

MPS.BR - Melhoria de Processo do Software Brasileiro

Guia de Implementação – Parte 4: Fundamentação para Implementação do Nível D do MR-MPS

Este guia contém orientações para a implementação do nível D do Modelo de Referência MR-MPS.

VIGÊNCIA E TRANSIÇÃO: O Guia Geral:2009 entra em vigor em 30 de junho de 2009. Assim, a partir desta data podem ser realizadas avaliações MPS usando o modelo de referência MR-MPS:2009. Entretanto, fica definido um período de transição, de 30 de junho a 31 de dezembro de 2009, durante o qual podem ser realizadas avaliações MPS usando o modelo de referência MR-MPS:2009 ou a versão anterior MR-MPS 1.2. A partir de 1º de janeiro de 2010 só serão válidas avaliações MPS usando o modelo de referência MR-MPS:2009. As implementações a serem feitas utilizando este Guia de Implementação deverão levar em conta estas vigências.

Maio de 2009

Atualizado em Agosto de 2009

Copyright © 2009 - SOFTEX

Direitos desta edição reservados pela Sociedade SOFTEX

A distribuição ilimitada desse documento está sujeita a *copyright*

ISBN 978-85-99334-16-4

Sumário

1	Prefácio	3
2	Introdução	5
3	Objetivo	6
4	Evoluindo do nível E para o nível D	6
5	Desenvolvimento de Requisitos (DRE)	7
5.1	Propósito.....	7
5.2	Fundamentação teórica	7
5.3	Resultados esperados	10
6	Integração do Produto (ITP)	15
6.1	Propósito.....	15
6.2	Fundamentação teórica	16
6.3	Resultados esperados	18
7	Projeto e Construção do Produto (PCP).....	23
7.1	Propósito.....	23
7.2	Fundamentação teórica	24
7.3	Resultados esperados	26
8	Validação (VAL)	30
8.1	Propósito.....	30
8.2	Fundamentação teórica	30
8.3	Resultados esperados	32
9	Verificação (VER).....	35
9.1	Propósito.....	35
9.2	Fundamentação teórica	36
9.3	Resultados esperados	37
10	Os atributos de processo no nível D.....	42
	Referências Bibliográficas	43
	Lista de colaboradores do Guia de Implementação – Parte 4:2009.....	47
	Lista de colaboradores do Guia de Implementação – Parte 4 versão 1.1 – Julho/2007	48
	Lista de colaboradores do Guia de Implementação – Parte 4 versão 1.0 – Dezembro/2006.....	49

1 Prefácio

O MPS.BR¹ é um programa mobilizador, de longo prazo, criado em dezembro de 2003, coordenado pela Associação para Promoção da Excelência do Software Brasileiro (SOFTEX), que conta com apoio do Ministério da Ciência e Tecnologia (MCT), Financiadora de Estudos e Projetos (FINEP), Serviço Brasileiro de Apoio às Micro e Pequenas Empresas (SEBRAE) e Banco Interamericano de Desenvolvimento (BID).

O objetivo do programa MPS.BR (acrônimo) é a Melhoria de Processo do Software Brasileiro, com duas metas a alcançar a médio e longo prazos:

a) meta técnica, visando à criação e aprimoramento do modelo MPS, com resultados esperados tais como: (i) guias do modelo MPS; (ii) Instituições Implementadoras (II) credenciadas para prestar serviços de consultoria de implementação do modelo de referência MR-MPS; (iii) Instituições Avaliadoras (IA) credenciadas para prestar serviços de avaliação seguindo o método de avaliação MA-MPS; (iv) Consultores de Aquisição (CA) certificados para prestar serviços de consultoria de aquisição de software e serviços relacionados;

b) meta de mercado, visando à disseminação e adoção do modelo MPS, em todas as regiões do país, em um intervalo de tempo justo, a um custo razoável, tanto em PME (foco principal) quanto em grandes organizações públicas e privadas, com resultados esperados tais como: (i) criação e aprimoramento do modelo de negócio MN-MPS; (ii) cursos, provas e workshops; (iii) organizações que implementaram o modelo MPS; (iv) organizações com avaliação MPS publicada (prazo de validade de três anos).

O programa MPS.BR conta com duas estruturas de apoio para o desenvolvimento de suas atividades, o Fórum de Credenciamento e Controle (FCC) e a Equipe Técnica do Modelo (ETM). Por meio destas estruturas, o MPS.BR obtém a participação de representantes de universidades, instituições governamentais, centros de pesquisa e de organizações privadas, os quais contribuem com suas visões complementares que agregam qualidade ao empreendimento.

Cabe ao FCC: (i) emitir parecer que subsidie decisão da SOFTEX sobre o credenciamento de Instituições Implementadoras (II) e Instituições Avaliadoras (IA); (ii) monitorar os resultados das Instituições Implementadoras (II) e Instituições Avaliadoras (IA), emitindo parecer propondo à SOFTEX o seu descredenciamento no caso de comprometimento da credibilidade do modelo MPS.

Cabe à ETM apoiar a SOFTEX sobre os aspectos técnicos relacionados ao Modelo de Referência (MR-MPS) e Método de Avaliação (MA-MPS), para: (i) criação e aprimoramento contínuo do MR-MPS, MA-MPS e seus guias específicos; (ii) capacitação de pessoas por meio de cursos, provas e workshops.

¹ MPS.BR, MR-MPS, MA-MPS e MN-MPS são marcas da SOFTEX. A sigla MPS.BR está associada ao programa MPS.BR – Melhoria do Processo de Software Brasileiro e a sigla MPS está associada ao modelo MPS – Melhoria do Processo de Software.

A criação e o aprimoramento deste Guia de Implementação são também atribuições da ETM, sendo que este guia faz parte do seguinte conjunto de documentos do modelo MPS:

- Guia Geral:2009 [SOFTEX, 2009a];
- Guia de Avaliação:2009 [SOFTEX, 2009b];
- Guia de Aquisição:2009 [SOFTEX, 2009c]; e
- Guia de Implementação (partes 1 a 10)

Este Guia de Implementação fornece orientações para implementar nas organizações os níveis de maturidade descritos no Modelo de Referência MR-MPS, detalhando os processos contemplados nos respectivos níveis de maturidade e os resultados esperados com a implementação dos processos.

O Guia de implementação está subdividido em dez partes, contemplando, respectivamente, os seguintes níveis de maturidade:

- Parte 1: Fundamentação para Implementação do Nível G do MR-MPS:2009;
- Parte 2: Fundamentação para Implementação do Nível F do MR-MPS:2009;
- Parte 3: Fundamentação para Implementação do Nível E do MR-MPS:2009;
- Parte 4: Fundamentação para Implementação do Nível D do MR-MPS:2009;
- Parte 5: Fundamentação para Implementação do Nível C do MR-MPS:2009;
- Parte 6: Fundamentação para Implementação do Nível B do MR-MPS:2009;
- Parte 7: Fundamentação para Implementação do Nível A do MR-MPS:2009; e
- Parte 8: Implementação do MR-MPS:2009 (Níveis G a A) em organizações que adquirem software;
- Parte 9: Implementação do MR-MPS:2009 (Níveis G a A) em organizações do tipo Fábrica de Software;
- Parte 10: Implementação do MR-MPS:2009 (Níveis G a A) em organizações do tipo Fábrica de Teste.

As alterações deste Guia de Implementação em relação à versão 1.0 são decorrentes de:

- mudanças realizadas na versão 1.2 do Guia Geral;
- melhoria da definição de alguns resultados de processo e resultados de atributos de processo, com o intuito de facilitar o entendimento e a aplicabilidade do MR-MPS;
- correção ortográfica e gramatical;
- alterações para compatibilidade com o CMMI-DEV versão 1.2; e
- adequação das referências bibliográficas.

2 Introdução

As mudanças que estão ocorrendo nos ambientes de negócios têm motivado as empresas a modificar estruturas organizacionais e processos produtivos, saindo da visão tradicional baseada em áreas funcionais em direção a redes de processos centrados no cliente. A competitividade depende, cada vez mais, do estabelecimento de conexões nestas redes, criando elos essenciais nas cadeias produtivas. Alcançar competitividade pela qualidade, para as empresas de software, implica tanto na melhoria da qualidade dos produtos de software e serviços correlatos, como dos processos de produção e distribuição de software.

Desta forma, assim como para outros setores, qualidade é fator crítico de sucesso para a indústria de software. Para que se tenha um setor de software competitivo, nacional e internacionalmente, é essencial que os empreendedores do setor coloquem a eficiência e a eficácia dos seus processos em foco nas empresas, visando à oferta de produtos de software e serviços correlatos conforme padrões internacionais de qualidade.

Busca-se que o modelo MPS seja adequado ao perfil de empresas com diferentes tamanhos e características, públicas e privadas, embora com especial atenção às micro, pequenas e médias empresas. Também se espera que o modelo MPS seja compatível com os padrões de qualidade aceitos internacionalmente e que tenha como pressuposto o aproveitamento de toda a competência existente nos padrões e modelos de melhoria de processo já disponíveis. Dessa forma, ele tem como base os requisitos de processos definidos nos modelos de melhoria de processo e atende a necessidade de implantar os princípios de engenharia de software de forma adequada ao contexto das empresas, estando em consonância com as principais abordagens internacionais para definição, avaliação e melhoria de processos de software.

O modelo MPS baseia-se nos conceitos de maturidade e capacidade de processo para a avaliação e melhoria da qualidade e produtividade de produtos de software e serviços correlatos. Dentro desse contexto, o modelo MPS possui três componentes: Modelo de Referência (MR-MPS), Método de Avaliação (MA-MPS) e Modelo de Negócio (MN-MPS).

O modelo MPS está descrito por meio de documentos em formato de guias:

- Guia Geral: contém a descrição geral do modelo MPS e detalha o Modelo de Referência (MR-MPS), seus componentes e as definições comuns necessárias para seu entendimento e aplicação [SOFTEX, 2009a].
- Guia de Aquisição: descreve um processo de aquisição de software. É descrito de forma a apoiar as instituições que queiram adquirir produtos de software apoiando-se no MR-MPS [SOFTEX, 2009c].
- Guia de Avaliação: descreve o processo e o método de avaliação MA-MPS, os requisitos para avaliadores líderes, avaliadores adjuntos e Instituições Avaliadoras (IA) [SOFTEX, 2009b].

- Guia de Implementação: série de dez documentos que fornecem orientações para implementar nas organizações os níveis de maturidade descritos no Modelo de Referência MR-MPS.

3 Objetivo

O Guia de Implementação fornece orientações para implementar nas organizações os níveis de maturidade descritos no Modelo de Referência MR-MPS, detalhando os processos contemplados nos respectivos níveis de maturidade e os resultados esperados com a implementação dos processos. Este documento corresponde à parte 4 do Guia de Implementação e aborda a implementação do nível de maturidade D.

Este documento é destinado, mas não está limitado, a organizações interessadas em utilizar o MR-MPS para melhoria de seus processos de software e a Instituições Implementadoras (II). O conteúdo deste documento é informativo, ou seja, não se espera que uma organização implementando o MR-MPS atenda a todos os itens citados na descrição dos resultados esperados. As observações presentes neste documento procuram apenas explicitar elementos importantes na interpretação dos resultados esperados. Durante uma avaliação MPS, só é requerido o atendimento aos resultados esperados definidos no Guia Geral. Os avaliadores MPS devem analisar se a implementação da organização atende a cada resultado, com abertura a múltiplas formas válidas de implementação.

4 Evoluindo do nível E para o nível D

A implementação do nível E numa organização tem como foco principal a padronização dos processos da organização, por meio da definição de processos padrão, o que inclui, além dos processos do nível E, todos os processos que pertencem aos níveis G e F do MR-MPS.

A evolução do nível E para o nível D não apresenta novidades em termos dos processos e atributos de processo já implantados no nível E, pois estes continuam com a mesma capacidade.

A evolução para o nível D do MR-MPS implica, portanto, apenas na definição e implementação de cinco novos processos com o mesmo nível de capacidade dos processos já implantados: Desenvolvimento de Requisitos (DRE), Integração do Produto (ITP), Projeto e Construção do Produto (PCP), Validação (VAL) e Verificação (VER). Estes processos, junto com Gerência de Requisitos (GRE), são geralmente mencionados como sendo relacionados à engenharia do software propriamente dita. Os processos de engenharia estão intimamente relacionados e, portanto, deve-se procurar não tratá-los de forma isolada em uma abordagem meramente sequencial, mas executá-los de forma interativa e alinhada com o ciclo de vida definido. Os processos são descritos em ordem alfabética nos guias, porém uma possível sequência de leitura mais compatível com a ordem com que são executados dentro de um processo de desenvolvimento seja: Desenvolvimento de Requisitos (DRE), Projeto e Construção do Produto (PCP), Integração do Produto (ITP), Verificação (VER) e Validação (VAL).

Neste nível são permitidas algumas exclusões de resultados esperados de alguns processos conforme descrito nas Partes 8, 9 e 10 do Guia de Implementação.

5 Desenvolvimento de Requisitos (DRE)

5.1 Propósito

O propósito do processo Desenvolvimento de Requisitos é definir os requisitos do cliente, do produto e dos componentes do produto.

Estes três tipos de requisitos atendem as diferentes necessidades de todos os envolvidos no projeto. Inicialmente as necessidades, expectativas, restrições e interfaces do cliente são levantadas e traduzidas em requisitos do cliente. Posteriormente os requisitos do cliente são refinados e descritos em termos técnicos originando os requisitos funcionais e não-funcionais do produto e dos componentes do produto. Uma definição desses requisitos, bem como dos cenários e conceitos operacionais requeridos também devem ser elaborados em um nível de detalhe que permita a realização de projetos (*design*) técnicos e a construção da solução do software para resolver o problema em questão. Os requisitos devem ser analisados, validados e gerenciados ao longo do ciclo de desenvolvimento ou de manutenção de um produto.

