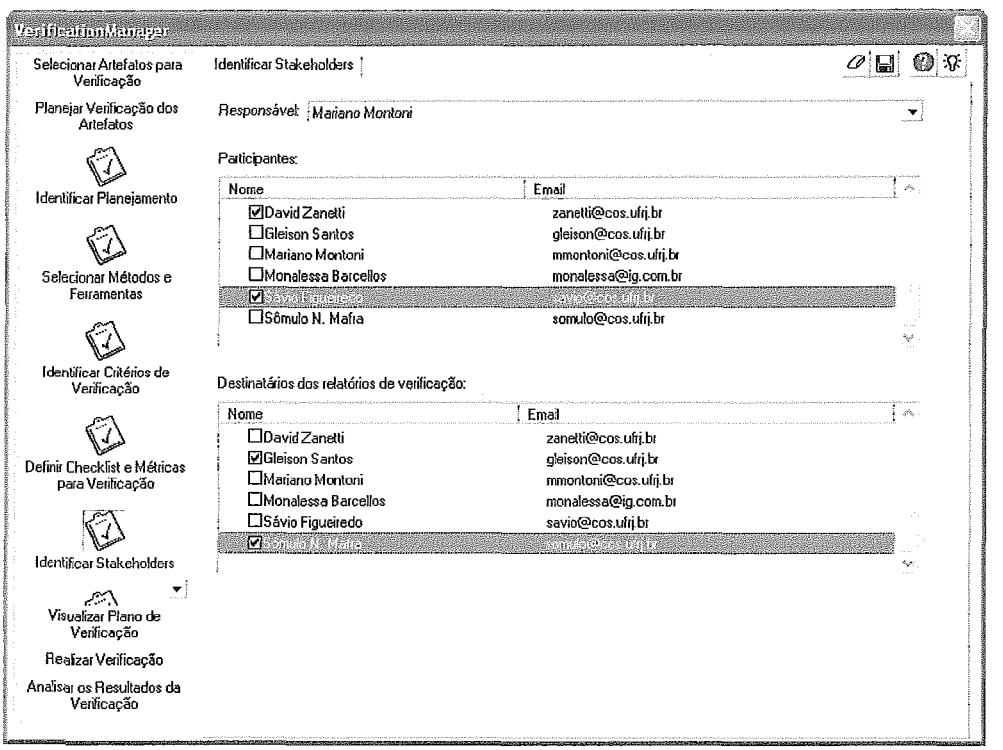


Figura 6.10 – Tela de identificação das pessoas envolvidas na verificação planejada

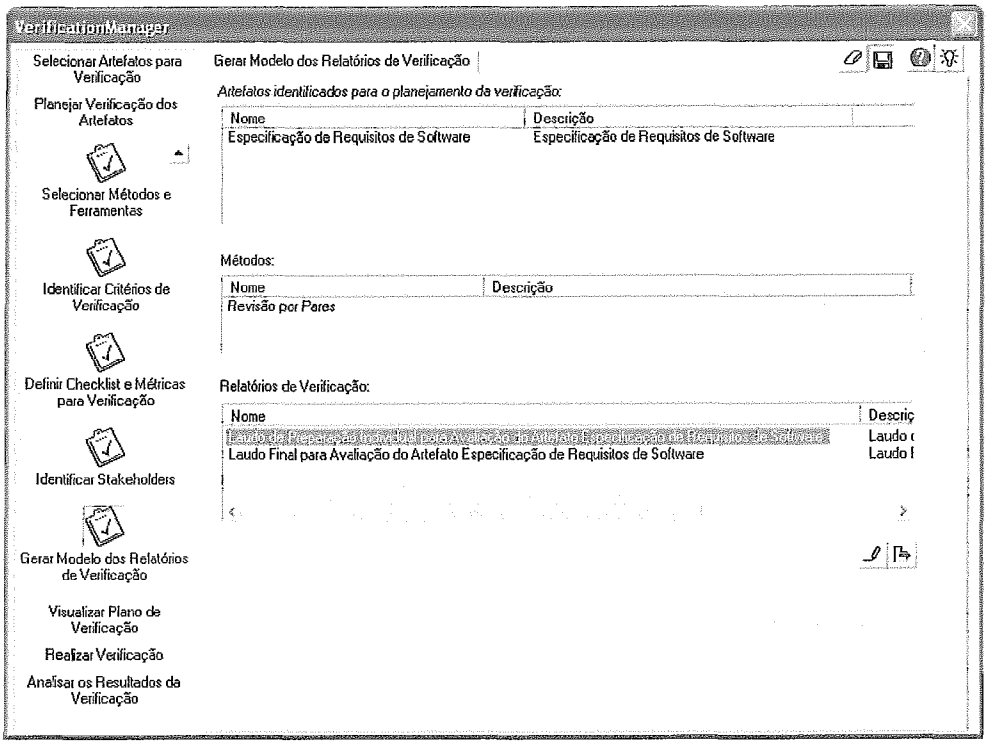


Figura 6.11 – Tela de geração dos modelos dos relatórios de verificação

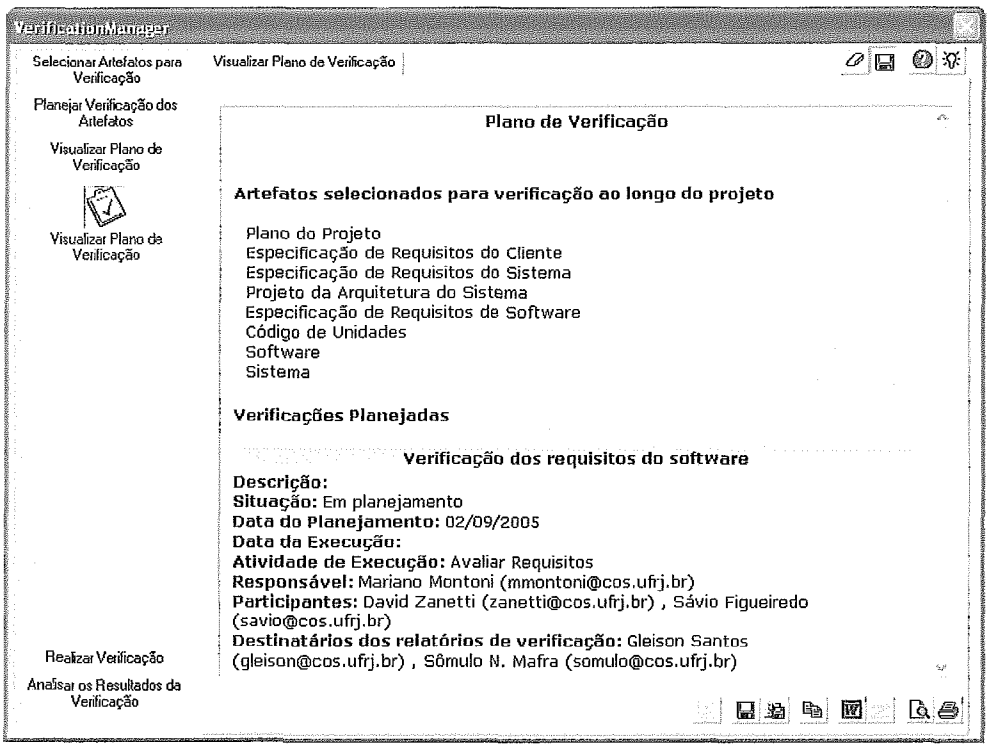


Figura 6.12 – Tela de visualização do plano de verificação definido

6.3.2 Execução da Verificação na Ferramenta *VerificationManager*

A execução da verificação é realizada no momento em que, de acordo com a evolução do projeto, a atividade de execução da verificação identificada no planejamento da verificação deva ser realizada. Nesse momento, a ferramenta *VerificationManager* é novamente utilizada.

A partir das informações disponibilizadas no plano de verificação, a verificação deve ser executada usando os métodos definidos. Nesse momento a ferramenta se propõe apenas a apoiar o registro da execução da verificação e análise dos resultados gerados ao realizar a verificação. A figura 6.13 apresenta a tela para registro da execução da verificação. Nesta tela, a verificação a ser executada pode ser identificada e as pessoas envolvidas podem registrar a data da execução da verificação, mudar a situação da verificação e, para cada método selecionado no planejamento, anexar os relatórios que foram produzidos durante a verificação. Além disso, nessa atividade é possível também registrar algumas medidas coletadas durante a verificação.

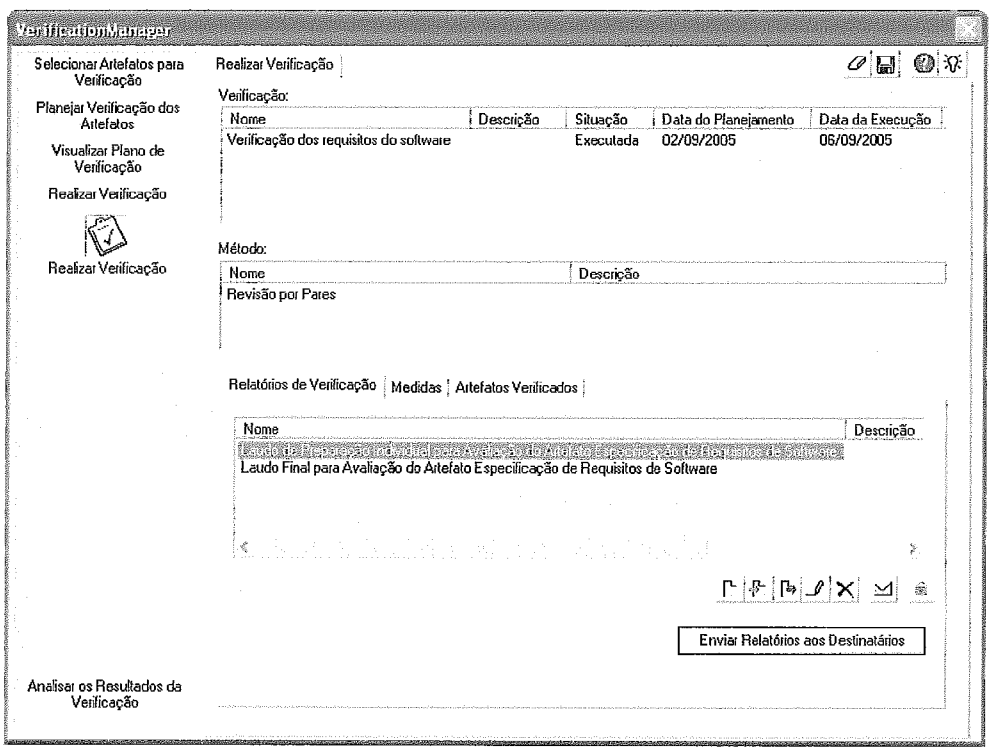


Figura 6.13 – Tela de registro da execução da verificação

Após registrar a execução da verificação, as pessoas envolvidas devem enviar os relatórios produzidos aos destinatários definidos no plano de verificação. A ferramenta disponibiliza um botão para que o usuário solicite o envio de um e-mail aos destinatários identificados no planejamento, com os relatórios associados à verificação anexados ao e-mail. Considerando o exemplo de uso, no planejamento somente o método revisão por pares foi selecionado e, portanto, somente os laudos de preparação individual e final devem ser associados à ferramenta. No exemplo considerado foi registrada uma medida referente à duração da reunião de revisão por pares, conforme mostra a figura 6.14.

