

Bruno Ferreira

**UMA TÉCNICA PARA VALIDAÇÃO DE PROCESSOS DE
DESENVOLVIMENTO DE SOFTWARE**

Bruno Ferreira

UMA TÉCNICA PARA VALIDAÇÃO DE PROCESSOS DE
DESENVOLVIMENTO DE SOFTWARE

Dissertação apresentada ao Curso de Mestrado em Modelagem Matemática e Computacional (MMC) do Centro Federal de Educação Tecnológica de Minas Gerais, como requisito parcial à obtenção do título de Mestre em Modelagem Matemática e Computacional.

Área de pesquisa: Sistemas Inteligentes.

Orientador: Prof. Dr. Gray Farias Moita

Belo Horizonte
Centro Federal de Educação Tecnológica de Minas Gerais
Diretoria de Pesquisa e Pós-Graduação 2008

Ferreira, Bruno
F383t Uma técnica para validação de processos de desenvolvimento de
2008 software. – 2008.
 146 f.

Orientador: Gray Farias Moita

Dissertação (mestrado) – Centro Federal de Educação Tecnológica
de Minas Gerais.

1. Engenharia de software – Teses. 2. Software – Verificação.
3. Software – Validação. I. Moita, Gray Farias. II. Centro Federal de
Educação Tecnológica de Minas Gerais. III. Título.

CDD 005.1

Folha de aprovação. Esta folha será fornecida pelo Programa de Pós-Graduação e deverá substituir esta página.

Dedico este trabalho aos meus pais, Divino e Maria e aos meus irmãos Efrem e Viviane, que sempre me incentivam a seguir em frente e a superar os desafios. À minha namorada Danielle, que compreendeu meus momentos de ausência e impaciência.

AGRADECIMENTOS

A Deus, por me proporcionar todas as oportunidades de aprendizado vivenciadas até o presente momento.

Ao meu orientador, Prof. Dr. Gray Farias Moita, pelas diretrizes de pesquisa, pelo incentivo durante todo o projeto e pela paciência e apoio mesmo nos momentos adversos.

A TOTVS/SA, pela confiança e pela participação nos estudos realizados nessa pesquisa. Um agradecimento especial para Isabela Iunes de Oliveira, Túlio Mendes Rodrigues e Sérgio Gontijo do Nascimento por aceitarem o convite de participação nas pesquisas.

Aos colegas e amigos da TOTVS/SA e ao Prof. Vitor da Fundação Educacional de Oliveira pelo apoio nos momentos de dificuldade.

Aos professores e colegas do Grupo de Pesquisa em Sistemas Inteligentes – GPSI, do CEFET-MG.

À todos aqueles que contribuíram de alguma forma com este trabalho.

*“Comece fazendo o que é necessário, depois o que
é possível, e de repente você estará fazendo o
impossível.” (São Francisco de Assis)*

RESUMO

A grande demanda e o crescente grau de complexidade do software nas últimas décadas obrigaram as empresas desenvolvedoras de sistemas a utilizarem processos de desenvolvimento de software. Embora, existam na literatura vários processos de desenvolvimento, com características diferentes que atendem os diversos ramos do desenvolvimento, tais processos não se mostram suficientes. As práticas de desenvolvimento diferem entre as organizações, o que as obrigam a criar ou aprimorar processos já conhecidos. No entanto, para que um processo possa garantir sua eficiência ele deve ser válido. Neste trabalho é proposto um modelo de validação de processo, baseado nos conceitos de Verificação & Validação (V&V), fazendo uso da técnica de Inspeção de Software para validar processos de desenvolvimento de software. Ao final, é apresentado um estudo de caso que foi realizado como uma primeira tentativa de analisar o modelo proposto. Os resultados apresentados indicam que é viável e simples validar processos pelo uso desses conceitos e técnicas de V&V.

ABSTRACT

The increasing demand for software in the last decades led to a most intensively use of specific software development processes. Although there are several software development processes with different characteristics in the literature, such processes are not enough to encompass the numerous fields of software applications. The practices of development differ among the developers and, therefore, sometimes it is necessary to improve or join processes already known to guarantee one needs. Consequently some software producers are still in need of specific processes that can attend their particular interests and, in some situations, forcing them to conceive new processes. However, a process can only guarantee its correctness and efficiency after being thoroughly tested and validated. The validation is conceptually complex and related to a wide range of issues, many times subjective, what leads to a possible absence of guidelines or benchmarks to validate a process of development of software. In this research a process validation model is proposed, based on the concepts of Verification and Validation (V&V), making use of the Software Inspection Technique to validate software development processes. It is presented a case study that was carried out as a first attempt of analysing the proposed model. The presented results indicate that it is viable and simple to validate processes with the use of V&V concepts and techniques.

LISTA DE ABREVIATURAS E SIGLAS

CASE Computer Aided Software Engineering

CEFET/MG Centro Federal de Educação Tecnológica de Minas Gerais

CMMI Capability Maturity Model Integration

ERP *Enterprise Resource Planning*

GPSI Grupo de Pesquisa de Sistemas Inteligentes

IBIS Internet Based Inspection System

IEEE Instituto de Engenheiros Eletricistas e Eletrônicos

ISPI Infra-estrutura de Suporte ao Processo de Inspeção

ISO International Organization for Standardization

IV&V Verificação e Validação Independente

LSI Laboratório de Sistemas Inteligentes

MVP Planos de Verificação Principal

NAS Manual de Engenharia de Sistemas

NASA Administração Nacional de Aeronáutica e Espaço Norte Americana

NA Não aplicável

PESC Processo de Desenvolvimento Específico para Software Científico

PU Processo Unificado

RAD Rapid Application Development

RE Resposta esperada

RVCD Documento de Cumprimento de Verificação dos Requisitos

SEI Software Engineering Institute

SDK Kits de Desenvolvimento de Software

SQA Garantia da Qualidade de Software

UML Unified Modeling Language

VRTM Matriz de Responsabilidade de Verificação é criado

VProcInsp Validação de Processos por Inspeção

V&V Verificação e Validação

XP Extreme Programming

LISTA DE FIGURAS

Figura 1.1 – Apresentação da dissertação em capítulo	21
Figura 2.1 – Camadas da engenharia de software (Fonte: Pressman, 2006) ..	23
Figura 2.2.1 – Modelo em Cascata (Fonte: Pressman, 2006).....	26
Figura 2.2.2 – Modelo Incremental (Fonte: Pressman, 2006).	27
Figura 2.2.3 – Modelo RAD (Fonte: Adaptado de Pressman, 2006).	28
Figura 2.2.4 – Modelo de Prototipagem (Fonte: Pressman, 2006).....	30
Figura 2.2.5 – Modelo Espiral (Fonte: Pressman, 2006).	31
Figura 2.3.1 – Termos relativos ao cronograma de um PU (Fonte: Larman, 2004)	33
Figura 2.3.2 – Artefatos mais importantes produzidos durante as fases do PU (Fonte: Pressman, 2006).....	34
Figura 2.3.4 – Ciclo de desenvolvimento proposto pelo PESC. (Fonte: Pereira Jr, 2007)	38
Figura 2.5 – Framework de comparação de eventos (Fonte: Cook e Wolf, 1999).	47
Figura 3.1 – Atividades de Verificação e Validação (Adaptada de NAS, 2006).	51
Figura 3.2 – Diagrama em “V” da Engenharia de Sistemas (Adaptada de NAS, 2006).	52
Figura 3.3 – Modelo clássico de V&V (Adaptada de Sargent, 2000).....	53
Figura 3.4.1 – Redistribuição do retrabalho pelas atividades de desenvolvimento de software. (Adaptado de Wheeler <i>et al.</i> , 1996)	65
Figura 3.4.2 – Custo relativo para corrigir um defeito (Adaptado de Boehm, 1981).	65
Figura 3.4.3 – Processo de inspeção de software (Fonte: Fagan, 1976)	67
Figura 3.4.4 – Processo de Inspeção de Software Assíncrono (Fonte: Sauer <i>et al.</i> , 2000).	69
Figura 4.1 – Inspeção de Software em diferentes artefatos (Fonte: Ackerman <i>et al.</i> , 1989)	73
Figura 4.2 – Versão padrão do Checklist proposto pelo VProclnsp	80
Figura 4.3 – Modelo de Inspeção proposto pelo VProclnsp (Adaptado de Sauer <i>et al.</i> , 2000).	82
Figura 4.3.2 – Atividades pertencentes à fase de Planejamento.....	84
Figura 4.3.3 – Documento de Planejamento de Validação por Inspeção proposto pelo VProclnsp	85

Figura 4.3.4 – Documento de Cadastro de Inspetores proposto pelo VProclnsp	86
Figura 4.3.5 – Documento de Atribuições de Inspetores proposto pelo VProclnsp.....	87
Figura 4.3.6 – Atividades pertencentes à fase de Análise e Comparação	88
Figura 4.3.7 – Relatório de Discrepância proposto pelo VProclnsp	89
Figura 5.1 – Processo de desenvolvimento criado pela RM Sistemas.....	99
Figura 5.2 – Fase de Especificação do processo de desenvolvimento criado pela RM	100
Figura 5.3 – Fase de Implementação do processo de desenvolvimento criado pela RM	101
Figura 5.4 – Contexto em que o VProclnsp foi aplicado no estudo de caso ..	103
Figura 5.5 – Relatório de Discrepância preenchido com os dados do estudo de caso.....	104
Figura 5.6 – Relatório de Discrepância preenchido com os dados do estudo de caso.....	106

LISTA DE TABELAS

Tabela 2.1 – Princípios da XP (Adaptado de Astels <i>et al.</i> , (2002).....	35
Tabela 2.2 – Método de validação de processos baseados em atividades. (Fonte: Moor e Delugach, 2006).....	46
Tabela 3.1 – Taxonomia de Técnicas de Verificação & Validação (Fonte: Balci, 1995).	56
Tabela 3.2 – Características da técnica de Desk-Checking. (Fonte: Perry, 1995).	57
Tabela 3.3 – Características da técnica de Peer-Review. (Fonte: Perry, 1995).	58
Tabela 3.4 – Características da técnica de Design Review (Fonte: Perry, 1995)	59
Tabela 3.5 – Características da técnica de Auditoria (Fonte: Perry, 1995).	60
Tabela 3.6 – Características da técnica de Walkthrough (Fonte: Perry, 1995).61	
Tabela 3.7 – Características da técnica de Inspeções (Fonte: Perry, 1995)	62
Tabela 4.1 – Taxonomia de defeitos propostos pelo VProclnsp	77
Tabela 4.2 – Técnica de Insp. de Software (Adaptado de Barcelos e Travassos, 2005).	78
Tabela 5.1 – Planejamento de experimento.....	97
Tabela 5.2 – Informações levantadas pelo VProclnsp antes da fase de Discriminação.....	109
Tabela 5.3 – Descrição dos itens com discrepâncias do checklist de especificação.....	111
Tabela 5.4 – Informações levantadas pelo VProclnsp na etapa de Especificação na fase de Discriminação	113

SUMÁRIO

CAPÍTULO 1	16
INTRODUÇÃO	16
1.1 Motivação.....	17
1.2 Caracterização do Problema.....	18
1.3 Objetivos: Geral e Específico	20
1.4 Organização da Dissertação.....	20
CAPÍTULO 2	22
PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE.....	22
2.1 Processo de desenvolvimento: visão geral	22
2.2 Modelos de Processo de Desenvolvimento	24
2.2.1 Modelo em Cascata	25
2.2.2 Modelos Incrementais.....	26
2.2.3 Modelos Evolucionários.....	29
2.3 Processos de Desenvolvimento de Software.....	31
2.4 Questões sobre avaliação e validação em processos de desenvolvimento de software	40
2.5 Análise das técnicas empregadas em processos de desenvolvimento de software	42
CAPÍTULO 3	49
VERIFICAÇÃO E VALIDAÇÃO DE SOFTWARE	49
3.1 Conceitos de Verificação & Validação	49
3.2 Técnicas de Verificação & Validação	55
3.3 Análise das Técnicas Informais.....	57
3.4 Conceitos de Inspeção de Software.....	63
3.4.1 Benefícios da Inspeção de Software.....	64
3.4.2 O Processo de Inspeção de Software.....	66
CAPÍTULO 4	71
PROPOSTA DE UMA TÉCNICA DE VALIDAÇÃO DE PROCESSO POR INSPEÇÃO.....	71
4.1 A técnica proposta: visão geral e justificativas	71
4.2 Validação de Processo de Desenvolvimento de Software por Inspeção	74

4.2.1 Detalhamento do Processo de Validação Proposto	74
4.2.1.1 Determinar os artefatos do processo.....	75
4.2.1.2 Definição do papel e do perfil da equipe participante do processo de inspeção para validação.....	75
4.2.1.3 Taxonomia de defeitos	77
4.2.1.4 Técnica de inspeção utilizada.....	78
4.2.1.5 Processo utilizado na inspeção	82
CAPÍTULO 5	92
AVALIAÇÃO DA TÉCNICA DE VALIDAÇÃO DE PROCESSO POR INSPEÇÃO PROPOSTA	92
5.1 Discussão	92
5.2 Definição	95
5.3 Planejamento	95
5.4 Operação	98
5.5 Análise dos Dados e Resultados Obtidos	107
5.5.1 Análise dos resultados da abordagem.....	107
5.5.2 Análise da utilização da abordagem	114
5.6 Considerações em relação à hipótese do estudo	114
CAPÍTULO 6	116
CONCLUSÕES E CONSIDERAÇÕES FINAIS	116
6.1 Resumo.....	116
6.2 Contribuições	119
6.3 Limitações	120
6.4 Trabalhos Futuros	120
REFERÊNCIAS BIBLIOGRÁFICAS	122
Apêndice A	128
Documentos do VProInsp Aplicados no Estudo de Caso	128
Apêndice B	143
Questionário de Avaliação Pós-Experimento	143

CAPÍTULO 1

INTRODUÇÃO

O desenvolvimento e o uso de software têm passado por profundas modificações, seguindo o aumento da capacidade de processamento e de memória das máquinas. Seu uso estende-se praticamente por todos os setores da atividade humana. A automatização de tarefas repetitivas, o aumento de controle e eficiência em procedimentos específicos, a possibilidade de antecipação de problemas e apresentação de uma solução prévia, como é o caso de simulações computacionais, são apenas algumas das possíveis aplicações dessa tecnologia. Mas, em consequência deste contexto, a criação e manutenção de software vêm apresentando um significativo aumento na complexidade, fato este que favorece a maior incidência de erros e, consequentemente, queda na qualidade.

Para contornar esta situação, técnicas de Engenharia de Software são empregadas nos casos em que se deseja obter a garantia da qualidade do software que será desenvolvido. Estas técnicas - conhecidas como Processos de Desenvolvimento de Software, ou simplesmente, processo de desenvolvimento - quando bem empregadas, possibilitam um desenvolvimento de software de alta confiabilidade e qualidade.

Apesar das vantagens no uso dos vários processos de desenvolvimento propostos, a implantação e a aplicação desses processos se revelam uma tarefa complexa, sendo altamente dependentes do ambiente em que eles são inseridos. Em consequência desse panorama, um considerável número de processos vêm sendo criados, aperfeiçoados ou simplesmente otimizados para atender as necessidades das equipes de projetos.

Para evitar problemas ao criar ou otimizar um processo, Humphrey (1995), apresentou uma metodologia de criação que consiste dos seguintes passos: (1)

Determinar as necessidades e as prioridades do novo processo; (2) Definir os objetivos e os critérios de qualidade; (3) Caracterizar o processo atual; (4) Caracterizar o processo desejado; (5) Estabelecer uma estratégia de desenvolvimento do processo; (6) Definir um processo inicial; (7) Validar o processo inicial; (8) Melhorar o processo. Esses passos são importantes porque podem diminuir a subjetividade de criação, formalizando as atividades necessárias a serem seguidas.

A validação do processo inicial é um passo importante e pode responder a uma preocupação natural da gerência de projetos – a garantia que o processo de desenvolvimento guia os participantes de forma correta e eficiente durante a criação do software. Burocracia excessiva ou uma orientação ambígua podem atrapalhar, ao invés de orientar, o desenvolvimento no ciclo de criação do produto.

Neste contexto, este trabalho tem o intuito de apresentar uma técnica para validar um processo de desenvolvimento de software, fundamentada em conceitos, diretrizes e técnicas de Verificação & Validação de Software, minimizando a subjetividade de criação de processos. Para essa técnica, definiram-se algumas características que são consideradas importantes e que seria interessante estarem presentes na atividade de validação: (1) Genérica e Adaptável – para que a técnica possa ser aplicada na avaliação de diferentes artefatos do processo de desenvolvimento, ficando independente, portanto, da abordagem utilizada para criá-los; (2) Simples – para que não seja necessária a execução de atividades elaboradas para sua aplicação, a exigência de tipos de conhecimento específicos ou a alocação de grandes quantidades de recursos; (3) Extensível – para que seja facilmente modificável e permita ser aplicada com outras técnicas de validação de processos de desenvolvimento.

1.1 Motivação

O cenário mundial nunca foi tão competitivo como hoje e isso não está restrito a um setor específico. No ambiente de competição, onde as empresas procuram alcançar uma posição de destaque sobre seus concorrentes, elas adotam estratégias visando criar e sustentar vantagens competitivas.

No campo da tecnologia da informação, o processo de desenvolvimento de software é um elemento fundamental para obtenção dessas vantagens no que diz respeito à criação de sistemas. As empresas devem produzir software com qualidade, e em tempo hábil, para que o produto final seja competitivo.

Diante deste cenário, o processo de desenvolvimento deve guiar os envolvidos nos projetos de software de forma clara e eficiente. Um processo excessivamente burocrático ou que não condiz com a realidade da empresa e, que exige recursos humanos e material inexistente, ou conhecimento e treinamento insuficiente, pode confundir os envolvidos na criação de software que utilizam tal processo.

Sendo assim, o processo de desenvolvimento deve ser eficiente e tangível (alcançável) de ser seguido. Ou seja, a execução do processo durante um projeto deve seguir o modelo formal do processo proposto. Indicando que o processo é válido.

Segundo Cook e Wolf (1999), o processo de validação serve a vários objetivos, entre eles garantir a confiança no modelo do processo formal, pois a execução do processo deve ser compatível com o modelo do processo descrito. Isto, por sua vez, aumenta a confiança nos resultados de qualquer análise executada no modelo formal do processo. A validação do processo pode ser usada também como uma ferramenta de garantia do processo, descobrindo diferenças entre o comportamento desejado e o comportamento real. Finalmente, o processo de validação pode revelar quando um processo talvez necessite evoluir para acomodar atividades e requisitos de um novo projeto.

1.2 Caracterização do Problema

Validar um processo de desenvolvimento não é uma tarefa simples. Um problema é que validação é conceitualmente complexa, pois se refere a uma larga escala de questões, muitas vezes subjetivas. Outro problema encontrado é a ausência de diretrizes, técnicas ou padrões consolidados e documentados para auxiliar os interessados em executar esta tarefa.

As poucas pesquisas nesta área são complexas e dependentes do contexto em que foram aplicadas. Logo, é evidente a ausência de uma técnica que seja: (1) Genérica e Adaptável; (2) Simples e (3) Extensível.

Neste ponto, é importante ressaltar que validar não é dizer que o produto criado foi construído corretamente utilizando o processo, mas sim provar que o processo atende às necessidades dos colaboradores, quanto às questões de Engenharia de Software. Mais especificamente, o processo deve representar com exatidão as atividades de comunicação, planejamento, modelagem, construção, implantação e manutenção, na perspectiva pretendida pelos envolvidos no projeto, que é justamente a idéia de Validação (Cook e Wolf, 1999).

Contudo, criar ou mesmo padronizar técnicas de validação de processos, tornando-as disponíveis, ajudaria qualquer instituição que tenha a necessidade de gerar seu próprio processo, evitando incertezas ou mesmo falhas no produto, por consequência de ambigüidade, inconsistências ou erros em um processo de software recém criado.

Diante dessas colocações, surge então o problema, objeto de discussão, para o qual se intenta encontrar respostas no decorrer da presente pesquisa. Como validar novos processos de desenvolvimento de software de forma simples, extensível e adaptável?

Baseada na questão em estudo nesta dissertação, a hipótese formulada é: “Se forem aplicados conceitos e técnicas de Verificação e Validação de Software, então a tarefa de validar um processo pode se tornar simples, adaptável e extensível”.

Sendo assim, focando no problema mencionado, propõe-se utilizar técnicas de Verificação e Validação de Software, mais precisamente, Inspeção de Software. Essa é uma técnica metódica, mas de fácil aplicação, e vem sendo utilizada para validar documentos conceituais em outros ramos da Engenharia de Software.

1.3 Objetivos: Geral e Específico

O objetivo geral deste trabalho é desenvolver uma técnica de validação de processos de desenvolvimento de software utilizando de conceitos e técnicas de Verificação & Validação empregadas em softwares.

Para tanto, este trabalho se propôs aos seguintes objetivos específicos:

- Apresentar um referencial teórico acerca das técnicas utilizadas para validar e verificar software e possíveis técnicas para validação de processos.
- Propor uma técnica de validação a partir da identificação das melhores práticas e técnicas de Verificação & Validação, que se aplique aos processos de desenvolvimento de software.
- Validar a técnica proposta aplicando-a em um processo já conhecido e utilizado em grande escala.

1.4 Organização da Dissertação

Esse trabalho foi organizado em seis capítulos. Após a introdução são apresentados conceitos de processos de desenvolvimento de software e Verificação & Validação de Software, além de uma análise sobre certificação de processo e de trabalhos correlatos. Em seguida, é apresentada a técnica de validação proposta, o estudo de caso realizado e uma análise dos resultados. Após as considerações finais são listadas as referências bibliográficas utilizadas e os apêndices com os documentos gerados pelo estudo de caso. A Figura 1.1 apresenta graficamente a organização dessa dissertação em capítulos.

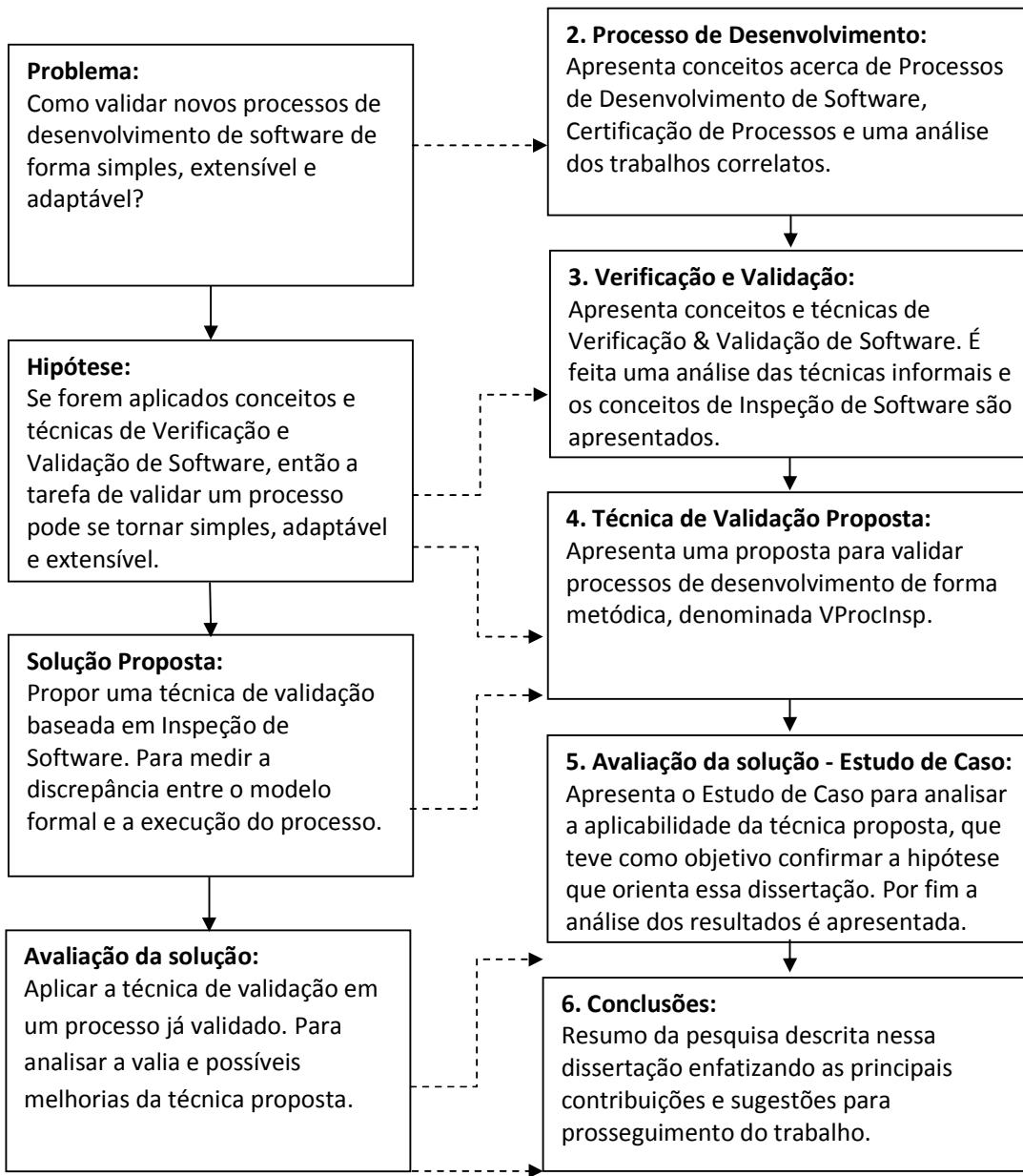


Figura 1.1 – Apresentação gráfica da organização da dissertação em capítulos

CAPÍTULO 2

PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE

Neste capítulo são apresentadas algumas definições que permitem identificar o contexto dos processos de desenvolvimento de software. Para embasar o trabalho, foram estudados os modelos de processos, dois processos de desenvolvimento de software convencionais, existentes na literatura e um processo proposto recentemente para a comunidade científica, mas que necessita de uma validação formal. É apresentada também a diferença entre avaliar e validar um processo e, para finalizar, um panorama sobre as técnicas de validação e de melhorias de processo existentes na literatura é levantado.