O Guia Geral [[SOFTEX, 2009a]] define componente de produto como uma parte do produto final ou algo usado no seu desenvolvimento, por exemplo, um subproduto, um processo ou uma ferramenta, que faz parte da entrega. Os componentes são integrados em sucessivos níveis para compor o produto final. Um produto é definido como artefato associado à execução de um processo que se pretende entregar para um cliente ou usuário final [[SOFTEX, 2009a]].

Os resultados esperados deste processo estão relacionados aos resultados esperados dos processos Projeto e Construção do Produto (PCP), Gerência de Requisitos (GRE), Verificação (VER) e Validação (VAL), ou por serem produtos requeridos para sua execução ou por terem uma interface com o processo propriamente dito.

O conjunto de requisitos produzido pelo Desenvolvimento de Requisitos (DRE), por exemplo, é o produto de trabalho requerido para se iniciar o processo Projeto e Construção do Produto (PCP). De forma semelhante, tanto os requisitos do cliente quanto os requisitos funcionais e não-funcionais do produto e de componentes do produto são produtos de trabalho que estão sob o escopo do processo Gerência de Requisitos (GRE).

Finalmente, existe uma interseção direta do último resultado esperado deste processo (DRE 8 – “Os requisitos são validados”) com o processo Validação (VAL).

5.2 Fundamentação teórica

Requisitos são a base de todo projeto de software. Um requisito é uma característica do sistema ou a descrição de algo que o sistema é capaz de realizar para atingir os seus objetivos [PFLEEGER, 2004]. No SWEBOK [IEEE, 2004], um requisito é descrito como uma propriedade que o software deve exibir para resolver algum

problema no mundo real. De acordo com o IEEE *Software Engineering Standards*, um requisito é descrito de duas formas: (i) uma condição ou capacidade necessária para um usuário resolver um problema ou alcançar um objetivo, ou (ii) uma condição ou uma capacidade que deve ser alcançada ou estar presente num sistema para satisfazer um contrato, padrão, especificação ou outro documento formalmente imposto.

Um desenvolvimento de requisitos criterioso é condição fundamental para o sucesso do projeto, pois os requisitos formam o alicerce para todo o ciclo do projeto, do desenvolvimento até a manutenção.

Diferentes tipos de requisitos precisam ser considerados durante o desenvolvimento. Os requisitos do cliente expressam os resultados desejados para superar os problemas reais. Os requisitos funcionais e não-funcionais do produto² definem as soluções computacionais desenvolvidas utilizando sistemas novos e existentes [ALEXANDER e ROBERTSON, 2004]. Estes tipos de requisitos precisam ser pensados de maneira diferente, tendo suas definições e correlações apresentadas de forma explícita. Segundo SOMMERVILLE [SOMMERVILLE, 2003], alguns dos problemas comuns no desenvolvimento de requisitos são resultantes da falta de uma nítida separação entre esses diferentes níveis de descrição de requisitos.

Desenvolver requisitos inclui as seguintes atividades:

- Elicitação, análise, validação e comunicação das necessidades, expectativas e restrições dos clientes, para obter os requisitos dos clientes, que constituem um entendimento sobre o que satisfará os envolvidos;
- Coleta e coordenação das necessidades dos envolvidos, com priorização e negociação de possíveis conflitos;
- Estabelecimento dos requisitos do cliente;
- Estabelecimento dos requisitos funcionais e não-funcionais do produto e dos componentes do produto consistentes com os requisitos dos clientes.

A Engenharia de Requisitos é definida como o processo de descobrir, analisar, documentar e verificar as funções e restrições do sistema [SOMMERVILLE, 2003] e pode ser dividida em dois grupos de atividades relacionadas: o Desenvolvimento de Requisitos (que inclui as atividades relacionadas à Elicitação, Análise e Modelagem) e a Gerência de Requisitos (incluindo as atividades de Identificação, Rastreabilidade e Gerência de Mudanças). O Desenvolvimento de Requisitos cria e interpreta os requisitos e a Gerência de Requisitos organiza, relaciona os requisitos entre si e com outros produtos de trabalho e mantém os registros destes requisitos.

Alguns dos maiores desafios na criação e manutenção de produtos de software estão diretamente relacionados aos requisitos [COAD e YOURDON, 1992]: (i) compreensão do domínio do problema; (ii) comunicação efetiva com reais usuários do produto; e (iii) evolução contínua dos requisitos. O processo Desenvolvimento de Requisitos deve propor atividades para minimizar os riscos associados aos desafios

² Os requisitos do produto podem se referir tanto aos requisitos de um sistema quanto aos produtos de software que o compõem.

(i); e (ii), enquanto que a combinação dos processos Gerência de Requisitos e Desenvolvimento de Requisitos objetiva minimizar os riscos causados pelo desafio (iii).

De acordo com o SWEBOOK [IEEE, 2004], o Desenvolvimento de Requisitos inclui os seguintes passos:

- Elicitação de requisitos – identificação de forma proativa dos requisitos;
- Análise e negociação de requisitos – exame dos requisitos coletados e negociação com os envolvidos, caso haja requisitos conflitantes;
- Especificação e Modelagem dos requisitos – documentação e criação de modelos dos requisitos com o propósito de obter uma melhor compreensão do problema a ser solucionado; e
- Validação de requisitos – exame da especificação para garantir que inconsistências, omissões e ambiguidades tenham sido detectadas e corrigidas.

A elicitación de requisitos se inicia com a aplicação de técnicas apropriadas para identificar requisitos do cliente, considerando as necessidades, expectativas e restrições impostas pelo cliente [PRESSMAN, 2005]. Existem diversas técnicas para elicitación de requisitos, entre as principais estão: entrevistas, prototipação, técnica FAST (como JAD) e *brainstorming*.

Entrevista é a técnica mais comumente utilizada. Para potencializar seus resultados, ela deve ser planejada e preparada cuidadosamente, identificando-se os candidatos à entrevista, definindo seus objetivos e listando as questões que devem ser obrigatoriamente formuladas.

A prototipação inclui os seguintes passos: estudo preliminar dos requisitos do usuário; construção do protótipo; e seu exame pelos usuários. Protótipos são apenas modelos do produto final e não precisam ser completos. São muito úteis para avaliação de requisitos críticos ou complexos.

Técnicas facilitadas de especificação de aplicações ou técnicas FAST (*Facilitated Application Specification Techniques*) encorajam a criação de uma equipe conjunta de clientes e desenvolvedores que trabalham juntos para [PRESSMAN, 2005]: identificar problemas; propor elementos da solução; negociar diferentes abordagens; e especificar um conjunto preliminar de requisitos. Utilizando uma técnica FAST, uma reunião é conduzida com participação de engenheiros de software e de clientes. São estabelecidas regras para preparação e participação dessa reunião. É sugerida uma agenda, com foco no problema a ser resolvido, e um “facilitador” controla a reunião. Um mecanismo de definição é utilizado (como *flip-charts*, quadro negro, planilhas). A meta é identificar o problema, propor elementos da solução, negociar diferentes abordagens e especificar um conjunto preliminar de requisitos. As abordagens mais populares de FAST são: JAD (técnica desenvolvida pela IBM) e *The Method* (criada pela Performance Resources Inc.).

A técnica *Brainstorming* consiste na condução de reuniões onde as pessoas sugerem e exploram ideias, sendo uma técnica muito utilizada para a geração de novas ideias. Uma sessão *brainstorming* consiste em duas fases: Geração de Ideias, na qual os participantes são encorajados a propor ideias sem críticas pelos demais;

e Consolidação, na qual é feita a avaliação de viabilidade e a priorização das ideias propostas.

Após a identificação, os requisitos devem ser modelados para se obter uma melhor compreensão do produto a ser desenvolvido. O modelo dos requisitos deve focar naquilo que o produto deve fazer, não em como ele o faz. Geralmente, usa-se uma notação gráfica para descrever as informações transformadas pelo produto, o processamento das informações, o comportamento do produto e outras características [PRESSMAN, 2005]. Os principais paradigmas de modelagem de requisitos são: Análise Estruturada e Análise Orientada a Objetos.

Na Análise Estruturada são criados modelos que representam o fluxo e o conteúdo da informação (dados e controle), o produto é dividido em participações funcionais e comportamentais e a essência daquilo que deve ser construído é descrita. Os seguintes modelos são geralmente elaborados:

- Diagramas de fluxo de dados (DFDs);
- Diagrama de Transição de Estado (DTE);
- Dicionário de Dados.

Na Análise Orientada a Objetos o objetivo é modelar os conceitos (objetos) do domínio do produto, seus relacionamentos e comportamentos. Esse modelo é refinado continuamente até se obter um modelo com detalhe suficiente para sua implementação na forma de código executável. Dentre os modelos elaborados estão:

- Modelo de Casos de Uso e Cenários;
- Modelo de Classes;
- Diagramas de Sequência e de Atividade;
- Diagramas de Estados.

5.3 Resultados esperados

5.3.1 DRE1 - As necessidades, expectativas e restrições do cliente, tanto do produto quanto de suas interfaces, são identificadas

O alcance deste resultado esperado envolve a utilização de métodos adequados para identificar necessidades, expectativas, restrições e interfaces do cliente. Deve-se buscar o envolvimento de representantes do cliente e utilizar técnicas de elicitação de requisitos para identificar de forma proativa requisitos adicionais não discutidos explicitamente pelos clientes.

Alguns exemplos de técnicas de elicitação de requisitos são [IEEE, 2004; SEI, 2006; PFLEEGER, 2004; PRESSMAN, 2005; SOMMERVILLE, 2003]: entrevistas; questionários; construção de cenários operacionais e análise de tarefas do usuário final; protótipos e modelos; técnicas facilitadoras de especificação de aplicações (como, por exemplo, JAD); casos de uso; *brainstorming*; observação de produtos e ambientes existentes; análise de casos de negócio; estudo de fontes de informação como documentos, padrões ou especificações; etnografia; QFD (*Quality Function*

Deployment); e engenharia reversa (para sistemas legados). Além desses, pode-se citar também histórias de usuários quando se desenvolve utilizando métodos ágeis. Qualquer que seja a técnica utilizada, o alcance deste resultado deve ser evidenciado por meio de registros que mostrem o levantamento das necessidades, expectativas e restrições do cliente em relação ao produto e suas interfaces.

Em algumas situações a organização, devido ao seu ramo de atividade, pode receber os requisitos do cliente ou requisitos funcionais e não-funcionais do produto e dos componentes do produto já especificados. Mesmo neste caso é necessário que haja revisão do conjunto de requisitos recebido, de forma proativa, buscando identificar incorreções, inconsistências e requisitos ausentes. Como resultado, haverá uma nova lista de requisitos ou a confirmação da anterior, caso não tenham sido feitas alterações.

5.3.2 DRE2 - Um conjunto definido de requisitos do cliente é especificado a partir das necessidades, expectativas e restrições identificadas

As necessidades, expectativas e restrições do cliente que foram identificadas anteriormente são traduzidas em requisitos do cliente. Para que isso ocorra pode ser necessária a resolução de conflitos entre os fornecedores de requisitos e demais envolvidos no projeto relacionados à especificação de requisitos. Além disso, podem surgir questões relevantes a serem verificadas e/ou validadas.

5.3.3 DRE3 - Um conjunto de requisitos funcionais e não-funcionais, do produto e dos componentes do produto que descrevem a solução do problema a ser resolvido, é definido e mantido a partir dos requisitos do cliente

O alcance deste resultado esperado compreende a consolidação das necessidades, expectativas e restrições do cliente em um conjunto de requisitos funcionais e não-funcionais do produto e dos componentes do produto.

Definir os requisitos funcionais envolve analisar os requisitos do cliente para identificar as funções requeridas no produto. Requisitos funcionais descrevem as funções ou os serviços que se espera que o sistema forneça. Um requisito funcional descreve uma interação entre o sistema e seu ambiente [IEEE, 2004; SEI, 2006; SOMMERVILLE, 2003]. São exemplos de requisitos funcionais: gerar relatório com os resultados dos testes clínicos de um paciente; formatar um texto; e cadastrar cliente.

Requisitos não-funcionais são requisitos que expressam condições ou qualidades específicas que o produto e/ou componentes do produto deve atender. Em vez de informar o que o produto fará, os requisitos não-funcionais apontam restrições que devem ser obedecidas. Requisitos não-funcionais são algumas vezes conhecidos como restrições ou requisitos de qualidade [IEEE, 2004; SEI, 2006]. São exemplos de requisitos não-funcionais: tempo de resposta máximo para consultas deve ser três segundos; e o sistema deve estar disponível para o cliente sete dias na semana, vinte e quatro horas por dia. Requisitos não-funcionais podem ser classificados de acordo com seu tipo em diferentes categorias como: requisitos de usabilidade; requisitos de desempenho; requisitos de confiabilidade; entre outros [SOMMERVILLE, 2003].

Ao especificar os requisitos funcionais e não-funcionais é possível perceber falta de informações, inconsistências e erros. Nessas situações, é necessário buscar informações complementares e resolver as inconsistências detectadas.

Durante a execução de um projeto, podem ocorrer mudanças nos requisitos. Essas mudanças devem ser gerenciadas por meio do processo Gerência de Requisitos (GRE) de forma a manter os requisitos funcionais e não-funcionais consistentes com os demais produtos de trabalho e minimizar o impacto das mudanças no projeto.

5.3.4 DRE4 - Os requisitos funcionais e não-funcionais de cada componente do produto são refinados, elaborados e alocados

Alcançar este resultado esperado significa elaborar os requisitos funcionais e não-funcionais de cada componente do produto nos termos técnicos necessários para o desenvolvimento do produto e dos componentes do produto. Para refinar os requisitos funcionais e não-funcionais podem ser utilizadas técnicas de modelagem como especificação de casos de uso de negócio, modelos de contexto [SOMMERVILLE, 2003] e outras como as citadas na seção 5.2.

Os requisitos do cliente podem ser descritos utilizando-se os termos usados pelos clientes e podem conter descrições não-técnicas. Os requisitos funcionais e não-funcionais de cada componente do produto são a expressão dos requisitos do cliente em termos técnicos, de modo a poderem guiar o projeto (*design*) do produto e dos componentes do produto. Requisitos funcionais podem ser descritos de muitas formas como, por exemplo: funções; opções do sistema; ou ainda como serviços ou métodos Orientados a Objetos (OO).