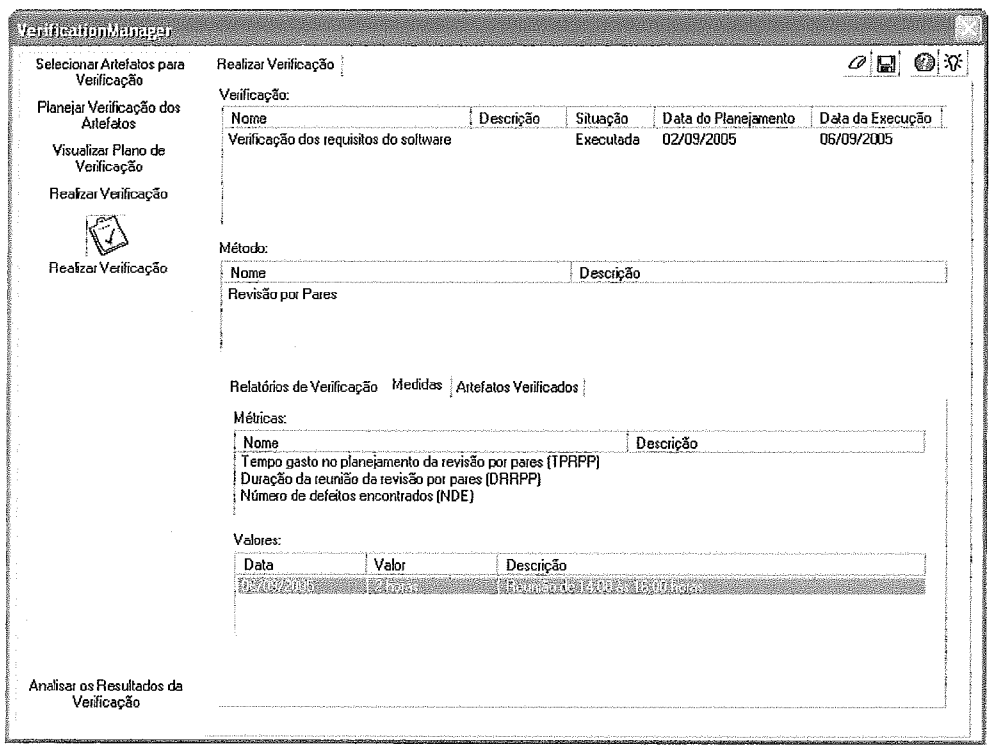


Figura 6.14 – Tela de registro das medidas coletadas durante a execução da verificação

Após a execução da verificação, uma análise dos resultados da verificação deve ser realizada. Para apoiar essa atividade, a ferramenta apresenta a tela mostrada na figura 6.15, que permite o registro de uma análise da verificação, bem como a geração de planos de ação que contenham as ações corretivas necessárias para a correção dos problemas encontrados no artefato durante a verificação.

Os planos de ação são criados a partir da ferramenta *ActionPlanManager*, descrita no capítulo 4, que pode ser chamada diretamente da ferramenta *VerificationManager*. A

integração entre essas duas ferramentas permite que planos de ação gerados a partir de verificações registradas na ferramenta *VerificationManager* possam ser gerenciados e executados ao longo do processo de desenvolvimento através da ferramenta *ActionPlanManager*. Considerando o exemplo de uso, o artefato verificado não apresentou problemas e por isso não foi necessária a criação de um plano de ação para corrigi-lo.

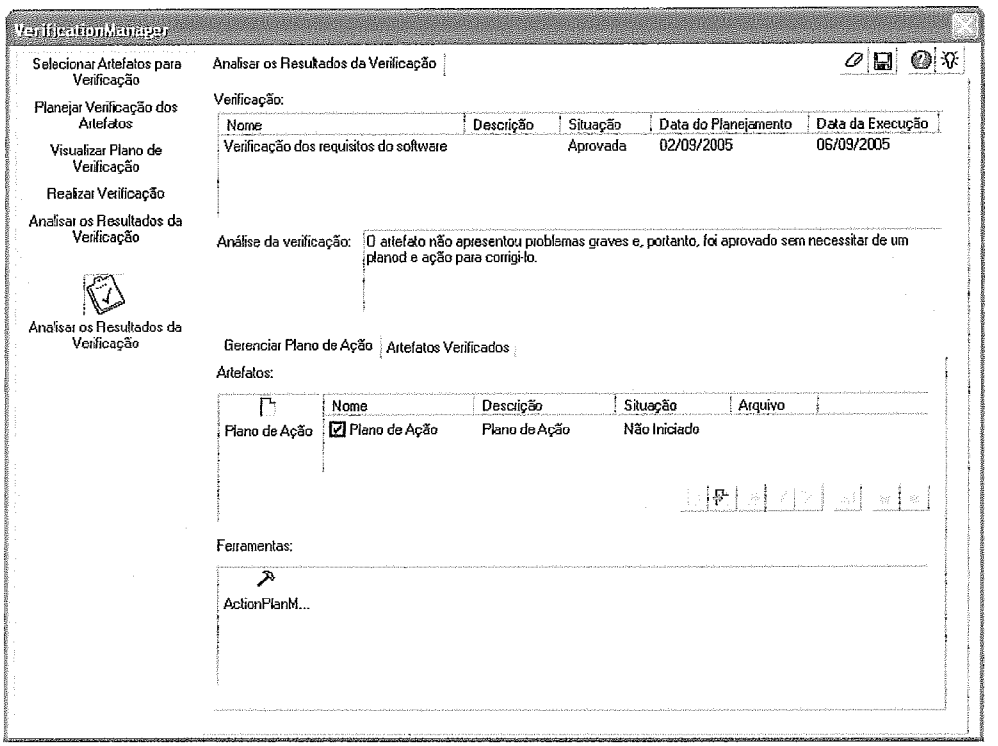


Figura 6.15 – Tela de registro da análise dos resultados da verificação

6.4 Conclusão

Neste capitulo foram descritos a implementação e o uso da abordagem proposta para a verificação de software na Estação TABA. Também foi apresentado o apoio oferecido à execução das atividades de verificação, além de uma descrição detalhada de como essas atividades devem ser executadas nos ADSOrgs.

A ferramenta *VerificationManager* foi apresentada e um exemplo de uso dessa ferramenta para apoiar o planejamento e o registro da execução da verificação de um artefato foi detalhadamente descrito.

No próximo capítulo, serão apresentadas as conclusões e considerações finais deste trabalho, além de suas perspectivas futuras.

CAPÍTULO 7

Conclusão

7.1 Considerações Finais

Com o crescimento do volume e complexidade dos produtos de software, tem se destacado a importância da qualidade dos mesmos. O mercado consumidor tem se tornado rigoroso na escolha de seus produtos e, nesse contexto, a qualidade das aplicações de software desenvolvidas passou a ser um fator crítico para o sucesso. Com o objetivo de atender a essa exigência do mercado, melhorando a qualidade dos produtos de software, atividades de verificação têm sido realizadas ao longo do desenvolvimento do produto.

Neste trabalho, foi apresentada uma proposta para apoiar a verificação ao longo do desenvolvimento de produtos de software. A proposta consiste em organizar as atividades de verificação em um processo que deve ser executado de forma integrada ao processo de desenvolvimento de software e oferecer conhecimento e apoio automatizado para auxiliar na execução dessas atividades.

Para isso, baseando-se na literatura existente sobre verificação de software, foi definido um processo de verificação de software. Uma forma de integração desse processo ao processo de desenvolvimento de software foi apresentada e, com o objetivo de auxiliar a execução do processo definido, foi sugerido um conjunto de critérios para a verificação de alguns artefatos.

A abordagem aqui proposta foi implementada na Estação TABA, mais especificamente nos Ambientes de Desenvolvimento de Software Orientados à Organização e já está em uso na indústria. O apoio automatizado à execução das atividades de verificação é oferecido através da ferramenta *VerificationManager* e da infra-estrutura oferecida pela Estação TABA. Apesar de a abordagem proposta ainda não ter sido avaliada formalmente, vale destacar que já está em uso em três empresas no Rio de Janeiro desde 2005. Essa interação com a indústria já permitiu, inclusive, que modificações fossem solicitadas e realizadas na abordagem proposta, de forma a melhor adequá-la à realidade das empresas.

Apesar de este trabalho estar inserido no contexto dos Ambientes de Desenvolvimento de Software Orientados à Organização, é de se esperar que o processo

definido, a sua integração com o processo de desenvolvimento e o conjunto de critérios aqui sugeridos possam ser utilizados em um contexto mais geral, não ficando restritos apenas aos ADSOrgs.

7.2 Contribuições

Dentre as contribuições deste trabalho podemos destacar como principais:

- A definição de um processo de verificação de software, baseado na literatura e aderente à norma ISO/IEC 12207 (ISO/IEC, 1998; ISO/IEC, 2002; ISO/IEC, 2004) e aos modelos CMMI (SEI, 2002) e MPS.BR (MPS.BR, 2005);
- A definição de uma estratégia de integração das atividades de verificação ao processo de desenvolvimento de software;
- A definição de um conjunto inicial de critérios para a verificação de artefatos. Este conjunto foi definido a partir de uma seleção da literatura existente, baseada no conhecimento de especialistas e buscando reunir o conhecimento que poderia ser mais facilmente aplicável à indústria. Vale salientar que estes critérios representam um conjunto que, apesar de ser baseado na literatura, ainda não foi avaliado através de experimentos formais. Apesar disso, ele já está em uso na indústria, o que possibilitará a sua avaliação formal e identificação de possíveis adequações e melhorias.
- A definição e implementação de uma ferramenta para auxiliar a execução das atividades de verificação;
- A integração da ferramenta definida e implementada neste trabalho aos Ambientes de Desenvolvimento de Software Orientados à Organização e às ferramentas já existentes nestes ambientes, sobretudo *AdaptPro*, *QFuzzy*, *Acknowledge*, *ActionPlanManager* e *ValidationManager*; e,
- A contribuição para que a Estação TABA ofereça um apoio explícito à área de processo Verificação do modelo CMMI (SEI, 2002), bem como ao processo de verificação do modelo de referência MPS.BR (MPS.BR, 2005). Nesse contexto espera-se que a Estação TABA possa contribuir para avaliações bem sucedidas em qualquer uma dessas abordagens nas empresas que hoje utilizam a Estação TABA.

Apesar de a implementação da abordagem proposta já estar em uso em três empresas no Rio de Janeiro, os benefícios dessa abordagem só poderão ser efetivamente avaliados em um procedimento de avaliação da ferramenta. Porém, a validação de uma ferramenta como *VerificationManager* implica em sua utilização em vários projetos e excede, em muito, o tempo esperado para o desenvolvimento de uma dissertação de mestrado. Portanto, a validação da abordagem proposta está sendo realizada no contexto do Projeto TABA englobando seu conjunto de ferramentas. Inclui-se, entretanto, no capítulo 6, um exemplo de sua utilização.

A abordagem proposta introduz o planejamento e a execução da verificação de software à Estação TABA e representa um passo importante na construção de ferramentas de apoio ao desenvolvimento de software centrado na utilização do conhecimento organizacional. Esta perspectiva pode trazer grandes benefícios para as organizações que desenvolvem software e é uma das principais tendências na área de Engenharia de Software.

7.3 Perspectivas Futuras

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

Uma perspectiva futura interessante é caracterizar a aplicabilidade da abordagem proposta neste trabalho na prática através do uso sistemático dessa abordagem por empresas.

Um outro desdobramento possível deste trabalho é planejar e executar um estudo experimental para avaliar o conjunto de critérios para verificação sugerido neste trabalho.

Outra perspectiva futura é a realização de diversas pesquisas na literatura para levantar e organizar o conhecimento existente sobre os métodos de verificação. Esse conhecimento organizado poderia ser disponibilizado na Estação TABA através da ferramenta de aquisição e gerência do conhecimento *Acknowledge* (MONTONI *et al.*, 2004) e auxiliaria tanto o gerente do projeto durante o planejamento da verificação, quanto a equipe do projeto no momento da aplicação dos métodos. Por exemplo, poderia ser disponibilizado o conhecimento referente ao planejamento de revisão por pares, tipos de revisão por pares e como aplicá-las, e ao planejamento de teste, tipos de teste, técnicas de teste e como aplicá-los.