2.1 Processo de desenvolvimento: visão geral

Na década de 70, a atividade de desenvolvimento de software era executada de forma desorganizada, desestruturada e sem planejamento, gerando um produto final de má qualidade. Essa época ficou conhecida como Crise do Software (Melo, 2004). A partir deste cenário, surgiu a necessidade de tornar o desenvolvimento de software um processo estruturado, planejado e padronizado.

Segundo Paula (2003, p.11), um processo é “um conjunto de passos parcialmente ordenados, constituídos por atividades, métodos, práticas e transformações, usados para atingir uma meta”.

Para Pressman (2006, p.17), os processos de software formam a base para o controle gerencial de projetos de software e estabelecem o contexto no qual os métodos técnicos são aplicados, os produtos de trabalho (modelos, documentos, dados, relatórios, formulários) são produzidos, os marcos são

estabelecidos, a qualidade é assegurada e modificações são adequadamente geridas.

Ainda segundo Pressman, o processo é o elo que mantém unidas as camadas de tecnologia e permite o desenvolvimento racional e oportuno de software de computador. A Figura 2.1 ilustra a camada de processo e as demais camadas da engenharia de software.



Figura 2.1 – Camadas da engenharia de software (Fonte: Pressman, 2006).

A engenharia de software tem como foco principal a qualidade final do produto gerado, e uma maneira de atingi-la é através do aperfeiçoamento do processo de desenvolvimento. Para auxiliar nessa tarefa os métodos fornecem as técnicas de construção, que abrangem um amplo conjunto de operações que incluem análise de requisitos, projeto, construção de programas, teste e manutenção. As ferramentas de engenharia de software fornecem apoio automatizado ou semi-automatizado tanto para o processo quanto para os métodos.

Somerville (2007, p.42) observa que embora existam muitos processos de software diferentes, algumas atividades fundamentais são comuns a todos eles, como:

- Especificação: a funcionalidade do software e as restrições sobre sua operação devem ser definidas;
- Projeto e implementação: o software que atenda à especificação deve ser produzido;
- Validação: o software deve ser validado para garantir que ele faça o que o cliente deseja;

- Evolução: O software deve evoluir para atender às necessidades mutáveis do cliente.

Assim, um processo é considerado importante principalmente porque fornece estabilidade, controle e organização para a atividade de desenvolvimento de software, que pode tornar-se caótica caso não seja bem acompanhada. Para tornar o desenvolvimento de software uma atividade mais efetiva, é importante definir modelos de processo de software.

Atualmente existe um grande número de processos de desenvolvimento software e esses processos seguem os modelos propostos pela literatura. As seções seguintes apresentam os principais modelos de processo existentes e três processos de desenvolvimento de software, que foram considerados importantes para o entendimento geral do problema descrito nessa dissertação. As seções a seguir descrevem sucintamente os modelos e processos fundamentando-se em Pressman (2006), Somerville (2007) e Paula (2003).

2.2 Modelos de Processo de Desenvolvimento

Em um processo de desenvolvimento de software o ponto de partida para a arquitetura de um processo é a escolha de um modelo de processo de software. Dependendo do tipo de projeto a ser desenvolvido, modelos diferentes serão utilizados. Segundo Pressman (2006, p. 19), independentemente do modelo de processo selecionado, os engenheiros de software têm, tradicionalmente, escolhido um arcabouço genérico que inclui as seguintes atividades:

- Comunicação: envolve alta comunicação e colaboração com o cliente e outros interessados e abrange o levantamento de requisitos e outras atividades relacionadas. Procura descobrir, definir, especificar e documentar as funcionalidades gerais do software;
- Planejamento: estabelece um plano para o trabalho de desenvolvimento do software, ou seja, descreve as tarefas técnicas a serem conduzidas,

os riscos do projeto, os recursos necessários, os artefatos a serem produzidos e um cronograma de trabalho;

- Modelagem: cria os modelos que permitem entender melhor os requisitos do software e o projeto que vai satisfazer esses requisitos. É a representação de engenharia do software que vai ser construído;
- Construção: combina a geração do código-fonte, que deve implementar as funcionalidades especificadas, e os testes necessários para descobrir os erros na função, no comportamento e no desempenho do software;
- Implantação: entrega do software ao cliente, que avalia o produto e fornece *feedback* com base na avaliação.

As atividades podem ser organizadas em seqüência ou intercaladas dependendo do tipo de software, pessoas e estruturas organizacionais envolvidas. A forma de organização das atividades pode ser definida segundo os modelos apresentados a seguir.

2.2.1 Modelo em Cascata

O Modelo em Cascata, proposto em 1970, é também conhecido como ciclo de vida clássico onde as fases definidas para o desenvolvimento do software são sistematicamente seguidas de maneira seqüencial. Essa abordagem começa pela especificação de requisitos pelo cliente e progride ao longo do planejamento, modelagem, construção, implantação e manutenção progressiva do software, como apresentado na Figura 2.2.1.

Segundo Pressman (2006, p.39), o modelo em cascata é o mais antigo, no entanto, sua eficácia é questionável devido alguns problemas levantados quando o modelo é aplicado. Projetos reais raramente seguem o fluxo seqüencial que o modelo propõe. Neste modelo, existe a necessidade de se estabelecer todos os requisitos na fase inicial, fato que em geral é difícil tanto para o cliente quanto para o desenvolvedor, já que os requisitos mudam constantemente. Outro problema é a demora para apresentação de uma versão executável do software, que só estará disponível para o cliente no final do projeto.

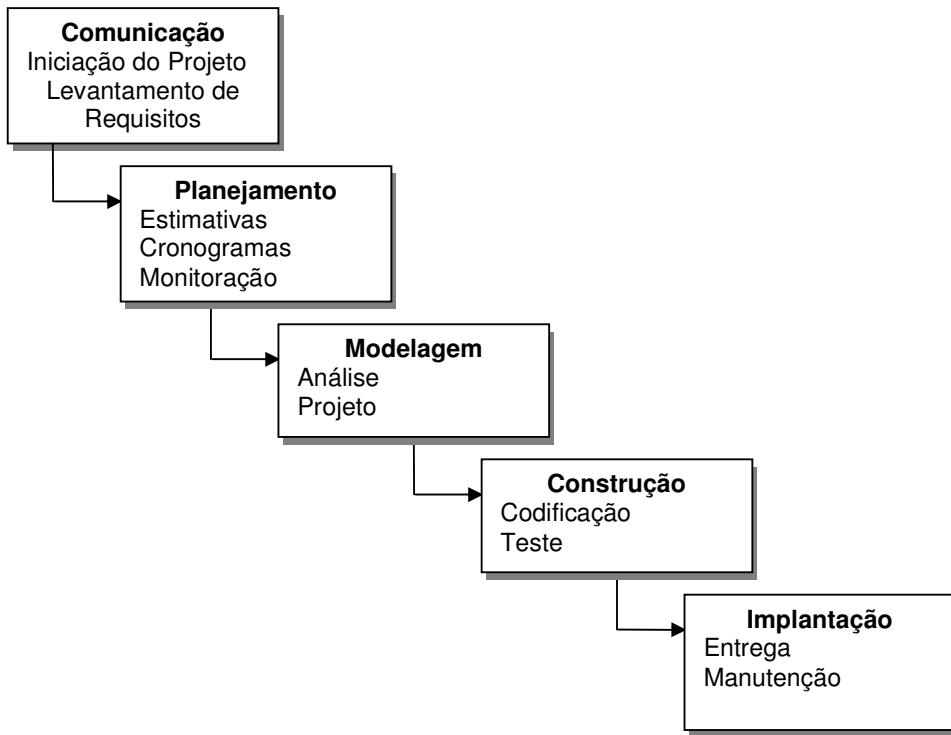


Figura 2.2.1 – Modelo em Cascata (Fonte: Pressman, 2006).

2.2.2 Modelos Incrementais

Esses modelos assumem que o software desenvolvido pode sempre crescer e agregar novas funcionalidades, sendo que cada uma dessas funcionalidades, ou o conjunto delas, será desenvolvido por um incremento e esse incremento segue todas as etapas descritas no modelo em cascata.

O primeiro incremento deste modelo é chamado de núcleo do produto, pois nele estarão os requisitos básicos do sistema. Os próximos incrementos desenvolvidos agregarão os requisitos do núcleo do produto e dos incrementos anteriores já desenvolvidos. Ao final de cada incremento, é gerada uma versão simplificada do produto final, que é apresentada ao cliente. Este modelo é um melhoramento do modelo em cascata, pois permite a alteração dos requisitos durante o desenvolvimento do software. A Figura 2.2.2 apresenta o modelo incremental.

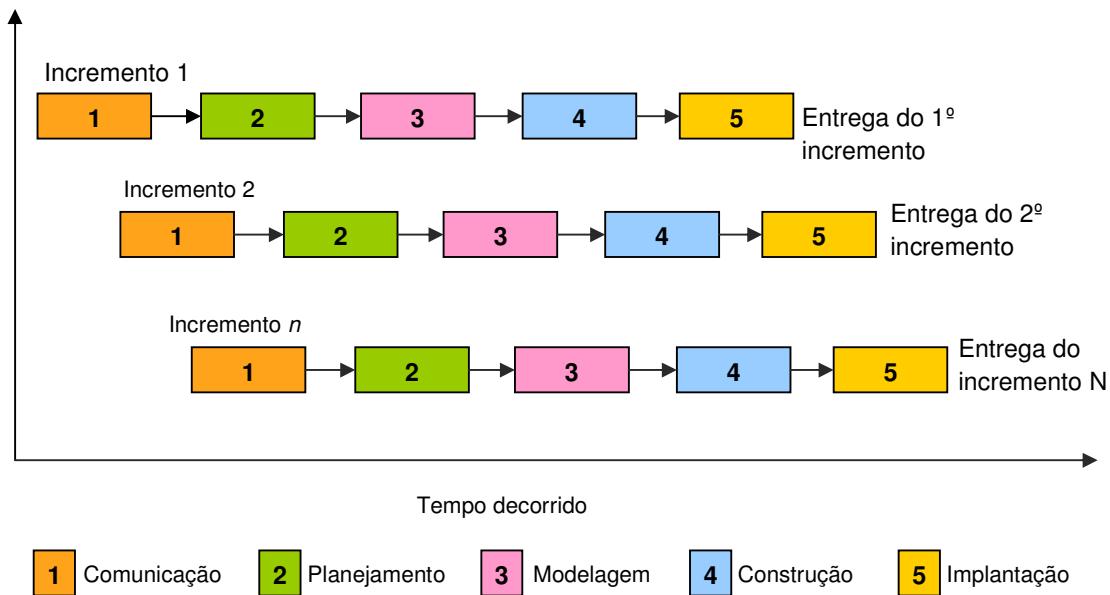


Figura 2.2.2 – Modelo Incremental (Fonte: Pressman, 2006).

Modelo RAD

O Modelo RAD (Rapid Application Development) é um exemplo de modelo incremental, considerado uma adaptação para projetos curtos, geralmente com prazo máximo de 90 dias. Sua principal característica é que o produto de software seja desenvolvido em componentes, pois a reutilização de código permite que a equipe de desenvolvimento possa desenvolver um sistema completamente funcional em pouco tempo.

Este modelo de desenvolvimento tem aplicação nos casos em que o sistema pode ser modularizado. Cada uma das equipes RAD de desenvolvimento fica responsável por um módulo, enquanto que outras equipes desenvolvem outros módulos, de forma concorrente. Ao final da construção de cada módulo há uma integração entre eles. Este ciclo se repete até que o software esteja completamente pronto. Este modelo possui cinco fases mostradas na Figura 2.2.3.

A *comunicação* trabalha para entender os problemas do negócio. O *planejamento* para gerenciar o trabalho paralelo das equipes. A fase de

modelagem abrange a modelagem do negócio, a qual visa à organização dos requisitos do sistema a ser construído. A modelagem dos dados identifica as características e relações entre objetos de dados, e na modelagem do processo, os objetos de dados definidos anteriormente são transformados para construir o fluxo de informação necessário para implementar uma função do negócio. A fase de *construção* enfatiza o uso de componentes de software preexistentes e a aplicação de geração automática de código. Finalmente a fase de *implantação* faz a integração entre os módulos e recolhe um feedback de utilização do sistema criado.

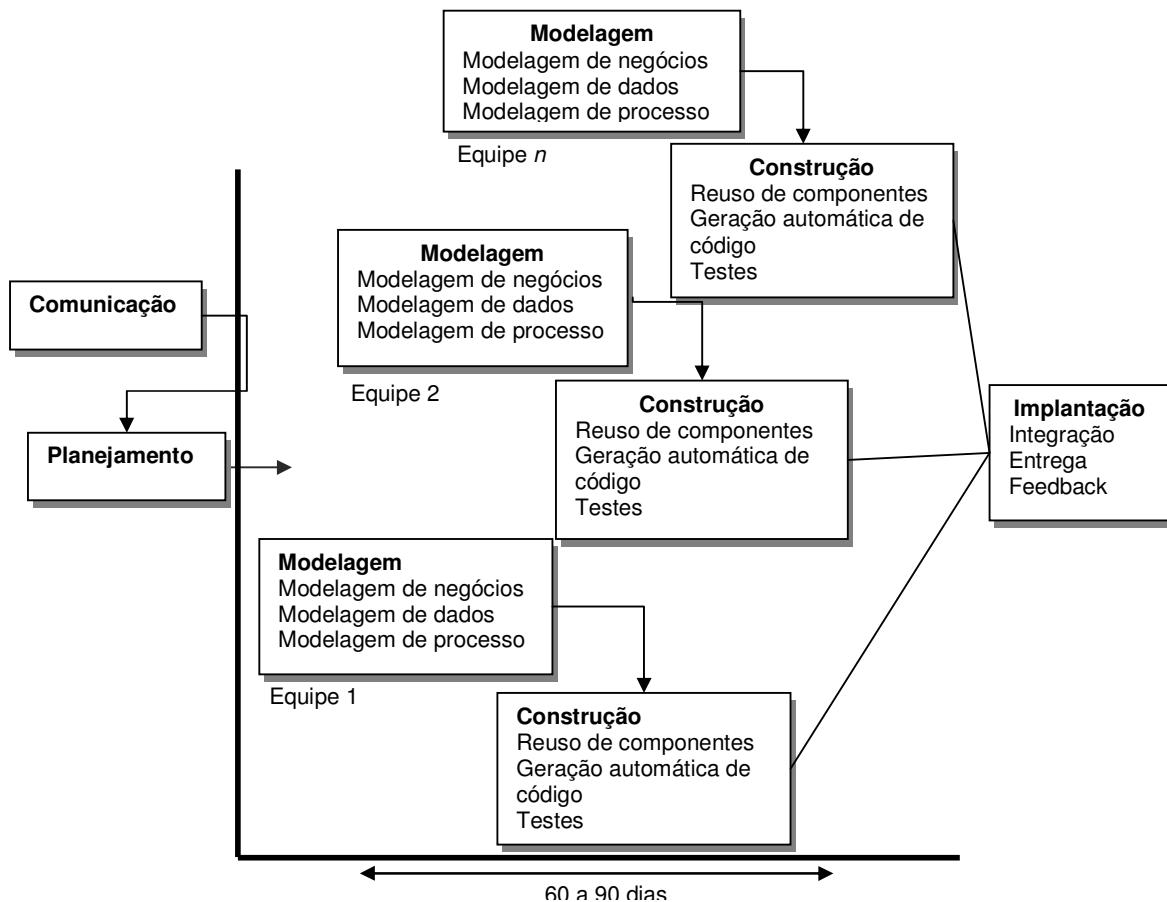


Figura 2.2.3 – Modelo RAD (Fonte: Adaptado de Pressman, 2006).

2.2.3 Modelos Evolucionários

Os Modelos Evolucionários baseiam-se na idéia de desenvolvimento de uma implementação inicial, expondo o resultado ao cliente e refinando esse resultado por meio de várias versões, para que então seja desenvolvido o sistema adequado.

Os modelos servem como um importante mecanismo para explorar os requisitos, uma vez que o desenvolvimento começa com as partes do sistema compreendidas e evolui por meio da adição de novas características propostas pelos clientes. Diferentemente do Modelo Incremental, as versões não necessariamente são produtos operacionais. Segundo Pressman (2006, p.47), partindo do ponto de vista da engenharia e do gerenciamento, a abordagem evolucionária traz algumas preocupações: o planejamento é incerto (não existe um método para calcular com precisão o número de ciclos que serão necessários) e se a evolução de desenvolvimento for muito rápida, o processo pode se transformar em caos (algumas atividades importantes do processo podem ser desconsideradas, gerando problemas na construção e manutenção do software).

São exemplos de Modelos Evolucionários: o modelo de prototipagem e o modelo espiral.

Modelo de Prototipagem

A prototipagem é um modelo de processo que possibilita ao desenvolvedor criar um modelo do software que deve ser construído. Assim, uma prévia avaliação tanto do cliente, quanto do desenvolvedor pode ser feita. Este modelo pode ser usado como um modelo de processo independente, ou como uma técnica, que pode ser implantada no contexto de qualquer um dos modelos de processo existentes. A vantagem da prototipagem é auxiliar o engenheiro de software e o cliente, a entenderem melhor o que deve ser construído, quando os requisitos estão confusos.

Este modelo começa pela comunicação. Os envolvidos no projeto identificam os objetivos gerais do software e as necessidades conhecidas. Em seguida

uma iteração é planejada e o protótipo é modelado de forma rápida. Após a modelagem; o protótipo é criado, implantado e depois avaliado pelo usuário. Esta avaliação é usada para refinar os requisitos e fornecer um melhor entendimento do problema.

Após a identificação de todos os requisitos, o protótipo deve então ser descartado, já que não foi construído observando-se critérios para a garantia da qualidade do produto. Em seguida, o produto final é construído, com foco na qualidade. A Figura 2.2.4 exibe o ciclo de desenvolvimento do modelo de prototipagem ressaltando a sua forma cíclica.

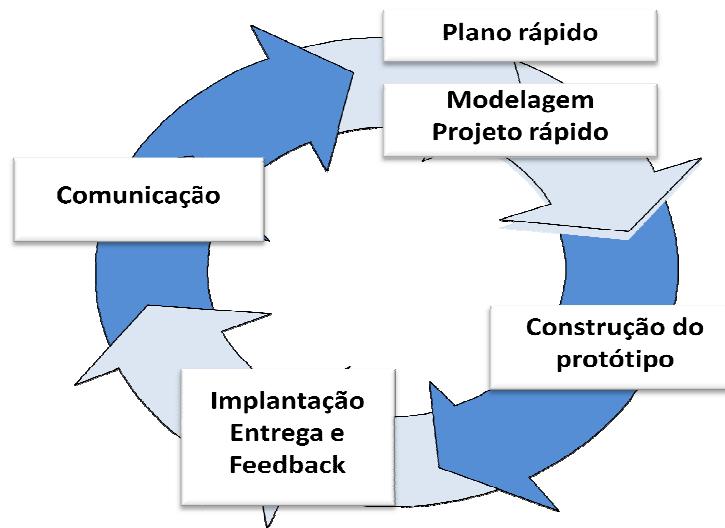


Figura 2.2.4 – Modelo de Prototipagem (Fonte: Pressman, 2006).

Modelo Espiral

O Modelo Espiral é um modelo iterativo, como o Modelo de Prototipagem, e sistemático como o Modelo em Cascata. Essas características facilitam com que sejam lançadas versões utilizáveis do projeto ao final de cada iteração do modelo, similar ao modelo incremental.

Esse modelo se subdivide em regiões que abrigam conjuntos de tarefas que crescem proporcionalmente ao risco e ao tamanho do projeto. São elas: Comunicação com o Cliente, a fim de manter contato com o cliente, adquirindo informações úteis ao sistema; Planejamento, que definirá custo e tempo de desenvolvimento, além de realizar uma análise de risco, onde os riscos da

implementação são avaliados para que o desenvolvimento não se torne inviável na próxima etapa do modelo; Modelagem, responsável pela criação da representação conceitual da aplicação; Construção, onde acontecem a implementação e testes; Implantação, responsável pela instalação, suporte e feedback do incremento desenvolvido.

Iniciado o processo, ele passa por todas as regiões e depois volta à primeira, executando novamente as regiões seguindo uma espiral até que se tenha o produto desejado, como mostra a Figura 2.2.5

Uma forte característica desse modelo, que o difere de outros modelos, é o fato de acompanhar toda a vida do software, mesmo depois da entrega ao cliente. Este modelo é considerado o mais realístico possível, pois assume que usuários, analistas e desenvolvedores adquirem maior conhecimento sobre o projeto com o decorrer do tempo. As falhas do software são identificadas e corrigidas, impedindo que se propaguem para as próximas iterações do ciclo de desenvolvimento do software.

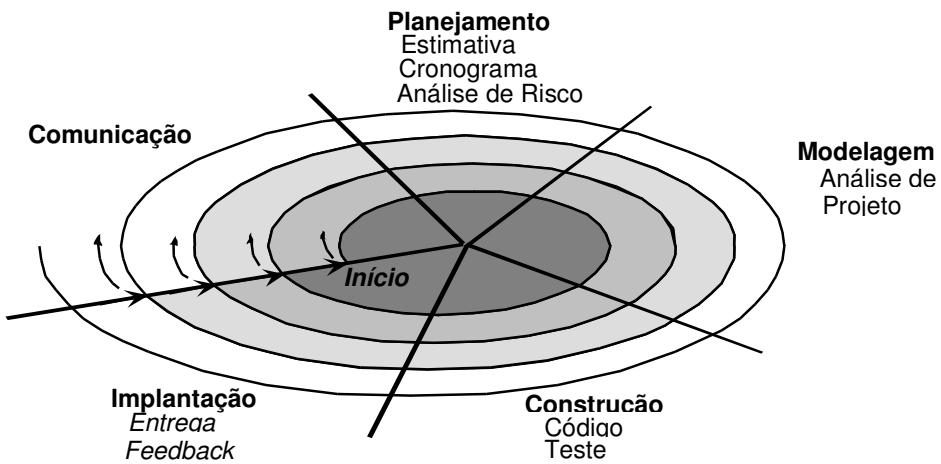


Figura 2.2.5 – Modelo Espiral (Fonte: Pressman, 2006).

2.3 Processos de Desenvolvimento de Software

Com base nos conceitos propostos pelos modelos de processo, vários processos de desenvolvimento de software são criados para atender as diversas exigências do contexto em que são aplicados. Nesta seção são apresentados três processos de desenvolvimento com algumas características semelhantes, como as de serem leves e adaptativos, ao invés de exigirem

muitos artefatos, criando uma atmosfera pesada (burocrática), e de serem preditivos, ao tentar planejar e prever as atividades em detalhes por um longo intervalo no ciclo de desenvolvimento.

Processo Unificado (PU)

O Processo Unificado é um processo de desenvolvimento de software que descreve uma abordagem para a construção, implantação e manutenção de projetos orientado a objetos. Ele foi proposto por Ivar Jacobson, James Rumbaugh e Grady Booch e utiliza a Linguagem de Modelagem Unificada (UML – Unified Modeling Language). Essa linguagem serve como notação para uma série de modelos que compõem os principais resultados das atividades deste processo (Jacobson, 1999).

Segundo Larman (2004, p. 42), a idéia central a ser apreciada e praticada no PU é a de usar iterações curtas com duração fixa, em um processo de desenvolvimento iterativo e adaptativo. Algumas das melhores práticas e conceitos-chave adotados pelo PU são:

- Enfrentar os problemas que envolvem altos riscos e alto valor já nas iterações iniciais;
- Envolver continuamente os usuários na avaliação, na realimentação e nos requisitos;
- Construir uma arquitetura central coesa nas iterações iniciais;
- Verificar continuamente a qualidade; fazer testes logo de início, com freqüência e em situações realísticas;
- Aplicar casos de uso;
- Modelar visualmente o software (com a UML);
- Gerenciar requisitos cuidadosamente;
- Usar solicitações de mudança e praticar a gerência de configuração.