Ao definir os requisitos funcionais e não-funcionais, uma prática comum é categorizar os requisitos em grupos, por meio de um critério. Exemplos de critérios para esse fim são [CACHERO e KOCH, 2002]: propósitos similares, dependência funcional e dados envolvidos.

O alcance deste resultado esperado pode envolver:

- Derivar requisitos funcionais e não-funcionais que resultem de decisões de projeto (*design*), tais como seleção de tecnologia;
- Alocar requisitos funcionais e não-funcionais e restrições para cada componente do produto;
- Estabelecer os relacionamentos entre os requisitos do cliente e os requisitos funcionais e não-funcionais de cada componente do produto, de acordo com o processo Gerência de Requisitos.

Como indicador de alcance deste resultado, deve-se evidenciar que o conjunto de requisitos funcionais e não-funcionais foi refinado, detalhado e documentado ao longo do ciclo de vida para o desenvolvimento do produto e dos componentes do produto. Os registros das atualizações realizadas nesses requisitos também devem ser documentados como evidência do alcance deste resultado.

5.3.5 DRE5 - Interfaces internas e externas do produto e de cada componente do produto são definidas

As interfaces internas e externas do produto e de cada componente do produto devem ser especificadas e documentadas de acordo com a arquitetura definida do produto. As definições dessas interfaces são úteis para projetar e construir as unidades de código dos componentes do produto, bem como para servir de base para verificar a integração entre cada componente do produto e para verificar a integração do produto com outros elementos externos.

As definições das interfaces geralmente são definidas em termos de tipos e formatos de dados de entrada e saída entre os componentes do produto e entre elementos do sistema, especificações de protocolos de comunicação, entre outros.

5.3.6 DRE6 - Conceitos operacionais e cenários são desenvolvidos

O alcance deste resultado esperado exige o desenvolvimento de conceitos operacionais e cenários para o produto e os componentes do produto.

Um conceito operacional para um produto depende do projeto (*design*) da solução e de um cenário, portanto são elaborados quando as decisões de projeto (*design*) são tomadas e os requisitos detalhados. Um cenário é uma sequência de eventos possível de ocorrer no uso de um produto e é utilizado para tornar explícitas algumas necessidades dos envolvidos [SEI, 2006].

Uma forma possível de descrever os cenários é utilizar a modelagem de cenários sugerida pela UML, na qual o cenário é uma sequência específica de ações que ilustra o comportamento de um caso de uso. Ao descrever um caso de uso, geralmente os seguintes elementos são considerados:

- Fluxo principal – descreve uma sequência de ações que serão executadas considerando que nada de errado acontecerá durante a execução das ações;
- Fluxos Alternativos – descrevem o que acontece quando o ator (papel que interage com o sistema) faz uma escolha alternativa, diferente da descrita no fluxo principal, para alcançar seu objetivo. Fluxos alternativos podem descrever escolhas exclusivas entre si;
- Fluxos de Exceção – correspondem à descrição das situações de exceção, quando algo inesperado ocorre na interação com o sistema;
- Pré-condição – define que hipóteses são assumidas como verdadeiras para que o cenário tenha início. Deve ser usada em casos de uso cuja realização não faz sentido em qualquer momento, mas somente quando o sistema está em um determinado estado com certas propriedades;
- Pós-condição – estado que o sistema alcança após o caso de uso ter sido realizado.

O alcance deste resultado esperado pode abranger:

- Definir o ambiente no qual o produto operará, incluindo limites e restrições;

- Elaborar um conceito operacional detalhado para cada produto ou componente do produto que defina a interação do produto, do usuário final, do ambiente e que satisfaça as necessidades de operação, manutenção e apoio;
- Revisar conceitos operacionais e cenários para refinar e descobrir novos requisitos.

5.3.7 DRE7 - Os requisitos são analisados, usando critérios definidos, para balancear as necessidades dos interessados com as restrições existentes

Este resultado visa garantir que os requisitos, em seus diferentes níveis, sejam analisados de forma a balancear as necessidades dos interessados com as restrições de projeto existentes.

Os requisitos podem ser analisados juntamente com cenários, conceitos operacionais e definições detalhadas dos requisitos, para determinar se eles são necessários, corretos, testáveis e suficientes para atingir os objetivos e requisitos de alto nível (requisitos do cliente) [SEI, 2006]. Técnicas de Verificação podem ser utilizadas para garantir que:

- Todos os requisitos tenham sido declarados de forma não ambígua;
- As inconsistências, omissões e erros tenham sido detectados e corrigidos;
- Os requisitos de diferentes níveis estejam consistentes entre si.

O alcance deste resultado esperado pode compreender:

- Analisar as necessidades, expectativas e restrições dos envolvidos, com o objetivo de organizá-las e remover possíveis conflitos;
- Analisar cenários e conceitos operacionais para refinar as necessidades, restrições e interfaces do cliente e descobrir novos requisitos;
- Analisar requisitos para assegurar que eles estão completos, são factíveis e verificáveis, de acordo com os critérios estabelecidos no processo Verificação (VER).

Para a análise de balanceamento entre necessidades e restrições podem ser utilizados modelos, simulações, protótipos e avaliações de riscos nos requisitos e na arquitetura funcional. Em [KELLNER *et al.*, 1999] são apresentados os principais conceitos relacionados com simulação de processos de software.

5.3.8 DRE8 - Os requisitos são validados

Este resultado esperado visa garantir que os requisitos sejam validados utilizando-se técnicas adequadas, de forma a garantir que o produto terá o desempenho adequado quando instalado no seu ambiente alvo. A validação aumenta a confiança de que os requisitos definidos são capazes de guiar o desenvolvimento satisfatoriamente. Quanto mais cedo problemas forem identificados, menos retrabalho e custo serão necessários para adequar os requisitos às expectativas do cliente.

As técnicas de validação são discutidas na seção que apresenta o processo Validação (VAL). Para atender a este resultado esperado, a validação deve estar de acordo com critérios estabelecidos pelo processo Validação (VAL).

6 Integração do Produto (ITP)

6.1 Propósito

O propósito do processo Integração do Produto é compor os componentes do produto, produzindo um produto integrado consistente com seu projeto, e demonstrar que os requisitos funcionais e não-funcionais são satisfeitos para o ambiente alvo ou equivalente.

O processo Integração do Produto diz respeito a como integrar um produto e qual a sequência de integração a ser usada. Trata, também, da criação de um ambiente operacional no qual se possa implantar o produto satisfatoriamente; da documentação dos procedimentos e critérios de integração do produto; de como assegurar a integração correta das partes; e da entrega do produto.

Os resultados esperados deste processo estão relacionados a resultados esperados dos processos Projeto e Construção do Produto (PCP), Verificação (VER), Validação (VAL) Gerência de Decisões (GDE), Gerência de Configuração (GCO) e Gerência de Requisitos (GRE).

A interseção deste processo com o processo Projeto e Construção do Produto (PCP) está presente no resultado esperado referente ao estabelecimento do ambiente de integração, no que diz respeito à compra, reutilização ou desenvolvimento do ambiente. Também está presente no resultado esperado referente ao gerenciamento das interfaces internas dos produtos e componentes do produto, no que diz respeito ao projeto de interfaces entre componentes do produto.

A interseção deste processo com o processo Verificação (VER) está presente nos resultados esperados referentes à verificação das interfaces, do ambiente de integração, dos componentes do produto e do produto integrado, no que diz respeito à realização de testes de unidades, integração e regressão, além de revisões por pares. De maneira similar, caso haja a necessidade de validar interfaces, ambiente de integração, componentes de produto ou produto integrado, pode ser identificada interação com o processo Validação (VAL).

A interseção deste processo com o processo Gerência de Decisões (GDE) está presente nos resultados esperados relacionados ao desenvolvimento e escolha da estratégia de integração, caso se deseje selecionar a estratégia de integração de acordo com um processo formal de decisão. Também pode haver interseção no resultado esperado relacionado ao estabelecimento do ambiente de integração, já que a decisão sobre adquirir, reutilizar ou desenvolver o ambiente também pode seguir um processo formal de seleção.

A interseção deste processo com o processo Gerência de Configuração (GCO) está presente no resultado esperado referente ao gerenciamento das interfaces internas dos produtos e componentes do produto, no que diz respeito ao controle das mudanças. Também existe relacionamento no resultado esperado referente à

entrega do produto e sua documentação ao cliente, no que diz respeito à liberação do produto.

A interseção deste processo com o processo Gerência de Requisitos (GRE) está presente no resultado esperado referente ao gerenciamento das interfaces internas dos produtos e componentes do produto, no que diz respeito ao gerenciamento das mudanças. Também existe relacionamento no resultado esperado referente a testes de regressão, uma vez que se pode fazer uso de uma matriz de rastreabilidade.

6.2 Fundamentação teórica

Em projetos pequenos, a integração pode envolver apenas algumas classes ou arquivos que precisam funcionar juntos. Em projetos grandes, pode envolver milhares de programas e componentes que formam um sistema maior. Independentemente do tamanho, alguns princípios básicos devem ser aplicados [McCONNELL, 2004].

A integração do produto pode abranger tanto a integração do software como a integração do sistema. Na integração do software, integram-se as unidades do software, produzindo itens de software integrados. Na integração do sistema, integram-se os elementos do sistema (incluindo itens de software, itens de hardware, operações manuais e outros sistemas, conforme necessário) para produzir um sistema completo [ISO/IEC, 2008].

A integração dos componentes do produto deve ser planejada, incluindo requisitos de teste, procedimentos, dados, responsabilidades e cronograma. Esse planejamento deve ser adequadamente documentado [ISO/IEC, 2008].

Um aspecto crítico da integração de produtos é o gerenciamento de interfaces internas e externas do produto ou componentes do produto, para garantir compatibilidade entre as interfaces [SEI, 2006]. Uma interface pode ser vista, de maneira geral, como sendo uma fronteira de comunicação entre componentes, tais como partes de um software, itens de hardware ou até mesmo um usuário. Normalmente se refere a uma abstração que um componente fornece de si mesmo para o exterior. Segundo a ISO/IEC 12119 [ISO/IEC, 1994] uma interface é uma fronteira compartilhada entre duas unidades funcionais, definida por características funcionais, características físicas comuns de interconexão e outras características, conforme apropriado. Deve-se atentar para a gerência de interfaces ao longo do projeto.

A ordem na qual se constroem os componentes do produto influencia na ordem na qual se pode integrá-los, já que não se pode integrar o que ainda não foi construído. Tanto a sequência de construção como a de integração são tópicos importantes a serem considerados, pois construir e integrar software em uma ordem errada pode tornar a codificação, os testes e a depuração mais difíceis [McCONNELL, 2004].

Os testes de integração também têm papel importante para garantir que as diferentes partes do produto possam interagir adequadamente em conjunto, de forma a atender corretamente aos requisitos funcionais e não-funcionais pretendidos [TIAN, 2005]. O teste de integração é o processo de verificar se os componentes, juntos, executam conforme está descrito nas especificações e no projeto de programas. A partir deste momento, outros tipos de teste são realizados: teste

funcional; teste de desempenho; teste de aceitação; e teste de instalação [PFLEEGER, 2004].

A integração do produto é mais que uma única montagem dos componentes do produto ao final do projeto e da construção. A integração do produto pode ser incremental, usando um processo iterativo de composição de componentes do produto, avaliação e composição de mais componentes do produto [SEI, 2006]. Uma integração incremental oferece algumas vantagens em relação a uma abordagem em que o produto é totalmente integrado de uma só vez, entre elas [McCONNELL, 2004]:

- É mais fácil localizar os erros, uma vez que a parte do produto que está sendo integrada é menor;
- Os membros do projeto conseguem ver o resultado de seu trabalho mais cedo no projeto, o que aumenta sua motivação;
- Melhor monitoramento do progresso, uma vez que o gerente pode ver claramente que porção do produto está ou não está pronta;
- Melhor relacionamento com o cliente, que também consegue ver progressos mais concretos no projeto;
- Os componentes do produto são testados de forma mais abrangente, uma vez que os mesmos componentes poderão ser testados diversas vezes ao longo dos testes de integração das partes; e
- É possível reduzir o tempo de desenvolvimento, pois é possível o paralelismo nas atividades do projeto.

É importante que seja definida uma estratégia de regressão, uma vez que o produto certamente sofrerá alterações durante o seu desenvolvimento ou após ser entregue ao cliente, decorrentes de manutenções (corretivas ou evolutivas) ou inclusões de novos elementos (requisitos ou módulos). Essa estratégia deve possibilitar que o produto seja testado novamente após uma mudança ter sido realizada, de forma a garantir que modificações ou correções no produto não afetem e danifiquem outras partes. Teste de regressão implica em executar novamente um conjunto de testes já conduzidos anteriormente para garantir que as mudanças realizadas não produziram efeitos colaterais indesejados [PRESSMAN, 2005].

O produto, depois de integrado, testado e empacotado, é entregue ao cliente e instalado no ambiente pretendido para operação.

De forma geral, pode-se resumir alguns dos benefícios esperados pelo uso de uma abordagem cuidadosa de integração de produtos [McCONNELL, 2004]: (i) detecção de defeitos mais fácil; (ii) menos defeitos; (iii) menor tempo para se chegar a produtos ou partes de produtos funcionais; (iv) menor tempo total de desenvolvimento; (v) melhor relacionamento com o cliente; (vi) moral elevado da equipe; (vii) maior chance de se completar o projeto; (viii) estimativas de tempo mais confiáveis; (ix) relatórios de status mais precisos; (x) maior qualidade do código; e (xi) menos documentação.

6.3 Resultados esperados

6.3.1 ITP1 - Uma estratégia de integração, consistente com o projeto (*design*) e com os requisitos do produto, é desenvolvida para os componentes do produto

Deve ser definida uma estratégia, incluindo procedimentos e critérios, para conduzir a integração dos componentes do produto, determinando quais componentes serão integrados e qual será a sequência de integração. É importante que a estratégia de integração escolhida seja consistente com o projeto (*design*), arquitetura e com os requisitos do produto.