Uma outra extensão possível para este trabalho é a definição e implementação de uma abordagem que apóie a execução da verificação, mais especificamente a aplicação dos métodos durante a verificação dos artefatos. Essa abordagem poderia ser implementada na Estação TABA e disponibilizada nos ADSOrgs.

REFERÊNCIAS BIBLIOGRÁFICAS

ACKERMAN, A.F., BUCHWALD, L.S., LEWSKI, F.H., 1989, "Software Inspections: An Effective Verification Process", *IEEE Software*, v. 6, n. 3, pp. 31-36.

ANDERSSON, C., 2003, *Exploring the Software Verification and Validation Process with Focus on Efficient Fault Detection*, Licentiate Thesis, Lund Institute of Technology (LTH), Lund University, Sweden.

ANDRADE, J.M.S., 2005, *Avaliação de Processos de Software em ADSOrg*, Dissertação de M. Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

BAHIA, A.S., 1992, *Avaliação da Complexidade de Software Científico*, Tese de M.Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

BARCELLOS, M.P., 2003, *Planejamento de Custos em Ambientes de Desenvolvimento de Software Orientados à Organização*, Dissertação de M. Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

BARCELLOS, M.P., FIGUEIREDO, S.M., ROCHA, A.R.C., TRAVASSOS, G.H., 2003, "Utilização de Métodos Paramétricos, Analogias, Julgamento de Especialistas e Conhecimento Organizacional no Planejamento de Tempo e Custos de Projetos de Software", *In: Anais do II Simpósio Brasileiro de Qualidade de Software*, pp. 17-31, Fortaleza, Brasil.

BARRETO, A.O.S., ROCHA, A.R.C., 2005, "Apoio à Área de Processo Verificação de Software em Ambientes de Desenvolvimento de Software Orientados à Organização", In: *Anais do IV Simpósio Brasileiro de Qualidade de Software*, pp. 213-226, Porto Alegre, Brasil, Junho.

BASILI, V.R., CALDIERA, G., ROMBACH, H.D., 1994, "Goal Question Metric Paradigm", *Encyclopedia of Software Engineering*, 2 Volume Set, John Wiley & Sons, Inc.

BEAUFOND, C.E.C., WERNER, C.M.L., ROCHA, A.R.C., 1997, "Manual para Controle da Qualidade de Especificações Orientadas a Objetos", *Relatório Técnico ES-428/97*, Programa de Engenharia.de Sistemas e Computação, COPPE/UFRJ.

BELCHIOR, A.D., 1992, *Qualidade de Software Financeiro*, Tese de M.Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

BELCHIOR, A.D., 1997, *Um modelo fuzzy para avaliação da qualidade de software*, Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

BERGER, P., 2003, *Instanciação de Processos de Software em Ambientes Configurados na Estação TABA*, Tese de M. Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

BESSA, A., 2005, *Melhoria de Processos nos Ambientes Configurados TABA*, Exame de Qualificação para o Doutorado, PESC, COPPE/UFRJ, Brasil.

BHATTI, S.N., KEPLER, J., 2005, "Why Quality? ISO 9126 Software Quality Metrics (Functionality) Support by UML", *ACM Software Engineering Notes*, v. 30, n. 2, pp. 1-5.

BIANCHI, F., 2005, *Melhoria de Processos e Evolução do Meta-ambiente TABA*, Exame de Qualificação para o Doutorado, PESC, COPPE/UFRJ, Brasil.

BINDER, R.V., 2001, *Testing Object Oriented Systems Models Patterns and Tools*, 3ª Edição, Addison-Wesley Publishing Company, Boston, Estados Unidos.

BLASCHECK, J.R.., 1995, *Planejamento estratégico de sistemas de informação*, Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

BOEHM, B., BASILI, V., 2001, "Software Defect Reduction Top 10 List", *IEEE Computer*, v. 34, n.1, pp. 135-137.

BOEHM, B., BROWN, J.R., KASPAR, H., LIPOW, M., McLEOD, G., MERRIT, M., 1978, *Characteristics of Software Quality*, North Holland.

CAMPOS, F., 1994, *Qualidade de aplicações hipermídia*, Tese de M.Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

CHRISSIS, M.B.., KONRAD, M., SHRUM, S., 2003, *CMMI: Guidelines for Process Integration and Product Improvement*, Addison-Wesley Publishing Company, Boston, Estados Unidos.

CHRISTENSEN, M.J., THAYER, R.H., 2001, *Software Metrics*, In: The Project Manager's Guide to Software Engineering Best Practices, Best Practices Series, Wiley.

CRESPO, A.N., SILVA, O.J., BORGES, C.A., SALVIANO, C.F., TEIVE, M., JINO, M., 2004, "Uma Metodologia para Teste de Software no Contexto da Melhoria de Processo", *In: Anais do III Simpósio Brasileiro de Qualidade de Software*, pp. 271-285, Brasília, Brasil.

CROSBY, P.B., 1988, *Quality Without Tears*, McGraw-Hill, Nova York, Estados Unidos.

D'OLIVEIRA, F.M., 2003, *Planejamento de Testes na Homologação de Software: uma Abordagem Orientada ao Cliente*, Dissertação de Pós Graduação, Universidade Católica de Brasília, Brasília, Brasil.

DONEGAN, P., BANDEIRA, L., MATOS, A., CUNHA, P., MAIA, C., PIRES, C.G.S., 2005, "Aplicabilidade da Automação de Testes Funcionais - A Experiência no Instituto Atlântico", In: *Anais do IV Simpósio Brasileiro de Qualidade de Software*, pp. 447-454, Porto Alegre, Brasil, Junho.

DROMEY, R.G., 1996, "Cornering the chimera", *IEEE Software*, v.13, n.1, pp. 33-43.

DUNN, R.H., 1984, *Software Defect Removal*, McGraw-Hill Book Company, Nova York, Estados Unidos.

ESA, 1995, "Guide to Software Verification and Validation", *ESA PSS-05-10*, European Space Agency.

ESTOLANO, M.H.R., 2005, *Base de Métricas para a Estação TABA*, Dissertação de M. Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

FAGAN, M.E., 1976, "Design and Code Inspections to Reduce Errors in Program Development", *IBM Systems Journal*, v. 15, n. 3, pp. 182-211.

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

FANTINATO, M., CUNHA, A.C.R., DIAS, S.V., 2004, "AutoTest – Um Framework Reutilizável para a Automação de Teste Funcional de Software", In: *Anais do III Simpósio Brasileiro de Qualidade de Software*, pp. 286-300, Brasília, Brasil.

FARIAS, L., 2002, *Planejamento de Riscos em Ambientes de Desenvolvimento de Software Orientados à Organização*, Dissertação de M. Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

FARIAS, L. *et al.*, 2003, *RiscManager: uma Ferramenta par Planejamento de Riscos com apoio de Gerência do Conhecimento*, I Workshop Tecnologias da Informação e Gerência do Conhecimento, Fortaleza, Brasil.

FENTON, N.E., PFLEEGER, S.L, 1997, *Software Metrics – A Rigorous & Practical Approach*, 2ª Edição, PWS Publishing Company, Boston, Estados Unidos.

FERREIRA, A.I.F., CERQUEIRA, R., ROCHA, A.R., SANTOS, G., MONTONI, M., MAFRA, S., FIGUEIREDO, S., 2005, "Implantação de Processo de Software na BL Informática Um Caso de Sucesso", *In: Anais do IV Simpósio Brasileiro de Qualidade de Software*, Porto Alegre, Brasil.

FIGUEIREDO, S., 2004, *Gerência de Configuração em Ambientes de Desenvolvimento de Software Orientados à Organização*, Monografia de Final de Curso de B.Sc., UFRJ, Rio de Janeiro, Brasil.

FIGUEIREDO, S., ROCHA, A.R., SANTOS, G., MONTONI, M., NATALI, A.C., 2005, "Apoio à Manutenção de Software através de Design Rationale em Ambientes de Manutenção de Software Taba" , *II Workshop de Manutenção de Software Moderna (WMSWM-05)*, Manaus, Brasil.

FREEDMAN, D.P., WEINBERG, G.M., 1982, *Handbook of Walkthroughs, Inspections, and Technical Reviews: Evaluating Programs, Projects, and Products*, 3ª Edição, Dorset House Publishing, Nova York, Estados Unidos

FUGGETTA, A., 2000, "Software Process: a roadmap", *In: The Future of Software Engineering*, A.Finkelstein (ed).

GALIN, D., AVRAHAMI, M., 2005, "Do SQA Programs Work – CMM Works. A Meta Analysis", *In: IEEE International Conference on Software - Science, Technology & Engineering (SwSTE'05)*, pp. 95-100.

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

GOLUBIC, S., 2003, "On Software Quality Verification in the Object-Oriented Development Environment", *In: 7th International Conference on Telecommunications – ConTEL*, pp.557-563, Zagreb, Croacia.

HILBURN, T.B., TOWHIDNEJAD, M., 2000, "Software Quality: A Curriculum Postscript?", *In: Proceedings of the SIGCSE 2000*, pp. 167-171, Texas, Estados Unidos.

HUMPHREY, W.S., 1989, *Managing the Software Process*, Addison-Wesley Publishing Company, Boston, Estados Unidos.

HUMPHREY, W.S., 1997, *Introduction to the Personal Software Process*, Addison-Wesley Publishing Company, Boston, Estados Unidos.

IEEE 1028:1998, IEEE Standard for Software Reviews, Institute of Electrical and Electronics Engineers.

IEEE 610.12, 1990, *IEEE Standard Glossary of software engineering terminology*, IEEE Press.

ISO/IEC 12207, 1998, *Tecnologia de Informação - Processos de ciclo de vida de Software*, ABNT - ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, Rio de Janeiro: ABNT.

ISO/IEC 12207, 2002, *ISO/IEC 12207 Information Technology - Amendment to ISO/IEC 12207*, INTERNATIONAL STANDARD ORGANIZATION, Montreal: ISO/IEC JTC1 SC7.

ISO/IEC 12207, 2004, *ISO/IEC 12207 Information Technology - Amendment to ISO/IEC 12207*, INTERNATIONAL STANDARD ORGANIZATION, Montreal: ISO/IEC JTC1 SC7.

ISO/IEC 14598, 1998, *Information Technology - Software Product Evaluation,* INTERNATIONAL STANDARD ORGANIZATION.

ISO/IEC 15504, 2003, *Information Technology – Software Process Assessment*, Parts 1-9, International Organization for Standardization and the International Electrotechnical Commission, Geneva, Switzerland.

ISO/IEC 15939, 2002, *Information Technology—Software Engineering—Software Measurement.*

ISO/IEC 9126, 2001, Information Technology – Software Evaluation – Quality Characteristics and Guidelines for their use, INTERNATIONAL STANDARD ORGANIZATION.

JUNG, H., KIM, S., CHUNG, C., 2004, "Measuring Software Product Quality: A Survey of ISO/IEC 9126," *IEEE Software*, v. 21, n. 5, pp. 88-92.

JURISTO, N., MORENO, A.M., VEGAS, S., 2003, "Limitations of Empirical Testing Technique Knowledge", *Lecture Notes on Empirical Software Engineering*, Series on Software Engineering and Knowledge Engineering, v. 12, pp.1-38.

KALINOWSKI, M., 2004, *Infra-Estrutura Computacional para Apoio ao Processo de Inspeção de Software*, Tese de M. Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

KALINOWSKI, M., SPINOLA, R., TRAVASSOS, G.H., 2004, "Infra-Estrutura Computacional para Apoio ao Processo de Inspeção de Software", *In: III Simpósio Brasileiro de Qualidade de Software*, pp. 176-190, Brasília, Brasil.