O Processo Unificado promove, principalmente, o desenvolvimento iterativo. Nesta abordagem, o desenvolvimento é organizado em uma série de mini-projetos, de duração fixa, chamados iterações. O produto de cada iteração é um sistema testado, integrado e executável. Cada iteração inclui suas próprias atividades de análise de requisitos, projeto, implementação e teste. Entre as

vantagens dessa abordagem pode-se citar (1) a mitigação precoce, em vez de tardia, de riscos, (2) o progresso visível desde o início e (3) a complexidade é administrada, a equipe não é sobrecarregada pela "paralisia da análise" ou por passos muito longos e complexos;

Como mostra a Figura 2.3.1 o PU organiza o trabalho de cada iteração em quatro fases principais e Larman (2004, p. 43) as descreve como a seguir:

1. Concepção – visão aproximada, regras de negócio, escopo e estimativas vagas. (estudo de viabilidade);
2. Elaboração – visão refinada, implementação iterativa da arquitetura central, resolução dos riscos, identificação da maioria dos requisitos e do escopo e estimativas mais realistas;
3. Construção – implementação iterativa dos elementos restantes e preparação para a implantação;
4. Transição – testes beta e implantação;

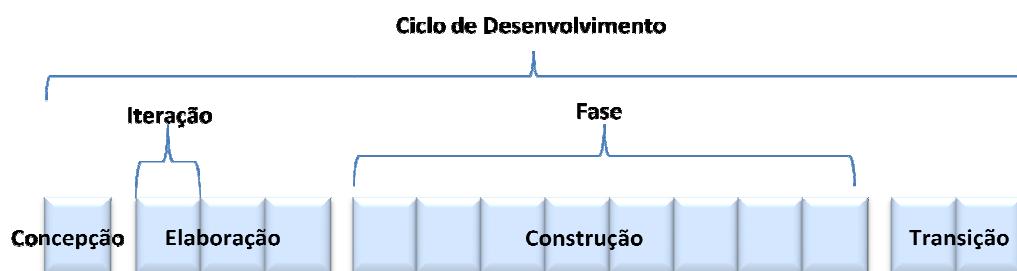


Figura 2.3.1 – Termos relativos ao cronograma de um PU (Fonte: Larman, 2004).

Em cada fase devem ser executadas diferentes atividades, produzindo artefatos específicos. O Processo Unificado trata as atividades de trabalho como disciplinas e define artefato como qualquer produto gerado durante as atividades do processo: documentos textuais, modelos, código, gráficos, casos de teste e manuais são exemplos de artefatos (Larman, 2004). Existem várias disciplinas no PU, como, por exemplo: Modelagem de Negócio, Requisitos, Projeto, Teste, Implantação, Gestão de Mudança e Configuração, Gestão de Projeto e Ambiente. A Figura 2.3.2 ilustra os artefatos mais importantes, produzidos em consequência, das quatro fases técnicas do PU, de acordo com Pressman (2006, p.54).

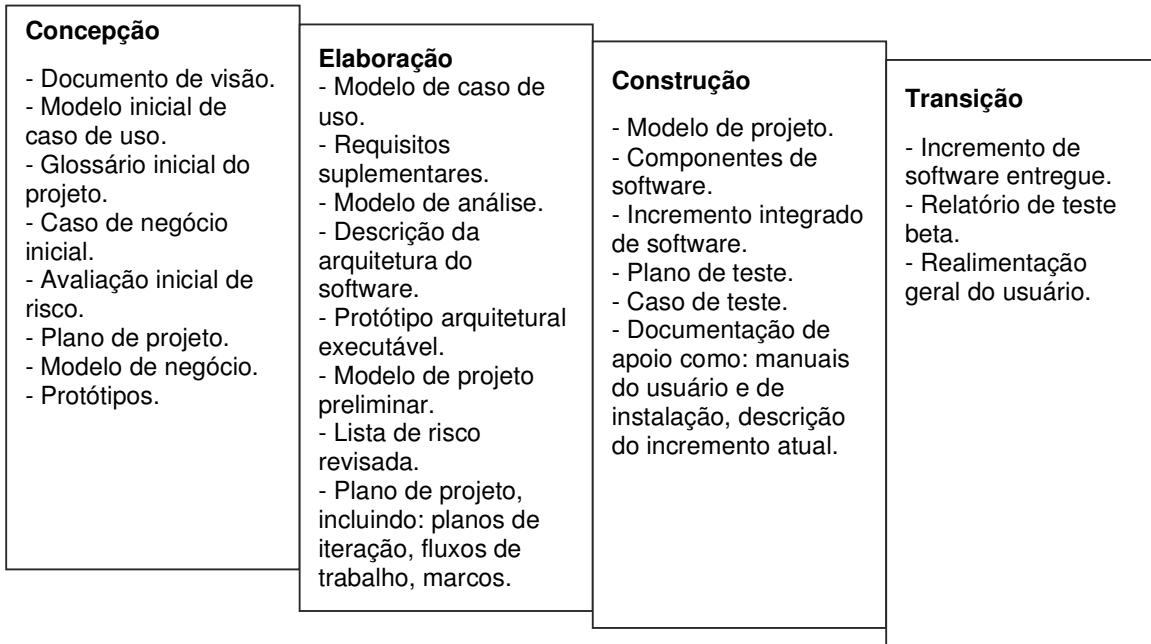


Figura 2.3.2 – Artefatos mais importantes produzidos durante as fases do PU (Fonte: Pressman, 2006).

Outro ponto a ser considerado sobre o PU é que algumas das práticas e dos princípios do PU são invariantes, como o desenvolvimento iterativo e orientado ao controle de riscos, bem como a retificação contínua da qualidade. Entretanto, um aspecto-chave do PU é que todas as atividades e os artefatos são opcionais, tornando o processo leve e adaptativo.

Extreme Programming (XP)

Extreme Programming é um processo de desenvolvimento ágil. Segundo Beck (2000), os processos ágeis aplicam-se com especial relevância em pequenos projetos ou projetos com equipes de trabalho co-localizadas. Apresentam uma visão semelhante sobre as boas práticas necessárias ao desenvolvimento de software e a obtenção da qualidade, como, por exemplo, o desenvolvimento iterativo, a preocupação com os requisitos e o envolvimento dos usuários finais no processo.

Segundo Astels *et al.* (2002), as práticas do XP são criadas para funcionar juntas e fornecer mais valor do que cada uma poderia fornecer individualmente. O XP apresenta uma grande preocupação com relação a alteração constante

dos requisitos do sistema. Sobre esta questão, a XP é enfática; o desenvolvedor deve permitir que o projeto seja flexível. Em outras palavras, o desenvolvedor deve aceitar as alterações e não lutar contra elas.

A Extreme Programming define 13 princípios básicos, que devem ser adotados em conjunto por qualquer projeto que se baseie nesta técnica. Estes 13 princípios são mostrados na Tabela 2.1, juntamente com uma explicação resumida.

Tabela 2.1 – Princípios da XP (Adaptado de Astels *et al.*, (2002).

Princípio	Descrição resumida
1. Cliente faz parte da equipe de desenvolvimento	O cliente deve participar do desenvolvimento, fornecendo o feedback necessário para a correção precoce de falhas.
2. Uso de metáforas	Deve-se utilizar metáforas para a definição de termos complexos, para permitir um conhecimento uniforme de toda a equipe.
3. Planejamento	O projeto deve ser constantemente planejado para permitir uma real identificação do seu progresso.
4. Reuniões curtas	As reuniões devem ser curtas, de preferência sem o uso de cadeiras, para que se discuta os termos estritamente relevantes.
5. Teste contínuo	Deve-se efetuar testes antes mesmo da implementação de um código. Este teste precoce permite um controle otimizado sobre as falhas de um projeto.
6. Simplicidade	O projeto deve ser o mais simples possível.
7. Programação em Pares	A programação deve ser feita em pares. Enquanto um programador tem um foco no código que está desenvolvendo, o observador tem uma visão macro do processo, permitindo a identificação facilitada de falhas.
8. Padrão de Codificação	Todos as pessoas envolvidas no processo devem definir e seguir um mesmo padrão de codificação.
9. Propriedade coletiva sobre o código-fonte	Todas as pessoas devem ter acesso livre ao código-fonte, de modo que possa acrescentar melhorias e corrigir falhas identificadas.
10. Integração contínua	Deve-se efetuar uma contínua integração do novo código gerado, para que seja possível identificar precocemente falhas na integração deste novo código.
11. Refatoração contínua	O desenvolvedor deve alterar a estrutura interna sempre, de forma a melhorar a sua performance, sem alterar o funcionamento externo do código.
12. Concepção de pequenas versões	Devem ser concebidas pequenas versões de código de cada vez, para que os usuários tenham rapidamente um contato com uma versão executável do software e consequentemente consigam gerar o feedback necessário.
13. Jornada de Trabalho	A equipe de desenvolvimento deve ter uma jornada de trabalho flexível e leve, para que não exista fadiga e estresse dos seus membros.

Processo de Desenvolvimento Específico para Software Científico (PESC)

O PESC é um processo de desenvolvimento específico para software científicos. Ele considera o termo software científico como, “software desenvolvido por pesquisadores em seus projetos de pesquisa científica. A grande maioria destes projetos é de natureza acadêmica, ou seja, projetos de pesquisa de iniciação científica, mestrado, doutorado, pós-doutorado, dentre outros, que necessitam da construção de software para auxiliar estas pesquisas” (Purri et al., 2006).

O PESC é fruto do trabalho de dois alunos de mestrado do curso de Modelagem Matemática e Computacional do CEFET/MG. Purri (2006), em sua dissertação de mestrado, formulou e validou 12 hipóteses acerca do desenvolvimento de software que é praticado atualmente no meio acadêmico. Para tanto, foi realizada uma pesquisa semiquantitativa, através da implementação de um questionário Web (disponibilizado na internet), onde pesquisadores das mais diversas áreas caracterizaram como desenvolvem seus softwares e o que julgam importante em seu desenvolvimento. Tal questionário foi fundamental para que as atividades propostas por Humphrey (1995) fossem realizadas. Com base nas respostas, Purri (2006), apresentou as seguintes diretrizes iniciais que o processo deveria contemplar:

1. Ciclo iterativo e incremental;
2. Base no Processo Unificado;
3. Deve ser um processo simples;
4. Deve ser voltado para uma equipe pequena de desenvolvimento;
5. Deve utilizar a linguagem UML como base;
6. Deve permitir o gerenciamento de código aberto e concorrente;
7. A utilização dos artefatos deve ser opcional.

Assim como o processo apresenta algumas características essenciais, os artefatos também deverão possuir algumas peculiaridades. Segundo Purri (2006), as características são as seguintes:

1. Descrição das funcionalidades gerais do software;
2. Projeto arquitetural;
3. Definição do tipo de licença do software;

4. Descrição da relevância científica do software;
5. Documentação do código-fonte e regras de codificação;
6. Descrição das restrições e viabilidade do projeto;
7. Levantamento completo e especificação detalhada dos requisitos;
8. Diagramas de casos de uso, descrição e especificação;
9. Diagrama de classes completo;
10. Citação das referências científicas nas quais a implementação irá se basear;
11. Descrição dos algoritmos complexos;
12. Documentação dos componentes reutilizados na construção do software;
13. Documentação de módulos lógicos reutilizáveis em outros projetos;
14. Plano e descrição dos testes;
15. Relatórios de erros encontrados e causas;
16. Registro das falhas e dos sucessos do projeto de software;
17. Manual do usuário do software.

Tomando como ponto de partida as diretrizes citadas acima, Pereira Jr (2007) propôs um processo de desenvolvimento inicial. Esse processo sofreu fortes influências do Processo Unificado e do Extreme Programming, adaptando os conceitos já consagrados do ciclo de vida iterativo e incremental do PU e a padronização e propriedade coletiva do código-fonte, a simplicidade do processo e a integração contínua propostos pelo XP.

O ciclo de desenvolvimento utilizado no PESC é definido como iterativo e incremental, que é empregado no Processo Unificado e na XP. Como o ciclo de vida é evolutivo, ele é composto por iterações. Ao final de cada iteração, tem-se uma parte executável do sistema, conhecido como liberação ou versão. Em cada uma das iterações são contempladas as seguintes fases:

- Planejamento – Cada uma das iterações deve ser extremamente reduzida. Isto permite um controle muito maior do desenvolvimento, já que se pensa apenas em uma pequena parte do projeto de cada vez. Cada uma das iterações deve ser planejada de modo que seja simples,

mas que não deixe de lado nenhum detalhe importante ao desenvolvimento.

- Desenvolvimento – Com base no planejamento rápido feito anteriormente, é desenvolvida a versão da iteração em questão, observando-se os padrões de codificação. É gerada então uma versão em cada iteração do ciclo de vida.
- Testes – Após o seu desenvolvimento, cada versão deve passar pelos testes de unidade e integração. Os testes de unidade servem para garantir que esta versão esteja funcionando de acordo com o planejamento efetuado. Após a aprovação desta versão no teste de unidade, deve-se proceder com o teste de integração, onde será verificada a compatibilidade deste novo código com o que já existia anteriormente.
- Implantação – Depois de passar pelos testes de unidade e integração, a nova versão deve ser incorporada ao código-fonte oficial.

A Figura 2.3.4 mostra o ciclo de desenvolvimento proposto, com seus artefatos definidos para cada fase do desenvolvimento. Logo em seguida, uma descrição dos artefatos é apresentada de acordo com Pereira Jr (2007).



Figura 2.3.4 – Ciclo de desenvolvimento proposto pelo PESC. (Fonte: Pereira Jr, 2007).

- Controle Geral de Desenvolvimento – este artefato é conhecido como artefato guarda-chuva. Sendo assim, este artefato é específico para o controle geral do desenvolvimento do software.
- Requisitos e Escopo da Versão – este artefato é utilizado e/ou modificado na fase de planejamento, que é a primeira etapa do ciclo de desenvolvimento do PESC. Neste artefato, o desenvolvedor deve registrar o que foi planejado para aquela iteração.
- Detalhamento dos Casos de Uso – este artefato é modificado ainda na fase de planejamento. Ele é confeccionado com base nas informações coletadas e documentadas no artefato de Controle Geral de Desenvolvimento. Este documento de detalhamento possui os diagramas de caso de uso e suas especificações.
- Plano de Codificação da Versão – este artefato é modificado na fase de planejamento e deve ser consultado e, se necessário, modificado na fase de desenvolvimento. Esse artefato é particularmente importante para o sucesso do desenvolvimento, pois define todos os detalhes pertinentes ao desenvolvimento do código-fonte propriamente dito, incluindo os diagramas de classes.
- Plano de Testes – este artefato é utilizado na fase de testes. O código-fonte gerado na fase anterior do ciclo de vida servirá de base para a aplicação dos testes que serão documentados neste artefato.
- Registro das Falhas e Sucessos – este artefato é utilizado ao final da fase de implantação do ciclo de vida do PESC. Após uma iteração completa da evolução do desenvolvimento, o pesquisador terá enfrentado dificuldades e possíveis falhas, bem como terá conseguido sucessos no seu desenvolvimento. Este artefato permite que o desenvolvedor possa registrar as falhas e sucessos obtidos no desenvolvimento de cada versão.

O PESC, como todo processo de desenvolvimento proposto ou customizado seguindo a metodologia de Humphrey (1995), deve executar o sétimo passo dessa metodologia, que consiste em validar o processo inicial. Neste contexto, o PESC sentiu a ausência de técnicas de validação de processo bem

documentadas e serviu para confirmar a caracterização e a pergunta problema desta pesquisa.

2.4 Questões sobre avaliação e validação em processos de desenvolvimento de software

Esta seção discute questões relevantes acerca de avaliação e validação de processos de desenvolvimento de software. Essa discussão leva em consideração os pontos levantados por Cook e Visconti (1993), que diferenciam as atividades de avaliação das atividades de validação. Estas definições são importantes, pois esta pesquisa tem o objetivo de criar uma técnica de validação e não de avaliação.

Avaliação

O propósito dos procedimentos de avaliação de processos de software é determinar onde os padrões organizacionais do processo se familiarizam com um modelo pré-determinado. A idéia da avaliação é determinar o estado atual das práticas da equipe de desenvolvimento. Essa determinação é a base para a criação de um plano para melhorias nos procedimentos, métodos e ferramentas de engenharia de software. O objetivo principal é melhorar a qualidade do produto e a produtividade das pessoas que estão usufruindo do processo. Portanto, uma avaliação de processo ajuda uma organização a caracterizar e entender o estado atual do seu processo de software e fornece informações e recomendações para facilitar as melhorias.

A avaliação é realizada através de instrumentos que podem ser um questionário de avaliação ou um especialista (ou time de especialistas) que avalia o estado das práticas, fazendo uso de entrevistas, observações, ou mesmo outros métodos subjetivos; ou uma mistura deles. O instrumento de avaliação deve ser aplicado em um contexto próprio e usado competentemente.

As avaliações são realizadas seguindo metodologias, que têm como idéia principal identificar o status ou o nível de maturidade do processo de software, estabelecendo assim, as questões mais críticas a serem melhoradas. Uma vez definido seu nível na estrutura de maturidade, a organização pode concentrar-se nos itens que os permitem avançar para outro nível. Logo, o resultado da avaliação é um rótulo com o nível de maturidade do processo de software e uma lista de ações recomendadas para melhorias. Em seguida, uma equipe de gerência estabelece uma estrutura para implementar as ações prioritárias necessárias para melhorar o processo.

Estes níveis são conhecidos na engenharia de software como níveis de maturidade, onde cada nível indica um procedimento gradativo de amadurecimento estabelecido pelo Instituto de Engenharia de Software (SEI – Software Engineering Institute). Para evitar que inconsistências de termos de nomenclatura, processo de avaliação e modo de implementação fossem gerados, a Organização Internacional de padrões (ISO – International Organization Standardization) em conjunto com o SEI criaram o padrão de maturidade (Wallin, 2002).

A vantagem de se usar o padrão ISO é que as empresas, de modo geral, procuram prestadores de serviço que sejam certificados por esta organização internacional de padrões e excluem fornecedores que não se preocupam em certificar-se. Como resultado, empresas devem ter explícita a certificação ISO, a fim de fornecer suas capacidades.

Validação

A avaliação é inevitável em qualquer plano de melhoria. É absolutamente necessário saber a posição atual da organização avaliada para identificar os possíveis problemas, ser capaz de propor ações de melhorias e medir os seus efeitos. O termo validação é considerado essencial e acaba sendo executada antes da avaliação, mesmo que de forma *Ad-hoc*¹.

¹ Expressão em latim que quer dizer “com este objetivo”. Geralmente significa uma solução designada para um problema ou tarefa específica, que não pode ser aplicada em outros casos. (Wikipedia, 2008).

Ejiogu (1993) afirma que a validação de um modelo significa provar que as suas definições, atributos de comportamento e os postulados de função matemáticas são corretos. Por necessidade um modelo só incorpora o que é acreditado ser importante e exclui o que não é considerado importante. Ejiogu (1993), ainda afirma que a validação pode ser feita por uma análise de correlação que tem como objetivo, calcular o grau de relação ou concordância entre dois ou mais objetivos do modelo.

Moor e Delugach (2006) apresentam uma definição aplicada a processos de desenvolvimento de software. Segundo estes autores, para um processo de desenvolvimento ser válido, sua execução deve ser coincidente com o modelo proposto pelo processo. Ou seja, o modelo formal deve ser executado como proposto, caso contrário, o modelo possui pontos desnecessários ou não abrangentes o suficiente.

Cook e Wolf (1999) acreditam que a validação de um modelo formal de um processo de software é uma atividade para desenvolver evidências convincentes que o modelo pode ser seguido, isto é, o modelo realmente apresenta as etapas, atividades e procedimentos de interesse dos envolvidos no desenvolvimento de software.

2.5 Análise das técnicas empregadas em processos de desenvolvimento de software

Visando a melhoria nos processos de desenvolvimento de software, vários trabalhos vêm sendo realizados com a idéia central em medir, analisar e propor mudanças nos processos de desenvolvimento. Alguns dos objetivos desses trabalhos são: (1) minimizar o número de defeitos durante todo o processo de desenvolvimento, (2) maximizar a eficácia e a confiabilidade do produto final e (3) dimensionar adequadamente o esforço dedicado a atividades do projeto. Nesta seção são apresentados dois trabalhos com o aspecto de melhoria de processos e dois trabalhos com o foco em validar os processos.

Em Chmura *et al.* (1990), é descrita uma técnica de análise de dados levantados a partir de alterações realizadas nos vários artefatos do software ao longo do processo de desenvolvimento. Segundo os autores, já existem estudos com este objetivo, mas eles tentam coletar dados durante todo o processo, o que gera um esforço adicional de 5 a 15% em um projeto. O método adotado por Chmura *et al.* (1990) analisa dados no fim de cada iteração do processo, aproveitando dados já coletados pela própria Gerência de Configuração do projeto.

Durante a análise, é gerado um documento conceitual, chamado de Formulário de Relatório de Alterações, o qual é capaz de descrever as alterações realizadas na especificação e na implementação do projeto de forma textual. Os formulários são analisados para garantir um grau de confiabilidade, evitando, por exemplo, duplicidades de relatórios, múltiplas informações em um mesmo formulário ou ausência de informações fundamentais, como as que indicam o tempo gasto e em que ponto do projeto as alterações ocorreram. Outra informação relevante é se as alterações são fruto de novas implementações ou correções.

Com os formulários armazenados, informações são levantadas e mostradas através de gráficos e tabelas. Com os gráficos são verificadas, por exemplo, se as alterações ocorreram durante o projeto, em funções de testes eficientes ou se as alterações são realizadas depois que o produto foi entregue.

Outra pesquisa aplicada a processos de desenvolvimento é apresentada por Garg *et al.* (1993). Esta pesquisa tem com objetivo criar um processo específico para a criação de Kits de Desenvolvimento de Software (SDK). Segundo os autores, atualmente o desenvolvimento de SDK é restrinido, pois as organizações de desenvolvimento de software precisam re-projetar seus processos de desenvolvimento. Com este intuito os autores buscam a criação de um processo e necessitam: (1) empiricamente validar o processo de software proposto e (2) monitorar e aperfeiçoar tal processo.

Garg *et al.* (1993), define SDK como um conjunto de ferramentas de desenvolvimento que permite os engenheiros de software criar aplicações

como, por exemplo, jogos eletrônicos ou sistemas operacionais, substituindo o grande esforço de codificação por operações de customização.

A fim de monitorar e aperfeiçoar tal processo, os estudos de Garg *et al.* (1993) buscam avaliar informações em cada iteração de um projeto, monitorando todas as etapas para que possa ser obtido um entendimento completo do processo sob estudo. Os dados são obtidos em entrevistas com os participantes, que são conduzidas para capturar a história do processo. Esta história é analisada para determinar os pontos fortes e fracos do processo e os resultados são apresentados a equipe de desenvolvimento. O histórico é capturado analisando quatro elementos, que são caracterizados por Garg *et al.* como: (1) Agentes – o pessoal envolvido no processo, assumindo seus diferentes papéis, (2) Tarefas Primárias (P-Tasks) – as atividades ou tarefas planejadas que foram executadas no processo para realizar os objetivos, (3) Tarefas de Articulação (A-Tasks) – tarefas realizadas para consertar erros nas P-Tasks e (4) Produtos – o resultado das P-Tasks e A-Tasks.

Com o histórico capturado, o objetivo passa a ser analisar as informações para entender o que aconteceu durante o processo, utilizando-se de: (1) Análise de Objetivos, (2) Análise de Tarefas e (3) Análises de Participantes, que são apresentadas em Garg *et al.* (1993). A primeira análise avalia se os objetivos dos agentes e das tarefas estão congruentes durante as etapas do processo. Como hipótese, se os objetivos dos agentes não estão alinhados com os objetivos da tarefa o trabalho não será executado com sucesso. Na Análise de Tarefa são avaliados os atributos e relacionamentos das tarefas, por exemplo, se a tarefa tem um único agente responsável, ou se a comunicação entre as tarefas concorrentes e interdependentes ocorrem com sucesso. Na última análise, as opiniões dos participantes e pontos importantes são levantadas buscando um *feedback* com o intuito de melhorar o processo.

Baseado nestas análises de histórico do processo, os autores relatam que vários problemas são encontrados e discutidos, entre eles o alinhamento dos objetivos, as responsabilidades das tarefas e a sincronização de tarefas concorrentes resultando em melhorias a partir da primeira iteração do processo

em um projeto. Outros trabalhos relacionados à melhoria de processos são apresentados em Ferreira e Moita (2008 a) e Ferreira e Moita (2008 b).

Com o foco em validar um processo de desenvolvimento de software, pode-se citar as pesquisas de Moor e Delugach (2006) e Cook e Wolf (1999). Seus trabalhos têm como ponto em comum a comparação entre a execução do processo com o modelo formal proposto, mas o fazem de formas distintas.

Moor e Delugach (2006) apontam que, embora os modelos formais de processo sejam importantes em fornecer uma orientação geral, eles são raramente seguidos à risca, ou seja, há uma diferença entre o modelo formal do processo, que é o modo normativo ou prescritivo, e as práticas, que são o que realmente os envolvidos no projeto executam. Práticas permitem que a equipe aprenda, inove, e lide com erros. Elas são, assim, uma parte essencial do desenvolvimento do software, especialmente em um ambiente ágil ou distribuído, onde as incertezas e os erros necessitam ser tratados o tempo todo.