A estratégia de integração a ser adotada é composta pela determinação da sequência de integração. A sequência de integração fornece um apoio à integração incremental e avaliação de componentes do produto. Geralmente contém informações sobre os produtos a serem integrados em cada incremento de integração, além das verificações a serem realizadas usando as definições das interfaces entre os componentes do produto.

Diversas estratégias de integração diferentes podem ser encontradas na literatura, tais como [PFLEEGER, 2004; McCONNELL, 2004]:

- Integração *Bottom-Up*: Cada componente no nível inferior da hierarquia do sistema é desenvolvido e testado individualmente. Os próximos componentes a serem integrados e testados são aqueles que “chamam” os que foram previamente integrados. Essa abordagem é seguida repetidamente até que todos os componentes sejam considerados;
- Integração *Top-Down*: Reverso da abordagem *Bottom-Up*. O nível mais superior (normalmente um componente de controle principal) é desenvolvido e testado. Em seguida, todos os componentes chamados pelo(s) componente(s) testado(s) são desenvolvidos, combinados e testados como uma grande unidade. Essa abordagem é seguida repetidamente até que todos os componentes sejam considerados;
- Integração Sanduíche: Combinação entre as abordagens *Top-Down* e *Bottom-Up*. Primeiro, são integrados e testados os componentes de mais alto nível (*Top-Down*). Depois, são integrados e testados os componentes de nível mais inferior (*Bottom-Up*). Então, a integração e os testes convergem para os componentes do sistema de nível intermediário. Essas três camadas justificam o nome da abordagem;
- Integração Orientada a Riscos: Identifica-se o nível de riscos associado a cada componente. Com isso, analisam-se quais serão as partes mais difíceis de implementar; estas são desenvolvidas e integradas primeiro. As partes mais simples são desenvolvidas, integradas e testadas mais tarde;
- Integração Orientada a Funcionalidade: Consiste em desenvolver, integrar e testar uma funcionalidade por vez. Assim, as funcionalidades vão sendo integradas de maneira incremental, compondo a funcionalidade total do sistema;

- Integração em Forma de T: É construída e integrada uma parte completa do sistema, do nível mais alto ao mais baixo de hierarquia, permitindo a verificação de questões arquiteturais (parte vertical do “T”). Depois disso, as demais funcionalidades e partes do sistema são construídas e integradas. Normalmente é usada em conjunto com as abordagens orientadas a riscos e a funcionalidades.

Uma boa prática consiste em definir sequências de integração alternativas para um dado produto, bem como critérios para a seleção de alternativas, e, então, selecionar, dentre as alternativas, baseando-se nos critérios definidos, aquela que é a mais adequada. Por exemplo, em um projeto se poderia selecionar uma dentre as seis estratégias de integração descritas acima com base em critérios como: experiência da equipe no uso da estratégia; tempo necessário para entrega de produtos intermediários; complexidade do produto; entre outros. É importante, também, documentar as razões pelas quais uma estratégia de integração foi selecionada e não outra.

A sequência de integração deve ser revista, sempre que necessário, uma vez que esta pode ser afetada por diversos fatores, como, por exemplo, atrasos na construção, mudanças no cronograma de entregas, mudanças nas prioridades de construção.

6.3.2 ITP2 - Um ambiente para integração dos componentes do produto é estabelecido e mantido

O ambiente de integração necessário em cada etapa do processo de integração do produto pode incluir ferramentas de testes, simuladores (funcionando como componentes de produtos ainda indisponíveis), partes de equipamentos ou componentes do produto reais, dispositivos de armazenamento, entre outros. O objetivo deste resultado esperado é garantir que o ambiente para integração dos componentes do produto foi definido e mantido conforme necessário.

Os requisitos para o ambiente de integração do sistema pode envolver requisitos de equipamentos, software ou outros recursos e geralmente são identificados a partir do desenvolvimento dos requisitos e da arquitetura do produto.

Critérios e procedimentos de verificação para o ambiente de integração do produto podem ser definidos para garantir o apoio adequado do ambiente na integração dos itens de produto.

Uma decisão importante em relação ao ambiente de integração é determinar se este será construído internamente, reutilizado ou adquirido de um fornecedor externo. Para isso, pode ser necessário realizar uma análise entre desenvolver, comprar ou reutilizar (ver o resultado esperado PCP5 do processo Projeto e Construção do Produto). Vale salientar que o ambiente de integração pode ser um ambiente do cliente.

6.3.3 ITP3 - A compatibilidade das interfaces internas e externas dos componentes do produto é assegurada

Muitos problemas relacionados à integração de produtos são originados de aspectos desconhecidos ou não controlados, tanto das interfaces internas como externas.

Para assegurar que as interfaces internas e externas dos componentes do produto são compatíveis e que sua descrição é completa, as interfaces devem ser revisadas.

Além das interfaces entre componentes do produto, podem ser consideradas, também, todas as interfaces com o ambiente de integração do produto e com outros ambientes, como o de validação, o de verificação, o de operação e o de suporte, conforme necessário.

Checklists com as situações mais comuns de problemas de consistência e completeza nas interfaces podem ser utilizados para tornar a revisão mais objetiva e aumentar sua efetividade.

As descrições das interfaces podem ser periódicas e continuamente revisadas para garantir que não haja diferença entre as descrições existentes e os produtos que estão sendo desenvolvidos.

6.3.4 ITP4 - As definições, o projeto (*design*) e as mudanças nas interfaces internas e externas são gerenciados para o produto e para os componentes do produto

As definições, projetos e mudanças nas interfaces internas e externas devem ser gerenciadas, ou seja, a consistência das interfaces deve ser mantida ao longo de todo o ciclo de vida do produto. Além disso, resolução de conflitos, não conformidades e questões relativas a mudanças devem ser tratadas.

Uma interface declara o conjunto de serviços que são fornecidos ou exigidos pelo componente. Podem ser consideradas interfaces internas a um produto de software, todas as interfaces que representam comunicação apenas entre componentes do produto em questão, tais como interfaces entre funções, objetos e módulos de um produto. Já as interfaces externas a um produto são aquelas que representam a comunicação de componentes internos ao produto com itens de outros sistemas externos ao produto, como interfaces com usuários, com itens de hardware, com o ambiente em que está inserido, entre outros.

Sempre que houver mudanças nas interfaces, estas devem ser documentadas, mantidas e disponibilizadas para todos os interessados, conforme pertinente. Esse gerenciamento de interfaces começa muito cedo no desenvolvimento de um produto. As definições e projetos para interfaces podem afetar não apenas os componentes e sistemas externos, mas também os ambientes de verificação e de validação [SEI, 2006].

6.3.5 ITP5 - Cada componente do produto é verificado, utilizando-se critérios definidos, para confirmar que estes estão prontos para a integração

Deve ser realizada a verificação dos componentes do produto para garantir que estes estão prontos para a integração, a ser realizada de acordo com a sequência de integração e procedimentos definidos. Essa verificação deve ser baseada em critérios definidos. Exemplos de critérios incluem: defeitos óbvios; conformidade com as descrições; consistência entre os componentes de produto e interfaces; entre outros.

Este resultado tem uma forte relação com o processo Verificação (VER), visto que exige a realização de verificações nos componentes de produto, baseando-se em critérios estabelecidos. Para isso, normalmente são utilizados testes de unidades e revisões por pares. Maiores informações sobre essas técnicas podem ser obtidas no processo Verificação (VER), resultados esperados VER3, VER4 e VER5.

Um exemplo poderia ser a realização de testes de unidades nos componentes do produto a serem integrados, de forma a garantir sua adequação. Pode-se ainda realizar revisões por pares avaliando o relatório de testes de unidades, de forma a garantir que a integração dos componentes é viável. A avaliação do relatório de testes de unidades permite responder questões como:

- Os resultados obtidos estão de acordo com o comportamento esperado?
- Todos os casos de testes planejados foram executados?
- As unidades estão prontas para a Integração do software e testes de Integração do software?

6.3.6 ITP6 - Os componentes do produto são integrados, de acordo com a sequência determinada e seguindo os procedimentos e critérios para integração

O objetivo deste resultado esperado é garantir que os componentes do produto sejam integrados de acordo com a sequência de integração (ver o resultado esperado ITP1) e seguindo os procedimentos e critérios previamente definidos.

6.3.7 ITP7 - Os componentes do produto integrados são avaliados e os resultados da integração são registrados

O objetivo deste resultado esperado é garantir que seja realizada a avaliação dos componentes do produto integrados e que os resultados encontrados na avaliação sejam documentados. Uma forma de executar a avaliação é pela realização de testes de integração e da análise de seus resultados.

O teste de integração representa um conjunto de atividades com intenção de descobrir defeitos na estrutura do software definida durante a fase de projeto. Neste contexto, os defeitos são comumente associados com as interfaces dos módulos (componentes) que compõem a arquitetura do software [BRIAND *et al.*, 200].

Quando as unidades são combinadas, os caminhos possíveis a serem percorridos pelo programa crescem bastante e podem ocorrer falhas impossíveis de serem identificadas quando as unidades são testadas separadamente [VIEIRA e TRAVASSOS, 1998]. Por meio do teste de integração será testado se as unidades conseguem trabalhar juntas de forma correta e se comunicam sem problemas.

O teste de integração pode ter sua execução iniciada assim que alguns componentes ficarem prontos. No contexto do ciclo de desenvolvimento, componente pronto significa componente implementado e com os testes de unidade aprovados, ou seja, componentes individuais funcionando corretamente e atingindo os seus objetivos [PFLEEGER, 2004; LIMA, 2005].

Assim, uma forma de alcance deste resultado é por meio da elaboração de um plano de testes de integração, que detalhe como os testes serão realizados, e um relatório de testes de integração, que contenha informações sobre os resultados obtidos após realização dos testes. A avaliação do relatório de testes de integração permite responder questões como:

- Os resultados obtidos estão de acordo com o comportamento esperado?
- Todos os casos de testes planejados foram executados?
- O código integrado está pronto para o teste do software?

6.3.8 ITP8 - Uma estratégia de teste de regressão é desenvolvida e aplicada para uma nova verificação do produto, caso ocorra uma mudança nos componentes do produto (incluindo requisitos, projeto (*design*) e códigos associados)

Este resultado esperado tem como objetivo garantir a existência de uma estratégia para testes de regressão a ser aplicada no caso de ocorrerem mudanças nos componentes do produto incluindo requisitos, projeto (*design*) e código associados.

Teste de regressão é realizado depois de uma melhoria funcional, reparo ou qualquer mudança no produto. Seu propósito é determinar se a mudança introduziu erros em outras partes do produto. É normalmente feito por meio de novas execuções de alguns dos casos de testes do produto. O teste de regressão é importante, pois mudanças e correções de erros tendem a ser mais suscetíveis a erros que a primeira codificação. Um planejamento para o teste de regressão – quem, como, quando – também é necessário [MYERS, 2004]. Vale salientar que por mudança no produto não se entende apenas manutenções corretivas ou evolutivas em um produto pronto. Por exemplo, em um desenvolvimento iterativo, a inclusão de novos módulos pode implicar em repetir testes realizados em módulos anteriores, o que pode ser considerado teste de regressão.

As técnicas para realizar testes de regressão são normalmente mais especializadas [ROTHERMEL e HARROLD, 1996; ROSENBLUM e WEYUKER, 1997], incluindo [TIAN, 2005]:

- Uma análise das diferenças entre a versão anterior do produto e a versão atual baseada em algum modelo formal ou informal para selecionar os casos de teste existentes e determinar se novos casos de testes precisam ser desenvolvidos; e
- Os novos casos de testes têm o foco em duas áreas: (i) a nova parte desenvolvida ou atualizada, que é similar ao teste de novos produtos, mas em escala menor; e (ii) as interações envolvendo tanto as partes novas e antigas, o que é similar a teste de integração, mas com um foco em tipos específicos de interfaces e de interações.

A matriz de rastreabilidade pode ser utilizada para determinar quais outros componentes estão relacionados àquele sendo analisado. Assim, pode auxiliar na identificação de quais casos de testes precisam ser executados novamente. Logo, manter a rastreabilidade desde os requisitos até os casos de testes é importante para auxiliar na garantia de que tudo que precise ser testado novamente, realmente o seja.

6.3.9 ITP9 - O produto e a documentação relacionada são preparados e entregues ao cliente

O produto e a documentação a serem entregues ao cliente devem ser organizados, em uma mídia adequada e entregues ao cliente. O alcance deste resultado esperado inclui, portanto, o empacotamento e distribuição de software com métodos que podem incluir fita magnética, disquetes, documentos impressos, CDs, DVDs, Internet [SEI, 2006]. O alcance adequado deste resultado esperado inclui a entrega do produto e sua documentação ao cliente, bem como a confirmação do recebimento pelo cliente.

Dependendo das restrições do projeto podem ser considerados requisitos para empacotamento e distribuição do produto, tais como tipo de armazenamento e mídia de distribuição, documentação requerida, *copyrights* e licenças [SEI, 2006].

O ambiente operacional pode precisar ser preparado para a instalação do produto. Essa preparação pode ser de responsabilidade do cliente.

7 Projeto e Construção do Produto (PCP)

7.1 Propósito

O propósito do processo Projeto e Construção do Produto é projetar, desenvolver e implementar soluções para atender aos requisitos.

Uma vez que os requisitos foram desenvolvidos, têm suas mudanças controladas e estão sob o nível apropriado de gerência de configuração, o objetivo do processo de Projeto e Construção do Produto (PCP) é projetar uma solução, dentre as inúmeras possíveis soluções existentes, para satisfazer aos requisitos, desenvolver e, então, implementar a solução projetada.