KALINOWSKY, M., TRAVASSOS, G.H., 2004, "A Computational Framework for Supporting Software Inspections", *In: IEEE 19th International Conference on Automated Software Engineering - ASE'04*, Los Angeles: IEEE Computer Press, v. 1. pp. 46-55.

KAN, S.H., 2003, *Metrics and Models in Software Quality Engineering*, 2ª Edição, Addison-Wesley Publishing Company, Boston, Estados Unidos.

KIRNER, T.G., CRUZ, E.R., MONTEBELO, M.I., 2005, "Avaliação das Técnicas OORT para Inspeção de Especificações de Requisitos em UML: um estudo empírico", In: *Anais do IV Simpósio Brasileiro de Qualidade de Software*, pp. 469-483, Porto Alegre, Brasil, Junho.

LAITENBERGER, O., VEGAS, S., CIOLKOWOSKI, M., 2002, *The State of the Practice of Review and Inspection Technologies in Germany*, Tech Report Number: ViSEK/011/E.

LANUBILE, F., MALLARDO, T.M., CALEFATO, F., 2003, "Tool Support for Geographically Dispersed Inspection Teams", *Software Process Improvement and Practice*, v. 8, n.1, pp. 217-231.

LIMA, G.M.P.S., 2005, "Heurísticas para Identificação da Ordem de Integração de Classes em Testes Aplicados a Software Orientado a Objetos", Tese de M. Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

MADU, C.N., KUEI, C.H., JACOB, R.A., 1996, "An Empirical Assessment of the Influence of Quality Dimensions on Organizational Performance," *International Journal of Production Research,* v. 34, n. 7, pp. 1943-62.

MAGUIRE, S., 1993, *Writing Solid Code*, Microsoft Press.

MALDONADO, J.C., FABRI, S.C.P.F., 2001, *Verificação e Validação de Software*, In: Qualidade de Software – Teoria e Prática, Prentice Hall, pp. 66-73, São Paulo, Brasil.

MARTINS, F., 2004, *Ambientes de Desenvolvimento de Software Orientado à Organização Baseado em Instrumentação Virtual*, Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

McCALL, J.L. *et al.*, 1977, *Factors in Software Quality*, National Technical Information Service.

McCONNELL, S., 2004, *Code Complete*, 2ª Edição, Microsoft Press, Washington, Estados Unidos.

McGARRY, J., CARD, D., 2001, JONES, C., LAYMAN, B., CLARK, E., DEAN, J., HALL, F., 2001, *Practical Software Measurement: Objective Information for Decision Makers*, Addison-Wesley Publishing Company, Boston, Estados Unidos.

MINISTÉRIO DA CIÊNCIA E TECNOLOGIA - MCT, 2002, *Qualidade e Produtividade no Setor de Software Brasileiro*, Brasília, Brasil.

MONTONI, M., 2003, *Aquisição de Conhecimento: Uma Aplicação no Processo de Software*, Dissertação de M. Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

MONTONI, M., MIRANDA, R., ROCHA, A.R., TRAVASSOS, G.H., 2004, "Knowledge Acquisition and Communities of Practice: an Approach to Convert Individual Knowledge into Multi-organizational Knowledge", *In: Proceedings of the LSO 2004*, pp. 110-121, Banff, Canadá.

MONTONI, M., ROCHA, A.R.C., TRAVASSOS, G.H., 2003, "ACKNOWLEDGE: uma Ferramenta para Aquisição do Conhecimento no Desenvolvimento de Software", *In: I Workshop Tecnologias da Informação e Gerência do Conhecimento*, Fortaleza, Brasil.

MONTONI, M., SANTOS, G., VILLELA, K., ROCHA, A.R., TRAVASSOS, G., FIGUEIREDO, S., MAFRA, S., ALBUQUERQUE, A., MIAN, P., 2005, "Enterprise-Oriented Software Development Environments to Support Software Products and Processes Quality Improvement", *In: Proceedings of the PROFES 2005*, pp.370-384, Oulu, Finlândia.

MOURA, L., 1992, *Taxonomia de Ambientes de Desenvolvimento de Software*, Tese de M. Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

MPS.BR – Melhoria de Processo do Software Brasileiro, Guia Geral (v. 1.0) (2005) URL: .http://www.softex.br/cgi/cgilua.exe/sys/start.htm?sid=191

MYERS, G., 2004, *The Art of Software Testing*, 2ª Edição, John Wiley & Sons, Hoboken, New Jersey, Estados Unidos.

NASA, 1993, "Software Formal Inspections Standard", *NASA-STD-2202-9*, National Aeronautics and Space Administration.

NATALI, A.C.C., ROCHA, A.R., TRAVASSOS, G.H., 2005, "An Approach to Support Software Verification and Validation", *III Workshop de Teses e Dissertações em Qualidade de Software, In: Anais do IV Simpósio Brasileiro de Qualidade de Software*, Porto Alegre, RS, Brasil.

NBR ISO 9000:2000, Sistemas de gestão da qualidade - Fundamentos e vocabulário.

NGWENYAMA, O., NIELSEN, P., 2003, "Competing Values in Software Process Improvement: Assumption Analysis of the CMM from an Organizational Culture Perspective", *IEEE Transactions on Engineering Management*, v. 50, n. 1, pp. 100-112.

NUNES, E. D., SILVA, R., ROCHA, A. R., NATALI, A. C., SANTOS, G., 2005, "Uma Abordagem para Implantação de Processos de Software com ISO 9001 e CMMI", *In: Anais do IV Simpósio Brasileiro de Qualidade de Software*, Porto Alegre, Brasil.

OLIVEIRA, K., 1999, *Modelo para Constrição de Ambientes de Desenvolvimento de Software Orientados a Domínio*, Tese de D. Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

OLIVEIRA, K.M., SANTOS, G., ZLOT, F., ROCHA, A.R.C., 2000, *A Estação TABA e Ambientes de Desenvolvimento de Software Orientados a Domínio*, Caderno de Ferramentas - XIV Simposio Brasileiro de Engenharia de Software, pp. 343-346, João Pessoa, Brasil.

OSTERWEIL, L., 1996, "Strategic Directions in Software Quality", *ACM Computing Surveys*, v. 28, n. 4, pp. 738-750.

PFLEEGER, S.L., 2004, *Engenharia de Software – Teoria e Prática*, 2ª edição, Prentice Hall, São Paulo, Brasil.

PORTER, A.A., JOHNSON, P.M., 1997, "Assessing Software Review Meetings: Results of a Comparative Analysis of Two Experimental Studies", *IEEE Transactions on Software Engineering*, v. 32, n. 3, pp. 129-145.

PORTER, A.A., VOTTA, L.G., BASILI, V.R., 1995, "Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment", *IEEE Transactions on Software Engineering*, v. 21, n. 6, pp. 563-575.

PRESSMAN, R.S., 2001, *Software Engineering: A Practitioner's Approach*, 5ª Edição, McGraw-Hill, Nova York, Estados Unidos.

PRICE, A.M., HERBERT, J.S., 2000, "Técnicas de teste de software orientado a objetos", In: Revista de Informática Teórica e Aplicada do Instituto de Informática da Universidade Federal do Rio Grande do Sul, v. 6, n. 1.

QUAQUARELLI, B., 1997, "Quality Improvement through VErification PROcess (PROVE)", *In: 1th International Software Quality Week Europe (QWE'97)*, Bruxelas, Bélgica.

RAKITIN, S.R., 2001, *Software Verification and Validation for Practitioners and Managers*, 2ª Edição, Artech House Publishers, Norwood, Estados Unidos.

REIS, L.N.M., 2005, *Apoio Automatizado para Aplicação de Técnicas de Leitura Orientada a Objetos (OORTs)*, Tese de M. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

ROCHA, A.R.C., 1983, *Um Modelo para Avaliação da Qualidade de Especificações*, Tese de D. Sc., PUC/RJ, Rio de Janeiro, Brasil.

ROCHA, A.R.C., 1987, *Análise e Projeto Estruturado de Sistemas*, Ed. Campus, São Paulo, Brasil.

ROCHA, A.R.C., MALDONADO, J.C., WEBER, K.C., 2001, *Qualidade de Software – Teoria e Prática*, Prentice Hall, São Paulo, Brasil.

ROCHA, A.R., MONTONI, M., SANTOS, G., OLIVEIRA, K., NATALI, A.C., MIAN, P., CONTE, T., MAFRA, S., BARRETO, A.., ALBUQUERQUE, A. FIGUEIREDO, S., SOARES, A., BIANCHI, F., CABRAL, R., DIAS, A., 2005a, "Fatores de Sucesso e Dificuldades na Implementação de Processos de Software Utilizando o MR-MPS e o CMMI", *PROQUALITY – Qualidade na Produção de Software*, pp. 13-18.

ROCHA, A.R.C., PALERMO, S., 1989, *Software Quality Assurance in HEP*, North Holland, Computer Physics Communications 57 (1989).

ROCHA, A.R., MONTONI, M., SANTOS, G., OLIVEIRA, K., NATALI, A.C., MIAN, P., CONTE, T., MAFRA, S., BARRETO, A.., ALBUQUERQUE, A. FIGUEIREDO, S., SOARES, A., BIANCHI, F., CABRAL, R., DIAS, A., 2005b, "Fatores de Sucesso e Dificuldades na Implementação de Processos de Software Utilizando o MR-MPS e o CMMI", *I Encontro de Implementadores de MPS.BR*, Brasília, Brasil.

ROCHA, A.R., SOUZA, J.M., AGUIAR, T.C., 1990, "TABA: A Heuristic Workstation for Software Development". In: *Proceedings of COMPEURO 90*, pp. 126-129, Tel Aviv, Israel, May.

SANDERS, J., CURRAN, E., 1994, *Software Quality – A Framework for Success in Software Development and Support*, Addison-Wesley Publishing Company, Boston, Estados Unidos.

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

SANTOS, G. *et al.*, 2003, *SAPIENS: uma Ferramenta para Descrição e Recuperação de Competências em uma Organização*, I Workshop Tecnologias da Informação e Gerência do Conhecimento, Fortaleza, Brasil.

SANTOS, G., MONTONI, M., ROCHA, A.R., FIGUEIREDO, S., MAFRA, S., ALBUQUERQUE, A., PARET, B.D., AMARAL, M., 2005, "Using a Software Development Environment with Knowledge Management to Support Deploying Software Processes in Small and Medium Size Companies", *In: Proceedings of the LSO 2005*, pp. 72-76, Kaiserslautern, Alemanha.

SANTOS, G., VILLELA, K., SCHNAIDER, L., ROCHA, A.R.C., TRAVASSOS, G.H., 2004, "Building Ontology Based Tools for a Software Development Environment", *In : Proceedings of the LSO 2004*, pp. 19-30, Banff, Canadá.

SAUER, C., JEFFERY, D., LAND, L., YETTON, P., 2000, "The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research", *IEEE Transactions on Software Engineering*, v. 26, n. 1, pp. 1-14.

SAYÃO, M., LEITE, J.C.S.P., TRAVASSOS, G.H., 2005, "Uso de Agentes no Processo de Requisitos em Ambientes Distribuídos de Desenvolvimento", *In: Anais do Workshop em Engenharia de Requisitos (WER05)*, v. 8. pp. 135-147, Porto, Portugal.

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

SCHNAIDER, L., SANTOS, G., MONTONI, M., ROCHA, A.R.C., 2004, "MedPlan: uma Abordagem para Medição e Análise em Projetos de Desenvolvimento de Software", in Anais do III Simpósio Brasileiro de Qualidade de Software, Brasília, Brasil.

SCHULMEYER, G.G., MACKENZIE, G.R., 1999, *Verification & Validation of Modern Software-Intensive Systems*, New Jersey, Prentice-Hall Inc.

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

SILVA, L.F.S., CHAPETTA, W.A., TRAVASSOS, G.H., 2004, "Inspeções de Requisitos de Software Utilizando PBR e Apoio Ferramental", *In: Anais do 3º Simpósio Brasileiro de Qualidade de Software*, pp.139-152, Brasília, Brasil.