Assim, a pesquisa destes autores foca em explorar as diferenças entre o modelo formal do processo e a prática, com o alvo de fornecer idéias úteis para melhorar ambos os processos de engenharia de software e seus resultados práticos, sendo essa comparação essencial para detectar, por exemplo, “Qual é o tamanho da diferença?”, “Quanto significativamente é a diferença?”, “Em quais passos o processo de desenvolvimento ocorre?” “De acordo com qual ponto de vista o processo está visível?” (Moor e Delugach, 2006).

Para que as diferenças sejam apontadas, Moor e Delugach (2006) adotam um formalismo baseados na teoria de gráfico conceitual, de forma que o modelo e a prática sejam representados e comparados. A Tabela 2.2 mostra o método criado por esses autores que tem como unidade básica de análise as atividades pertencentes ao processo.

Tabela 2.2 – Método de validação de processos baseados em atividades.
(Fonte: Moor e Delugach, 2006)

<ul style="list-style-type: none"> • Criar o padrão de atividades para o modelo • Criar dois modelos <ul style="list-style-type: none"> ○ Modelos podem ser o modelo prático e o modelo do processo ○ Identifique os passos dos modelos ○ Construa modelos instanciando ou especializando padrões de atividades gerais para cada passo • Compare os modelos <ul style="list-style-type: none"> ○ Identifique atividades nos modelos, e delimita passos em uma seqüência ○ Compare os passos entre os modelos ○ Crie a diferença de modo gráfico • Interprete as diferenças entre os modelos <ul style="list-style-type: none"> ○ Apresente as diferenças de forma gráfica aos profissionais envolvidos no processo ○ Adapte o processo e/ou a prática ○ Execute as alterações no programa

Cook e Wolf (1999) também compartilham da idéia de que quando os modelos de processo e as execuções do processo divergem, alguma coisa significante está acontecendo. Com essa idéia em mente, eles desenvolveram sua própria técnica para descobrir e medir as discrepâncias entre modelos e execuções, que é chamada pelos autores de Processo de Validação. Segundo Cook e Wolf, a validação serve a vários objetivos entre eles, garantir a confiança no modelo do processo formal aumentando a confiança nos resultados.

O fundamento que a técnica de Cook e Wolf utiliza é uma visão de processos como uma seqüência de ações executadas por agentes humanos ou autômatos, ou seja, usam um modelo baseado em eventos de ações de processo, onde um evento é usado para caracterizar o comportamento dinâmico de um processo em termos identificáveis. Esses eventos são tipados e podem ter atributos como, por exemplo, o tempo em que ocorreu o evento, itens como agentes e artefatos associados ao evento, os resultados tangíveis da ação, por exemplo, passar/falhar em uma revisão de projeto; erro/execução de uma compilação, e qualquer outra informação que dá o caráter à ocorrência específica daquele tipo de evento. Os eventos são extraídos, por exemplo, de sistemas de controle de versão e mensagens de compiladores. Os eventos não computacionais são registrados manualmente.

A Figura 2.5 mostra o modelo conceitual do *framework* implementado por Cook e Wolf (1999). Observe que há uma distinção entre o modelo formal do processo e a execução do processo, ou seja, existem dois universos de tipos de eventos. Um universo é o conjunto de tipos de evento associados com o modelo de um processo, enquanto o outro é o conjunto de tipos de evento associados com a execução do processo. O *framework* é capaz de armazenar, consultar e medir *Stream²* de Eventos, que os autores classificam como uma sequência de eventos capturados em uma atividade.

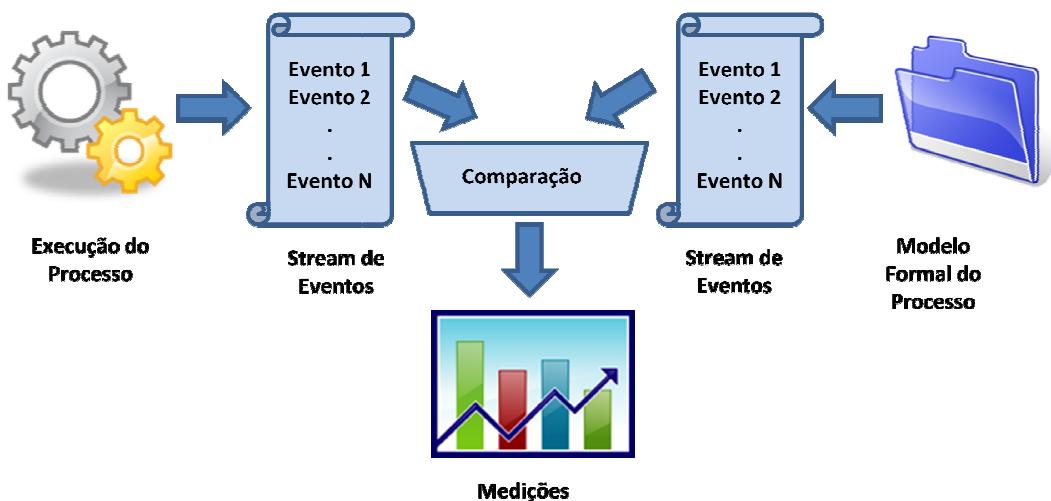


Figura 2.5 – Framework de comparação de eventos (Fonte: Cook e Wolf, 1999).

O método adotado por Cook e Wolf para medir as diferenças entre os streams é a Edição de *String*, onde o número de inserções, deleções e substituições simbólicas (*tokens*) necessárias para transformar uma string em outra são analisadas. Para mais detalhes sobre Edição de *String* consulte (Kruskal, 1983).

Uma investigação sobre validação de processos mostra a existência de poucos trabalhos nesta área. As poucas referências encontradas (Cook e Wolf, Moor e Delugach) adotam como princípio a comparação entre modelo e prática, mas o fazem de formas distintas, conforme mostrado acima, e não consideram questões como a eficiência e os resultados finais obtidos pelo processo. Preocupação com a adaptabilidade destas técnicas aos vários ambientes de desenvolvimento e a possibilidade de extensão das técnicas não estão

² Cook e Wolf (2006) implementam Streams através de vetores para armazenar os eventos capturados pelo framework de validação.

presentes na literatura pesquisada. Não foi encontrada documentação publicada comprovando ou demonstrado a aplicabilidade das técnicas de validação em uma quantidade relevante de ambientes reais.

Apesar disto, a proposta de comparar o modelo formal do processo e sua execução, como defendida por Cook e Wold (1999) e Moor e Delugach (2006), mostra-se capaz de produzir resultados interessantes para validar um processo. Contudo, a utilização de conceitos de teoria dos grafos ou edição de *string* pode inviabilizar a validação de processos de desenvolvimento ágeis, que pregam a flexibilidade e a facilidade de uso.

Processos como o PESC, apresentado na Seção 2.3, são criados com o intuito de prover qualidade no desenvolvimento de software sem a aplicação de grande quantidade de recursos humanos ou financeiros, e se a validação do processo não seguir esse intuito, ela pode inviabilizar a utilização do mesmo. Dessa forma, nesta dissertação é proposta uma nova abordagem para comparar o modelo formal com a execução do processo, adotando conceitos e técnicas de verificação e validação de software, para simplificar e dar flexibilidade a esta comparação.

CAPÍTULO 3

VERIFICAÇÃO E VALIDAÇÃO DE SOFTWARE

Visando diminuir a complexidade e a subjetividade ao validar processos de desenvolvimento, buscou-se embasamento teórico na validação de software. Este capítulo apresenta uma visão geral de Verificação e Validação, analisando os conceitos, diretrizes e técnicas aplicadas em software para a utilização dos mesmos em processos de desenvolvimento.

3.1 Conceitos de Verificação & Validação

Softwares são amplamente utilizados para resolver problemas e tomar decisões. Os usuários passam a acreditar e a trabalhar nos resultados apresentados por eles. Para isto, o software deve ser construído atendendo às especificações do projeto e o produto final deve servir às necessidades reais do usuário. A Verificação e a Validação, ou simplesmente V&V, têm respectivamente estes interesses (Oberkampf e Trucano, 2007).

Em uma definição formal, Pressman (2006) afirma que “Verificação se refere a um conjunto de atividades que garantem que o software implementa corretamente uma função específica e a Validação se refere a um conjunto de atividades diferentes que garante que o software construído corresponde aos requisitos do cliente”. Segundo o mesmo autor, a definição de V&V engloba muitas das atividades que são abrangidas pela Garantia da Qualidade de Software (SQA), como por exemplo: revisões técnicas formais, auditoria de qualidade e configuração, monitoramento de desempenho, estudo de viabilidade, revisão da documentação, entre outras.

Boehm (1981) faz a seguinte definição. “Verificação: Estamos construindo certo o produto? Validação: Estamos construindo o produto certo?”. Wallin (2002) diz

que o “propósito da verificação é garantir que o componente de software ou sistemas baseados nele cumpra suas especificações” e que o “propósito da validação é demonstrar que um componente de software ou um sistema customizado atinja o uso desejado em um ambiente desejado”.

Pode-se notar que as definições acerca de Verificação e Validação são semelhantes e servem ao mesmo propósito, mas a V&V pode ser executada de diversas maneiras dependendo do campo de pesquisa. Ramos *et al.* (1999) usa ferramentas de verificação baseados em métodos formais para detectar anomalias em bases de conhecimento de Sistemas Inteligentes. Wallin (2002) faz uso de técnicas de Inspeções de Software e Testes de Caixa Branca e Caixa Preta em sistemas comerciais. Pressman ressalta que:

[...] há uma forte divergência de opinião sobre que tipos de teste constituem validação. Algumas pessoas acreditam que todo teste é verificação e que validação é conduzida quando requisitos são revisados e aprovados, e posteriormente, pelo usuário, quando o sistema estiver operacional. Outras pessoas consideram teste unitário e de integração como verificação e testes de alta ordem como validação (PRESSMAN, 2006).

John e Steve (2000) destacam, em um estudo de caso realizado na Administração Nacional de Aeronáutica e Espaço Norte Americana (NASA), que o uso de V&V é um dos fatores para garantir a qualidade de software em sistemas desenvolvidos por meio do Desenvolvimento de Aplicações Rápidas (RAD). A questão é usufruir das vantagens desse desenvolvimento, como a alta produtividade e o baixo custo, sem que os problemas levantados no estudo comprometam tais vantagens. Neste caso, a V&V fornece análises úteis ao grupo de projeto em uma contínua tarefa para monitorar a co-evolução das ferramentas CASE usadas e analisa os diferentes códigos gerados de acordo com a versão das ferramentas.

Pode-se citar também o uso de V&V em Wallin (2002), o qual ressalta o sucesso e as vantagens do desenvolvimento de softwares baseado em componentes e diz que esta tecnologia requer, entre outras coisas, que o componente usado tenha alta funcionalidade, qualidade e que seja bem documentado, o que é obtido por organizações de software maduras com alta qualidade, fazendo uso de um processo que previnam problemas por meio da Verificação & Validação.

A V&V pode se relacionar com o processo de desenvolvimento de diversas maneiras. A literatura trata essa forma de relacionamento como paradigma. Cada paradigma dita quais atividades e em que momento elas serão aplicadas. A especificação dos documentos ou artefatos de entrada e a apresentação dos resultados também são definidas. A Figura 3.1 mostra o paradigma de V&V apresentado pelo Manual de Engenharia de Sistemas (NAS, 2006) e sumariza as atividades de V&V da seguinte forma:

1. Os Documentos de requisitos do sistema alimentam a validação. Durante as atividades de validação, uma tabela de validação é criada.
2. Com os requisitos validados, a Tabela de Validação torna-se a base para as atividades de Verificação. Nesse momento as atividades de planejamento da Verificação são iniciadas e o documento chamado de Planos de Verificação Principal (MVP) é criado e será atualizada durante todo o ciclo de desenvolvimento.
3. As técnicas de verificação são especificadas para analisar cada requisito descrito na Tabela de Validação. Neste momento, um documento chamado Matriz de Responsabilidade de Verificação é criado (VRTM).
4. Depois que as atividades de verificação são concluídas, o VRTM é atualizado e o Documento de Cumprimento de Verificação dos Requisitos (RVCD) é criado para registrar a realização dos esforços de verificação.

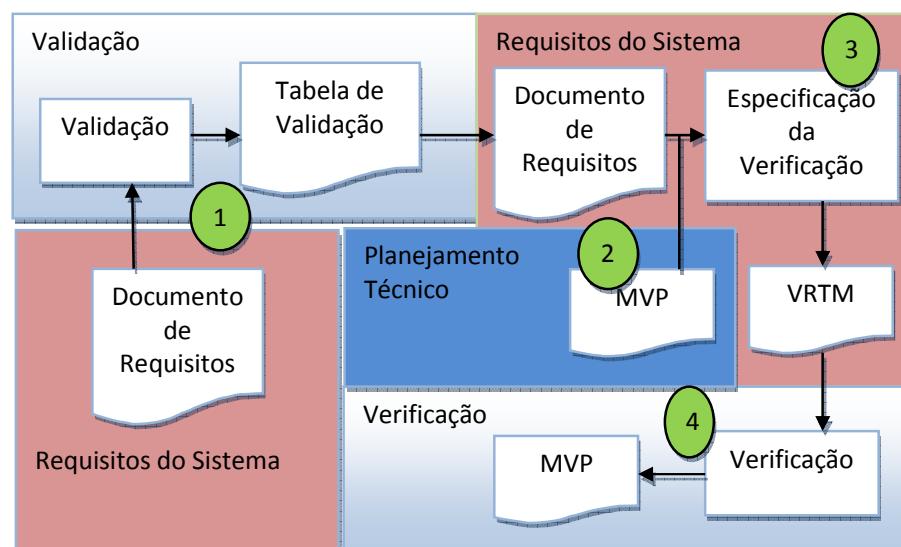


Figura 3.1 – Atividades de Verificação e Validação (Adaptada de NAS, 2006).

Como mostra a Figura 3.2, o modelo proposto pelo Manual de Engenharia de Sistemas aplica Verificação e Validação durante todo o processo de desenvolvimento de software. Esse modelo propõe que a V&V seja aplicada incrementalmente sob todos os requisitos durante o ciclo de desenvolvimento, seguindo a evolução natural e hierárquica da criação de software. Assim, quando cada incremento de requisitos é levantado e desenvolvido, eles sofrem Validação e Verificação.

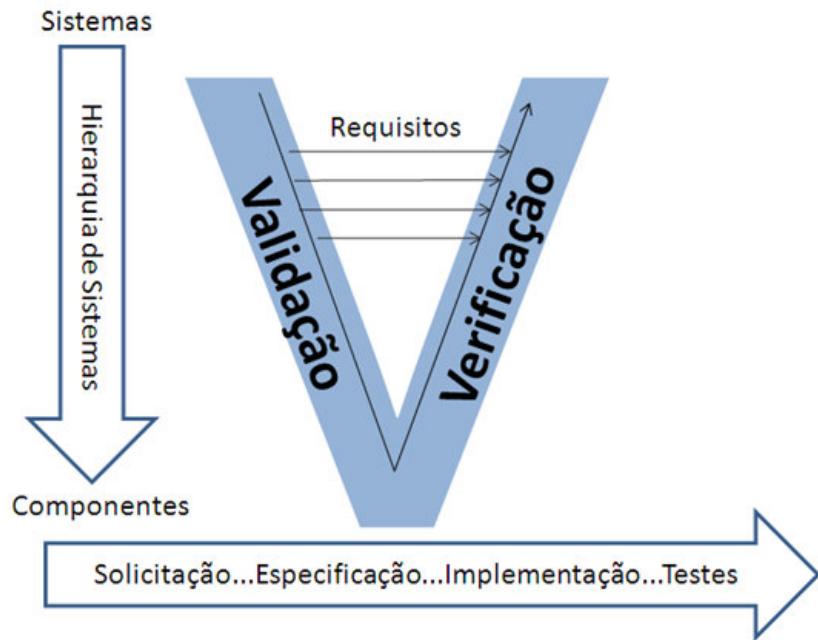


Figura 3.2 – Diagrama em “V” da Engenharia de Sistemas (Adaptada de NAS, 2006).

Ainda sobre a Figura 3.2, fica claro que o modelo proposto em NAS (2006) tem dois momentos distintos na aplicação de V&V, onde a validação é aplicada às fases iniciais e a verificação é aplicada no decorrer do ciclo de desenvolvimento. Alguns autores como Sargent (2000) e Balci (1994) propõem a aplicação de Validação em outras fases do desenvolvimento, criando uma maior evidência que o produto final está correto e que atende as necessidades do usuário. A Figura 3.3 mostra um paradigma clássico de V&V, onde fica claro que além da validação da especificação ou do modelo conceitual, existem as atividades de validação operacional e dos dados.

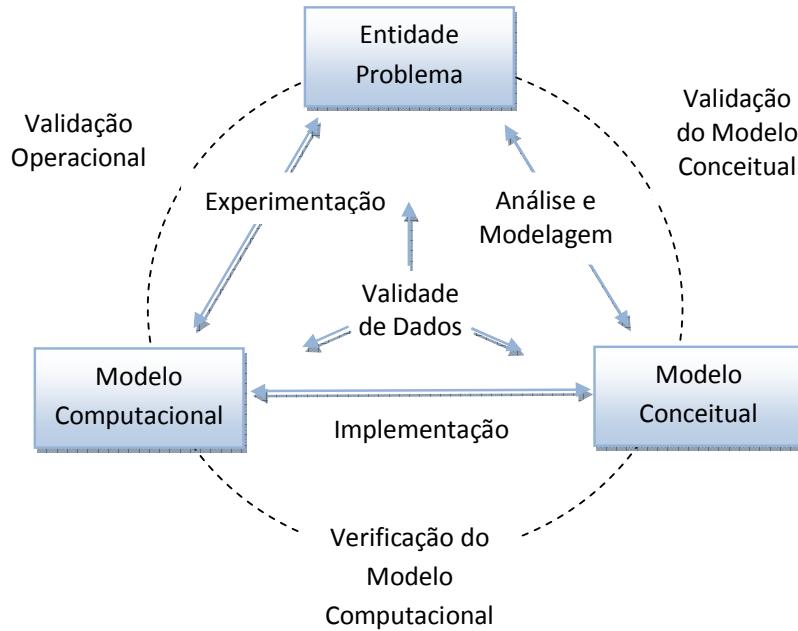


Figura 3.3 – Modelo clássico de V&V (Adaptada de Sargent, 2000).

Sargent (2000) descreve este modelo como: A **Entidade Problema** é um sistema (real ou proposto) de uma idéia, uma situação, uma política, ou um fenômeno a ser modelado; o **Modelo Conceitual** é a representação matemática/lógica/verbal da Entidade Problema desenvolvida por um estudo particular; e o **Modelo Computacional** é o Modelo Conceitual implementado em um computador. O modelo conceitual é desenvolvido por meio de uma etapa de **Análise e Modelagem**. O Modelo Computacional é desenvolvido com o uso de um compilador na fase de **Implementação**. Logo, inferências sobre o problema são obtidas por experimentos de computador aplicados ao Modelo Computacional na fase de **Experimentos**.

Na Figura 3.3 pode-se notar que o Modelo Conceitual passa por uma fase de **Validação**. Na qual se assume que a conceitualização está correta e que o modelo representacional do problema está “razoável”³ para o uso pretendido. Nota-se também que o Modelo Computacional passa pela fase de **Verificação**, a qual garante que o programa de computador implementado está correto. Na **Validação Operacional** é definido como será garantido que o comportamento

³ Segundo Balci (1995), a exatidão dos resultados é julgada de acordo com os objetivos do estudo, por exemplo, em alguns casos 60% de exatidão é suficiente, em outros casos são exigidos 95%.

das saídas do modelo terá a exatidão desejada. É neste ponto onde muitos dos testes de validação ocorrem, sendo possível encontrar diferenças e apurar em quais das etapas do modelo apresentam falhas. Na **Validade dos Dados** é definido como garantir que os dados necessários para a construção e avaliação do software são adequados e corretos. É importante ressaltar que esse paradigma, apresentado na Figura 3.3, é aplicado durante todo o ciclo de desenvolvimento de forma iterativa. Várias versões de softwares são criadas e melhoradas até que se obtenha um produto que satisfaça a exatidão pretendida.

Os paradigmas apresentados por NAS (2006) e Sargent (2001) mostram pontos em comuns e algumas dissonâncias. Este fato é comum entre os diversos paradigmas e pode ser percebido em outros trabalhos como Banks *et al.* (1988) e Knepell e Arangno (1993). Mas, independente das diferenças, todos os paradigmas exigem que uma equipe de profissionais conduza as atividades de verificação e validação.

Sargent (2001) aponta quatro formas diferentes ao se aplicar V&V em relação à equipe envolvida:

1. A própria equipe de desenvolvimento toma as decisões de como o software é válido. Estas decisões são subjetivas e feitas baseadas nos resultados de vários testes e avaliações conduzidas como parte do processo de desenvolvimento do modelo.
2. Outro método é ter os usuários do modelo fortemente envolvidos com o time de desenvolvimento para determinar a validade do modelo de simulação. Neste método, o foco de quem determina a validade do software deve mudar dos desenvolvedores para os usuários, melhorando assim a credibilidade.
3. O próximo método é conhecido como Verificação e Validação Independente (IV&V), onde uma empresa ou equipe independente dos desenvolvedores e dos usuários do software irão validá-lo. Este é um método aplicado quando existem várias equipes de desenvolvimento envolvidas no projeto e com desenvolvimento de sistemas em larga escala. É importante ressaltar que este método envolve um custo

elevado e que pode ser feito de duas maneiras: de modo concorrente com o desenvolvimento do projeto ou quando o projeto estiver implementado ou concluído.

4. O Modelo de Pontos é o último método apresentado por Sargent para determinar se um modelo é válido. Pontos (ou pesos) são determinados subjetivamente sobre as questões referentes ao processo de validação. Um modelo de sistema é considerado válido se o total geral dos pontos adquiridos de acordo com cada questão ultrapassarem a nota mínima, mas, segundo o próprio Sargent, este método é raramente utilizado na prática.

3.2 Técnicas de Verificação & Validação

Como dito anteriormente, a literatura propõe diversos paradigmas que se diferem uns dos outros pela forma de interação com o processo de desenvolvimento. Mas, apesar das diferenças, esses paradigmas utilizam técnicas comuns para realizar as atividades de Verificação e de Validação. Alguns autores (Balci, 1994 e Nance, 1994) descrevem essas técnicas como Técnicas de Testes e afirmam que elas têm o propósito de demonstrar a inexatidão e a existência de erros no software. Logo, alguns testes são conduzidos sob o modelo conceitual e outros sob o modelo computacional, durante todo o ciclo de vida de desenvolvimento do sistema proposto.

A Tabela 3.1 mostra uma taxonomia apresentada em Balci (1995), a qual categoriza as técnicas de V&V em seis perspectivas. O nível de formalismo matemático de cada categoria aumenta de muito informal na primeira linha de classificação da tabela, a muito formal na ultima linha, implicando também no aumento da complexidade.

Tabela 3.1 – Taxonomia de Técnicas de Verificação & Validação (Fonte: Balci, 1995).

Técnicas de Verificação & Validação e Testes	
Classificação	Técnicas
Informal	Auditória, Validação de Face, Inspeções, Revisões, Teste de Turing, Walkthroughs
Estática	Checagem de Inconsistência, Análise de Fluxo de Dados, Análise Baseada em Grafos, Análise Semântica, Analise Estrutural, Análise Sintática
Dinâmica	Teste de Caixa Branca, Teste Button-Up, Monitoração de Execução, Execução de Perfil, Execução de Traço, Testes de Campo, Comparação Gráfica, Validação Preditiva, Teste de Regressão, Análise Sensitiva, Técnicas Estatísticas, Teste de estresse, Testes de Sub-Modelo, Debug Simbólico, Teste Top-down, Visualização, Teste de Caixa Branca
Simbólica	Gráfico de Causa Efeito, Análise de Partição, Análise de Caminhos, Execução Simbólica
Restrições	Checagem de Suposições, Análise de Limites, Suposição Indutiva
Formal	Indução, Inferência, Dedução Lógica, Cálculos Preditivos, Transformação Preditiva, Prova de Exatidão

Segundo Balci (1995), as técnicas informais de V&V estão entre as mais usadas. São chamadas informais porque as ferramentas e métodos usados confiam pesadamente no raciocínio e na subjetividade humana sem formalismos matemáticos estritos. O rótulo "informal" não implica em qualquer falta de estrutura ou diretrizes formais para o uso das técnicas.

As técnicas estáticas são aplicadas no código fonte dos sistemas sem exigir a execução do mesmo. O compilador da linguagem utilizada no desenvolvimento do software é um exemplo de ferramenta estática. Já as técnicas dinâmicas requerem a execução do sistema tendo como função avaliar seu comportamento.