O raciocínio que explica o porquê da escolha desta solução deve ser mantido. A implementação da solução, no caso de software, corresponde à codificação das unidades de software. A documentação que descreve o projeto (*design*) e a implementação da solução deve ser desenvolvida. A rastreabilidade do projeto (*design*) deve ser mantida em relação aos requisitos, para que possa ser verificado se os requisitos realmente foram satisfeitos.

Estes requisitos, desenvolvidos durante o processo Desenvolvimento de Requisitos (DRE), estão sob o apropriado nível de gerência de configuração e possuem suas alterações controladas de acordo com o processo Gerência de Requisitos (GRE). A matriz de rastreabilidade deve prover a ligação entre os requisitos e os componentes projetados durante a execução do processo Projeto e Construção do Produto (PCP) de modo que seja possível determinar que componentes estejam satisfazendo determinados requisitos, apoiando a análise de impacto no caso de ser necessário realizar alguma mudança.

Esse processo interage com os processos Desenvolvimento de Requisitos (DRE), Integração do Produto (ITP), Verificação (VER) e Validação (VAL). O processo Projeto e Construção do Produto (PCP) recebe como entrada os requisitos desenvolvidos para projetar e construir a solução. O processo Integração do Produto

(ITP) recebe como insumo os requisitos desenvolvidos durante a execução do processo Desenvolvimento de Requisitos (DRE) e os componentes do produto projetados e construídos durante a execução do processo Projeto e Construção do Produto (PCP), a fim de combiná-los e verificar se as interfaces satisfazem os requisitos de interface desenvolvidos durante a execução do processo Desenvolvimento de Requisitos (DRE). Além disso, os componentes projetados e construídos são verificados durante o processo Verificação (VER) em relação aos requisitos e o produto final é validado de forma incremental durante o processo Validação (VAL).

7.2 Fundamentação teórica

A execução do processo Projeto e Construção do Produto (PCP) é iniciada quando os requisitos para o problema a ser resolvido pelo software estiverem definidos, desenvolvidos e aprovados. Este processo pode ser executado tanto no contexto do software a ser desenvolvido quanto no contexto do sistema onde o software é integrado. O objetivo do processo Projeto e Construção do Produto (PCP) é definir atividades que permitam a elaboração do projeto (*design*) do software e, também, possibilitem a implementação da solução de projeto (*design*) para os requisitos em questão.

Projeto (*design*) é o processo criativo de transformar o problema em uma solução, sendo que a descrição da solução também é chamada de projeto (*design*) [PFLEEGER, 2004]. Em [PRESSMAN, 2005] define-se projeto (*design*) como a representação significativa de alguma coisa a ser construída. O projeto (*design*) enfatiza uma solução conceitual que satisfaça os requisitos e não a sua implementação. Uma descrição de um esquema de banco de dados e objetos de software ou da arquitetura do sistema é um bom exemplo [LARMAM, 2004]. O processo Projeto e Construção do Produto (PCP) engloba tanto a elaboração do projeto (*design*) quanto a implementação do projeto (*design*) definido. Os clientes sabem o que o sistema deve fazer e os construtores precisam saber como o sistema funcionará. Por isso, o projeto (*design*) é um processo iterativo constituído de duas partes: o projeto (*design*) conceitual ou projeto (*design*) do sistema, mostra ao cliente exatamente o que o sistema fará; e o projeto (*design*) técnico, que é uma tradução detalhada do projeto (*design*) conceitual, permite que os construtores do sistema saibam quais são o hardware e software necessários para solucionar o problema do cliente [PFLEEGER, 2004].

De acordo com [SHAW e GARLAN, 1996] a definição da arquitetura do software é o primeiro passo a ser tomado durante o projeto (*design*) de software, sendo possível identificar três níveis de projeto (*design*):

- Projeto da Arquitetura: Este nível de projeto (*design*) associa as capacidades do sistema identificadas na especificação de requisitos com os componentes do sistema que as implementarão, bem como mostra a interconexão entre estes componentes. Os componentes da arquitetura são, geralmente, os módulos do sistema;

- Projeto do Código: Este nível de projeto (*design*) envolve algoritmos e estruturas de dados. Os componentes são primitivas de linguagens de programação como, por exemplo, números, caracteres, ponteiros e estruturas de controle;
- Projeto Executável: Este nível de projeto (*design*) detalha ainda mais o código, discutindo detalhes, por exemplo, de alocação de memória, de formato dos dados e de padrões de bits.

Durante o projeto (*design*) da arquitetura também é definido o estilo arquitetural a ser utilizado na construção do sistema. O estilo arquitetural envolve os componentes do sistema, os conectores e as restrições sobre a combinação dos componentes. Como exemplos de estilos arquiteturais pode-se citar: *pipe and filter*, objetos; chamada implícita; formação de camadas; repositórios; interpretadores; e controle de processos [SHAW e GARLAN, 1996; PFLEEGER, 2004].

O processo Projeto e Construção do Produto (PCP) é intenso em conhecimento e durante a sua execução é necessário tomar diversas decisões, pois podem existir diversas maneiras de solucionar um mesmo problema. É importante armazenar o conhecimento organizacional adquirido durante a execução do processo mantendo o registro do raciocínio pelo qual uma determinada decisão foi tomada durante a etapa de projeto (*design*). Isso fica claro quando se tem um problema de rotatividade de pessoal na organização. A alta rotatividade da equipe na indústria de software, associada com longos períodos de vida dos produtos, aumenta a probabilidade de que os projetistas originais não estejam presentes quando evoluções nos produtos forem necessárias e os problemas começarem a ocorrer [BURGE, 2005].

Além disso, muito frequentemente, o entendimento necessário para realizar manutenções depende da compreensão de quais decisões de projeto (*design*) foram consideradas, que suposições foram feitas, que soluções alternativas foram rejeitadas e que critérios e requisitos foram satisfeitos no processo de deliberação [CONKLIN, 1989]. Este tipo de conhecimento raramente é registrado e está acessível para consulta durante o período de manutenção do produto.

Outro problema comum nas organizações é a dificuldade de aproveitar o conhecimento de membros mais experientes durante o treinamento de novos membros das equipes, pois a dinâmica de trabalho não permite que os experientes parem a execução de suas atividades para compartilhar o seu conhecimento. Assim, em situações de tomada de decisão, os membros iniciantes tendem a repetir os mesmos erros cometidos por outros membros das equipes que já passaram por situações semelhantes. Estes são alguns dos motivos que ressaltam a importância em se manter o raciocínio por trás das decisões tomadas durante a execução do processo Projeto e Construção do Produto.

Em relação à avaliação do projeto (*design*), é necessário estabelecer guias que orientem a avaliação. Em [PRESSMAN, 2005; PFLEEGER, 2004] estão relacionadas as características desejadas para um bom projeto (*design*):

- Possuir alternativas de solução e selecionar a alternativa mais adequada com base nos requisitos, recursos disponíveis e nos conceitos de projeto (*design*);
- Ser rastreável em relação ao modelo de análise;
- Não “reinventar a roda”, ou seja, observar padrões que possam ser reutilizados;

- Possuir a mesma estrutura que o domínio do problema a ser resolvido pelo projeto (*design*);
- Exibir uniformidade e integração;
- Permitir mudanças;
- Acomodar eventos não esperados de maneira adequada;
- Estar em um nível de abstração maior do que o código;
- Ser avaliado em relação à qualidade enquanto é criado e não apenas depois de ter sido totalmente definido; e
- Ser revisado para identificar erros de semântica.

7.3 Resultados esperados

7.3.1 PCP1 - Alternativas de solução e critérios de seleção são desenvolvidos para atender aos requisitos definidos de produto e componentes de produto

Para um determinado problema podem existir diversas maneiras diferentes de solucioná-lo. Por exemplo, caso seja necessário utilizar um algoritmo de ordenação, pode-se ter inúmeras diferentes soluções: *quicksort*, *bubblesort*, *mergesort*, ordenação por seleção direta etc. Durante o projeto (*design*), deve-se analisar estas soluções alternativas e selecionar a solução mais adequada como base em critérios pré-estabelecidos. Como exemplos de critérios podem-se citar as características e subcaracterísticas de qualidade presentes na norma ISO/IEC 9126-1 [ISO/IEC, 2001]. Outras escolhas típicas tomadas são a escolha da arquitetura a ser utilizada, desenvolvimento personalizado versus de prateleira, modularização dos componentes, escolhas referentes às interfaces entre os componentes, escolha da linguagem de programação, escolha do banco de dados e escolha da ferramenta de modelagem a ser utilizada pela organização.

Um processo de seleção de alternativas possui, basicamente, as seguintes atividades: definição dos objetivos de seleção; estabelecimento dos critérios de seleção; desenvolvimento das soluções a serem avaliadas; avaliações das soluções com base nos critérios pré-estabelecidos e seleção da solução mais adequada. É importante destacar que o desenvolvimento das soluções alternativas deve satisfazer os requisitos desenvolvidos.

7.3.2 PCP2 - Soluções são selecionadas para o produto ou componentes do produto, com base em cenários definidos e em critérios identificados

Tendo identificado os critérios de seleção a serem utilizados, é necessário avaliar as soluções alternativas com base nestes critérios e selecionar a solução mais adequada. A partir da seleção de uma alternativa, novas decisões podem precisar ser tomadas e por isso a abordagem de seleção de alternativas necessita ser novamente executada.

A evolução dos cenários criados durante o desenvolvimento dos requisitos pode ajudar na seleção da alternativa de solução mais adequada. Em [BASS *et al.*, 2003] é possível encontrar a descrição de *templates* que podem ser utilizados para a

definição de cenários. Estes cenários são definidos com base nos requisitos desenvolvidos. Estes cenários são desenvolvidos durante o processo Desenvolvimento de Requisitos (DRE) e evoluídos durante o processo Projeto e Construção do Produto (PCP).

Supondo que uma organização queira determinar, por exemplo, se, em um determinado projeto, as chaves primárias das tabelas serão geradas automaticamente pelo banco de dados selecionado ou se serão geradas pelo software a ser desenvolvido. Tal organização poderia selecionar dois critérios: desempenho e portabilidade. Por um lado, se o banco de dados gerar as chaves primárias automaticamente, o desempenho será maior, mas por outro lado, a portabilidade será menor pois pode ser que outro banco de dados não consiga entender aquele formato. O oposto ocorre caso o software gere as chaves primárias, a portabilidade aumenta, pois a chave pode ser gerada em um formato que qualquer banco de dados consiga armazenar, mas o desempenho diminui. A criação de cenários apoia a avaliação dessas alternativas em relação aos critérios. Um determinado cenário poderia avaliar o desempenho de cada opção ao criar, apagar e recuperar n registros no banco de dados, enquanto que outro determinado cenário poderia avaliar a utilização de cada solução com bancos de dados diferentes para avaliar a portabilidade.

7.3.3 PCP3 - O produto e/ou componente do produto é projetado e documentado

Deve-se projetar o produto ou os componentes do produto de acordo com os requisitos especificados. Este projeto (*design*) geralmente é constituído de duas atividades: o projeto (*design*) da arquitetura do sistema e o projeto (*design*) do software. O projeto (*design*) da arquitetura do sistema visa identificar que requisitos do sistema devem ser alocados a que elementos do sistema [ISO/IEC, 2008]. A arquitetura deve identificar itens de hardware, software e operações manuais [ISO/IEC, 2008].

O projeto (*design*) do software especifica, para cada componente definido na arquitetura, um projeto (*design*) que atenda os requisitos definidos. Este projeto (*design*) é refinado em níveis cada vez menores até chegar ao nível de unidades de software que possam ser codificadas e testadas. O produto das atividades de projeto (*design*) é a documentação necessária para a implementação do produto ou dos componentes do produto, assim como para a execução de outras atividades do ciclo de vida do produto como a manutenção ou instalação.

7.3.4 PCP4 - As interfaces entre os componentes do produto são projetadas com base em critérios predefinidos

Um recurso fundamental dos componentes é a capacidade de definir interfaces. O componente precisa expor alguns dos meios para os outros componentes se comunicarem com ele [PENDER, 2004]. Uma interface declara o conjunto de serviços que são fornecidos ou exigidos pelo componente.

O projeto (*design*) deve conter a descrição das interfaces, assim como os critérios utilizados para a seleção destas interfaces. Estas descrições das interfaces englobam as interfaces entre: componentes e componentes; componentes de mais

baixo nível e componentes de mais alto nível; componentes e produtos do processo de ciclo de vida; e componentes e itens externos. A escolha da interface deve levar em consideração os requisitos definidos durante o processo Desenvolvimento de Requisitos (DRE).

O processo Integração do Produto (ITP) utilizará as interfaces projetadas de acordo com os requisitos desenvolvidos para integrar os componentes de forma consistente.

7.3.5 PCP5 - Uma análise dos componentes do produto é conduzida para decidir sobre sua construção, compra ou reutilização

Devido a fatores como a globalização, a forte competitividade e o crescimento da complexidade dos produtos de software, as organizações buscam melhores formas de se organizarem. As organizações tentam focar em suas atividades principais, a fim de melhorar a gerência dos custos. Isto levanta questões como [BOUCHRIHA, *et. al.*, 2002]: Que recursos deveriam ser desenvolvidos para aumentar as competências da organização? Que atividades deveriam ser feitas por outras organizações e que parceiros em potencial deveriam ser contratados para executar estas atividades? Que atividades internas deveriam ser preservadas e desenvolvidas pela própria organização? Como os recursos da organização deveriam ser alocados em relação às atividades? É vital para as organizações que os custos de desenvolvimento dos produtos sejam reduzidos, assim como seu prazo de lançamento no mercado (*time to market*).

Um meio de conseguir estes dois objetivos é delegar a outras organizações a confecção de componentes do produto, desde que estes componentes não façam parte da competência central da organização. É necessário, portanto, a existência de uma abordagem que permita a uma organização decidir o que lhe é mais vantajoso: desenvolver um determinado componente internamente; contratar uma outra organização para fazer este desenvolvimento; ou reutilizar um componente já disponível na organização. As organizações devem ser capazes de escolher as partes do produto que serão produzidas internamente e as partes dos produtos que serão produzidas por organizações contratadas. As abordagens existentes na literatura para responder a estas questões possuem duas correntes principais de pensamento: a primeira busca responder a pergunta sob um ponto de vista econômico e a segunda foca no ponto de vista estratégico [CÁNEZ *et al.*, 2001].