SOFTEX, 2005, *Associação para Promoção da Excelência do Software Brasileiro*, URL: http://www.softex.br.

SOLINGEN, R., BERGHOUT, E., 1999, The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development, McGrawHill.

SPAWAR – Space and Naval Warfare Systems Center, 1997, Formal Inspection Process, Versão 2.2.

SPEHAR, G., 2005, "Quality Management of Software Projects", Full Knowledge LLC, disponível em http://www.full-knowledge.com/papers/12-Full-Knowledge-Quality-2005-10-27-V07.pdf. Site acessado em Janeiro de 2006.

STRONG, D.M., LEE, Y.W., WANG, R.Y., 1997, "Data quality in context", *Communications of the ACM*, v. 40, n. 9, pp. 103-110.

STYLIANOU, A.C., KUMAR, R.L., 2000, "An integrative framework for IS quality management", *Communications of the ACM*, v. 43, n. 9, pp. 99-104.

SWEBOK, 2004, *Software Engineering Body of Knowledge*, http://www.swebok.org.

THELIN, T., 2002, *Empirical Evaluations of Usage-Based Reading and Fault Content Estimation for Software Inspections*, Doctoral Thesis, Lund University, Sweden.

TIAN, J., 2005, *Software Quality Engineering - Testing, Quality Assurance, and Quantifiable Improvement*, John Willey & Sons, Hoboken, Estados Unidos.

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

TRAVASSOS, G.H., SHULL, F., CARVER, J., 2001, "Working with UML: A Software Design Process Based on Inspections for the Unified Modeling Language", *In: Advances in Computers*, v.54, n.1, pp.35-97.

TRAVASSOS, G.H., SHULL, F., CARVER, J., BASILI, V., 2002, "Reading Techniques for OO Design Inspections", *Relatório Técnico ES-575/02*, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ.

VEGAS, S., 2002, *Characterization Schema for Selecting Software Testing Techniques*, PhD Thesis, Universidad Politécnica de Madrid, Madri, Espanha.

VILLELA, K., 2004, Definição e Construção de Ambientes de Desenvolvimento de Software Orientados a Organização, Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

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

VILLELA, K., TRAVASSOS, G.H., ROCHA, A.R., 2001, "Ambientes de Desenvolvimento de Software Orientados a Organização", *IDEAS'2001 - Workshop Ibero-americano de Ingeniería de Requisitos y Ambientes de Software*, San Jose, Costa Rica, Abril.

VOTTA, L.G., 1993, "Does Every Inspection Need a Meeting", *ACM Software Engineering Notes*, v. 8, n. 5, pp. 107-114.

WAGNER, S., SEIFERT, T., 2005, "Software Quality Economics for Defect-Detection Techniques Using Failure Prediction", *ACM Software Engineering Notes*, v. 30, n. 4, pp. 1-6.

WALLACE, D.R., IPPOLITO, L.M., CUTHILL, B., 1996, "Reference Information for the Software Verification and Validation Process", *National Institute of Standards and Technology*, NIST Special Publication 500-234.

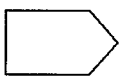
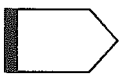
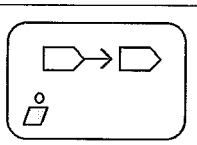
WINKLER, D., BIFFL, S., THURNHER, B., 2005, "Investigating the Impact of Active Guidance on Design Inspection", *6th International Conference, PROFES 2005*, Springer, Lecture Notes in Computer Science, v. 3547 (2005), pp. 458 - 473.

ZLOT, F., 2002, *Conhecimento de Tarefa em Ambientes de Desenvolvimento de Software Orientados a Domínio*, Dissertação de M. Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

ZLOT, F. *et al.*, 2002, *Modeling Task Knowledge to Support Software Development, In: Proceedings of the 14th international conference on Software engineering and knowledge engineering - SEKE'02*, pp. 35-42, Ischia, Itália.

ANEXO I

Notação utilizada na Modelagem do Processo de Verificação de Software

Objeto	Notação	Definição
Ator		Objeto que representa uma pessoa, agente ou unidade organizacional.
Atividade		Objeto referente ao conceito de atividade.
Atividade Composta		Objeto referente ao conceito de atividade composta, o que significa que ela pode ser decomposta em sub-atividades.
Estado Inicial		Objeto puramente notacional, proveniente dos diagramas de estado e que indica onde é iniciado o fluxo de atividades que definem um processo ou uma atividade composta.
Estado Final		Objeto puramente notacional, proveniente dos diagramas de estado e que indica onde é encerrado o fluxo de atividades que definem um processo ou uma atividade composta.
Documento		Objeto referente a um documento.
Operação Lógica OR		Adorno que indica a operação lógica OU.
Junção após uma operação lógica		Notação sem adorno, que é utilizada como elemento de junção após uso de uma operação lógica que atua como elemento de divisão.
Área de Ator		Área que agrupa atividades executadas por um ator ou grupo de atores. O ator ou o grupo de atores também precisa estar contido na área.
Fluxo de Entrada/Saída		Ligação que estabelece um insumo (se o fluxo é de entrada) ou um produto de uma atividade (se o fluxo é de saída).

ANEXO II

Conjunto de Características de Qualidade, Critérios, Questões e Métricas para Verificação de Software

Este anexo contém o conjunto de características de qualidade, critérios, questões e métricas para verificação de software que foram consideradas na ferramenta *VerificationManager*.

As características de qualidade presentes no conjunto definido foram organizadas em dois grupos:

- Características de Qualidade presentes na norma internacional ISO/IEC 9126 (ISO/IEC, 2001) mostradas na tabela A2.1; e,
- Características Gerais de Qualidade mostradas na tabela A2.3. Neste grupo estão as características comuns a qualquer software, que não estão presentes na ISO/IEC 9126, mas que foram definidas a partir dos diversos trabalhos analisados.

A tabela A2.2 apresenta o mapeamento entre as características de produto e os artefatos nos quais elas podem ser avaliadas com o objetivo de facilitar a análise da cobertura das características de qualidade desejáveis durante a verificação.

A tabela A2.4 apresenta o mapeamento entre as características gerais e os artefatos nos quais elas podem ser avaliadas com o objetivo de facilitar a análise da cobertura das características de qualidade desejáveis durante a verificação.

As demais tabelas apresentam os conjuntos definidos para a verificação de alguns artefatos, a saber:

1. Plano do Projeto;
2. Especificação de Requisitos do Cliente;
3. Especificação de Requisitos do Sistema;
4. Projeto da Arquitetura do Sistema;
5. Especificação de Requisitos do Software;
6. Modelo de Análise;
7. Modelo de Projeto;

8. Modelo de Análise & Projeto: Análise e projeto do software juntos em um único *artefato*;
9. Plano de Testes de Unidades;
10. Código de Unidades;
11. Relatório do Teste de Unidades;
12. Plano de Testes de Integr. do Software;
13. Relatório do Teste de Integr. do Software;
14. Relatório do Teste de Integr. do Sistema;
15. Plano de Testes do Software;
16. Software;
17. Relatório do Teste do Software;
18. Plano de Testes de Integr. do Sistema;
19. Plano de Testes do Sistema;
20. Sistema; e,
21. Relatório do Teste do Sistema.

Tabela A2.1 - Características de Qualidade Presentes na ISO 9126

Característica	Sub-Característica	Descrição
Funcionalidade	Adequação	Capacidade de o produto de software de fornecer um conjunto de funções adequado para as tarefas especificadas e os objetivos dos usuários.
	Acurácia	Capacidade de o produto de software de fornecer os resultados corretos ou acordados com o grau necessário de precisão.
	Interoperabilidade	Capacidade de o produto de software interagir com um ou mais sistemas especificados.
	Segurança	Capacidade de o produto de software proteger as informações e os dados de forma que pessoas ou sistemas não autorizados não possam lê-los ou modificá-los e pessoas e sistemas autorizados não tenham negado o acesso aos mesmos.
	Conformidade de funcionalidade	Capacidade de o produto de software aderir a normas, convenções ou regulamentações previstas em leis e prescrições similares, relacionadas à funcionalidade.
Confiabilidade	Maturidade	Capacidade de o produto de software evitar falhas provocadas por defeitos no software.
	Tolerância a defeitos	Capacidade de o produto de software manter um nível de desempenho especificado em casos de defeitos no software ou de violação de suas interfaces especificadas.
	Recuperabilidade	Capacidade de o produto de software restabelecer o nível de desempenho especificado e recuperar os dados diretamente afetados, em caso de uma falha.
Eficiência	Comportamento em relação ao tempo	Capacidade de o produto de software fornecer tempo de resposta e de processamento e velocidade na execução de suas funções sob condições estabelecidas.
	Utilização de recursos	Capacidade de o produto de software usar quantidade e tipos adequados de recursos quando o software realiza suas funções sob condições estabelecidas.
Manutenibilidade	Analisabilidade	Capacidade de o produto de software ser diagnosticado com relação a deficiências ou causas de falhas no software, ou para identificar as partes a serem modificadas.
	Modificabilidade	Capacidade de o produto de software permitir que uma modificação especificada seja implementada.
	Estabilidade	Capacidade de o produto de software evitar efeitos inesperados, ocasionados por modificações no software.
	Testabilidade	Capacidade de o produto de software permitir que o software modificado seja validado.
Portatilidade	Adaptabilidade	Capacidade de o produto de software ser adaptado a diferentes ambientes especificados, sem a necessidade de aplicação de outras ações ou meios além daqueles fornecidos para essa finalidade pelo software considerado.
	Capacidade para ser instalado	Capacidade de o produto de software ser instalado em um ambiente especificado.
	Capacidade para substituir	Capacidade de o produto de software ser usado para substituir um outro software especificado, para o mesmo objetivo no mesmo ambiente.
	Conformidade da portabilidade	Capacidade de o produto de software aderir a padrões ou convenções relacionadas à portabilidade.

Tabela A2.2 - Mapeamento entre as Características Presentes na ISO 9126 e os Artefatos nos quais elas podem ser avaliadas

Característica	Sub-Característica de Qualidade	Espec. de Req. do Cliente	Espec. de Req. do Sistema	Projeto da Arquitet. do Sistema	Espec. de Req. do Software	Modelo de Análise	Modelo de Projeto	Modelo de Análise & Projeto	Plano de Testes de Unidades	Código de Unidades	Rel. do Teste de Unidades	Plano de Testes de Integr. do SW.	Rel. do Teste de Integr. (Software e Sistema)	Plano de Testes do SW.	Software	Rel. do Teste do SW.	Plano de Testes de Integr. do Sistema	Plano de Testes do Sistema	Sistema	Rel. do Teste do Sistema
Funcionalidade	Adequação	RPP	RPP		RPP					RPP				RPP	T					
	Acurácia									T					T					
	Interoperabilidade		RPP	RPP															T	
	Segurança	RPP	RPP		RPP										T				T	
	Conformidade de funcionalidade														T				T	
Confiabilidade	Maturidade									T	RPP		RPP		T	RPP			T	RPP
	Tolerância a defeitos									RPP T					T					
	Recuperabilidade									T					T				T	
Utilizabilidade	Inteligibilidade																			
	Apreensibilidade																			
	Operacionalidade																			
Eficiência	Comportamento em relação ao tempo									T					T				T	
	Utilização de recursos									T					T				T	
Manutenibilidade	Analisabilidade									RPP										
	Modificabilidade					RPP	RPP	RPP		RPP										
	Estabilidade									T					T					
	Testabilidade	RPP	RPP		RPP		RPP	RPP	RPP			RPP	RPP	RPP			RPP	RPP		

Tabela A2.2 - Mapeamento entre as Características Presentes na ISO 9126 e os Artefatos nos quais elas podem ser avaliadas (Continuação)