As técnicas simbólicas também avaliam o comportamento dinâmico do sistema fornecendo entradas simbólicas e expressões são produzidas como resultado da transformação dos dados simbólicos durante o caminho de execução do software. As técnicas de restrições de V&V são empregadas para avaliar a exatidão do modelo usando afirmação de checagem, análise do limite, e afirmações indutivas.

Por último, as técnicas formais de V&V são baseadas na prova formal matemática de exatidão. Segundo Balci (1995), se atingível, esse tipo de prova é o meio mais eficaz, mas estas técnicas, e principalmente as técnicas

baseadas em testes estatísticos, necessitam de um número muito grande de ponto de dados. Portanto, mesmo quando as hipóteses são observáveis e satisfeitas, os testes estatísticos são considerados significativos somente se o volume de dados for suficiente. E que em alguns casos é impraticável do ponto de vista financeiro.

3.3 Análise das Técnicas Informais

Esta seção apresenta uma análise mais detalhada das técnicas informais. Técnicas às quais têm a característica de serem aplicadas a todo o ciclo de desenvolvimento do software de forma metódica e simples, analisando os artefatos e atividades de todas as etapas de um processo de software (Balci, 1995). Esta característica é importante para este trabalho, pois a validação de um processo não pode ser feita analisando somente uma etapa ou mesmo um único artefato, e sim o processo de desenvolvimento como um todo.

Abaixo são descritas seis técnicas informais comumente usadas na verificação e validação de software. A descrição de cada técnica é apresentada separadamente nas Tabelas 3.2 à 3.7 para auxiliar na escolha da técnica ou técnicas que podem vir a ser usadas na validação de processos.

Tabela 3.2 – Características da técnica de Desk-Checking. (Fonte: Perry, 1995).

Nome: Revisão de Mesa (Desk-Checking),			
Tipo:	<input checked="" type="checkbox"/> Manual <input type="checkbox"/> Automática	<input type="checkbox"/> Dinâmica <input checked="" type="checkbox"/> Estática	<input checked="" type="checkbox"/> Estrutural <input checked="" type="checkbox"/> Funcional
Descrição: Desk checking é o meio mais tradicional de análise de um software. Essa técnica é conduzida pelo próprio analista ou programador. O processo envolve uma revisão completa do produto para garantir que ele está estruturalmente correto e que os padrões e requisitos foram obtidos.			
Sugestão de Utilização: Realizar Desk Cheking é normalmente uma parte integral de projeto e da codificação. Essa técnica requer que o desenvolvedor valide a forma de trabalhar do produto desenvolvido de acordo com os requisitos. O processo envolve uma avaliação passo a passo do produto final. Vários testes individuais são usados como base para conduzir esta avaliação.			
Custo de Uso: Desk Cheking é executada por um indivíduo completamente informado dos requisitos do produto.			
Habilidades para Uso: As habilidades necessárias para utilizar essa técnica são as mesmas necessárias para desenvolver o produto analisado.			
Vantagens: Fornece uma checagem redundante na validação do produto desenvolvido.			
Desvantagens: É freqüentemente difícil para o indivíduo que desenvolveu o produto identificar os defeitos.			

Tabela 3.3 – Características da técnica de Peer-Review. (Fonte: Perry, 1995).

Nome: Revisão por Pares ⁴ (Peer-Review)			
Tipo:	<input checked="" type="checkbox"/> Manual <input type="checkbox"/> Automática	<input type="checkbox"/> Dinâmica <input checked="" type="checkbox"/> Estática	<input checked="" type="checkbox"/> Estrutural <input type="checkbox"/> Funcional
Descrição: A revisão por pares é uma análise em dupla aonde os envolvidos realizam as mesmas funções durante a revisão. Por exemplo, os analistas de sistemas revisando as especificações do projeto e os programadores revisando os códigos fontes. Esse tipo de revisão pode ser conduzida por uma ou mais duplas. A revisão é normalmente conduzida observando a eficiência, estilo e aderência aos padrões.			
Sugestão de Utilização: O uso de revisão em pares deve ser um processo formal e empregado a todo setor de desenvolvimento de software. Quando essa técnica é usada, todos os sistemas devem ser sujeitos às revisões e não apenas selecionando um projeto. Os pares devem receber diretrizes durante as revisões para garantir sua consistência. Geralmente, os relatórios formais não estão preparados ao final da revisão em pares, mas os resultados são enviados diretamente ao desenvolvedor.			
Custo de Uso: Revisão em pares é um trabalho intensivo em que o custo depende do número de pares conduzindo a revisão e do escopo das revisões. Normalmente ela é uma técnica de custo médio.			
Habilidades para Uso: Os pares que conduzem a revisão devem ser instruídos em como conduzir essas revisões e os indivíduos que formam as duplas devem ter o mesmo nível de conhecimento sobre os principais tópicos.			
Vantagens: Revisão em pares aplica uma dupla análise para garantir um produto de qualidade. Freqüentemente, essa análise em dupla pode incentivar os membros da equipe quando não existe um supervisor.			
Desvantagens: Membros dos pares podem ser relutantes em criticar outras duplas, ou fazê-lo em excesso, o que pode causar ressentimento dentro das equipes.			

⁴ O termo “Pares” pode ter o sentido de pessoas com funções semelhantes

Tabela 3.4 – Características da técnica de Design Review (Fonte: Perry, 1995)

Nome: Revisões de Projetos (Design Review)
Tipo:
<input checked="" type="checkbox"/> Manual <input type="checkbox"/> Automática
<input type="checkbox"/> Dinâmica <input checked="" type="checkbox"/> Estática
<input checked="" type="checkbox"/> Estrutural <input type="checkbox"/> Funcional
Descrição: Revisão de projetos é comumente realizada em muito dos diferentes estágios do desenvolvimento de software. Revisões formais freqüentemente ocorrem perto da conclusão dos requisitos do sistema, do projeto do sistema, do projeto preliminar, do projeto de codificação, da programação, e na conclusão dos testes. A Revisão é freqüentemente direcionada mais para o projeto do que o produto.
Sugestão de Utilização: O processo de revisão de projeto pode ser parte da metodologia do projeto de sistema. Muitas metodologias formais incluem a revisão de projeto como uma parte integral do processo. Algumas metodologias recomendam que a revisão de projeto seja conduzida pelo pessoal da garantia da qualidade. Freqüentemente as revisões incluem checklist formalizados para garantir que todos os passos da metodologia foram realizados
Custo de Uso: Revisão de projetos são revisões de trabalho intensivo e o custo depende do tamanho do time de revisão e do tamanho da revisão. As revisões são normalmente limitadas em escopo, logo o custo é médio.
Habilidades para Uso: O time de revisão de projeto necessita ter habilidade no processo de revisão de projetos e deve ter habilidades em projetos de sistemas
Vantagens: O uso de revisões de projeto são normalmente necessárias para garantir a confiança e o entendimento do desenvolvimento do sistema e da metodologia de manutenção.
Desvantagens: As revisões de projeto geralmente não são necessárias quando os desenvolvedores têm grande conhecimento da metodologia de projeto e são fornecidos recursos adequados para que eles possam seguir completamente a metodologia.

Tabela 3.5 – Características da técnica de Auditoria (Fonte: Perry, 1995).

Nome: Auditoria			
Tipo:			
<input checked="" type="checkbox"/> Manual	<input type="checkbox"/> Automática	<input type="checkbox"/> Dinâmica	<input checked="" type="checkbox"/> Estática
<input checked="" type="checkbox"/> Estrutural			
<p>Descrição: Auditoria de Software é um tipo de revisão no qual um ou mais auditores que não são membros da organização responsável pelo desenvolvimento do produto examinam de forma independente o software garantido um alto grau de confiança nas especificações, padrões e relatórios.</p>			
<p>Sugestão de Utilização: A Auditoria deve ser independente, realizada sob um ponto de vista ético e com o devido cuidado profissional. Seguindo um planejamento, onde o objetivo e escopo é traçado. Logo em seguida a auditoria é realizada utilizando-se de checklists e questionários aplicados em reuniões com os envolvidos no processo. Na auditoria, as conformidades, oportunidades de melhorias e boas práticas são levantadas e apresentadas.</p>			
<p>Custo de Uso: Auditoria é uma tarefa de trabalho intensivo envolvendo a participação de terceiros. Ou seja, uma equipe com experiência em auditoria deve ser contratada para manter a independência das avaliações implicando em um custo elevado.</p>			
<p>Habilidades para Uso: Os auditores de software necessitam ter habilidade em auditoria executando boas práticas como motivar a identificação de problemas e não atacar pessoas e sim fatos concretos, além de possuir um conhecimento em processos e desenvolvimento de software.</p>			
<p>Vantagens: Auditoria de software fornece uma avaliação independente das conformidades do software</p>			
<p>Desvantagens: Auditores tendem a ficar defasados tecnologicamente em relação ao ambiente computacional da organização devido à crescente complexidade do ambiente computacional. Tornando-se difícil encontrar auditores experientes.</p>			

Tabela 3.6 – Características da técnica de Walkthrough (Fonte: Perry, 1995).

Nome: Walkthrough			
Tipo:			
<input checked="" type="checkbox"/> Manual	<input type="checkbox"/> Automática	<input checked="" type="checkbox"/> Dinâmica	<input type="checkbox"/> Estática
Descrição: Durante o Walkthrough, o criador de um produto, ou o líder da equipe repassa ou faz uma checagem desse produto, através de uma simulação manual do sistema usando dados de testes. Os dados de testes devem ser simples, considerando os constrangimentos da simulação humana. A equipe deve fazer perguntas e levantar questões sobre o produto que pode identificar defeitos.			
Sugestão de Utilização: Walkthroughs são uma evolução da técnica de Desk Checking. Eles usam um método em equipe, onde essa equipe é normalmente composta de vários peritos sobre o assunto em questão, sobre análise de sistemas e testes. Diretrizes gerais devem ser estabelecidas em como conduzir a análise e os participantes devem ser treinados nesses procedimentos. É geralmente aconselhável para o time responsável em executar essa função aconselhe o desenvolvedor sobre pontos específicos que ele deve cobrir durante a avaliação.			
Custo de Uso: Walkthrough é um processo de uso intensivo de pessoas com o custo variando de acordo com o número de participantes, da equipe e do tamanho do projeto analisado. Mas normalmente é classificado como tendo um custo médio não sendo considerado nem alto nem baixo.			
Habilidades para Uso: Em avaliações de especificações de alto-nível requer habilidades de usuário do sistema e para avaliações de baixo-nível requer habilidades de programadores			
Vantagens: Walkthrough fornece uma base para questionar a lógica do programador e ajudar o desenvolvedor a identificar problemas no sistema.			
Desvantagens: É relativamente desestruturado, fazendo com que sua eficiência seja dependente da habilidade e do interesse da equipe envolvida.			

Tabela 3.7 – Características da técnica de Inspeções (Fonte: Perry, 1995)

Nome: Inspeções			
Tipo:			
<input checked="" type="checkbox"/> Manual	<input type="checkbox"/> Automática	<input type="checkbox"/> Dinâmica	<input checked="" type="checkbox"/> Estática
<input checked="" type="checkbox"/> Estrutural			
Descrição: As inspeções são revisões formalizadas que usa pessoal além do desenvolvedor para avaliar a exatidão de um produto. O produto é verificado contra critérios de entrada predeterminados ou contra as especificações que foram usadas para construir o produto. O processo necessita de uma revisão minuciosa da especificação de entrada e do produto, por pessoas competentes que foram reunidos para formar uma equipe de inspeção.			
Sugestão de Utilização: As inspeções necessitam de uma equipe de inspeção treinada e uma lista predeterminada de critérios ou especificações de entrada anteriormente verificadas que formam a base da inspeção. A equipe de inspeção normalmente inclui um moderador, em conjunto com representantes de pares de vários grupos. A eficácia do processo de inspeção depende da habilidade da equipe, o tempo de preparação gasto, e a escolha dos critérios de inspeção.			
Custo de Uso: As inspeções são um processo de trabalho intensivo, com o preço dependendo do tamanho da equipe e a profundidade da inspeção. Geralmente ela tem um alto preço.			
Habilidades para Uso: As inspeções necessitam uma equipe com conhecimento no processo de inspeção e do processo usado para produzir o produto que é inspecionado (isto é, codificação, análise e projeto). Contudo, as inspeções não necessitam de conhecimento detalhado da aplicação que é inspecionada.			
Vantagens: As inspeções são um processo formalizado de avaliar sistemas. O processo estruturado fornece uma alta probabilidade para descobrir problemas.			
Desvantagens: As inspeções são muito detalhistas, o processo consome tempo olhando cada componente de um sistema. Mas por causa da eficácia, os analistas e os programadores confiam pesadamente nas inspeções.			

As tabelas apresentadas nesta seção com as técnicas de validação contêm classificações de tipos com o propósito de orientar o leitor a escolher a técnica adequada. Esses tipos são explicados por Perry (1995) como a seguir:

- Avaliações Estruturais versus Funcionais – O conjunto de testes baseados na análise estrutural tende a descobrir erros que ocorrem durante "a codificação" do programa, enquanto o conjunto de testes com base na análise funcional tende a descobrir erros que ocorrem no levantamento de requisitos ou especificações de projetos. Ou seja, o teste funcional assegura que as exigências sejam propriamente satisfeitas na aplicação do sistema.
- Avaliações Dinâmicas versus Estáticas – A análise dinâmica necessita que o programa seja utilizado. Isto é, o programa é executado sob alguns casos de teste e os resultados da execução do programa são examinados para verificar se ele funcionou como esperado. A análise

estática não implica normalmente na execução do programa real. As técnicas de análise estáticas comuns incluem tarefas como verificação de sintaxe.

- Avaliações Automáticas versus Manuais – As técnicas manuais são executadas por uma equipe e as técnicas automatizadas pelo computador. Quanto mais automatizado é o processo de desenvolvido, mais fácil fica para automatizar o processo de teste. Por outro lado, se a equipe executa a análise, a documentação, e o desenvolvimento de forma manual, os testes não podem ser automatizados.

3.4 Conceitos de Inspeção de Software

A inspeção de software é uma importante técnica de V&V. Segundo Somerville (2007 p.358), através de inspeções pode-se analisar as representações do sistema como documentos de requisitos, diagramas de projeto e o código fonte do programa. As inspeções podem ser aplicadas em todos os estágios do processo e não necessitam que o software seja executado. Se aplicada ao modelo conceitual, a inspeção é considerada como uma técnica de validação. Mas, se a técnica é aplicada sob o código fonte, ela toma o foco de verificação.

Somerville (2007 p.363) afirma que erros podem ser encontrados de um modo mais barato, por meio da inspeção, do que com extensivos programas de testes. Isso foi demonstrado em um experimento feito por Gilb e Graham (1993), que empiricamente compararam a eficácia das inspeções.

As técnicas de inspeções de software têm como principal objetivo encontrar erros. Para tanto, as técnicas de inspeções adotam como padrão a taxonomia de classificação de erros apresentadas pelo Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE 830, 1998). Ela define atributos de qualidade para requisitos de software e a falta de qualquer um desses atributos constituiria um tipo de defeito (Shull, 1998). Assim, os tipos são:

- Omissão – (1) Algum requisito importante relacionado à funcionalidade, ao desempenho, às restrições de projeto, ao atributo, ou à interface externa não foi incluído; (2) não está definida a resposta do software para todas as possíveis situações de entrada de dados; (3) faltam seções na especificação de requisitos; (4) faltam referências de figuras, tabelas, e diagramas; (5) falta definição de termos e unidades de medidas.
- Ambigüidade – Um requisito tem várias interpretações devido a diferentes termos utilizados para uma mesma característica ou vários significados de um termo para um contexto em particular.
- Inconsistência – Dois ou mais requisitos são conflitantes
- Fato Incorreto – Um requisito descreve um fato que não é verdadeiro, considerando as condições solicitadas para o sistema.
- Informação Estranha – As informações fornecidas no requisito não são necessárias ou mesmo usadas.
- Outros – Outros defeitos como a inclusão de um requisito numa seção errada do documento.

3.4.1 Benefícios da Inspeção de Software

De acordo com Biffl e Grossmann (2001), o esforço gasto por organizações de software com retrabalho varia em média entre 40% e 50% do esforço total do desenvolvimento de um projeto. Uma estimativa da distribuição do retrabalho pelas atividades de desenvolvimento de software feita por Wheeler *et al.* (1996) está ilustrada na Figura 3.4.1.

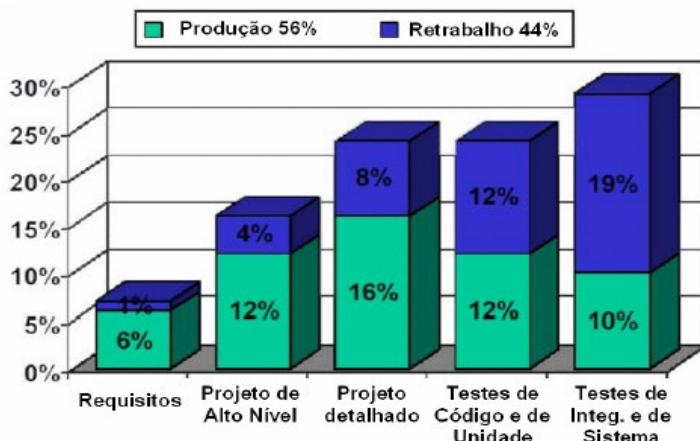


Figura 3.4.1 – Redistribuição do retrabalho pelas atividades de desenvolvimento de software. (Adaptado de Wheeler *et al.*, 1996).

Analizando a figura acima, é possível verificar que o retrabalho tende a aumentar na medida em que o desenvolvimento progride. Uma das razões para isto é o aumento no esforço para corrigir defeitos nas atividades finais do processo de desenvolvimento. Através da análise de 63 projetos, Boehm (1981) apresenta o custo relativo da correção de defeitos encontrados em cada uma das atividades de desenvolvimento (Figura 3.4.2).

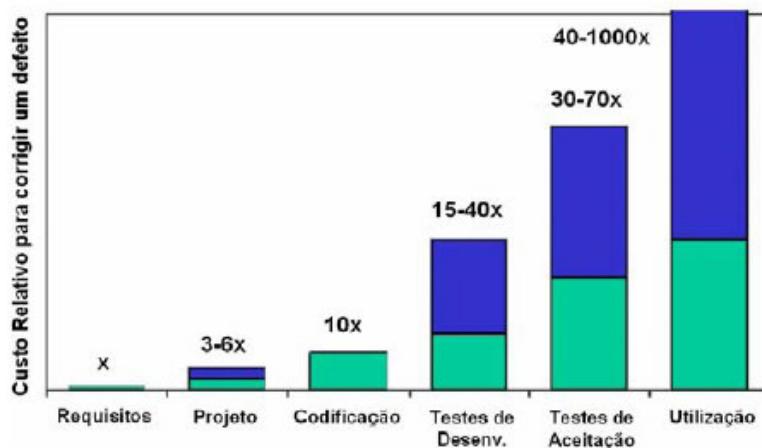


Figura 3.4.2 – Custo relativo para corrigir um defeito (Adaptado de Boehm, 1981).

Assim, um dos maiores benefícios de se utilizar inspeções de software é a detecção de defeitos nas fases iniciais do processo de desenvolvimento de software, facilitando a correção destes defeitos com menor esforço e custo. Desta forma, de acordo com Jones (1991), o esforço com retrabalho é reduzido em média para 10% a 20% do esforço total de desenvolvimento. Esta redução no retrabalho pode implicar em melhorias significativas para a produtividade de software. Laitenberger e Atkinson (1999) e Gilb e Graham (1993) afirmam em suas pesquisas que além da redução do esforço com retrabalho, a produtividade, o tempo de projeto e o custo são reduzidos significantemente.

De acordo com Kalinowski (2004) a aplicação de inspeções de forma bem planejada pode trazer diversos outros benefícios. Dentre eles se encontram:

- Aprendizado – Inspetores experientes podem tentar detectar padrões de como os defeitos ocorrem e definir diretrizes que ajudem na detecção destes. Além disto, bons padrões de desenvolvimento podem ser

observados durante a inspeção, sendo possível sua descrição como melhores práticas para a organização.

- Integração entre processos de detecção e de prevenção de defeitos – Saber onde e quando os defeitos ocorrem pode ajudar a estabelecer planos de contingência que evitem a sua ocorrência.
- Produtos mais inteligíveis – Os autores dos diversos artefatos, sabendo que estes serão inspecionados, passarão a produzir artefatos de uma forma que sua compreensão seja facilitada. A produção de artefatos mais inteligíveis, além de facilitar a inspeção, trará benefícios para as fases seguintes do processo de desenvolvimento, incluindo principalmente a fase de manutenção.
- Dados defeituosos ajudam a melhorar o processo de software do projeto – Analisando a causa dos defeitos encontrados é possível fazer ajustes no processo para evitar a ocorrência de defeitos deste mesmo tipo.

3.4.2 O Processo de Inspeção de Software

Fagan (1976) desenvolveu o processo tradicional de inspeção de software. Esse processo é largamente utilizado e serve como base para vários outros processos de inspeções propostos. A Figura 3.4.3 ilustra o processo de Fagan. Neste processo, existem cinco atividades principais:

- Planejamento – Um usuário, desempenhando o papel de moderador da inspeção, define o contexto da inspeção (descrição da inspeção, técnica a ser utilizada na detecção de defeitos, documentos a serem inspecionados, autor do documento, entre outros), seleciona os inspetores e distribui o material a ser inspecionado.
- Preparação – Os inspetores estudam os artefatos individualmente, e eventualmente, fazem anotações sobre estes, produzindo uma lista de discrepâncias. O fornecimento de técnicas de leitura pode facilitar a execução desta tarefa.
- Reunião – Uma reunião em equipe ocorre, envolvendo o moderador, os inspetores e os autores do documento. Discrepâncias são discutidas e classificadas como defeito ou falso positivo. A decisão final sobre a classificação de uma discrepancia sendo discutida é do moderador. A

solução dos defeitos não é discutida durante a reunião, que não deve exceder duas horas, uma vez que após este tempo a concentração e a capacidade de análise dos inspetores costuma reduzir drasticamente.

No caso em que uma reunião precisar de mais de duas horas, é sugerido que o trabalho de inspeção continue no próximo dia.

- Retrabalho – O autor corrige os defeitos encontrados pelos inspetores e confirmados pelo moderador.
- Continuação – O material corrigido pelos autores é repassado para o moderador, que faz uma análise da inspeção como um todo e re avalia a qualidade do artefato inspecionado. Ele tem a liberdade de decidir se uma nova inspeção deve ocorrer ou não.

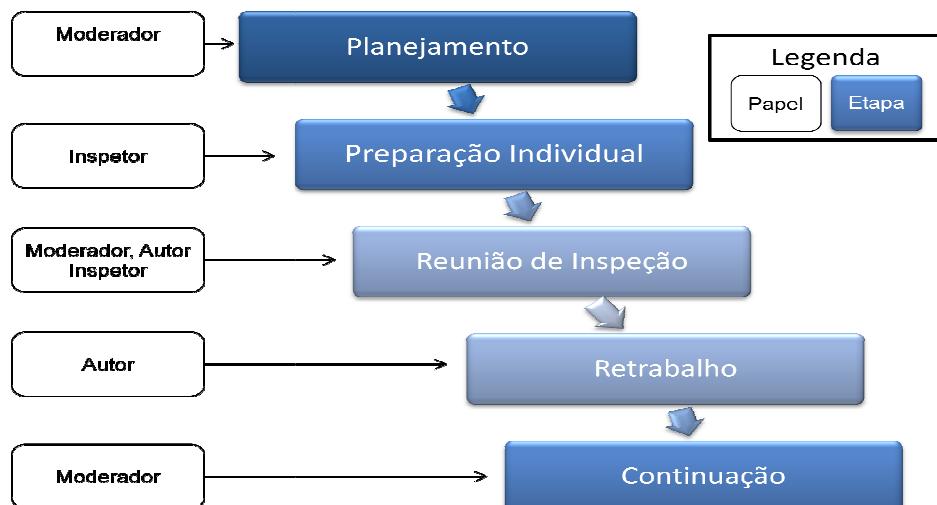


Figura 3.4.3 – Processo de inspeção de software (Fonte: Fagan, 1976)

Kalinowski (2004) mostra que outros processos vêm sendo propostos como o processo de Humphrey (1989), que muda o foco da atividade de preparação individual, pois, ao invés de entender o artefato a ser inspecionado, o inspetor tenta encontrar defeitos. Assim, cada um dos inspetores entrega uma lista de discrepâncias para o moderador da inspeção antes da reunião de inspeção se iniciar. O objetivo da reunião passa então a ser discutir as discrepâncias encontradas e classificá-las como defeito ou falso positivo.

Seguindo a idéia de Humphrey, Johnson (1994) introduz o conceito de Reunião Assíncrona, ou seja, um moderador pode se reunir apenas com o número de inspetores que a análise do artefato necessita, não mobilizando assim toda a

equipe de inspeção. Segundo o autor, ao se evitar reuniões, custos e conflitos de alocação de recurso são reduzidos sem sacrificar a eficiência das inspeções. O processo proposto por Johnson é composto das fases de planejamento, apresentação, revisão particular e revisão pública.