7.3.6 PCP6 - Os componentes do produto são implementados e verificados de acordo com o que foi projetado

Os componentes do produto devem ser implementados de acordo com o que foi especificado no projeto (*design*) e a documentação relacionada a estes componentes deve ser desenvolvida.

A implementação do projeto (*design*) é feita por meio da utilização de um método como, por exemplo: programação orientada a objetos; programação estruturada; programação orientada a aspectos; programação imperativa; programação orientada a eventos; reutilização de código; e geração automática de código.

Também é necessário revisar cada componente do produto. Esta revisão pode ser feita, por exemplo, por meio de revisão por pares e/ou testes de unidades. Assim, uma vez que as unidades sejam implementadas é necessário verificá-las em relação aos requisitos e ao projeto (*design*). Esta verificação é feita de acordo com critérios definidos - ver processo Verificação (VER).

7.3.7 PCP7 - A documentação é identificada, desenvolvida e disponibilizada de acordo com os padrões estabelecidos

Deve-se desenvolver a documentação associada ao projeto (*design*) e para o usuário final, de acordo com padrões. Também é necessário identificar que documentos serão produzidos, quando serão produzidos, por quem serão produzidos e quem são os interessados nestes documentos.

A satisfação deste resultado implica na produção de um pacote de dados técnico que descreva o produto, de uma forma que permita às diferentes pessoas que trabalharão realizarem suas tarefas de desenvolvimento ou de manutenção. Este pacote de dados técnico contém a documentação produzida durante o projeto que tem início com a avaliação das alternativas de solução. O pacote de dados técnico pode conter, por exemplo, o projeto (*design*) da arquitetura do sistema, o raciocínio por trás das decisões tomadas, o projeto (*design*) dos componentes, o projeto (*design*) das interfaces e a rastreabilidade entre os requisitos e os componentes do projeto e interfaces.

7.3.8 PCP8 - A documentação é mantida de acordo com os critérios definidos

A documentação necessária para a manutenção, operação e instalação do produto deve ser mantida e revisada, de acordo com critérios previamente definidos, para garantir a consistência em relação aos requisitos e ao projeto (*design*).

Padrões podem facilitar o entendimento e a leitura da documentação. Alguns exemplos de possíveis padrões são:

- Fontes a serem utilizadas, dependendo do contexto (rodapé, título, conteúdo etc.);
- Uso de abreviações;
- Tamanho das fontes;
- Compatibilidade com processadores de texto;
- Imagens geralmente utilizadas pela organização na documentação.

Os critérios incluem, por exemplo, como a documentação será mantida no sentido de controlar as suas modificações durante o projeto, mantendo-a consistente com os demais artefatos gerados durante o projeto, ou então se a documentação será desenvolvida aos poucos, durante as atividades do projeto, ou se será desenvolvida toda de uma vez só, no final do projeto. Estes critérios também podem indicar como a documentação estará organizada e que itens/requisitos do sistema deverá descrever.

8 Validação (VAL)

8.1 Propósito

O propósito do processo Validação é confirmar que um produto ou componente do produto atenderá a seu uso pretendido quando colocado no ambiente para o qual foi desenvolvido.

O processo Validação diz respeito, portanto, a como avaliar a qualidade de um produto ou componente de produto, garantindo que atenda às necessidades de seus usuários, quando colocado em seu ambiente de uso. O objetivo da validação é garantir que o produto correto está sendo desenvolvido.

Os resultados esperados deste processo estão relacionados a resultados esperados dos processos Desenvolvimento de Requisitos (DRE) e Integração do Produto (ITP).

A interseção deste processo com o processo Desenvolvimento de Requisitos (DRE) está presente no resultado esperado referente à validação dos requisitos desenvolvidos, desde as fases iniciais do ciclo de vida, para assegurar que o desempenho do produto, quando instalado no seu ambiente de uso, será adequado.

A interseção com o processo Integração do Produto (ITP) está presente na avaliação dos componentes do produto integrados onde pode ser feita uma validação desses componentes e também no momento da entrega do produto ao cliente, garantindo que este se comporta adequadamente em seu ambiente alvo.

8.2 Fundamentação teórica

A qualidade de software destaca-se como um diferencial de mercado visto que sua importância está no fato de produzir sistemas cada vez melhores e, assim, assegurar a satisfação do cliente. Nesse contexto, a validação é considerada um elemento importante para a garantia da qualidade e, portanto, deve ser planejada e executada, com eficácia, durante o desenvolvimento do software.

A validação de software visa avaliar a qualidade de produtos ou componentes de produto. Como apresentado na seção 5.1, um componente de produto é uma parte do produto final ou algo usado no seu desenvolvimento que faz parte da entrega; e, produto é qualquer artefato associado à execução de um processo que se pretende entregar ao cliente ou usuário final [[SOFTEX, 2009a]].

Validação é a confirmação, por exame e fornecimento de evidência objetiva, de que os requisitos específicos, para um determinado uso pretendido, são atendidos [ISO/IEC, 2008]. 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].

Frequentemente, a validação de software está fortemente associada à verificação de software sendo que, normalmente, elas são executadas em conjunto, pois muitas vezes é difícil determinar onde uma 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 assegurar que se

está desenvolvendo o produto correto, isto é, o produto que o cliente deseja [BOEHM, 1981].

A norma internacional ISO/IEC 12207 [ISO/IEC, 2008] define 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 usando um protótipo. A validação dos requisitos visa garantir que o produto a ser desenvolvido se comportará adequadamente no seu ambiente de uso. Esta norma também apresenta um processo de validação que pode ser usado como referência auxiliar na implementação deste processo e na interpretação de seus resultados esperados.

Atividades de validação normalmente são executadas nas etapas iniciais e finais do desenvolvimento de um produto, começando geralmente com a validação dos requisitos e, posteriormente, terminando com a validação do produto final no ambiente operacional [TIAN, 2005]. Uma vez que a validação se preocupa com o atendimento das necessidades dos clientes e usuários do produto, é recomendável que, sempre que possível, os usuários finais sejam envolvidos nas atividades de validação.

Uma forma de realizar a validação é por meio de uma série de testes que demonstrem que o produto correto está sendo desenvolvido. O teste é um dos mais importantes métodos de garantia da qualidade do produto e, frequentemente, o mais usado [TIAN, 2005]. Consiste em sua análise dinâmica, ou seja, na sua execução com o objetivo de verificar a presença de defeitos e aumentar a confiança de que o produto esteja correto [ROCHA *et al.*, 2001].

Quando o software é desenvolvido para um cliente específico, uma série de testes de aceitação pode ser conduzida para permitir ao cliente validar o produto. Este tipo de teste pode ser conduzido pelo cliente ou pelos desenvolvedores, mas o principal participante é o cliente. Os testes de aceitação podem ser conduzidos por um período de tempo de dias a meses, descobrindo erros que poderiam degradar o sistema ao longo do tempo. Por outro lado, quando o software é desenvolvido como um produto para ser usado por diversos clientes, é impraticável realizar testes de aceitação formais com cada um deles. A maioria dos fabricantes de software utiliza um processo chamado de testes alfa e beta para detectar erros que apenas os usuários finais parecem ser capazes de encontrar. Nas duas abordagens, os testes são realizados pelo cliente, sendo que, no primeiro, o teste é realizado em um ambiente controlado, com o acompanhamento do desenvolvedor, enquanto no segundo, o teste é realizado em ambiente real de uso pelo cliente, que reporta os erros ao desenvolvedor [PRESSMAN, 2005].

Vale salientar que testes no contexto de verificação e de validação são complementares. Em hipótese alguma, estes deverão ser encarados como

atividades redundantes. Tanto um quanto o outro possui natureza e objetivos distintos, fortalecendo o processo de detecção de erros e aumentando a qualidade final do produto [BARTIÉ, 2002].

No entanto, a validação de software não é somente teste. Requisitos devem ser especificados e o uso pretendido para estes requisitos deve ser avaliado por meio da validação. Portanto, além dos testes, diversas outras técnicas podem ser utilizadas na validação, dentre elas: prototipação, simulações, *model checking*, utilização de cenários de caso de uso [PFLEEGER, 2004; PRESSMAN, 2005; SUKUMARAN *et al.*, 2006; SEYBOLD *et al.*, 2005; SOMÉ, 2005].

Dentre estas técnicas, a prototipação merece destaque. Um protótipo é um produto parcialmente desenvolvido, que possibilita aos clientes e desenvolvedores examinarem certos aspectos do sistema proposto e decidir se eles são ou não apropriados ou adequados para o produto acabado [PFLEEGER, 2004]. A prototipação é uma técnica de validação de requisitos bastante útil, pois normalmente é uma forma de validar a interpretação que os engenheiros de software têm dos requisitos e também de elicitar novos requisitos. A vantagem do protótipo é que ele pode tornar mais fácil interpretar as suposições feitas e, quando necessário, dar um retorno muito útil sobre as interpretações erradas [IEEE, 2004].

É importante ressaltar que a realização da validação em um projeto de software envolve um bom planejamento que determine quais produtos serão validados, quais métodos e técnicas serão utilizados, além de incluir também a definição dos ambientes necessários para validação, ferramentas e demais recursos que serão utilizados na validação.

Uma das possíveis formas de se executar o processo de validação ao longo do desenvolvimento do produto é começar com o planejamento da validação integrado ao planejamento do projeto no início do desenvolvimento, seguindo com a execução da validação ao longo do projeto, conforme planejado, iniciando pela validação dos requisitos e terminando com os testes do produto em seu ambiente de uso.

É de se esperar que a validação bem sucedida tenha como benefícios, dentre outros: (i) maior satisfação do cliente, devido ao melhor atendimento às suas necessidades; (ii) redução de retrabalho, devido, principalmente, à validação dos requisitos; (iii) redução de custos, decorrente da redução de retrabalho; e (iv) maior competitividade, originada na maior satisfação do cliente.

8.3 Resultados esperados

8.3.1 VAL1 - Produtos de trabalho a serem validados são identificados

Este resultado esperado visa garantir que sejam identificados os produtos ou componentes de produto que serão validados ao longo do projeto.

Esta identificação pode ocorrer nos estágios iniciais do projeto, com base nos artefatos que serão produzidos pelo processo. Uma boa prática é definir critérios para seleção dos produtos ou componentes de produto que serão validados e selecioná-los segundo estes critérios. Pode-se, por exemplo, selecionar os produtos mais relevantes com base nas necessidades do cliente ou levando-se em consideração os riscos associados aos produtos, pois, uma vez que produtos com

grande risco associado precisam ser mais confiáveis, podem precisar ser mais fortemente validados.

É possível, também, definir, em nível organizacional, uma lista de produtos ou componentes de produto que normalmente são validados, de forma que os projetos só precisem adaptar essa lista às suas necessidades. Diretrizes também podem ser utilizadas para guiar a seleção.

8.3.2 VAL2 - Uma estratégia de validação é desenvolvida e implementada, estabelecendo cronograma, participantes envolvidos, métodos para validação e qualquer material a ser utilizado na validação

Este resultado esperado tem como objetivo garantir que as atividades de validação sejam planejadas com a definição de procedimentos, infra-estrutura necessária, cronograma, recursos e responsabilidades. Os métodos a serem usados nas atividades de validação também devem ser identificados.

Alguns métodos de validação requerem um planejamento específico que deve ser realizado, por exemplo, o planejamento dos casos de testes. Deve ser definido ainda um cronograma para as atividades de validação e os recursos necessários à execução das atividades devem ser planejados. Este cronograma e a alocação dos recursos, tanto humanos quanto outros recursos em geral, devem estar integrados ao Plano do Projeto.

Prototipação é um dos métodos para a validação de requisitos. Para isto pode ser construído um protótipo descartável ou um protótipo evolutivo. Protótipos têm como objetivo aprender mais sobre o problema ou explorar a viabilidade de possíveis soluções. Protótipos descartáveis não são utilizados posteriormente. Protótipos evolutivos podem ser utilizados como a base de uma parte ou de todo o software a ser fornecido [PFLEEGER, 2004].

A validação é realizada, principalmente, por meio de testes. Após se ter o produto desenvolvido, é possível realizar uma série de testes de desempenho para avaliar o comportamento do produto em seu ambiente de uso, tais como testes de estresse, testes de volume, testes de tempo, testes de usabilidade, dentre outros [RAKITIN, 2001]. Após testar o sistema e garantir que ele se comporta conforme especificado, tanto para os requisitos funcionais quanto para os não-funcionais, o cliente pode avaliar o sistema para determinar se o sistema construído é o que ele deseja. Neste momento, o teste de aceitação pode ser realizado. Após a aceitação do sistema por parte do cliente, ele pode ser avaliado quanto à sua instalação no ambiente de uso. Essa avaliação é feita com o teste de instalação que tem por objetivo permitir que os usuários executem as funções do sistema e documentem problemas adicionais específicos do ambiente de operação [PFLEEGER, 2004].

8.3.3 VAL3 - Critérios e procedimentos para validação dos produtos de trabalho a serem validados são identificados e um ambiente para validação é estabelecido

O objetivo deste resultado esperado é garantir que os critérios e procedimentos a serem utilizados para a validação foram identificados e que foi estabelecido um ambiente para validação, semelhante ao ambiente operacional.

Devem ser definidos os critérios para a validação de cada produto ou componente do produto. Para ajudar a determinar se um critério foi ou não atendido, algumas métricas podem ser definidas. Alguns exemplos de critérios para validação dos requisitos são: adequação funcional e usabilidade. Para validação de um software alguns critérios de validação poderiam ser: tempo de resposta, tolerância a falhas, recuperabilidade, uso de memória, confiabilidade e portabilidade. Algumas sugestões de critérios e métricas que podem ser usados na validação de produtos de software podem ser encontradas nos relatórios técnicos [ISO/IEC, 2003a; ISO/IEC, 2003b; ISO/IEC, 2004].