Característica	Sub-Característica de Qualidade	Espec. de Req. do Cliente	Espec. de Req. do Sistema	Projeto da Arquitet. do Sistema	Espec. de Req. do Software	Modelo de Análise	Modelo de Projeto	Modelo de Análise & Projeto	Plano de Testes de Unidades	Código de Unidades	Rel. do Teste de Unidades	Plano de Testes de Integr. do SW.	Rel. do Teste de Integr. (Software e Sistema)	Plano de Testes do SW.	Software	Rel. do Teste do SW.	Plano de Testes de Integr. do Sistema	Plano de Testes do Sistema	Sistema	Rel. do Teste do Sistema
Portatilidade	Adaptabilidade														T				T	
	Capacidade para ser instalado														T					
	Capacidade para substituir														T					
	Conformidade da portabilidade														T				T	

LEGENDA	
RPP	Verificação através de Revisão por Pares
T	Teste

Tabela A2.3 - Características Gerais de Qualidade

Característica	Sub-Característica	Descrição
Implementabilidade	Viabilidade Econômica	Viabilidade de o produto poder ser construído com uma relação custo/benefício aceita pelos usuários e desenvolvedores.
	Viabilidade Financeira	Viabilidade de o produto poder ser construído mediante a disponibilidade do capital necessário para o seu desenvolvimento.
	Viabilidade Tecnológica	Viabilidade de o produto poder ser construído considerando-se a existência e a disponibilidade da tecnologia necessária para o seu desenvolvimento.
	Viabilidade de Cronograma	Viabilidade de o produto poder ser construído dentro do limite de tempo planejado, considerando possíveis ocorrências de imprevistos e com flexibilidade para introdução de atividades não projetadas e/ou contingenciais, mantendo a qualidade definida para o produto.
	Viabilidade de mão de obra	Viabilidade de o produto poder ser construído, considerando-se a existência e a disponibilidade da mão de obra necessária para o seu desenvolvimento.
	Viabilidade Legal	Viabilidade de o produto poder ser construído, considerando-se as leis vigentes.
Fidedignidade	Completeza	Característica de o produto e a sua documentação serem completos em relação à funcionalidade para a qual foi projetado, possuindo apenas informações úteis e necessárias para atender os objetivos dos seus usuários finais.
	Consistência	Característica de o produto e a sua documentação serem consistentes nas diversas partes que os compõem não possuindo informações contraditórias ou mesma informação com significados diferentes na várias partes que os compõem.
Legibilidade	Clareza	Característica de o produto estar especificado, modelado e implementado da forma mais clara possível, isento de práticas que o tornem complexo e de difícil entendimento.
	Correção da representação	Característica de o produto estar correto do ponto de vista do uso da linguagem de especificação adotada, no que se refere a notação, semântica, sintaxe e formato de documentação.
Manipulabilidade	Rastreabilidade	Característica de o produto possuir uma documentação e código que permitam a busca de informações através da sequência de agregação de detalhes de um determinado aspecto, desde sua visão mais geral até a mais detalhada, e vice-versa.

Tabela A2.4 - Mapeamento entre as Características Gerais e os Artefatos nos quais elas podem ser avaliadas

Característica	Sub-Característica	Plano do Projeto	Espec. de Requisitos do Cliente	Espec. de Requisitos do Sistema	Projeto da Arquitetura do Sistema	Espec. de Requisitos do Software	Modelo de Análise	Modelo de Projeto	Modelo de Análise & Projeto	Plano de Testes de Unidades	Código de Unidades	Relat. do Teste de Unidades	Plano de Testes de Integr. do Software	Plano de Testes do Software	Relat. do Teste do Software	Plano de Testes de Integr. do Sistema	Plano de Testes do Sistema
Implementabilidade	Viabilidade Econômica	RPP															
	Viabilidade Financeira	RPP															
	Viabilidade Tecnológica	RPP	RPP	RPP	RPP	RPP	RPP	RPP	RPP			RPP			RPP		
	Viabilidade de Cronograma	RPP															
	Viabilidade de mão de obra	RPP															
	Viabilidade Legal	RPP															
Fidedignidade	Completeza	RPP	RPP	RPP	RPP	RPP	RPP	RPP	RPP	RPP			RPP	RPP		RPP	RPP
	Consistência		RPP	RPP	RPP	RPP	RPP	RPP	RPP	RPP	RPP		RPP	RPP		RPP	RPP
Legibilidade	Clareza		RPP	RPP	RPP	RPP	RPP	RPP	RPP	RPP	RPP		RPP	RPP		RPP	RPP
	Correção da representação		RPP	RPP		RPP											
Manipulabilidade	Rastreabilidade		RPP	RPP	RPP	RPP	RPP	RPP	RPP	RPP	RPP			RPP			RPP

LEGENDA	
RPP	Verificação através de Revisão por Pares

Tabela A2.5 – Conjunto para a Verificação do Plano do Projeto

Artefato	Método	Característica de Qualidade	Critério	Checklist
Plano do Projeto	Revisão por Pares	Completeza	Completeza	A estrutura analítica do projeto está definida?
				A estrutura organizacional do projeto está definida?
				O cronograma do projeto está definido?
				Os recursos humanos necessários estão identificados?
				Os custos associados ao projeto estão identificados?
				Os riscos associados ao projeto estão identificados?
		Viabilidade Econômica	Aceitabilidade de custos	O custo estimado para o desenvolvimento do produto é aceitável?
			Relevância de benefícios	Os benefícios econômicos associados ao produto estão identificados?
			Compatibilidade Custo/Benefício	A relação custo/benefício é aceitável?
				A comercialização do produto poderá ser lucrativa?
		Viabilidade Financeira	Existência de capital	Existe capital suficiente para custear o desenvolvimento?
			Disponibilidade de capital	Existem recursos financeiros disponíveis?
		Viabilidade Tecnológica	Disponibilidade da tecnologia	Existem recursos tecnológicos para desenvolver o produto?
		Viabilidade de Cronograma	Adequabilidade de Cronograma	O prazo de entrega do produto é possível de ser atingido considerando as restrições específicas do projeto?
				Todas as fases do desenvolvimento estão consideradas no cronograma?
			Flexibilidade de Cronograma	Existe flexibilidade de cronograma?
		Viabilidade de mão de obra	Disponibilidade de mão de obra	A mão de obra estimada é suficiente para o projeto?
				A mão de obra alocada possui o conhecimento e as habilidades necessárias ao projeto?
		Viabilidade Legal	Viabilidade Legal	Os direitos de propriedade do produto estão adequadamente protegidos?
				A operação do produto é viável do ponto de vista legal?

Tabela A2.6 – Conjunto para a Verificação da Especificação de Requisitos do Cliente

Artefato	Método	Característica de Qualidade	Critério	Checklist
Especificação de Requisitos do Cliente	Revisão por Pares	Clareza	Clareza	O significado de cada requisito é compreensível?
				A terminologia usada é de conhecimento do usuário/patrocinador?
				Os requisitos estão descritos com um nível de detalhes suficiente para o entendimento?
				Os requisitos podem ser entendidos e desenvolvidos por um grupo independente?
		Completeza	Completeza	Todas as necessidades do cliente (inclusive as implícitas) estão identificadas?
				Todas as restrições e suposições do cliente estão identificadas?
				Todos os requisitos e restrições estão priorizados?
		Consistência	Consistência Interna	Os requisitos são consistentes entre si?
			Consistência Externa	Os requisitos do cliente são consistentes com os requisitos de documentos relacionados (licitações, contratos, pedidos de proposta, etc)?
				Os requisitos do cliente são consistentes com as necessidades de aquisição?
		Rastreabilidade	Rastreabilidade para as necessidades de aquisição	Há rastreabilidade bidirecional entre os requisitos do cliente e as necessidades de aquisição?
		Testabilidade	Viabilidade de testes	Cada requisito é testável?
		Viabilidade Tecnológica	Disponibilidade da tecnologia	Existe tecnologia disponível para implementar todos os requisitos especificados?
		Adequação	Adequação às necessidades do cliente	Os requisitos descritos são adequados às necessidades do cliente?
		Correção da representação	Aderência às normas estabelecidas pelo cliente	A especificação foi gerada considerando as normas estabelecidas pelo cliente?
			Nível adequado de abstração	Os requisitos descrevem o problema (necessidades do cliente) e não sua solução?
		Segurança	Controle de acesso	Os requisitos de proteção e segurança estão especificados?

Tabela A2.7 – Conjunto para a Verificação da Especificação de Requisitos do Sistema

Artefato	Método	Característica de Qualidade	Critério	Checklist
Especificação de Requisitos do Sistema	Revisão por Pares	Clareza	Clareza	O significado de cada requisito é compreensível?
				Os requisitos estão descritos com um nível de detalhes suficiente para o entendimento?
				Os requisitos podem ser entendidos e desenvolvidos por um grupo independente?
		Completeza	Completeza	Os objetivos do sistema estão definidos?
				Todos os requisitos, restrições e suposições do sistema estão identificados?
				Todos os requisitos e restrições estão priorizados?
		Consistência	Consistência Interna	Os requisitos são consistentes entre si?
			Consistência Externa	Os requisitos do sistema são consistentes com as necessidades de aquisição?
		Rastreabilidade	Rastreabilidade para os requisitos do cliente	Há rastreabilidade bidirecional entre os requisitos do cliente e os requisitos do sistema?
		Viabilidade Tecnológica	Disponibilidade da tecnologia	Existe tecnologia disponível para implementar todos os requisitos especificados?
			Viabilidade do projeto da arquitetura do sistema	Existe tecnologia disponível para desenvolver o projeto da arquitetura do sistema de acordo com os requisitos especificados?
		Testabilidade	Viabilidade de testes	Cada requisito é testável?
		Adequação	Adequação às necessidades do cliente	Os requisitos descritos são adequados às necessidades do cliente?
		Segurança	Controle de acesso	Todos os usuários do sistema estão identificados?
				Os requisitos de proteção e segurança estão especificados?
		Correção da representação	Aderência às normas estabelecidas pelo cliente	A especificação foi gerada considerando as normas estabelecidas pelo cliente?
		Interoperabilidade	Consistência de interface	Os protocolos de interface com o sistema estão definidos?

Tabela A2.8 – Conjunto para a Verificação do Projeto da Arquitetura do Sistema

Artefato	Método	Característica de Qualidade	Critério	Checklist
Projeto da Arquitetura do Sistema	Revisão por Pares	Clareza	Clareza	O projeto da arquitetura do sistema é fácil de ler e entender?
		Completeza	Completeza	Os componentes do projeto da arquitetura contemplam todos os requisitos do sistema?
		Consistência	Consistência Externa	Os componentes do projeto da arquitetura são consistentes com os requisitos do sistema?
		Rastreabilidade	Rastreabilidade para os requisitos do sistema	Há rastreabilidade bidirecional entre os componentes do projeto da arquitetura e os requisitos do sistema?
		Viabilidade Tecnológica	Disponibilidade da tecnologia	Existe tecnologia disponível para implementar todos os componentes do projeto da arquitetura?
		Interoperabilidade	Consistência de interface	Os protocolos de interface entre os componentes do sistema estão definidos?