A detecção de defeitos propriamente dita começa na fase de Revisão Particular. Os revisores podem cadastrar pontos de dúvida, requisitar que alguma ação seja tomada ou simplesmente tecer um comentário. Somente os comentários estarão visíveis para os demais revisores. Entretanto, o moderador tem acesso a todos estes registros, o que lhe permite monitorar o progresso das revisões particulares.

Na fase de Revisão Pública todos os registros (dúvidas, ações e comentários) são visíveis aos inspetores que podem votar (concordância, discordância ou neutralidade) de forma assíncrona ou gerar novos registros. A fase só se encerra quando todos os registros estiverem estabilizados.

Sauer *et al.* (2000) usa das idéias de Humphrey e Johnson e propõe uma reestruturação do modelo tradicional de Fagan, adicionando suporte a reuniões assíncronas com equipes geograficamente separadas. A idéia de Sauer *et al.*, é manter a estrutura rígida e os aspectos colaborativos do processo tradicional e substituir as atividades de preparação e de reunião do processo tradicional por três novas atividades seqüenciais: detecção de defeitos, coleção de defeitos e discriminação de defeitos. Estas atividades podem ser descritas da seguinte forma:

- Detecção de Defeitos – Cada um dos inspetores selecionados pelo moderador no planejamento realizará uma atividade de detecção de defeitos. A principal tarefa desta atividade consiste em buscar defeitos no documento a ser inspecionado e assim produzir uma lista de discrepâncias.
- Coleção de Defeitos – O moderador agrupa as listas de discrepâncias dos inspetores. Esta atividade envolve eliminar discrepâncias repetidas (encontradas por mais de um inspetor), mantendo só um registro para cada discrepancia.

- Discriminação de Defeitos – O moderador, o autor do documento e os inspetores discutem as discrepâncias de forma assíncrona. Durante esta discussão, algumas discrepâncias serão classificadas como falso positivo e outras como defeito. Os falsos positivos serão descartados e os defeitos serão registrados em uma lista de defeitos, que então será passada para o autor para que a correção possa ocorrer.

Segundo Sauer *et al.* (2000), este processo é mais eficiente no que se diz respeito ao tempo, pois os inspetores trabalham em paralelo e discrepâncias encontradas por mais de um inspetor vão direto para a fase de retrabalho e as discrepâncias encontradas por um inspetor não precisa ser discutida com todos os outros inspetores. A Figura 3.4.4 mostra o processo proposto.

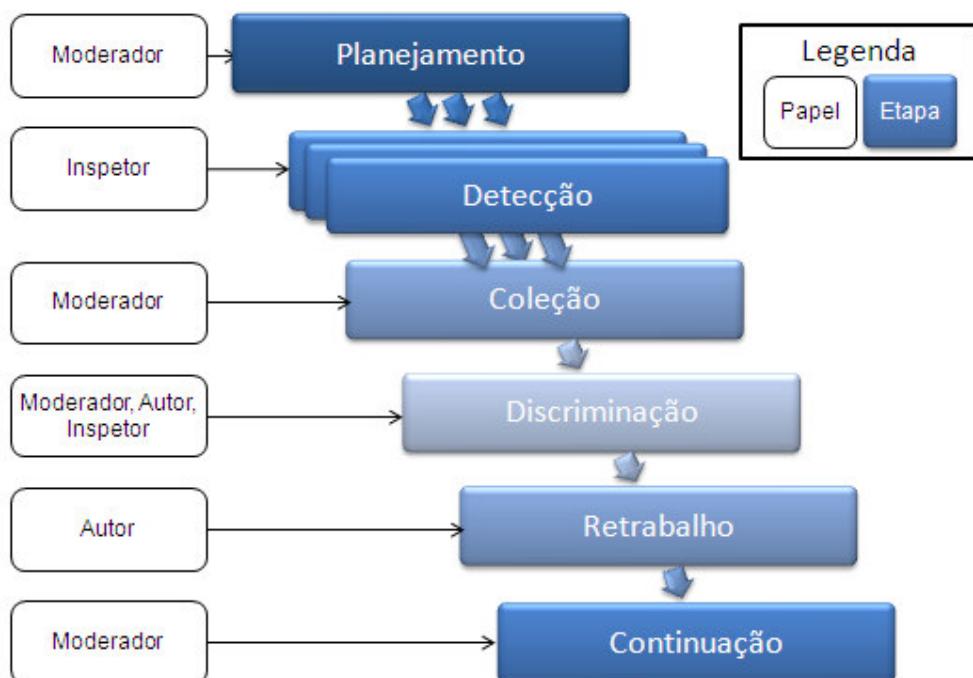


Figura 3.4.4 – Processo de Inspeção de Software Assíncrono (Fonte: Sauer *et al.*, 2000).

Visando utilizar a Inspeção de Software na prática de forma mais sistemática fazendo uso do modelo sugerido por Sauer *et al.* (2000), alguns autores propõe o uso de apoio ferramental, dentre estes autores pode-se citar o trabalho de Travassos *et al.* (2007) e Lanubile *et al.* (2003). Estas ferramentas podem realizar inspeções de diferentes artefatos com equipes geograficamente distribuídas.

A ferramenta proposta por Lanubile *et al.* (2003), chamada de Internet Based Inspection System (IBIS) utiliza a *Web* em conjunto com notificações por e-mail para apoiar inspeções assíncronas com equipes geograficamente distribuídas. A IBIS não limita o tipo de artefato a ser inspecionado e na detecção de defeitos atualmente permite apenas a utilização das técnicas ad-hoc e de checklists. Na reunião de inspeção desta proposta as discrepâncias encontradas na detecção de defeitos são tratadas como tópicos de discussão, onde mensagens podem ser acrescentadas e o moderador pode realizar a classificação das discrepâncias como defeito ou falso positivo. A IBIS não fornece apoio aos pontos de tomada de decisão do processo de inspeção, sendo as atividades de planejamento e de continuação do processo tratadas como simples cadastros, sem apoio para a realização de suas tarefas.

Segundo Travassos *et al.* (2007), a ferramenta proposta por eles, intitulada de Infra-estrutura de Suporte ao Processo de Inspeção (ISPIS) é a única ferramenta que disponibiliza recursos para a ajuda na tomada de decisão apoiadas por resultados de técnicas de estimativa e dados quantitativos representando a realidade da própria organização. A ISPIS pode ser utilizada para coordenar e apoiar a inspeção de qualquer tipo de artefato de software (documentos de requisitos, de projeto, código) e, opcionalmente, faz uso de um mecanismo de integração para se comunicar com ferramentas de detecção de defeitos existentes para artefatos específicos. Outra característica da ferramenta é permitir reuniões de inspeção tanto síncronas quanto assíncronas.

Os conceitos, técnicas e ferramentas de V&V de software apresentados neste capítulo são utilizados com eficiência, para minimizar a possibilidade de que um sistema não apresenta falhas e que realmente atenda as necessidades do usuário final. Estes mesmos conceitos e em particular a técnica de Inspeção de Software e suas ferramentas foram utilizados como base para a criação da técnica proposta, que é apresentada no próximo capítulo para validar processos de desenvolvimento de software.

CAPÍTULO 4

PROPOSTA DE UMA TÉCNICA DE VALIDAÇÃO DE PROCESSO POR INSPEÇÃO

Este capítulo apresenta uma proposta para validar processos de desenvolvimento a ser aplicada em ambientes de construção de software. O modelo proposto foi baseado em conceitos e técnicas de Verificação & Validação de software. A técnica de Inspeção foi investigada para prover soluções ao problema descrito nessa dissertação. A seguir são apresentadas as justificativas para a seleção e uma descrição detalhada da técnica.

4.1 A técnica proposta: visão geral e justificativas

Nos últimos anos, a Simulação Computacional vem assumindo uma importância cada vez maior como ferramenta de aquisição de conhecimento. Simular é o meio de confrontar teorias com experimentação, de antecipar resultados experimentais ou de realizar experiências de outro modo inacessíveis (Oberkampf e Trucano, 2007). Os softwares de Simulação Computacional são usados em muitas vezes em situações críticas, o que exige um nível elevado de credibilidade nos resultados exibidos.

Segundo Oberkampf e Trucano (2007), Sargent(2001) e Balci (1995), a credibilidade de tais resultados pode ser obtida através de técnicas de Verificação e Validação. Devido ao histórico bem sucedido desses trabalhos a técnica proposta para validar processos tem como base a V&V.

Entre alguns conceitos apresentados em Oberkampf e Trucano (2007), destaca-se uma diretriz, a qual indica que um experimento de validação deve ser conjuntamente projetado por pesquisadores, desenvolvedores do modelo e usuário, trabalhando próximos durante todo o ciclo de vida de um projeto. No

entanto, a independência deve ser mantida para obter os resultados pretendidos.

A técnica proposta no presente trabalho preocupa-se em provar que o processo atende às necessidades dos colaboradores quanto às questões de Engenharia de Software (Validação) e não em verificar se um produto foi construído corretamente de acordo com o processo (Verificação).

Como exposto na Seção 3.3, existem várias técnicas de validação formais ou informais que podem ser aplicadas em conjunto ou separadamente. Assim, deve-se selecionar a mais adequada para a validação de processos de software.

Segundo Balci (1995), as técnicas formais necessitam de dados matemáticos e são o meio mais eficaz de avaliar um software. Porém, em um processo de software, dados matemáticos não são gerados. Existem na literatura trabalhos como de Paula e Borges (2003) que realizam medições do processo, mas com o intuito de apresentar possíveis melhorias. No entanto, a validação é realizada na criação do processo ou em um estágio inicial de utilização e nesses períodos, dados estatísticos não existem. Além disso, as técnicas formais são consideradas complexas e uma outra preocupação deste trabalho foi selecionar uma técnica simples, pois um processo de validação não deve ter um grau de complexidade maior que o processo que será validado.

As técnicas informais segundo Balci (1994) são as mais utilizadas, simples e podem ser aplicadas durante todo o ciclo de desenvolvimento. Considerando uma outra diretriz defendida por Sargent (2000) e Oberkampf e Trucano (2007) de que o processo de validação de software deve ser aplicado durante todo o ciclo de desenvolvimento, as técnicas informais atendem a essa diretriz.

Dentre as técnicas informais apresentadas na Seção 3.3, a inspeção de software se destaca por ser uma técnica muito utilizada em diversos artefatos durante o ciclo de vida do software. A inspeção vem sendo usada em pesquisas para validar documentos como casos de testes e modelos arquiteturais de software, fornecendo conhecimento e exemplos práticos de

uso da validação em documentos conceituais (Perry, 1995; Barcelos e Travassos, 2005). A Figura 4.1 ilustra a possibilidade de se realizar inspeções nos diferentes artefatos de software.

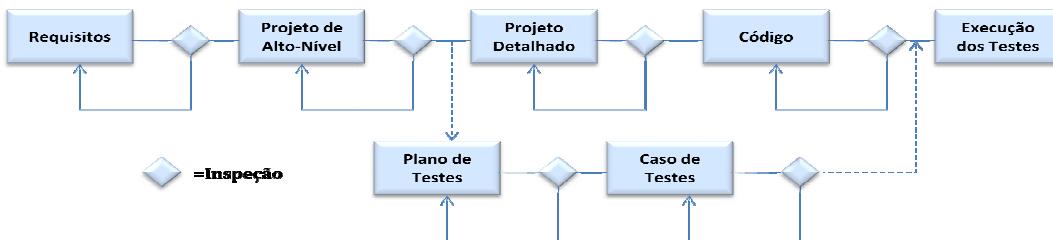


Figura 4.1 – Inspeção de Software em diferentes artefatos (Fonte: Ackerman *et al.*, 1989)

Para um processo de desenvolvimento de software ser considerado válido, ele deve ser tangível de ser seguido e atender ao que se propõem. Se a metodologia do processo não está sendo seguida, ou seja, se os usuários não obedecem a seus elementos (etapas, tarefas ou atividades), algo não está correto. O processo pode ser burocrático ao ponto de não ser seguido ou alguns desses elementos são desnecessários no dia a dia de uma equipe de projeto. Assim, com o intuito de examinar se o processo está sendo seguido, a inspeção de software pode ser utilizada.

Como mencionado na Seção 3.3, na inspeção de software, o produto é analisado contra critérios de entrada predeterminados ou contra as especificações que foram usadas para construir o produto. Aplicando essa técnica à validação de processos, tem-se como critério predeterminado a documentação do processo, ou seja, o modelo formal, o qual será comparado com todos os documentos produzidos, e o próprio software. Esses artefatos são considerados as entradas para que as comparações sejam feitas.

4.2 Validação de Processo de Desenvolvimento de Software por Inspeção

A técnica de inspeção proposta para a validação de processos consiste na adaptação da técnica de Sauer *et al.* (2000) para validação de software, já detalhada na Seção 3.4.2. Algumas modificações tiveram que ser realizadas afim de atender a inspeção de processos. A seguir, são descritas as principais características que compõem a técnica a ser executada para realizar a validação, detalhando as modificações realizadas.

4.2.1 Detalhamento do Processo de Validação Proposto

De acordo com Laitenberger e Atikson (1998), do ponto de vista técnico, uma abordagem para inspeção de software é formada pelo tipo do artefato de software a ser avaliado, pela forma como a equipe de profissionais será definida para realizar a inspeção, pela técnica de avaliação utilizada e pelo processo de execução seguido. Portanto, uma possível forma para se definir uma abordagem para inspeção é através da realização das seguintes tarefas:

1. Determinar o artefato de software que deve ser avaliado e o tipo de informação que o compõem;
2. Identificar os perfis das pessoas que devem ser envolvidas nessa inspeção, assim como o papel que desempenham durante a avaliação;
3. Definir os tipos de defeitos que a abordagem busca identificar;
4. Definir a técnica de avaliação utilizada para identificar os defeitos no artefato a ser avaliado, e;
5. Especificar o processo composto pelas atividades a serem executadas para atingir o objetivo final da avaliação.

A seguir, é apresentada a técnica de validação de processos de desenvolvimento proposta neste trabalho, seguindo a abordagem de Laitenberger e Atikson (1998). Essa técnica é denominada VProInsp (Validação de Processos por Inspeção).

4.2.1.1 Determinar os artefatos do processo

Geralmente artefatos de software (ou simplesmente artefato) são considerados como qualquer produto gerado em qualquer etapa do processo de desenvolvimento de software. Um artefato de software normalmente envolve a compreensão do problema, a modelagem do problema, a identificação de possíveis alternativas de solução para este problema, a análise destas alternativas, a tomada de decisão sobre quais soluções serão usadas na construção do software, o próprio software e sua documentação.

É importante ressaltar que no VProclnsp, o artefato tem uma abrangência um pouco maior, pois a idéia da técnica proposta de validação é comparar o modelo formal, ou seja, a documentação do processo, com os produtos gerados na execução de tal processo. Logo, esse modelo formal se torna um importante artefato para a validação de processo.

Além da documentação do processo, as informações dos software (comunicadores eletrônicos, softwares de controle de versões, compiladores, entre outros) utilizados no dia a dia da equipe de desenvolvimento também são artefatos considerados no VProclnsp. Através da análise das informações desses softwares pode-se descobrir:

- a) Análise de logs e informações dos softwares de controle de versões – pode-se garantir que a gerência de configuração está sendo feita e o padrão do código seguido, caso o processo de desenvolvimento especifique esses pontos.
- b) Comunicadores eletrônicos – pode-se descobrir se reuniões técnicas e contatos com o cliente estão sendo feitos, caso o processo de desenvolvimento exija.

Sendo assim o VProclnsp expandiu o termo de artefato de software para artefato de processos de desenvolvimento.

4.2.1.2 Definição do papel e do perfil da equipe participante do processo de inspeção para validação

O VProclnsp é constituído de quatro papéis com características bem definidas de atuação. A diferença para o modelo de inspeção proposto por Sauer *et al.*

(2000) é a criação do papel chamado de Analista. Os papéis são definidos abaixo:

- Moderador – é uma pessoa independente, individual e imparcial que coordena o planejamento, a preparação, a conduta de inspeção, e reuniões. Durante a inspeção, ele assegura que a inspeção seja eficientemente conduzida quanto à utilização de recurso e a realização do esforço máximo em direção ao descobrimento de problemas.
- Autor – é a pessoa ou uma equipe responsável pelo modelo formal do processo de desenvolvimento de software. Essa pessoa ou equipe pode ser a mesma que desenvolveu o processo sob validação ou uma pessoa com um conhecimento avançado nesse processo. Muitas vezes, essa pessoa pode ser um gerente de projeto que tem um treinamento suficiente. A responsabilidade do autor é explicar e introduzir a documentação do processo, além de esclarecer os objetivos, peculiaridades e expectativas sobre a validação. Se o processo for bem documentado, o moderador pode assumir o papel de autor, desde que ele tenha o conhecimento avançado do processo sob validação.
- Inspetor – é o responsável por identificar, analisar e comparar o modelo formal do processo, com o processo executado na prática, ou seja, comparar os artefatos criados pelos *stakeholders* com a documentação fornecida pelo moderador (modelo formal). O inspetor não tem a função de encontrar erros nos artefatos e sim analisar se o que foi proposto pelo processo está sendo feito.
- Analista – é a pessoa, ou grupo de pessoas, envolvidas no projeto de desenvolvimento de software, que estão fazendo uso do processo de software sob validação. Os analistas também são conhecidos como *stakeholders*.

Em relação ao número de moderadores e de autores, a realização de uma inspeção do processo geralmente necessita somente de uma pessoa alocada para cada um desses papéis, ou como mencionado anteriormente, o moderador pode desempenhar o papel do autor, desde que ele tenha o conhecimento necessário.

Quanto ao número de inspetores, pode-se ter a idéia que quanto mais pessoas forem alocadas para esse papel, maior seria o número de defeitos identificados. Contudo, o custo da avaliação poderia se tornar muito alto e, além disso, estudos experimentais mostram que não é a quantidade de inspetores que importa, mas sim a utilização de profissionais com diferentes níveis de experiência (Svahnberg, 2003). E, é a realização da revisão, a partir de diferentes perspectivas, que influenciam no tipo e na quantidade de defeitos encontrados (Shull, 1998).

4.2.1.3 Taxonomia de defeitos

Como dito anteriormente, o VProclnsp não tem o objetivo de achar defeitos nos artefatos e sim validar os artefatos quanto a utilização e criação de acordo com o processo, descobrindo se o processo está sendo executado como previsto. Shull (1998), não apresenta como podem ser identificados os possíveis tipos de defeitos, mas define uma taxonomia que foi utilizada por Barcelos e Travassos (2005) para inspeção de documentos arquiteturais. O VProclnsp utiliza a mesma taxonomia e a interpreta para o contexto da inspeção de processos. A Tabela 4.1 apresenta a taxonomia.

Tabela 4.1 – Taxonomia de defeitos propostos pelo VProclnsp

Tipos de Defeitos	Descrição dos Defeitos
Omissão	<ol style="list-style-type: none"> 1. Quando um dos elementos (artefato, atividade, tarefa, etapa) necessários ao processo não foram definidos. 2. Quando faltam seções de especificação de como os artefatos, atividades, tarefa e etapas serão definidas. 3. Quando falta definição de termos utilizados no processo.
Ambigüidade	<ol style="list-style-type: none"> 4. Quando a forma como os elementos do processo foi definida dificultam ou impossibilitam o seu uso. 5. Quando elementos possuem o mesmo nome, mas responsabilidades diferentes (Homônimos).
Inconsistência	<ol style="list-style-type: none"> 6. Quando elementos descritos possuem mesma responsabilidade, mas nomes distintos (Sinônimos). 7. Quando um elemento presente na execução do processo não foi definido no modelo formal. 8. Quando a representação não condiz com o significado estabelecido pela abordagem de documentação. 9. Quando as responsabilidades dos <i>stakeholders</i> não foram definidas.
Outro	10. Quando o defeito não se encaixa nas categorias acima.

4.2.1.4 Técnica de inspeção utilizada

O principal diferencial de uma abordagem de inspeção está no formalismo e na maneira como ela auxilia o inspetor a encontrar os defeitos. O que determina essa maneira é a técnica de detecção de defeitos utilizada. A Tabela 4.2 apresenta as técnicas e suas características.

Tabela 4.2 – Técnica de Insp. de Software (Adaptado de Barcelos e Travassos, 2005).

Técnica	Descrição
Ad-hoc	Técnicas ad-hoc são as técnicas mais simples para identificar defeitos em artefatos de software. Elas não oferecem nenhum tipo de apoio ou procedimento de execução formal e sistemático de inspeção. Sendo assim, os defeitos identificados dependem exclusivamente da capacidade, competência e experiência do inspetor. Um dos pontos negativos em usar técnicas ad-hoc é o fato delas não oferecerem auxílio aos inspetores para identificar defeitos.
Checklist	<i>Checklist</i> é uma evolução da técnica <i>ad-hoc</i> . Ao se utilizar <i>checklist</i> para revisar artefatos de software, é fornecido ao inspetor um conjunto de questionamentos que o auxiliam a identificar em que parte do artefato avaliado ele deve procurar por defeitos
Técnicas de Leitura	Técnicas de leitura são procedimentos que visam guiar individualmente os inspetores no entendimento de um artefato de software e, por consequência, na identificação de discrepâncias. Porém, para que esse tipo de técnica seja utilizado, o artefato deve ser representado de forma específica e padronizada, característica que ainda não pode ser atingida nos processos de desenvolvimento de software

Levando em consideração as peculiaridades da área de processos de desenvolvimento de software e as características apresentadas na Tabela 4.2, a técnica de *checklist* mostra-se mais adequada ao contexto de inspeção para validação de processos, pois não existem técnicas de leitura para processos e as técnicas *Ad-hoc* não orientam e são altamente dependentes dos inspetores. Logo, o VProInsp procurará validar processos usando como guia, os questionamentos definidos em um *checklist* de avaliação.

A idéia não é criar um *checklist* fixo, já definido e sim, mostrar um *template* com diretrizes para permitir que o moderador crie o *checklist* de acordo com as necessidades impostas por cada processo de desenvolvimento. Espera-se que o uso de *checklist* como técnica de inspeção possibilite comparar o modelo formal com a execução do processo.

Para auxiliar na identificação de discrepâncias o *checklist* deve atender algumas características que visam maximizar os resultados da validação. São elas:

- Os itens de avaliação que compõem o *checklist* devem usar conceitos oriundos da Engenharia de Software. Com isso, pretende-se reduzir a subjetividade em relação ao que é avaliado;
- O procedimento deve ser de fácil execução da avaliação, sem a necessidade de elaboradas atividades, como as necessárias para a especificação de cenários, permitindo a realização de avaliações com baixos custos;
- O moderador deve entender que é preciso existir pelo menos um *checklist* para cada etapa do processo. Isso não será difícil de ser criado, pois, apesar de existirem inúmeros processos de desenvolvimento todos eles se encaixam em algum dos modelos de processo de desenvolvimento existentes, que, por sua vez, tem suas etapas bem definidas;
- Para orientar o moderador na criação do *checklist*, o VProclnsp tem definidas algumas seções, em que orientações podem ser dadas aos inspetores e grupos de perguntas podem ser subdivididas de forma a organizar melhor as questões avaliadas. Além desses itens, o moderador tem a liberdade de criar novas seções para que mais de uma técnica de validação possa ser usada. Por exemplo, criar uma seção de perguntas abertas, seguindo os moldes da técnica de Validação de Face;

A Figura 4.2 apresenta o *checklist* inicial proposto pelo VProclnsp. Nessa figura, todas as seções identificadas com o caractere “*” são de preenchimento obrigatório e, consequentemente, essas seções não podem ser removidas do *checklist*. As outras seções são opcionais, ou seja, o Moderador pode customizá-las como bem entender. Por exemplo, se a seção *Templates* não se aplica ao processo sob validação ela pode ser retirada ou até mesmo renomeada.

Em cada item de checagem (pergunta) existem as seguintes informações: (1) Número seqüencial único de um item dentro de uma seção; (2) Descrição do item a ser checado; (3) Resposta afirmativa de um item checado; (4) Resposta negativa de um item checado e (5) Resposta Esperada (RE) pelo Moderador. Estes três últimos campos são fundamentais para a comparação entre processo e prática, pois, espera-se que a resposta afirmativa ou negativa coincida com a resposta esperada evidenciando que a atividade do processo representada pelo item do *checklist* está sendo executada. Em seguida, é feita uma descrição das seções com itens de checagem (Entradas, Saídas, Passos, Ferramentas, *Templates* e Responsáveis).