A preparação do ambiente de validação pode envolver identificar e disponibilizar as ferramentas, tais como ferramentas de apoio ao planejamento e execução dos testes, os recursos de hardware, infra-estrutura de rede, entre outros recursos necessários.

8.3.4 VAL4 - Atividades de validação são executadas para garantir que o produto esteja pronto para uso no ambiente operacional pretendido

Este resultado esperado visa garantir que as atividades de validação foram realizadas nos produtos e componentes de produto conforme o planejado.

Uma das principais formas de se realizar a validação é executando testes. A realização dos testes ao longo de todo o processo de desenvolvimento do software é possível por meio da execução de quatro etapas distintas [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;
- Avaliação dos resultados: Consiste em analisar os resultados dos testes confrontando-os com os resultados esperados.

8.3.5 VAL5 - Problemas são identificados e registrados

Este resultado esperado visa garantir que os problemas identificados durante a execução das atividades de validação foram documentados e que foram definidos quais problemas serão tratados. Estes problemas devem ser acompanhados até sua conclusão.

Nem todos os problemas encontrados precisam ser corrigidos. A organização pode definir critérios que facilitem essa análise considerando os riscos para o projeto e o impacto na qualidade do produto.

8.3.6 VAL6 - Resultados de atividades de validação são analisados e disponibilizados para as partes interessadas

O alcance deste resultado esperado envolve realizar uma análise dos resultados obtidos em decorrência da execução das atividades relacionadas a validação e

disponibilizar estes resultados para o cliente, ou seu representante na execução das atividades, e outras partes interessadas.

Assim, uma forma de alcance deste resultado é por meio da análise de laudos de avaliação e relatórios de testes que contenham informações sobre os resultados obtidos após a realização das atividades de validação. A avaliação destes resultados permite responder questões como:

- Os critérios definidos foram satisfeitos?
- As ações corretivas planejadas foram concluídas?
- A validação foi executada conforme planejado?
- Os resultados obtidos permitem a aprovação do artefato validado?
- O produto final está pronto para o uso pretendido?

8.3.7 VAL7 - Evidências de que os produtos de software desenvolvidos estão prontos para o uso pretendido são fornecidas

Quando as atividades de teste são realizadas e há evidências que o produto satisfaz os requisitos e as expectativas do cliente, o produto pode ser considerado validado. Para isso o produto deve ser testado em seu ambiente real de uso ou em uma reprodução deste ambiente.

É necessário registrar os resultados da validação, evidenciando que o produto está pronto para o uso. Uma das formas de garantir que o produto está pronto para ser usado é realizar uma reunião com os clientes e/ou usuários finais onde sejam apresentados os resultados da validação, a correção dos problemas detectados e se obtenha o aceite de que o produto está pronto para o uso.

9 Verificação (VER)

9.1 Propósito

O propósito do processo Verificação é confirmar que cada serviço e/ou produto de trabalho do processo ou do projeto atende apropriadamente os requisitos especificados.

O processo Verificação trata de como avaliar produtos de trabalho e serviços, garantindo que atendam a seus requisitos, por meio da identificação dos itens a serem verificados, do planejamento da verificação de cada um destes itens e da execução da verificação conforme planejado ao longo do desenvolvimento do produto.

Os resultados esperados deste processo estão relacionados a resultados esperados dos processos Desenvolvimento de Requisitos (DRE), Projeto e Construção do Produto (PCP) e Integração do Produto (ITP).

A interseção deste processo com o processo Desenvolvimento de Requisitos (DRE) está presente no resultado esperado referente à análise dos requisitos desenvolvidos para garantir que estes são necessários e suficientes, onde pode ser realizada uma verificação desses requisitos.

A interseção com o processo Projeto e Construção do Produto (PCP) está presente no resultado esperado referente à implementação e verificação, de acordo com o que foi especificado no projeto (*design*) dos componentes do produto e a sua documentação associada. A verificação desses componentes do produto pode ser realizada por meio de revisão por pares e/ou testes.

A interseção com o processo Integração do Produto (ITP) está presente ao longo de toda a integração, uma vez que há uma forte dependência dos testes de integração, que são técnicas de verificação. Vale destacar a interseção nos resultados esperados referentes à: (i) verificação dos componentes do produto para garantir que estes estão prontos para a integração, baseando-se em critérios predefinidos; (ii) avaliação dos componentes do produto integrados onde pode ser feita uma verificação desses componentes; (iii) desenvolvimento e aplicação de uma estratégia de regressão para uma nova verificação do produto quando ocorre uma mudança nos componentes do produto; e (iv) preparação e entrega do produto ao cliente realizando a verificação final no produto antes da entrega.

9.2 Fundamentação teórica

O principal objetivo da engenharia de software é, sem dúvida, melhorar a qualidade do software. Uma vez que a qualidade de um produto de software está diretamente relacionada à sua quantidade de defeitos, os defeitos de um produto de software devem ser detectados o mais cedo possível evitando o retrabalho [PUTNAM e MYERS, 2003]. De fato, há algumas evidências de que 40% a 50% do esforço de um projeto é gasto com retrabalho que poderia ser evitado [BOEHM e BASILI, 2001]. 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. Além de possibilitar a detecção de defeitos mais cedo, uma verificação efetiva pode aumentar a produtividade em projetos de software [BARRETO, 2006].

O termo “verificação” está definido de diversas formas na literatura. A verificação pode ser vista como a garantia de que produtos de uma fase particular do processo estão consistentes com os requisitos desta fase e da fase anterior [SCHULMEYER e MACKENZIE, 1999]. A norma internacional ISO/IEC 12207 [ISO/IEC, 2008] 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. Esta norma também apresenta um processo de verificação que pode ser usado como referência auxiliar na implementação deste processo e na interpretação de seus resultados esperados.

Atividades de verificação devem ser executadas ao longo do desenvolvimento de um produto, começando geralmente com a verificação dos requisitos, seguindo com a verificação dos produtos intermediários (projeto (*design*), código, dentre outros) e

concluindo com a verificação do produto final. Estas avaliações da qualidade, distribuídas ao longo de todo o ciclo de vida, têm como objetivos: (i) assegurar que os requisitos estabelecidos podem ser alcançados; (ii) identificar os requisitos que não podem ser alcançados; (iii) garantir que o software é desenvolvido de forma uniforme; (iv) identificar erros para tomar medidas corretivas o mais cedo possível; e (v) tornar o projeto mais gerenciável [PRESSMAN, 2005].

A avaliação da qualidade de um produto de software deve ser baseada nos requisitos de qualidade especificados para o produto. Com o objetivo de auxiliar nesta avaliação, alguns critérios que atendam aos requisitos especificados devem ser identificados e, se satisfeitos, indicam o atendimento aos requisitos especificados. Para auxiliar na determinação da satisfação dos critérios, podem-se definir questões (*checklist*) a serem respondidas, ajudando na avaliação [BARRETO, 2006].

A verificação de produtos de software deve ser realizada por meio da aplicação de métodos e técnicas específicos ao longo do desenvolvimento do produto. Dois métodos de verificação são destacados na literatura [THELIN, 2002; SEI, 2006]: revisão por pares e testes.

Revisão por pares é um método estático de verificação no qual um artefato é examinado por qualquer integrante da equipe do projeto, exceto o autor do artefato, com o propósito de detectar defeitos [LAITENBERGER *et al.*, 2002]. Em uma revisão por pares são usados critérios objetivos para a avaliação. 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.

Testes têm como objetivo examinar o comportamento do software por meio 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 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].

Verificação de software não é simples, uma vez que várias avaliações devem ser realizadas ao longo de um projeto e cada avaliação requer planejamento, controle e uso de técnicas de verificação adequadas. O planejamento da verificação pode ser iniciado no planejamento do projeto e refinado ao longo do projeto. Este planejamento deve estar integrado ao plano do projeto.

9.3 Resultados esperados

9.3.1 VER1 - Produtos de trabalho a serem verificados são identificados

Para atender a este resultado esperado deve-se analisar os produtos de trabalho que serão produzidos ao longo do projeto e selecionar aqueles a serem verificados.

Uma boa estratégia para seleção de produtos de trabalho leva em consideração as contribuições para o alcance dos objetivos e requisitos do projeto, considerando também os riscos do projeto. Desta forma, os principais produtos de trabalho em um projeto são geralmente objeto de atividades de verificação. Alguns possíveis produtos de trabalho selecionados para a verificação, por sua importância, podem ser o plano do projeto, o documento de requisitos, o documento de análise, o documento de projeto e o código-fonte.

9.3.2 VER2 - Uma estratégia de verificação é desenvolvida e implementada, estabelecendo cronograma, revisores envolvidos, métodos para verificação e qualquer material a ser utilizado na verificação

O alcance deste resultado esperado envolve definir uma estratégia de verificação descrevendo os procedimentos, a infra-estrutura necessária e as responsabilidades pelas atividades de verificação. Os métodos que serão usados para verificação de cada produto de trabalho selecionado para verificação devem ser identificados, garantindo, em cada projeto, a realização de algum tipo de revisão por pares e testes. As ferramentas que apoiarão a execução das atividades de verificação também devem ser definidas.

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, 2006; LAITENBERGER *et al.*, 2002; SCHULMEYER e MACKENZIE, 1999].

O método *Walkthrough* tem por objetivo avaliar um ou mais artefatos para identificar defeitos e considerar possíveis soluções alternativas. Consiste de uma preparação e uma reunião de avaliação de, no máximo, duas horas e da qual participam, no máximo, sete pessoas. Os participantes assumem papéis específicos, tais como moderador, autor, secretário e membros [SCHULMEYER e MACKENZIE, 1999]. 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. 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, se encontrarem problemas muito graves, os participantes podem decidir pela rejeição dos artefatos. Posteriormente, o autor corrige os problemas encontrados e, caso a decisão tenha sido a rejeição dos artefatos, outra avaliação deve ser realizada [SCHULMEYER e MACKENZIE, 1999].

Uma inspeção é realizada de acordo com um processo bem definido tendo como principais características 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 avaliam os artefatos com base nos critérios definidos nos *checklists* antes da reunião [SCHULMEYER e MACKENZIE, 1999]. Assim como no

walkthrough, os inspetores, com base nos problemas encontrados, decidem pela aceitação ou não dos artefatos avaliados.

Além da inspeção e do *walkthrough*, a revisão por pares pode ser implementada com uma revisão simples, onde somente uma pessoa revisa o artefato, desde que: o revisor não seja o próprio autor do documento; o revisor seja um “par” do autor, isto é, o revisor exerça uma função semelhante à do autor ou, no mínimo, tenha conhecimento sobre o documento para revisar o seu conteúdo; e que sejam usados critérios objetivos para a revisão.

Testes têm como objetivo verificar dinamicamente o comportamento de um programa, usando um conjunto de casos de teste adequadamente selecionados, em relação ao seu comportamento esperado [IEEE, 2004]. A execução dos testes envolve várias fases e abrange verificação e validação. Primeiro, cada unidade do programa é testada, isolada das demais unidades. Esse teste, conhecido como teste de unidade, verifica se a unidade funciona de forma adequada aos tipos de entrada esperados, a partir do estudo do projeto (*design*) 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. O teste do sistema envolve: teste funcional; teste de desempenho; teste de aceitação; e teste de instalação. O teste funcional verifica se o sistema integrado realiza as funções especificadas nos requisitos. Após garantir que as funções do sistema são executadas conforme especificado, pode-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. Neste ponto, o produto deve operar conforme os desenvolvedores pretendem e pode-se dizer que o produto está verificado [PFLEEGER, 2004; WALLACE *et al.*, 1996].

Para guiar os desenvolvedores durante a realização dos testes, diversas técnicas de teste estão definidas na literatura [TIAN, 2005; IEEE, 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;
- 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 tem o foco 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 coloca o foco na implementação interna, considerando o objeto a ser testado como uma caixa branca que nos permite ver seu interior.

Além de identificar os métodos de verificação que serão usados, é necessário definir os participantes envolvidos na verificação e seus respectivos papéis. Também devem ser definidos os interessados nos resultados de cada verificação a ser realizada bem como os procedimentos para disponibilizar tais resultados. Deve ser definido, ainda, um cronograma para as atividades de verificação e os recursos necessários à execução das atividades devem ser planejados. Este cronograma e a alocação dos recursos, tanto humanos quanto outros recursos em geral, devem estar integrados ao plano do projeto.

O planejamento específico requerido para cada método de verificação também deve ser documentado como, por exemplo, o planejamento das revisões por pares identificando os papéis, a documentação para a preparação individual, a data e o local da reunião preliminar (se houver) e da reunião de inspeção, se é uma re-inspeção ou não e critérios para decidir se será feita uma nova inspeção e, também, o planejamento dos testes. Além disso, qualquer material necessário à verificação de um determinado produto de trabalho como, por exemplo, o material para preparação individual das revisões por pares, deve ser identificado neste planejamento e disponibilizado no momento da execução da verificação.

9.3.3 VER3 - Critérios e procedimentos para verificação dos produtos de trabalho a serem verificados são identificados e um ambiente para verificação é estabelecido

O alcance deste resultado esperado implica na definição dos critérios e procedimentos que serão utilizados para a verificação de cada produto de trabalho e na preparação do ambiente para verificação, disponibilizando ferramentas, recursos de hardware, infra-estrutura de rede e outros recursos necessários à execução das atividades planejadas.

Para ajudar a determinar se um critério foi ou não atendido, questões (*checklist*) e/ou métricas para cada critério podem ser definidas. Algumas sugestões de critérios e questões para alguns artefatos podem ser encontradas em [ISO/IEC, 2008; ISO/IEC, 2001; RAKITIN, 2001; McCONNELL, 2004; MYERS, 2004; BARRETO, 2006].

Os relatórios técnicos [ISO/IEC, 2003a; ISO/IEC, 2003b; ISO/IEC, 2004] apresentam um conjunto de critérios para avaliação de produto associados às características de qualidade do produto e métricas relacionadas a esses critérios. Para cada métrica 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 [ISO/IEC, 2008] e os usuários interessados nos resultados da métrica. A Tabela 5.1 mostra um exemplo com características de qualidade, critérios e

questões para verificação de um documento de requisitos do software utilizando revisão por pares [BARRETO, 2006].