Tabela A2.9 – Conjunto para a Verificação da Especificação de Requisitos do Software

Artefato	Método	Característica de Qualidade	Critério	Checklist
Especificação de Requisitos do Software	Revisão por Pares	Clareza	Clareza	O significado de cada requisito é compreensível?
				Os requisitos estão descritos com um nível de detalhes suficiente para o entendimento?
				Os requisitos podem ser entendidos e desenvolvidos por um grupo independente?
		Completeza	Completeza	Todos os requisitos, restrições e suposições do software estão identificados?
				Todas as funcionalidades do software estão descritas?
				Os requisitos não-funcionais do software estão identificados?
				Os requisitos legais estão especificados?
				Os documentos que complementam a especificação de requisitos (editais, processos de negócio, pedidos de proposta, etc) estão referenciados na especificação e disponíveis para acesso dos interessados?
				Todos os requisitos e restrições estão priorizados?
				A especificação de requisitos do software apresenta o diagrama de casos de uso?
				Todos os casos de uso estão especificados?
				Os requisitos de interface com o usuário estão definidos?
				Os processos que serão apoiados pelo software estão definidos?
		Consistência	Consistência Interna	Os requisitos são consistentes entre si?
			Consistência Externa	Os requisitos do software são consistentes com os requisitos do cliente?
				Os requisitos do software são consistentes com os requisitos do sistema?
		Rastreabilidade	Rastreabilidade para os requisitos do cliente	Há rastreabilidade bidirecional entre os requisitos do cliente e os requisitos do software?
			Rastreabilidade para os requisitos do sistema	Há rastreabilidade bidirecional entre os requisitos do sistema e os requisitos do software?
			Rastreabilidade para o projeto da arquitetura do sistema	Há rastreabilidade bidirecional entre o projeto da arquitetura do sistema e os requisitos do software?

Tabela A2.9 – Conjunto para a Verificação da Especificação de Requisitos do Software (Continuação)

Artefato	Método	Característica de Qualidade	Critério	Checklist
Especificação de Requisitos do Software	Revisão por Pares	Viabilidade Tecnológica	Viabilidade do projeto de software	Existe tecnologia disponível para implementar todos os requisitos especificados?
		Testabilidade	Viabilidade de testes	Cada requisito é testável?
		Adequação	Adequação às necessidades do cliente	Os requisitos descritos são adequados às necessidades do cliente?
		Segurança	Controle de acesso	Todos os usuários do software estão identificados?
				Os requisitos de segurança estão especificados?
		Correção da representação	Aderência às normas estabelecidas pelo cliente	A especificação foi gerada considerando as normas estabelecidas pelo cliente?

Tabela A2.10 – Conjunto para a Verificação do Modelo de Análise (Características comuns aos paradigmas Estruturado e OO)

Artefato	Método	Característica de Qualidade	Critério	Checklist
Modelo de Análise	Revisão por Pares	Clareza	Clareza	O modelo de análise é fácil de ler e entender?
				Os diagramas de transição de estados estão claros e compreensíveis?
				Os modelos estão representados de forma a evitar detalhes de implementação?
		Completeza	Completeza	Todas as funcionalidades identificadas na especificação de requisitos estão modeladas?
		Consistência	Consistência Interna	Os diagramas presentes no modelo de análise são consistentes entre si?
			Consistência Externa	Os diagramas de transição de estados são consistentes com os requisitos do software?
		Viabilidade Tecnológica	Viabilidade do projeto de software	É viável desenvolver o projeto do software a partir do modelo de análise?
		Modificabilidade	Modularidade	O modelo de análise é modular?

Tabela A2.11 – Conjunto para a Verificação do Modelo de Análise (Características para o paradigma Estruturado)

Artefato	Método	Característica de Qualidade	Critério	Checklist
Modelo de Análise	Revisão por Pares	Clareza	Clareza	O significado de cada entidade presente no MER é compreensível?
				O MER está claro e compreensível?
				Os diagramas de fluxo de dados estão claros e compreensíveis?
		Completeza	Completeza	As hierarquias existentes entre as entidades estão modeladas?
				As cardinalidades dos relacionamentos entre as entidades estão definidas?
				Todas as entidades ou relacionamentos com atributos que possuem um número considerável de estados distintos durante seu ciclo de vida foram modelados em diagramas de transição de estados?
				Todos os processos do DFD possuem pelo menos um fluxo de dados de entrada e um de saída?
				Os DFDs representam apenas fluxos de dados e não fluxos de controle?
		Consistência	Consistência Interna	As entidades modeladas são consistentes entre si?
			Consistência Externa	O MER é consistente com os requisitos do software?
				Os DFDs são consistentes com os requisitos do software?
		Rastreabilidade	Rastreabilidade para os requisitos do software	Há rastreabilidade bidirecional entre as entidades modeladas e os requisitos do software?

Tabela A2.12 – Conjunto para a Verificação do Modelo de Análise (Características para o paradigma Orientado a Objetos)

Artefato	Método	Característica de Qualidade	Critério	Checklist
Modelo de Análise	Revisão por Pares	Clareza	Clareza	O significado de cada classe presente nos diagramas de classes é compreensível?
				Os diagramas de classes estão claros e compreensíveis?
				Os diagramas de interação estão claros e compreensíveis?
		Completeza	Completeza	Todas as classes necessárias para atender os requisitos do software no nível do domínio do problema foram identificadas?
				As hierarquias existentes entre as classes estão modeladas?
				As associações, atributos e métodos das classes estão identificados?
				As cardinalidades das associações entre as classes estão definidas?
				Todas as classes cujos objetos possuem um número considerável de estados distintos durante seu ciclo de vida foram modeladas em diagramas de transição de estados?
				Todos os cenários de casos de uso suficientemente complexos foram modelados em diagramas de interação?
Modelo de Análise	Revisão por Pares	Consistência	Consistência Interna	As classes modeladas são consistentes entre si?
			Consistência Externa	Os diagramas de classes são consistentes com os requisitos do software?
				Os diagramas de interação são consistentes com os requisitos do software?
		Rastreabilidade	Rastreabilidade para os requisitos do software	Há rastreabilidade bidirecional entre as classes modeladas e os requisitos do software?

Tabela A2.13 – Conjunto para a Verificação do Modelo de Projeto (Características comuns aos paradigmas Estruturado e OO)

Artefato	Método	Característica de Qualidade	Critério	Checklist
Modelo de Projeto	Revisão por Pares	Clareza	Clareza	Todas as interfaces entre os itens do software estão claramente identificadas?
				O modelo de projeto é fácil de ler e entender?
				Os diagramas de transição de estados estão claros e compreensíveis?
		Completeza	Completeza	Todas as decisões de projeto estão documentadas?
				Todas as funcionalidades identificadas na especificação de requisitos estão detalhadamente modeladas?
				O modelo de projeto considera detalhes específicos da linguagem de programação escolhida e outros detalhes de implementação da solução?
				O modelo de projeto trata adequadamente os requisitos não-funcionais?
				O modelo de projeto inclui detalhes suficientes para desenvolver e manter o software?
				O projeto da interface está descrito em um nível de detalhes suficiente para a construção?
				O projeto do banco de dados está descrito em um nível de detalhes suficiente para a construção?
		Consistência	Consistência Interna	Os diagramas e demais componentes do modelo de projeto são consistentes entre si?
			Consistência Externa	O modelo de projeto é consistente com os requisitos do software?
		Rastreabilidade	Rastreabilidade para os requisitos do software	Há rastreabilidade bidirecional entre o modelo de projeto e os requisitos do software?
		Viabilidade Tecnológica	Viabilidade de codificação	Existe tecnologia disponível para implementar o modelo de projeto?
		Testabilidade	Viabilidade de testes	O modelo de projeto é testável?
		Modificabilidade	Modularidade	O modelo de projeto é modular?

Tabela A2.14 – Conjunto para a Verificação do Modelo de Projeto (Características para o paradigma Estruturado)

Artefato	Método	Característica de Qualidade	Critério	Checklist
Modelo de Projeto	Revisão por Pares	Clareza	Clareza	O MER está claro e compreensível?
				O diagrama hierárquico de funções está claro e compreensível?
		Completeza	Completeza	As hierarquias existentes entre as entidades estão modeladas?
				As cardinalidades dos relacionamentos entre as entidades estão definidas?

Tabela A2.15 – Conjunto para a Verificação do Modelo de Projeto (Características para o paradigma Orientado a Objetos)

Artefato	Método	Característica de Qualidade	Critério	Checklist
Modelo de Projeto	Revisão por Pares	Clareza	Clareza	Os diagramas de classes estão claros e compreensíveis?
				Os diagramas de interação estão claros e compreensíveis?
		Completeza	Completeza	Todas as classes necessárias para atender os requisitos do software foram identificadas?
				As hierarquias existentes entre as classes estão modeladas?
				As associações, atributos e métodos das classes estão identificados?
				As cardinalidades, visibilidades e navegabilidades das associações entre as classes estão definidas?
				Todas as classes cujos objetos possuem um número considerável de estados distintos durante seu ciclo de vida foram modeladas em diagramas de transição de estados?
				Todos os cenários de casos de uso suficientemente complexos foram modelados em diagramas de interação?

Tabela A2.16 – Conjunto para a Verificação do Modelo de Análise&Projeto (Características comuns aos paradigmas Estruturado e OO)

Artefato	Método	Característica de Qualidade	Critério	Checklist
Modelo de Análise & Projeto	Revisão por Pares	Clareza	Clareza	Todas as interfaces entre os itens do software estão claramente identificadas?
				O modelo é fácil de ler e entender?
				Os diagramas de transição de estados estão claros e compreensíveis?
		Completeza	Completeza	Todas as funcionalidades identificadas na especificação de requisitos estão modeladas?
				Todas as decisões de projeto estão documentadas?
				O modelo considera detalhes específicos da linguagem de programação escolhida e outros detalhes de implementação da solução?
				O modelo trata adequadamente os requisitos não-funcionais?
				O modelo inclui detalhes suficientes para desenvolver e manter o software?
				O projeto da interface está descrito em um nível de detalhes suficiente para a construção?
				O projeto do banco de dados está descrito em um nível de detalhes suficiente para a construção?
		Consistência	Consistência Interna	Os diagramas e demais componentes do modelo são consistentes entre si?
			Consistência Externa	O modelo é consistente com os requisitos do software?
		Rastreabilidade	Rastreabilidade para os requisitos do software	Há rastreabilidade bidirecional entre o modelo e os requisitos do software?
		Viabilidade Tecnológica	Viabilidade de codificação	Existe tecnologia disponível para implementar o modelo especificado?
		Testabilidade	Viabilidade de testes	O modelo é testável?
		Modificabilidade	Modularidade	O modelo é modular?

Tabela A2.17 – Conjunto para a Verificação do Modelo de Análise&Projeto (Características para o paradigma Estruturado)

Artefato	Método	Característica de Qualidade	Critério	Checklist
Modelo de Análise & Projeto	Revisão por Pares	Clareza	Clareza	O significado de cada entidade presente no MER é compreensível?
				O MER está claro e compreensível?
				O diagrama hierárquico de funções está claro e compreensível?
		Completeza	Completeza	As hierarquias existentes entre as entidades estão modeladas?
				As cardinalidades dos relacionamentos entre as entidades estão definidas?
				Todas as entidades ou relacionamentos com atributos que possuem um número considerável de estados distintos durante seu ciclo de vida foram modelados em diagramas de transição de estados?
				Todos os processos do DFD possuem pelo menos um fluxo de dados de entrada e um de saída?
				Os DFDs representam apenas fluxos de dados e não fluxos de controle?

Tabela A2.18 – Conjunto para a Verificação do Modelo de Análise&Projeto (Características para o paradigma Orientado a Objetos)

Artefato	Método	Característica de Qualidade	Critério	Checklist
Modelo de Análise & Projeto	Revisão por Pares	Clareza	Clareza	O significado de cada classe presente nos diagramas de classes é compreensível?
				Os diagramas de classes estão claros e compreensíveis?
				Os diagramas de interação estão claros e compreensíveis?
		Completeza	Completeza	Todas as classes necessárias para atender os requisitos do software foram identificadas?
				As hierarquias existentes entre as classes estão modeladas?
				As associações, atributos e métodos das classes estão identificados?
				As cardinalidades, visibilidades e navegabilidades das associações entre as classes estão definidas?
				Todas as classes cujos objetos possuem um número considerável de estados distintos durante seu ciclo de vida foram modeladas em diagramas de transição de estados?
				Todos os cenários de casos de uso suficientemente complexos foram modelados em diagramas de interação?