VProclnsp – Validação de Processo de Desenvolvimento de Software por Inspeção				
Checklist de Inspeção				
*Instruções [seção reservada a instruções aos inspetores]				
*Inspetor.....:				
*Fase/Atividade...:				
*Principais Objetivos [seção reservada para descrever os objetivos gerais propostos pelo processo ou etapa do processo que está sob validação]				
*Importância [seção reservada para descrever a importância do processo ou etapa sob validação]				
Entradas				
Nº	Descrição do Item de Avaliação	Sim	Não	RE
Saídas				
Nº	Descrição do Item de Avaliação	Sim	Não	RE
Passos				
Nº	Descrição do Item de Avaliação	Sim	Não	RE
Ferramentas				
Nº	Descrição do Item de Avaliação	Sim	Não	RE
Templates				
Nº	Descrição do Item de Avaliação	Sim	Não	RE
Responsáveis				
Nº	Descrição do Item de Avaliação	Sim	Não	RE
Observações [seção reservada ao moderador ou inspetores para descrever informações relevantes para validação do processo]				

Figura 4.2 – Versão padrão do Checklist proposto pelo VProclnsp

Na seção **Entradas**, busca-se agrupar os itens relacionados à avaliação de todas as entradas exigidas nas atividades de cada etapa do processo sob validação. Por exemplo, o modelo formal do processo exige como entrada para a atividade de “Criação do Glossário” (Etapa de Especificação) um documento de especificação.

A seção denominada **Saídas** agrupa os itens que buscam avaliar se todos os artefatos de saída exigidos pelo modelo formal do processo foram cumpridos. Por exemplo, o processo de desenvolvimento exige que a atividade de Definição do Projeto (foco, tempo e iterações do projeto) na fase de Planejamento tenha como resultado um documento textual com as informações pré-estabelecidas.

Passos é o nome da seção que busca avaliar se todos os passos especificados no modelo formal do processo foram seguidos para uma determinada atividade ou fase. Por exemplo, na fase de Teste a atividade de “Teste de Caixa Preta” segue todo o roteiro pré-determinado pela equipe de qualidade de software.

A subdivisão **Ferramentas** é uma agrupadora de itens com o objetivo de checar se todas as ferramentas especificadas pelo processo de desenvolvimento foram utilizadas, evitando assim incompatibilidade de arquivos e perda de informações. Por exemplo, todos os diagramas foram criados e salvos por um mesmo software que gera imagens com uma mesma resolução e extensão, todos os códigos do sistema foram gerados com a mesma versão do compilador.

Templates agrupa os itens que buscam avaliar se todos os padrões de documentação e codificação foram seguidos. Por exemplo, todos os documentos de “Caso de Uso” seguem o padrão da empresa.

E, por fim, a seção de **Responsáveis** tem como função reunir os itens do processo de desenvolvimento referentes às especificações de responsabilidade dos *stakeholders* em cada atividade ou fase. Por exemplo, o gerente responsável pela aprovação de alterações críticas no sistema aprovou as

alterações assinando os documentos de codificação necessários na fase de implementação.

4.2.1.5 Processo utilizado na inspeção

Como exposto anteriormente, a aplicação da técnica de validação de inspeção de software em processo de desenvolvimento, não tem a função de encontrar defeitos e sim validar se o processo está sendo seguido como proposto. Portanto, a metodologia de inspeção será usada para que os artefatos e o fluxo do processo sejam inspecionados e comparados com os artefatos e o fluxo do modelo formal. Para isso o modelo de Sauer *et al.* (2000), apresentado na seção 3.4.2, foi alterado e a etapa de Detecção foi substituída pela etapa de Análise e Comparação.

Outra mudança ocorreu na etapa de planejamento. A idéia é que essa etapa seja executada pelo moderador e pelo autor (ou autores) do processo. Essa participação possibilita ao autor fazer uma apresentação do processo indicando pontos que devem ser atingidos, tipo de ambiente proposto para utilização (comercial, acadêmico), expectativas e anseios. A Figura 4.3 mostra o modelo proposto pelo VProclnsp e a descrição desse modelo é feita em seguida.

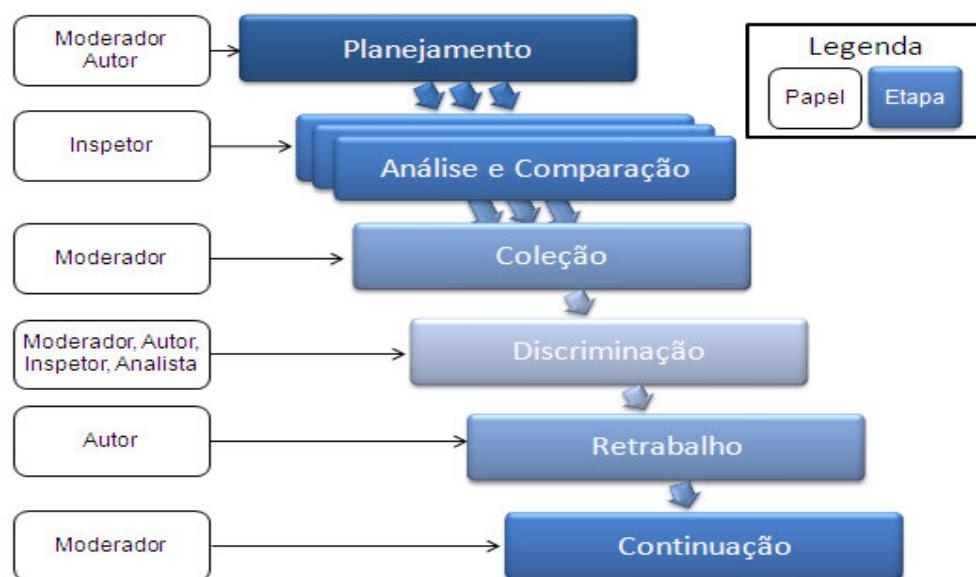


Figura 4.3 – Modelo de Inspeção proposto pelo VProclnsp (Adaptado de Sauer *et al.*, 2000).

Planejamento

Nessa fase, o primeiro passo é identificar um profissional que exercerá o papel do moderador da inspeção. O Planejamento é similar a uma Gerência de Projetos, pois nessa atividade o Moderador define a equipe, prazos, custos, entre outras funções cabíveis à gerência.

Durante essa fase, o Moderador deve definir a abrangência em que a inspeção de validação do processo vai ser realizada (por etapa, por iteração, ao final do ciclo de desenvolvimento, entre outros), além de informações como datas e tamanho da equipe. O Moderador define também os participantes, com base na experiência de desenvolvimento, conhecimento do domínio e experiência com validação de processos.

A principal atividade do Planejamento é a configuração do *checklist* para a inspeção de validação. Para configurar o *checklist*, o Moderador conta com a documentação ou modelo formal do processo sob validação e com o apoio do Autor do processo. Em uma reunião, os profissionais que assumiram os papéis de Moderador e Autor criam o *checklist* seguindo diretrizes propostas na atividade de Definição do Checklist, que foi apresentada na Seção 4.2.1.4, e identificam os artefatos a serem avaliados.

A última atividade cronológica do Planejamento é a apresentação do processo sob validação, dos objetivos da inspeção e dos *checklists* aos inspetores selecionados. Essa atividade pode ser omitida se os inspetores possuem conhecimento sobre o processo e sobre os artefatos a serem inspecionados. Nessa apresentação, os documentos a serem utilizados durante a inspeção são entregues. A Figura 4.3.2 mostra o fluxo das atividades e os documentos necessários e produzidos durante a etapa de planejamento. Nessa figura, as setas sólidas indicam o fluxo das atividades e as setas tracejadas representam os documentos de entrada e saída das atividades.

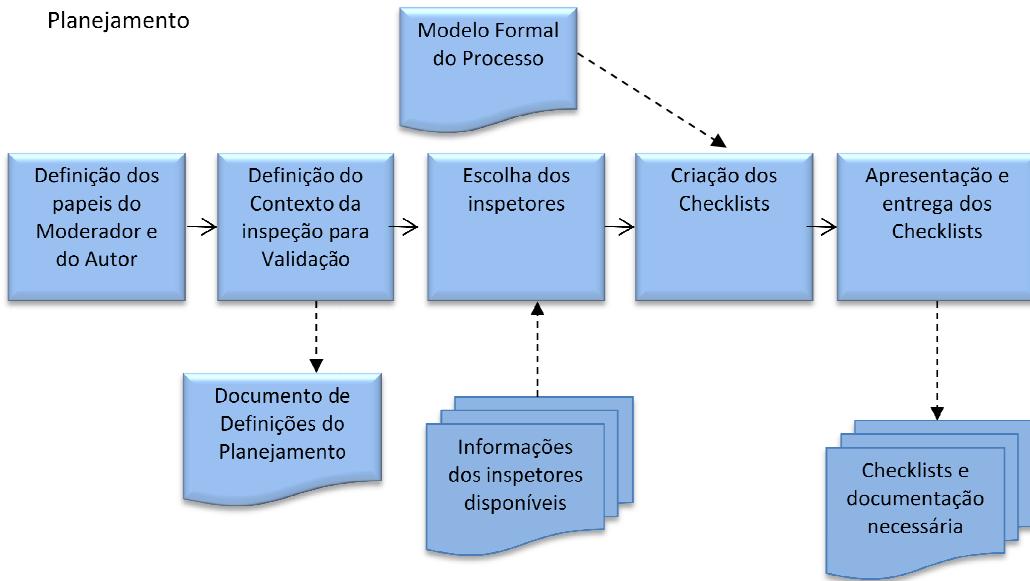


Figura 4.3.2 – Atividades pertencentes à fase de Planejamento

Como mostra a Figura 4.3.2 acima, a atividade de Planejamento conta com o Modelo Formal do Processo que é de responsabilidade do Autor. O Moderador, por sua vez, tem a responsabilidade de levantar e manter as informações dos Inspecionadores disponíveis para a Inspeção. Com esses documentos, o Moderador produz o Documento de Definição, os *checklists* e um relatório com a relação de inspetores por *checklists*.

A Figura 4.3.3 mostra o Documento de Definições do Planejamento. Ele contém informações definidas no planejamento, levantando informações do contexto em que a validação será realizada. Uma vez armazenadas, as informações ficam disponíveis para futuras consultas e atende a uma diretriz proposta por Balci (1995), que afirma que os documentos produzidos durante a validação devem estar disponíveis para consulta e de preferência junto com os documentos do software ou modelo validado.

Nesse documento constam os nomes dos participantes assumindo seus papéis, além de informações do processo e sobre e quando a validação ocorreu. Para isso, todos os campos são de preenchimento obrigatório.

VProclnsp – Validação de Processo de Desenvolvimento de Software por Inspeção	
Documento de Definições do Planejamento	
*Moderador.: _____	
*Autor(res): _____	
*Data de Início:	*Data de Término prevista:
*Nome do Processo de Desenvolvimento:	
*Descrição do Processo de Desenvolvimento: [seção reservada para a descrição de informações sobre o processo. O próprio documento formal do processo pode ser anexado.]	
*Aplicar inspeção no final das (os): <input type="checkbox"/> Etapas <input type="checkbox"/> Iterações <input type="checkbox"/> Projeto <input type="checkbox"/> Outros	
*Inspectores Selecionados:	
1) _____	Checklist entregue <input type="checkbox"/>
2) _____	Checklist entregue <input type="checkbox"/>
3) _____	Checklist entregue <input type="checkbox"/>
4) _____	Checklist entregue <input type="checkbox"/>
5) _____	Checklist entregue <input type="checkbox"/>
*Apresentação Realizada <input type="checkbox"/> Sim <input type="checkbox"/> Não *Data da Realização:	
Observações: [seção reservada ao moderador para descrever informações relevantes para o planejamento da validação do processo]	

Figura 4.3.3 – Documento de Planejamento de Validação por Inspeção proposto pelo VProclnsp

A Figura 4.3.4 mostra o Documento de Informações dos Inspectores. O documento funciona como uma ficha cadastral dos inspectores, onde cada inspector recebe um identificador único, para indicar quem é o responsável por um *checklist*, ou pelas discrepâncias encontradas entre o modelo e a execução do processo. Depois que o processo de validação terminar, o Moderador deve atualizar a ficha cadastral com o número de discrepâncias encontradas e incluir no histórico do inspector a sua participação no processo de validação atual. Essas informações ajudarão o próprio Moderador na seleção da próxima equipe de validação em trabalhos futuros.

VProclsp – Validação de Processo de Desenvolvimento de Software por Inspeção				
Documento de Informações dos Inspetores				
*ID:	*Inspetor..: _____			
*Nível de experiência com desenvolvimento de software <input type="checkbox"/> 1 <input type="checkbox"/> 2 <input type="checkbox"/> 3 <input type="checkbox"/> 4 <input type="checkbox"/> 5				
*Nível de experiência com o domínio do ambiente processo de desenvolvimento <input type="checkbox"/> 1 <input type="checkbox"/> 2 <input type="checkbox"/> 3 <input type="checkbox"/> 4 <input type="checkbox"/> 5				
*Nível de experiência com validação de processos <input type="checkbox"/> 1 <input type="checkbox"/> 2 <input type="checkbox"/> 3 <input type="checkbox"/> 4 <input type="checkbox"/> 5				
1-Nenhum	2-Pouco	3-Razoável	4-Suficiente	5-Alto
*Nome dos Processos de Desenvolvimento já validados pelo inspetor 1) _____ 2) _____ 3) _____ 4) _____ 5) _____				
*Estatística com o número de erros por categorias levantados pelo inspetor Omissão _____ Ambigüidade _____ Inconsistência _____ Outro _____				
Observações: [seção reservada ao moderador para descrever informações relevantes sobre o inspetor]				

Figura 4.3.4 – Documento de Cadastro de Inspetores proposto pelo VProclsp

A Figura 4.3.5 mostra o Documento de Atribuição de Inspetores. Este documento é o último documento criado na fase de Planejamento. Ele descreve os deveres de cada inspetor no processo de validação. Essa informação é útil para que o Moderador tenha conhecimento de quem foi responsável por cada item pertencente a cada *checklist*. Assim os inspetores podem responder dúvidas existentes durante as reuniões.

Neste documento todos os campos são de preenchimento obrigatório e os dados são basicamente o identificador único do inspetor, o identificador único do *checklist* e a lista com os itens do *checklist*. Com essas informações, o Moderador consegue descobrir o que cada inspetor inspecionou.

VProclnsp – Validação de Processo de Desenvolvimento de Software por Inspeção	
Documento de atribuições de Inspetores	
ID:	Inspetor...:
Lista com os identificadores dos checklists sob responsabilidade do inspetor.	
ID:	Inspetor...:
Lista com os identificadores dos checklists sob responsabilidade do inspetor.	
ID:	Inspetor...:
Lista com os identificadores dos checklists sob responsabilidade do inspetor.	
Observações: [seção reservada ao moderador para descrever informações relevantes sobre as atribuições dos inspetores]	

Figura 4.3.5 – Documento de Atribuições de Inspetores proposto pelo VProclnsp

Análise e Comparação

Na fase de Análise e Comparação, os inspetores individualmente analisam os artefatos e os compararam com o modelo formal do processo, guiados pelos *checklists*, identificando discrepância entre os dois conjuntos de documentos.

É importante ressaltar que na maioria das vezes, os inspetores terão que investigar todas as informações produzidas durante o processo de desenvolvimento. Como mencionado anteriormente, o termo artefato no VProclnsp é mais abrangente e envolve, por exemplo, logs e informações produzidos por softwares utilizados durante o processo de desenvolvimento.

Durante a execução dessa fase, o inspetor deve registrar e classificar as discrepâncias em um relatório. A classificação é feita de acordo com o tipo de característica da discrepância, usando para isso a taxonomia de defeitos que

fora definida na Seção 4.2.1.3. A Figura 4.3.6 mostra o fluxo de atividades dessa fase e os documentos de entrada e saída.

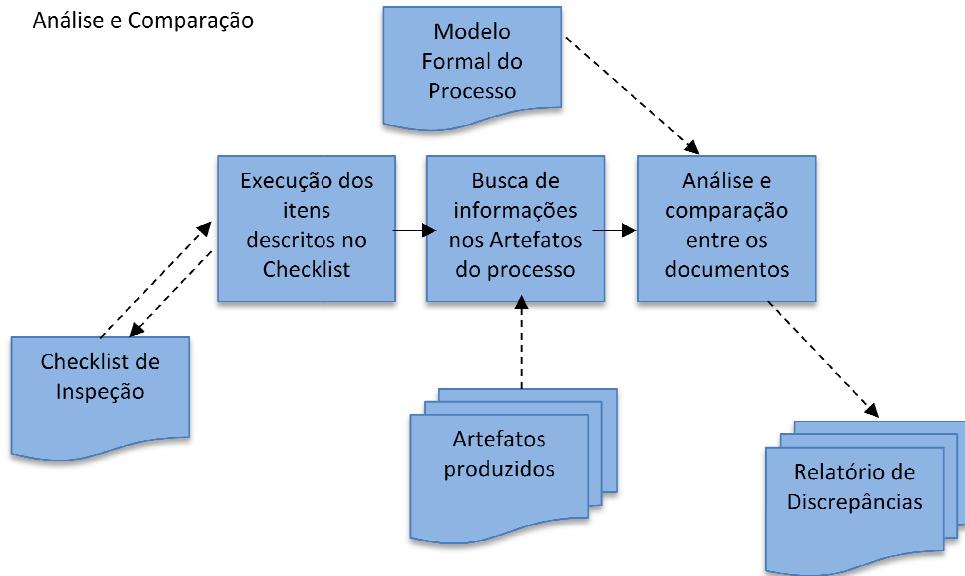


Figura 4.3.6 – Atividades pertencentes à fase de Análise e Comparação

A fase de Análise e Comparação conta com os *checklists* criados pelo Moderador na fase de Planejamento, com os documentos do modelo formal do processo, caso os inspetores julguem insuficientes os dados do *checklist*, e com os artefatos criados durante a execução do processo de desenvolvimento. Com esses documentos em mãos, os inspetores comparam o modelo formal com a execução. Os resultados dessa fase são o *checklist* e os relatórios de discrepâncias preenchidos.

O documento de *checklist* foi apresentado na Seção 4.2.1.4 e o Relatório de Discrepância é mostrado na Figura 4.3.7. Neste documento, o Inspetor deve preencher o seu identificador único e qual item do *checklist* foi encontrado uma discrepância. Deve ser preenchido, também, o campo “Descrição” com informações de como a discrepancia acontece. O inspetor não deve preencher o campo “Discrepância Aceita”. Ele é preenchido pelo Moderador na fase de Coleção e confirmado, ou não, na fase de Discriminação. Logo, o inspetor deve desconsiderar esse campo.

VProInsp – Validação de Processo de Desenvolvimento de Software por Inspeção	
Relatório de Discrepâncias	
Taxonomia de Defeitos: [seção informada pelo moderador para descrever a taxonomia de defeitos utilizada. Esta seção tem o intuito de orientar o inspetor a classificar as discrepâncias encontradas.]	
*ID Inspetor.....: _____	*Inspetor...: _____
*Nº Discrepância: _____	*ID Checklist.: _____ *ID Item do Checklist.: _____
*b Descrição: 	
*b Classificação da Discrepância: <input type="checkbox"/> Omissão <input type="checkbox"/> Ambigüidade <input type="checkbox"/> Inconsistência <input type="checkbox"/> Outros Discrepância aceita: <input type="checkbox"/> Sim <input type="checkbox"/> Não – Motivo: _____	
*Nº Discrepância: _____	*ID Checklist.: _____ *ID Item do Checklist.: _____
*b Descrição: 	
*b Classificação da Discrepância: <input type="checkbox"/> Omissão <input type="checkbox"/> Ambigüidade <input type="checkbox"/> Inconsistência <input type="checkbox"/> Outros Discrepância aceita: <input type="checkbox"/> Sim <input type="checkbox"/> Não – Motivo: _____	
*Nº Discrepância: _____	*ID Checklist.: _____ *ID Item do Checklist.: _____
*b Descrição: 	
*b Classificação da Discrepância: <input type="checkbox"/> Omissão <input type="checkbox"/> Ambigüidade <input type="checkbox"/> Inconsistência <input type="checkbox"/> Outros Discrepância aceita: <input type="checkbox"/> Sim <input type="checkbox"/> Não – Motivo: _____	
Observações: 	
Quantidade total de discrepâncias levantadas: _____	

Figura 4.3.7 – Relatório de Discrepância proposto pelo VProclnsp

Coleção

Na etapa de Coleção, o Moderador da inspeção tem acesso a todas as listas de discrepâncias através dos Relatórios de Discrepâncias produzidos pelos Inspetores na atividade de Análise e Comparação. O Moderador poderá, então, selecionar discrepâncias destas listas e as descartar ou classificar como replicadas, caso haja mais de uma discrepância representando o mesmo defeito. Quando uma discrepância é descartada, ela é retirada das próximas atividades do VProInsp.

Quando discrepâncias são classificadas como replicadas, por sua vez, fica sob responsabilidade do moderador selecionar uma das versões da discrepância. A versão selecionada poderá ainda ser editada, de modo que informações importantes contidas nas réplicas possam ser acrescentadas. Deve-se lembrar que essas réplicas devem entrar nos dados estatísticos do inspetor que levantou tais discrepâncias.

O Inspetor nessa etapa deve preencher o campo do Relatório de Discrepâncias informando se a discrepância foi, ou não, aceita. Em caso negativo, ele deve informar também o motivo.

Discriminação

O Moderador, o Autor e os Inspetores têm participação nesta atividade. Durante a discriminação, as discrepâncias são tratadas como tópicos de discussão. Cada participante pode acrescentar seus comentários relativos a cada uma das discrepâncias, que fica disponível, como tópico de discussão enquanto o Moderador e o Autor não decidem se o item representa realmente uma discrepância ou não.

Por fim, se de fato a discrepância for constatada, o Moderador deve marcar no Relatório de Discrepâncias que ela foi realmente aceita e deve-se atualizar as estatísticas do inspetor responsável por essa discrepância.

É importante ressaltar que a solução para as discrepâncias não são discutidas nessa etapa. Esta é uma tarefa do autor do processo. Logo, cabe ao autor gerar, da forma que lhe convier, à documentação para que alterações no processo sejam feitas.

Retrabalho

Nessa fase, o autor corrige as discrepâncias detectadas durante a inspeção.

Continuação

O material corrigido pelos autores é repassado para o Moderador, que faz uma análise da inspeção como um todo e re avalia a qualidade do artefato

inspecionado. Ele tem a liberdade de decidir se uma nova inspeção deve ocorrer ou não.

Este capítulo apresentou uma proposta de uma técnica de validação de processos de desenvolvimento utilizando-se de inspeção de software. O uso de inspeção visa comparar o modelo formal do processo com sua execução, validando o processo de forma metódica e simples.

Metódica por ser composta de atividades e documentos pré-estabelecidos para cada uma das cinco fases que compõe a técnica, e simples por utilizar *checklists* compostos por um conjunto de questionamentos com a função de guiar os inspetores durante a validação.

Os conceitos e aplicação da técnica de inspeção em software têm resultados conhecidos, mas a aplicação de inspeção em processos é uma nova proposta que deve ser avaliada. Assim, o primeiro passo em direção a avaliar a técnica proposta foi um estudo experimental apresentado no capítulo seguinte.

CAPÍTULO 5

AVALIAÇÃO DA TÉCNICA DE VALIDAÇÃO DE PROCESSO POR INSPEÇÃO PROPOSTA

É descrito neste capítulo o estudo de caso realizado para verificar a viabilidade da solução implementada, que teve como objetivo avaliar a hipótese que orienta essa dissertação. O estudo de caso foi realizado em um cenário real de desenvolvimento de software fazendo uso de um processo de desenvolvimento de software bem documentado.

5.1 Discussão

O VProcInsp, apresentado no capítulo anterior, foi desenvolvido com um embasamento teórico adquirido nos estudos de Processo de Desenvolvimento de Software e de conceitos de Verificação e Validação apresentados na literatura, além da experiência profissional dos envolvidos na pesquisa com o uso de boas práticas da Engenharia de Software no desenvolvimento de sistemas. Mas, não se pode assegurar, apenas com esse conhecimento adquirido, que a técnica proposta obtenha os resultados esperados. Portanto, além de desenvolver o método de validação, é importante avaliar se ele atende aos objetivos inicialmente definidos.

Para que isso seja possível, diversas perguntas devem ser respondidas até que se possa afirmar que o VProcInsp realmente atende aos seus objetivos. Entre os questionamentos possíveis, procura-se responder, inicialmente, se é viável utilizar a abordagem proposta na validação de processos e se ela é eficaz.

Segundo Amaral (2003), experimentação é o centro do processo científico e novas técnicas, métodos, linguagens e ferramentas não devem ser apenas propostas, elas devem passar pelo menos por uma avaliação inicial.

Em suma, a utilização de métodos experimentais pode trazer os seguintes benefícios (Tichy, 1998):

- Ajudar a construir uma base confiável de conhecimento e desta forma reduzir a incerteza sobre quais teorias, métodos e ferramentas são adequados;
- Poder levar a novas compreensões sobre áreas atualmente pesquisadas;
- Explorar áreas desconhecidas, permitindo estender a fronteira do conhecimento;
- Acelerar o progresso através da rápida eliminação de abordagens não fundamentadas, suposições errôneas e modismos, ajudando a orientar a engenharia e a teoria para direções promissoras;
- Ajudar no processo de formação da Engenharia de Software como ciência, através do crescimento do número de trabalhos científicos com uma avaliação experimental significativa.
- Na demonstração dos benefício de uma nova tecnologia e no registro de experiências em relação a sua utilidade;

Segundo Wohlin (2000), a escolha dos métodos experimentais depende dos pré-requisitos da investigação, do propósito, da disponibilidade de recursos e de como se pretende analisar os dados coletados. Wohlin classifica os métodos como:

- *Survey*: investigação executada em retrospecto quando, por exemplo, uma ferramenta ou técnica tem sido utilizada em uma empresa e pretende-se avaliá-la sob algum aspecto. Algumas formas para coleta de dados como entrevistas e questionários são bastante conhecidas e utilizadas.
- Estudos de caso: usados para monitorar projetos, atividades ou exercícios, objetivando rastrear um atributo específico, ou estabelecer

relacionamentos entre diferentes atributos, sem um controle muito formal sobre as atividades relacionadas ao método experimental.