Tabela 5.1 – Exemplo de Critérios para Avaliação de Requisitos de Software [BARRETO, 2006]

Característica de Qualidade	Critério	Checklist
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?
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?
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?

9.3.4 VER4 - Atividades de verificação, incluindo testes e revisões por pares, são executadas

Este resultado esperado visa garantir que as atividades de verificação são executadas conforme planejado, o que inclui, obrigatoriamente, a realização de revisão por pares e testes.

9.3.5 VER5 - Defeitos são identificados e registrados

Este resultado esperado visa garantir que os defeitos identificados durante a execução da verificação são documentados e registrados. Para registro dos defeitos identificados pode-se usar uma classificação de defeitos, por exemplo, por severidade (crítico, sério, moderado) ou por origem (requisitos, projeto (*design*), código, testes).

Após a eliminação dos defeitos, deve-se julgar a necessidade de executar nova verificação para garantir que os defeitos foram removidos adequadamente e que novos defeitos não foram introduzidos no produto ou componente do produto.

Nem todos os defeitos encontrados precisam ser corrigidos. A organização pode definir critérios que facilitem essa análise considerando os riscos para o projeto e o impacto na qualidade do produto.

9.3.6 VER6 - Resultados de atividades de verificação são analisados e disponibilizados para as partes interessadas

O alcance deste resultado esperado envolve realizar uma análise dos resultados obtidos em cada atividade de verificação e disponibilizar estes resultados para as partes interessadas.

Assim, uma forma de alcance deste resultado é pela análise de laudos de avaliação e relatórios de testes que contenham informações sobre os resultados obtidos após a realização das atividades de verificação. Exemplos de perguntas que podem ser respondidas com esta avaliação incluem:

- Os critérios definidos foram satisfeitos?
- As ações corretivas planejadas foram concluídas?
- A verificação foi executada conforme planejado?
- Os resultados obtidos permitem a aprovação do artefato verificado?

10 Os atributos de processo no nível D

A evolução do nível E para o nível D não apresenta novidades em termos dos atributos de processo já implantados no nível E. A evolução para o nível D do MR-MPS implica, portanto, como descrito na seção 4, apenas na definição e implementação dos cinco novos processos com a mesma capacidade dos processos já implantados.

Referências Bibliográficas

- [ALEXANDER e ROBERTSON, 2004] ALEXANDER, I., ROBERTSON, S. **Understanding Project Sociology by Modeling Stakeholders**. IEEE Software, vol. 21, no. 1, pp. 23-27, Jan/Feb, 2004.
- [ANDERSSON, 2003] ANDERSSON, C., **Exploring the Software Verification and Validation Process with Focus on Efficient Fault Detection**, Licentiate Thesis, Lund Institute of Technology (LTH), Lund University, Sweden, 2003.
- [BARRETO, 2006] BARRETO, A.O.S., “**Apoio à Verificação de Software em Ambientes de Desenvolvimento de Software Orientados à Organização**”, Dissertação de M. Sc., COPPE/UFRJ, Rio de Janeiro, Brasil, 2006.
- [BARTIÉ, 2002] BARTIÉ, A., “**Garantia da Qualidade de Software**”, Editora Campus, São Paulo, 2002
- [BASS *et al.*, 2003] BASS, L., CLEMENTS, P., KAZMAN, R., **Software Architecture in Practice**. Addison-Wesley, 2 edition, 2003.
- [BOEHM, 1981] BOEHM, B. **Software Engineering Economics**. Prentice-Hall, 1981
- [BOEHM e BASILI, 2001] BOEHM, B., BASILI, V., “**Software Defect Reduction Top 10 List**”, *IEEE Computer*, v. 34, n.1, pp. 135-137, 2001.
- [BRIAND *et al.*, 2002] BRIAND, L.C., FENG, J., LABICHE, Y., “**Experimenting with Genetic Algorithms and Coupling Measures to Devise Optimal Integration Test Orders**”. In: Technical Report TR SCE-02-03-Version 3, Outubro, Carleton University, Ottawa, Canadá, 2002.
- [BOUCHRIHA, *et al.*, 2002] BOUCHRIHA, H., D’AMOURS, S., LADET, P., “**A ‘make or buy’ decision model with economies of scale**”, Article de conférence avec actes, 2002.
- [BURGE, 2005] BURGE, J. E., “**Software Engineering Using design Rationale**”, PhD Dissertation, CS Dept., WPI, May, 2005
- [CACHERO e KOCH, 2002] CACHERO, C., KOCH, N. **Navigation Analysis and Navigation Design in OO-H and UWE**. Technical Report 0205, Institute of Computer Science, Ludwig-Maximilians University of Munich, 2002.
- [CÁNEZ *et al.*, 2001] CÁNEZ, L., PROBERT, D., PLATTS, K., **Testing a Make-or-Buy Process**, *Proceedings of the Twelfth Annual Conference of the Production and Operations Management Society*, POM-2001, March 30 – April 2, Orlando, 2001.
- [COAD e YOURDON, 1992] COAD, P., YOURDON, E. **Análise Baseada em Objetos**, Editora Campus, 1992.
- [CONKLIN, 1989] CONKLIN, J., “**Design Rationale and Maintainability**”. Vol. II: Software Track, In: Proceedings of the Twenty-Second Annual Hawaii International Conference, Volume 2, 3-6 Jan. Page(s): 533-539.
- [IEEE, 2004] IEEE COMPUTER SOCIETY PROFESSIONAL PRACTICES COMMITTEE. **Software Engineering Body of Knowledge – SWEBOK**, 2004 Disponível em: <http://www.swebok.org>, verificado em Abril/2009.

[ISO/IEC, 1994] THE INTERNATIONAL ORGANIZATION FOR STANDARDIZATION AND THE INTERNATIONAL ELECTROTECHNICAL COMMISSION. **ISO/IEC 12119: Information Technology – Software packages – Quality requirements and testing**, Geneve: ISO, 1994.

[ISO/IEC, 2001] THE INTERNATIONAL ORGANIZATION FOR STANDARDIZATION AND THE INTERNATIONAL ELECTROTECHNICAL COMMISSION. **ISO/IEC TR 9126-1: Software engineering - Product quality - Part 1: Quality model**. Geneve: ISO, 2001.

[ISO/IEC, 2003a] THE INTERNATIONAL ORGANIZATION FOR STANDARDIZATION AND THE INTERNATIONAL ELECTROTECHNICAL COMMISSION. **ISO/IEC TR 9126-2: Software engineering - Product quality - Part 2: External metrics**. Geneve: ISO, 2003.

[ISO/IEC, 2003b] THE INTERNATIONAL ORGANIZATION FOR STANDARDIZATION AND THE INTERNATIONAL ELECTROTECHNICAL COMMISSION. **ISO/IEC TR 9126-3: Software engineering - Product quality - Part 3: Internal metrics**. Geneve: ISO, 2003.

[ISO/IEC, 2004] THE INTERNATIONAL ORGANIZATION FOR STANDARDIZATION AND THE INTERNATIONAL ELECTROTECHNICAL COMMISSION. **ISO/IEC TR 9126-4: Software engineering - Product quality - Part 4: Quality in Use**. Geneve: ISO, 2004.

[ISO/IEC, 2008] THE INTERNATIONAL ORGANIZATION FOR STANDARDIZATION AND THE INTERNATIONAL ELECTROTECHNICAL COMMISSION. **ISO/IEC 12207:2008 Systems and software engineering — Software life cycle processes**, Geneve: ISO, 2008.

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

[KELLNER *et al.*, 1999] KELLNER, M., MADACHY, R., RAFFO, D. **Software Process Modeling and Simulation: Why, What, How**. Journal of Systems and Software, Vol. 46, No. 2/3, 1999.

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

[LIMA, 2005] LIMA, G. M. P. S., “**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, 2005.

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

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

- [MYERS, 2004] MYERS, G., **The Art of Software Testing**, 2ª Edição, John Wiley & Sons, Hoboken, New Jersey, Estados Unidos, 2004.
- [PENDER, 2004] PENDER, T., **UML, A Bíblia**. Rio de Janeiro: Elsevier, 1ª Edição, 2004
- [PFLEEGER, 2004] PFLEEGER, S.L. **Engenharia de Software – Teoria e Prática**, 2ª edição, Prentice Hall, São Paulo, Brasil, 2004.
- [PRESSMAN, 2005] PRESSMAN, R. S. **Software Engineering: A Practitioner's Approach**, 6th. ed., McGraw-Hill.
- [PUTNAM e MYERS, 2003] PUTNAM, L.H., MYERS, W., “**Five Core Metrics**”, Dorset House Publishing, 2003.
- [RAKITIN, 2001] RAKITIN, S.R., **Software Verification and Validation for Practitioners and Managers**, 2ª Edição, Artech House Publishers, Norwood, Estados Unidos, 2001.
- [ROCHA *et al.*, 2001] ROCHA, A.R.C., MALDONADO, J.C., WEBER, K.C., **Qualidade de Software – Teoria e Prática**, Prentice Hall, São Paulo, Brasil, 2001.
- [ROSENBLUM e WEYUKER, 1997] ROSENBLUM, D. S. and WEYUKER, E. J., **Using coverage information to predict the cost effectiveness of regression testing strategies**. IEEE Trans. on Software Engineering, 23(3): 146-156, 1997.
- [ROTHERMEL e HARROLD, 1996] ROTHERMEL, G. and HARROLD, M. J., **Analyzing regression test selection techniques**. IEEE Trans. on Software Engineering, 22(8):529-551, 1996.
- [SCHULMEYER e MACKENZIE, 1999] SCHULMEYER, G.G., MACKENZIE, G.R., **Verification & Validation of Modern Software-Intensive Systems**, New Jersey, Prentice-Hall Inc, 1999.
- [SEI, 2006] SOFTWARE ENGINEERING INSTITUTE - SEI. **CMMI® for Development, Version 1.2**, Carnegie Mellon University, Software Engineering Institute, Pittsburgh, agosto, 2006. Disponível em: <http://www.sei.cmu.edu>, verificado em Abril/2009.
- [SEYBOLD *et al.*, 2005] SEYBOLD, C., GLINZ, M., MEIER, S., “**Simulation-based Validation and Defect Localization for Evolving, Semi-Formal Requirements Models**”, In: Proceedings of the 12th Asia-Pacific Software Engineering Conference, pp.408-420, 2005.
- [SHAW e GARLAN, 1996] SHAW, M., GARLAN, D., “**Software Architectures-- Perspectives on an Emerging Discipline**”, Prentice Hall, 1996.
- [SOFTTEX, 2009a] ASSOCIAÇÃO PARA PROMOÇÃO DA EXCELÊNCIA DO SOFTWARE BRASILEIRO – SOFTTEX. **MPS.BR – Guia Geral:2009**, maio 2009. Disponível em www.softex.br.
- [SOFTTEX, 2009b] ASSOCIAÇÃO PARA PROMOÇÃO DA EXCELÊNCIA DO SOFTWARE BRASILEIRO – SOFTTEX. **MPS.BR – Guia de avaliação:2009**, maio 2009. Disponível em www.softex.br.

[SOFTEX, 2009c] ASSOCIAÇÃO PARA PROMOÇÃO DA EXCELÊNCIA DO SOFTWARE BRASILEIRO – SOFTEX. **MPS.BR – Guia de Aquisição:2009**, maio 2009. Disponível em www.softex.br.

[SOMÉ, 2005] SOMÉ, S., "**Use Cases based Requirements Validation with Scenarios**", In: Proceedings of the 2005 13th IEEE International Conference on Requirements Engineering, pp.465-466, 2005.

[SOMMERVILLE, 2003] SOMMERVILLE, I. **Engenharia de Software**, Addison Wesley, 6a edição, 2003.

[SUKUMARAN *et al.*, 2006] SUKUMARAN, S., SREENIVAS, A., VENKATESH, R., "**A rigorous approach to requirements validation**", In: Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods, pp.236-245, 2006.

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

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

[VIEIRA e TRAVASSOS, 1998] VIEIRA, M.E.R., TRAVASSOS, G.H., "**An Approach to Perform Behavior Testing in Object-Oriented Systems**". Technology of Object-Oriented Languages and Systems, Setembro, pp 318-328, 1998.

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

Lista de colaboradores do Guia de Implementação – Parte 4:2009

Editores:

Ana Regina C. Rocha	COPPE/UFRJ (Coordenadora da ETM)
Gleison dos Santos Souza	COPPE/UFRJ
Mariano Angel Montoni	COPPE/UFRJ

Revisores

Ana Liddy C. C. Magalhães	QualityFocus e Universidade FUMEC
Ana Regina C. Rocha	COPPE/UFRJ (Coordenadora da ETM)
Edmeia Leonor Pereira de Andrade	EMBRAPA e UCB

**Lista de colaboradores do Guia de Implementação – Parte 4 versão 1.1 –
Julho/2007**

Editores:

Ana Regina C. Rocha	COPPE/UFRJ (Coordenadora da ETM)
Mariano Angel Montoni	COPPE/UFRJ

Revisores:

Danilo Scalet	CELEPAR
Edmeia Leonor Pereira de Andrade	MAPA
Fábio Bianchi Campos	Universidade Católica de Brasília

**Lista de colaboradores do Guia de Implementação – Parte 4 versão 1.0 –
Dezembro/2006**

Editoras:

Ana Regina C. Rocha	COPPE/UFRJ (Coordenadora da ETM)
Káthia Marçal de Oliveira	Universidade Católica de Brasília

Colaboradores:

Ahilton Barreto	COPPE/UFRJ
Andrea Soares Barreto	COPPE/UFRJ
Sávio Figueiredo	COPPE/UFRJ
Tayana Conte	COPPE/UFRJ

Revisores:

Danilo Scalet	CELEPAR
Káthia Marçal de Oliveira	Universidade Católica de Brasília