Tabela A2.19 – Conjunto para a Verificação do Plano de Testes de Unidade

Artefato	Método	Característica de Qualidade	Critério	Checklist
Plano de testes de unidade	Revisão por Pares	Clareza	Clareza	Os procedimentos de teste estão claramente definidos?
		Completeza	Completeza	Os casos de teste contemplam todos os requisitos aplicáveis à unidade?
				Os principais caminhos de execução estão contemplados no plano de testes de unidades?
				Os resultados esperados estão definidos?
		Consistência	Consistência Externa	O plano de testes de unidades está consistente com os requisitos aplicáveis à unidade?
		Testabilidade	Viabilidade de testes	É possível realizar os testes a partir do plano de testes de unidades?
		Rastreabilidade	Rastreabilidade para os requisitos do software	Existe rastreabilidade entre os casos de teste, o modelo de projeto e os requisitos do software?

Tabela A2.20 – Conjunto para a Verificação do Código de Unidades

Artefato	Método	Característica de Qualidade	Critério	Checklist
Código de unidades	Revisão por Pares	Clareza	Clareza	O código é fácil de ler e entender?
				O código possui comentários adequados e suficientes para o entendimento?
				As mensagens de erro e códigos de retorno são claras e precisas?
		Consistência	Consistência Externa	O código da unidade está consistente com os requisitos aplicáveis à unidade?
				Todas as funcionalidades implementadas para a unidade correspondem a algum requisito aplicável à unidade?
				O código da unidade está consistente com o modelo de projeto?
		Rastreabilidade	Rastreabilidade para os requisitos do software	Há rastreabilidade bidirecional entre o código da unidade, o modelo de projeto e os requisitos aplicáveis à unidade?
		Adequação	Completeza da implementação funcional	Todas as funcionalidades especificadas para a unidade foram implementadas?
		Tolerância a defeitos	Impedimento de defeitos	Todas as prováveis condições de erro foram tratadas?
		Modificabilidade	Modularidade	O código possui forte coesão?
				O código possui baixo acoplamento?
			Inclusão de funcionalidade	O código está estruturado de forma a possibilitar o fácil desenvolvimento de novas funcionalidades?
		Analisabilidade	Facilidade de depuração	O código está estruturado de forma a permitir fácil depuração?

Tabela A2.20 – Conjunto para a Verificação do Código de Unidades (Continuação)

Artefato	Método	Característica de Qualidade	Critério	Métrica
Código de unidades	Teste	Acurácia	Precisão	
		Estabilidade	Taxa de sucesso em mudanças	
		Maturidade	Maturidade dos testes	Maturidade do teste: X=A/B A= Número de casos de teste aprovados B= Número de casos de teste executados
			Densidade de defeitos	Densidade de defeitos: X=A/B A = Número de defeitos encontrados na unidade B= Tamanho da unidade (número de linhas de código)
		Tolerância a defeitos	Impedimento de defeitos	
		Recuperabilidade	Restaurabilidade	Propósito: determinar o quanto o sistema é capaz de se restaurar após um evento ou requisição não esperado Restaurabilidade: X=A/B A=Número de restaurações efetuadas com sucesso B=Número de casos de restauração testados
		Comportamento em relação ao tempo	Tempo de resposta	
		Utilização de recursos	Uso de memória	

Tabela A2.21 – Conjunto para a Verificação do Relatório do Teste de Unidade

Artefato	Método	Característica de Qualidade	Critério	Checklist
Relatório do teste de unidade	Revisão por Pares	Maturidade	Conformidade com os resultados esperados	Os resultados obtidos estão de acordo com o comportamento esperado?
			Cobertura dos testes	Todos os casos de testes planejados foram executados?
		Viabilidade Tecnológica	Viabilidade de Integr.	As unidades estão prontas para a Integr. do software e testes de Integr. do software?

Tabela A2.22 – Conjunto para a Verificação do Plano de Testes de Integração do Software

Artefato	Método	Característica de Qualidade	Critério	Checklist
Plano de testes de Integr. do software	Revisão por Pares	Clareza	Clareza	Os casos de teste descritos podem ser entendidos e executados por um grupo independente?
				Os procedimentos de teste estão claramente definidos?
		Completeza	Completeza	Os casos de teste contemplam todas as interfaces entre as unidades?
				Os resultados esperados estão definidos?
		Consistência	Consistência Externa	O plano de testes de Integr. do software está consistente com a estratégia de Integr. do software?
		Testabilidade	Viabilidade de testes	É possível realizar os testes a partir do plano de testes de Integr. do software?

Tabela A2.23 – Conjunto para a Verificação do Relatório de Teste de Integração do Software

Artefato	Método	Característica de Qualidade	Critério	Checklist
Relatório do teste de Integr. do Software	Revisão por Pares	Maturidade	Conformidade com os resultados esperados	Os resultados obtidos estão de acordo com o comportamento esperado?
			Cobertura dos testes	Todos os casos de testes planejados foram executados?
		Testabilidade	Viabilidade de testes	O código integrado está pronto para o teste do software?

Tabela A2.24 – Conjunto para a Verificação do Plano de Testes do Software

Artefato	Método	Característica de Qualidade	Critério	Checklist
Plano de testes do software	Revisão por Pares	Clareza	Clareza	Os casos de teste descritos podem ser entendidos e executados por um grupo independente?
				Os procedimentos de teste estão claramente definidos?
		Completeza	Completeza	Os resultados esperados estão definidos?
		Consistência	Consistência Externa	O plano de testes do software está consistente com os requisitos do software?
		Rastreabilidade	Rastreabilidade para os requisitos do software	Há rastreabilidade bidirecional entre os casos de testes e os requisitos do software?
		Testabilidade	Viabilidade de testes	Os testes podem ser realizados com o mínimo de apoio da equipe de desenvolvimento?
				É possível realizar os testes a partir do plano de testes do software?
		Adequação	Completeza da implementação funcional	Os casos de teste contemplam todas as funcionalidades especificadas?

Tabela A2.25 – Conjunto para a Verificação do Software

Artefato	Método	Característica de Qualidade	Critério	Métrica
Software	Teste	Adequação	Adequação funcional	Adequação Funcional: X=1-A/B A= Número de funcionalidades nas quais foram detectados problemas durante os testes B= Número de funcionalidades testadas
		Segurança	Controle de acesso	
		Conformidade da funcionalidade	Conformidade funcional	
		Tolerância a defeitos	Impedimento de defeitos	
		Comportamento em relação ao tempo	Tempo de resposta	
		Utilização de recursos	Uso de memória	
		Estabilidade	Taxa de sucesso em mudanças	
		Capacidade para ser instalado	Facilidade de instalação	
		Capacidade para substituir	Consistência funcional de suporte ao usuário	
		Conformidade da portabilidade	Conformidade com os padrões de portabilidade	
		Maturidade	Maturidade dos testes	Maturidade do teste: X=A/B A= Número de casos de teste aprovados B= Número de casos de teste executados
			Densidade de defeitos	Densidade de defeitos: X=A/B A = Número de defeitos encontrados no software B= Tamanho do software (Pontos por função ou Pontos por caso de uso)

Tabela A2.25 – Conjunto para a Verificação do Software (Continuação)

Artefato	Método	Característica de Qualidade	Critério	Métrica
Software	Teste	Acurácia	Precisão	
		Recuperabilidade	Restaurabilidade	Propósito: determinar o quanto o sistema é capaz de se restaurar após um evento ou requisição não esperado Restaurabilidade: X=A/B A=Número de restaurações efetuadas com sucesso B=Número de casos de restauração testados
		Adaptabilidade	Adaptabilidade de software	

Tabela A2.26 – Conjunto para a Verificação do Relatório do Teste do Software

Artefato	Método	Característica de Qualidade	Critério	Checklist
Relatório do teste do software	Revisão por Pares	Maturidade	Conformidade com os resultados esperados	Os resultados obtidos estão de acordo com o comportamento esperado?
			Cobertura dos testes	Todos os casos de testes planejados foram executados?
		Viabilidade Tecnológica	Viabilidade de Integr.	Os componentes de hardware e software estão prontos para a Integr. do sistema e testes de Integr. do sistema?

Tabela A2.27 – Conjunto para a Verificação do Plano de Testes de Integração do Sistema

Artefato	Método	Característica de Qualidade	Critério	Checklist
Plano de testes de Integr. do sistema	Revisão por Pares	Clareza	Clareza	Os casos de teste descritos podem ser entendidos e executados por um grupo independente?
				Os procedimentos de teste estão claramente definidos?
		Completeza	Completeza	Os casos de teste contemplam todas as interfaces entre os componentes de hardware e software?
				Os resultados esperados estão definidos?
		Consistência	Consistência Externa	O plano de testes de Integr. do sistema está consistente com a estratégia de Integr. do sistema?
		Testabilidade	Viabilidade de testes	É possível realizar os testes a partir do plano de testes de Integr. do sistema?

Tabela A2.28 – Conjunto para a Verificação do Relatório do Teste de Integração do Sistema

Artefato	Método	Característica de Qualidade	Critério	Checklist
Relatório do teste de Integr. doSistema	Revisão por Pares	Maturidade	Conformidade com os resultados esperados	Os resultados obtidos estão de acordo com o comportamento esperado?
			Cobertura dos testes	Todos os casos de testes planejados foram executados?
		Testabilidade	Viabilidade de testes	O sistema integrado está pronto para o teste do sistema?

Tabela A2.29 – Conjunto para a Verificação do Plano de Testes do Sistema

Artefato	Método	Característica de Qualidade	Critério	Checklist
Plano de testes do sistema	Revisão por Pares	Clareza	Clareza	Os casos de teste descritos podem ser entendidos e executados por um grupo independente?
				Os procedimentos de teste estão claramente definidos?
		Completeza	Completeza	Os casos de teste contemplam todos os requisitos especificados para o sistema?
				Os resultados esperados estão definidos?
		Consistência	Consistência Externa	O plano de testes do sistema está consistente com os requisitos do sistema?
		Rastreabilidade	Rastreabilidade para os requisitos do sistema	Há rastreabilidade bidirecional entre os casos de testes e os requisitos do sistema?
		Testabilidade	Viabilidade de testes	Os testes podem ser realizados com o mínimo de apoio da equipe de desenvolvimento?
				É possível realizar os testes a partir do plano de testes do sistema?

Tabela A2.30 – Conjunto para a Verificação do Sistema

Artefato	Método	Característica de Qualidade	Critério	Métrica
Sistema	Teste	Interoperabilidade	Consistência de interface	
		Comportamento em relação ao tempo	Tempo de resposta	
		Utilização de recursos	Uso de memória	
		Adaptabilidade	Adaptabilidade de hardware	
		Conformidade da portabilidade	Conformidade com os padrões de portabilidade	
		Maturidade	Maturidade dos testes	Maturidade do teste: X=A/B A= Número de casos de teste aprovados B= Número de casos de teste executados
		Recuperabilidade	Restaurabilidade	Propósito: determinar o quanto o sistema é capaz de se restaurar após um evento ou requisição não esperado Restaurabilidade: X=A/B A=Número de restaurações efetuadas com sucesso B=Número de casos de restauração testados
		Segurança	Controle de acesso	
		Conformidade da funcionalidade	Conformidade funcional	

Tabela A2.31 – Conjunto para a Verificação do Relatório do Teste do Sistema

Artefato	Método	Característica de Qualidade	Critério	Checklist
Relatório do teste do sistema	Revisão por Pares	Maturidade	Conformidade com os resultados esperados	Os resultados obtidos estão de acordo com o comportamento esperado?
			Cobertura dos testes	Todos os casos de testes planejados foram executados?

ANEXO III

Modelo de Classes da Ferramenta *VerificationManager*