- Experimentos: atividades com o propósito de descobrir algo desconhecido ou de testar uma hipótese envolvendo uma investigação de coleta de dados e de execução de uma análise para determinar o significado dos dados. Experimentos são apropriados para confirmar teorias, o conhecimento convencional, explorar os relacionamentos, avaliar a predição dos modelos ou validar as medidas. As suas principais características são de permitir o controle total sobre processo e variáveis e de ser possível repeti-lo.

No contexto dessa pesquisa, adotou-se o estudo de caso levando em consideração a idéia de introduzir o VProclnsp em um ambiente industrial, que já utiliza um processo de desenvolvimento validado no meio em que está sendo aplicado. Caso contrário, seria questionável aplicar uma técnica que está sendo avaliada em um processo de desenvolvimento que também precisa passar por uma avaliação. É importante ressaltar que aplicar um método imaturo em um ambiente real com o intuito de obter resultados confiáveis é perigoso.

O principal objetivo do estudo de caso realizado é o de identificar e solucionar possíveis problemas observados durante o experimento, para que a tecnologia em estudo se torne mais madura e seus resultados não sejam oriundos de variações humanas ou erros experimentais. O estudo de caso proposto por Wohlin *et al.* (2000) consiste nas seguintes etapas, assim ordenadas: 1) Definição, 2) Planejamento, 3) Operação, 4) Análise e Interpretação e 5) Apresentação e Empacotamento.

A definição é a primeira atividade, onde o experimento é expresso em termos dos problemas e objetivos. A atividade de planejamento vem em seguida, onde o projeto do experimento é determinado, a instrumentação é considerada e os aspectos da validade do experimento são avaliados. A Execução do experimento segue o planejamento. Neste momento, os dados experimentais são coletados para serem analisados e avaliados na atividade de análise e interpretação. Finalmente, os resultados são apresentados e empacotados.

As três subseções abaixo apresentam as primeiras etapas desse processo experimental. A Análise e Interpretação são realizadas na Seção 5.5 e 5.6 fazendo uso de alguns documentos produzidos pela técnica proposta, a fim de simplificar e proporcionar um maior entendimento dos resultados, a etapa de Apresentação e Empacotamento pode ser vista nos Apêndice A e B, que contém o restante dos documentos produzidos no experimento.

5.2 Definição

A fase de definição do estudo experimental descreve os objetivos de sua realização. No contexto desse trabalho, o objetivo é avaliar se o apoio fornecido pelo VProclnsp para a atividade de validação de processos de desenvolvimento é adequado e se traz benefícios. Assim pretende-se responder as seguintes perguntas:

Q1 – O método é facilmente aplicável (simples, adaptável, extensível) e conduz os envolvidos na validação do processo de forma correta?

Q2 – A documentação proposta pelo VProclnsp é realmente necessária e fornece informações úteis?

Q3 – É possível comparar o modelo formal com a execução do modelo?

5.3 Planejamento

Segundo Wohlin *et al.* (2000), a fase de planejamento descreve como o experimento deve ser realizado. Esta descrição envolve: (1) a seleção do contexto, (2) a formulação das hipóteses, (3) a seleção das variáveis, (4) a seleção dos indivíduos, (5) o projeto do experimento, (6) instrumentação e (7) a avaliação da validade.

De acordo com Wohlin *et al.* (2000), a **seleção do contexto** pode ser caracterizada em quatro dimensões: (1) processo – monitoramente contínuo ou

não contínuo, (2) participantes – acadêmicos ou profissionais, (3) realidade – problema real ou modelado e (4) generalidade – geral ou específica. A **formulação da hipótese** estabelece uma ou mais hipóteses preliminares que deduzem as conclusões e ao final do estudo as hipóteses devem ser falseadas ou confirmadas. A **seleção das variáveis** é a escolha das características medidas, controladas ou manipuladas durante a pesquisa. Elas podem ser: (1) Independentes da reação inicial, intenções ou ações dos envolvidos na pesquisa, ou (2) Dependentes da manipulação ou das condições experimentais, ou seja, elas dependem do que as pessoas farão. Na **seleção dos indivíduos**, pessoas com o perfil desejado para executar ou fornecer dados para a pesquisa são selecionadas. O **projeto de experimento** determina a maneira como um experimento será conduzido. A decisão sobre a alocação dos objetos e dos participantes é feita nesse momento, logo, a **instrumentação**, que são todos os objetos ligados à pesquisa, está vinculada ao projeto de experimento. Por fim, a **avaliação da validade** é executada mostrando o quão válido são os resultados.

A Tabela 5.1 mostra o planejamento do estudo de caso realizado nesta dissertação seguindo a descrição apresentada acima.

Tabela 5.1 – Planejamento de experimento.

Seleção de Contexto: Esse estudo de caso se propõe a ser executado sem um monitorando contínuo das atividades, pois os dados serão coletados de informações existentes apenas em duas fases do processo de desenvolvimento, e os participantes são profissionais da área de desenvolvimento e qualidade de softwares aplicando o VProclnsp em um ambiente real de desenvolvimento de sistemas.
Formulação da Hipótese: Com vista na idéia apresentada anteriormente de que um processo é válido se é tangível de ser seguido e faz o que se propõe, a seguinte hipótese nula foi elaborada: <ul style="list-style-type: none">• Hipótese Nula: A utilização do VProclnsp não permite aos participantes do processo de validação comparar o modelo formal com a execução do processo. Com os resultados da fase de análise e interpretação dos dados do estudo de caso, pretende-se chegar a uma conclusão sobre a possibilidade da rejeição desta hipótese nula.
Seleção das variáveis: Esse experimento utiliza as seguintes variáveis: <ul style="list-style-type: none">• Variável Independente:<ul style="list-style-type: none">○ A documentação do Modelo formal do Processo de Desenvolvimento de Software sob validação.• Variável Dependente:<ul style="list-style-type: none">○ Os documentos gerados pelo VProclnsp.○ Os artefatos gerados pela execução do processo.
Seleção dos indivíduos: Os participantes devem deter conhecimento sobre o VProclnsp e/ou sobre o Processo de desenvolvimento de software e se possível experiência com Inspeções de Software. Os participantes selecionados irão assumir os papéis pré-estabelecidos pelo VProclnsp. Assim, um participante assume o papel do Moderador, outro, o papel de Autor e duas pessoas têm a função de Inspetor.
Projeto do Experimento: O projeto de estudo experimental segue o fluxo normal das fases proposta pelo VProclnsp, atribuindo, assim, os papéis, as atividades e os documentos que devem ser executados. Logo, os participantes devem aplicar o VProclnsp em um ambiente industrial coletando as informações necessárias para a validação do processo, mas com o foco central em analisar o VProclnsp. Como o estudo de caso tem o foco em analisar a técnica proposta e não o processo de software, a fase de Continuação apresentada na Seção 4.2.1.5, a princípio, não se faz necessária, pois, a continuação é a re-execução da validação em pontos reestruturados do processo.
Instrumentação: Na realização do experimento, os dois inspetores utilizarão como instrumento os checklists criados pelo Moderador. A partir destes checklists, os inspetores criam os relatórios de discrepâncias que se tornam o instrumento para que o moderador e o autor discutam e definam se realmente as informações são relevantes à validação do processo ou não. Sendo que, o resultado dessa definição é utilizado pelo Autor na tentativa de executar as adaptações necessárias para que o processo seja considerado válido.
Avaliação da Validade: O estudo envolve profissionais da área de desenvolvimento e de engenharia de software em um ambiente real de desenvolvimento. Mas, somente com esse estudo de caso, não é possível generalizar os resultados para todos os ambientes de desenvolvimento, sendo necessário para isto realizar novos estudos de casos. Esse estudo, porém, irá dar subsídios para que melhorias e confirmações sejam realizadas no processo e, principalmente, no VProclnsp.

5.4 Operação

O experimento foi realizado na empresa TOTVS S.A. que segundo a 19ª Pesquisa de Informática realizada pela Fundação Getúlio Vargas é a líder do mercado nacional em pacotes ERP e atua em países como México, Argentina e Portugal. A empresa agrupa as marcas Microsiga, Logocenter e RM Sistemas tendo uma base de aproximadamente quinze mil clientes. O estudo foi realizado exclusivamente sobre a marca RM Sistema que tem sua matriz na cidade de Belo Horizonte (FGV, 2008).

A empresa foi selecionada por usar um processo validado que controla com eficiência todo o setor de desenvolvimento de sistemas, gerenciando os mais de 200 participantes que estão envolvidos diretamente com os produtos RM. Outro ponto importante, é que o processo de desenvolvimento dessa empresa é certificado CMMI nível 2 ou MPS.BR nível F.

Não é o intuito deste trabalho, e nem de interesse da empresa, detalhar todo o processo utilizado pela RM Sistemas. Mas o processo segue a tendência mundial no uso de um processo iterativo e incremental com suas fases bem definidas, organizando a construção, implantação e manutenção. O processo lembra as características do Processo Unificado, propondo artefatos comuns e deixando a cargo dos *stakeholders* a escolha do que é realmente necessário, para que a burocracia não atrapalhe o desempenho dos mesmos.

O processo utilizado pelo RM faz uso da UML para modelagem dos seus projetos que adotam o paradigma orientado a objetos com uma arquitetura compostas em duas ou três camadas. A Figura 5.1, mostra, de forma geral, o processo de desenvolvimento criado pela RM Sistemas.

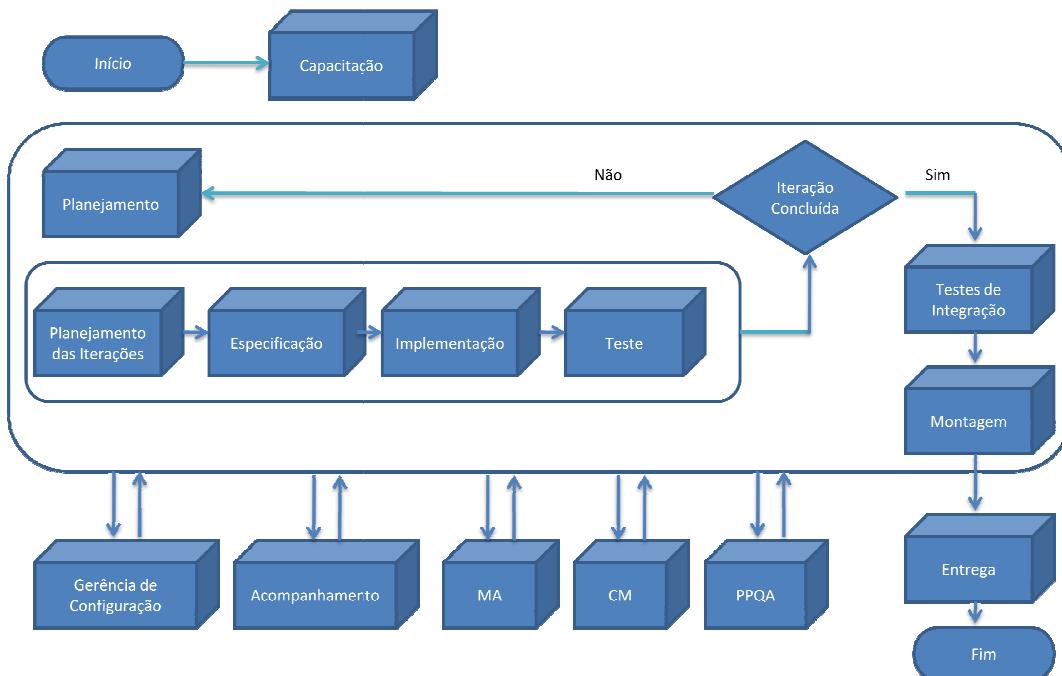


Figura 5.1 – Processo de desenvolvimento criado pela RM Sistemas

Para a aplicação do VProClnsp foi selecionado o produto RM Solum que visa a Gestão de Projetos e Obras. Logo, toda a análise foi realizada com a equipe e os dados gerados por esse produto. O RM Solum foi escolhido por ser um software que está há vários anos no mercado e já passou por todas as fases de desenvolvimento. Atualmente, o sistema vem sendo migrado para a plataforma *Dot Net* e essa migração acontece com uma grande preocupação com a qualidade, que não pode ser alcançada sem o uso do processo de desenvolvimento de software da empresa.

A proposta inicial do estudo de caso era validar todo o processo de software da RM Sistemas. Contudo, devido a limitações impostas pelo contexto em que o estudo foi realizado, somente duas etapas do processo de desenvolvimento foram validadas. Essa limitação ocorreu por motivos estratégicos da empresa de não divulgar, mesmo que em um meio acadêmico, todo o processo de desenvolvimento de software e porque não foi possível para a empresa disponibilizar pessoal nos diversos setores que o processo de desenvolvimento se aplica.

Outra limitação foi quanto a questão de permissão de acesso aos softwares e a documentos, que é uma tarefa cara e sigilosa. Primeiramente, os participantes

da pesquisa deveriam receber tais permissões e isto demandaria um trabalho extra para o pessoal de infra-estrutura de tecnologia. Segundo, informações sigilosas seriam expostas sem necessidade do ponto de vista estratégico da empresa.

Sendo assim, as etapas escolhidas dentre as opções apresentadas para a realização do estudo de caso foram a de Especificação e Implementação. As Figuras 5.2 e 5.3 mostram respectivamente essas duas fases do processo, nota-se, nas duas figuras, que as atividades seguem um fluxo pré-estabelecido e envolve pessoas com três papéis definidos fazendo uso e produzindo artefatos.

A fase de Especificação possibilita o entendimento da funcionalidade do sistema, em um nível voltado ao negócio, para todos os envolvidos no projeto define, também, a terminologia específica do problema de domínio, explicando termos que possam ser desconhecidos ao leitor, com respeito às descrições dos artefatos gerados.

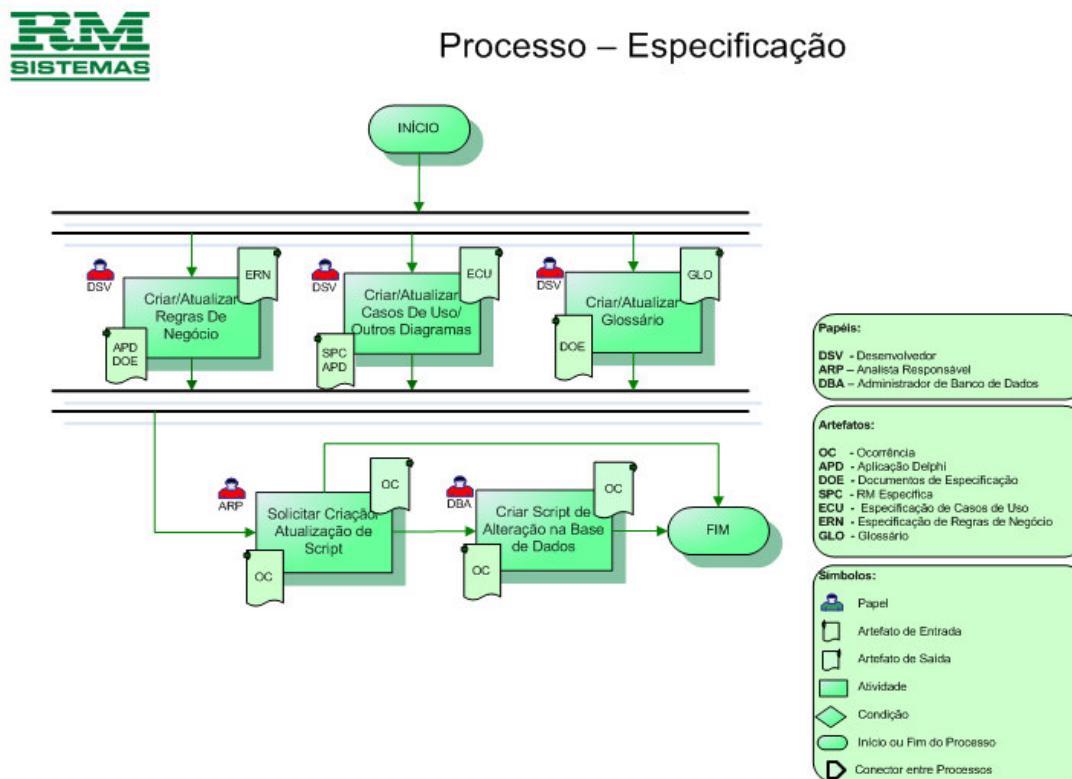


Figura 5.2 – Fase de Especificação do processo de desenvolvimento criado pela RM

A fase de Implementação verifica se todas as informações necessárias à implementação estão contidas na especificação dos requisitos, identificando eventuais inconsistências. Também são atividades dessa fase: escrever o código que implementa o requisito, armazenar o código fonte em lugar centralizado, de forma a possibilitar execução de backups, e verificar se o código fonte funciona efetivamente.

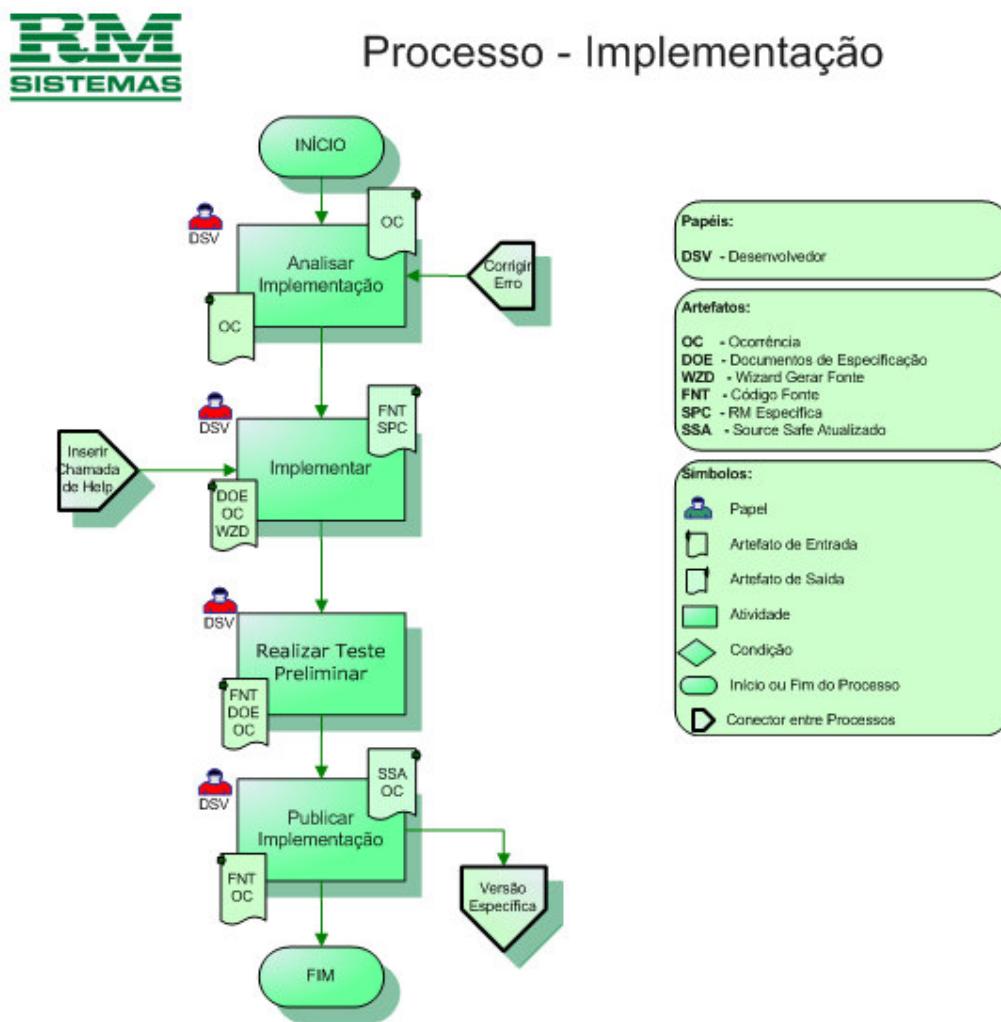


Figura 5.3 – Fase de Implementação do processo de desenvolvimento criado pela RM

O estudo de caso foi conduzido, seguindo a técnica de validação definida pelo VProclnsp, onde a primeira atividade a ser realizada é a de planejamento da inspeção. Durante esse planejamento, o escopo da inspeção foi definido e os participantes foram escolhidos. Os *checklists* foram configurados e apresentados aos inspetores.

A definição do escopo, ou seja, a escolha de atuar em apenas duas fases do processo, foi tomada em uma reunião com uma funcionária que faz parte da equipe de engenharia de software da empresa. Nessa ocasião a mesma aceitou o convite para fazer parte do estudo de caso, assumindo o papel de Autor (a).

Nessa mesma reunião ficou acordado que a análise contemplaria dez requisitos de software. Assim, os inspetores puderam aplicar o *checklist* nas duas fases do processo, comparando o modelo formal com a execução do aplicado sob dez requisitos escolhidos aleatoriamente. O objetivo foi analisar se os requisitos seguiram as atividades ou tarefas como o modelo formal propõe.

É importante ressaltar que a quantidade de requisitos analisados e a escolha das duas fases do processo, não seriam suficientes para validar inteiramente um processo de desenvolvimento. Mas, como o objetivo não é validar o processo, e sim analisar a aplicabilidade e eficiência do VProclnsp, considera-se esse número razoável.

Com o escopo definido, o próximo passo foi a definição dos participantes do estudo de caso. O próprio pesquisador exerceu o papel de Moderador. Essa decisão foi tomada para facilitar a identificação das necessidades de melhorias nos documentos propostos e para que o mesmo tenha um contato direto no processo de validação. Foi definido entre o Moderador e o Autor que dois inspetores seriam suficientes onde cada um inspecionou uma etapa do processo.

Os inspetores foram selecionados levando em consideração a experiência com o processo da empresa e com o desenvolvimento de software, além de terem acesso aos artefatos produzidos pelo produto RM Solum. A Figura 5.4 mostra de forma simples o escopo e o papel dos inspetores definido na etapa de planejamento do VProclnsp.

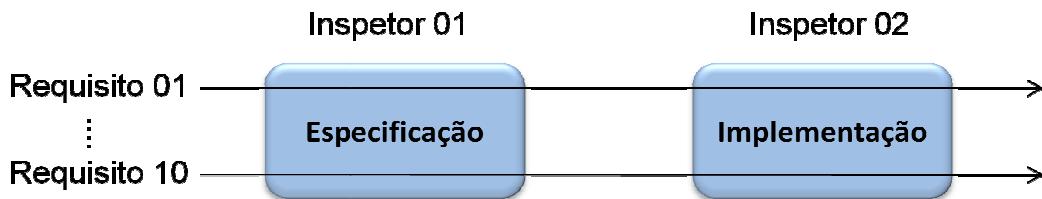


Figura 5.4 – Contexto em que o VProclnsp foi aplicado no estudo de caso

A etapa de planejamento foi encerrada com uma reunião entre os quatro participantes do VProclnsp e com os analistas da empresa. Neste Momento foram entregues os *checklists*, os objetivos da pesquisa foram explicados, alguns conceitos de inspeção de software foram apresentados e dúvidas foram esclarecidas.

Com os *checklists* em mãos, os dois inspetores analisaram os artefatos produzidos seguindo os itens elaborados pelo moderador. Ao final dessa atividade os inspetores preencheram o Relatório de Discrepância, que é um centralizador das informações coletadas e ajuda o Moderador na etapa de Coleção.

Seguindo o fluxo normal das fases propostas pelo VProclnsp, o Moderador analisou as discrepâncias como propõe a etapa de Coleção apresentada na Seção 4.2.1.5, e apesar de constatar que os cinco primeiros itens do relatório estavam ligados à criação de diagramas UML, ele optou por não agrupar ou descartar esses itens, pois cada um estava em atividades diferentes do processo (Veja os itens na Figura 5.6). Logo, esses itens poderiam incrementar a discussão sobre o processo na etapa de discriminação do VProclnsp. Nesse momento, o Moderador também atualizou as estatísticas dos inspetores informando a quantidade de discrepâncias por omissão, ambigüidade e inconsistência que os mesmos levantaram na participação dessa inspeção.

A validação do processo encerrou-se com a etapa de discriminação, pois o VProclnsp não contempla a fase de Retrabalho e como descrito na Tabela 5.1, não iria ser realizado outra validação do processo que corresponde a fase de Continuação.

A discriminação foi realizada pelo Moderador, pelo Autor, por dois inspetores e por um analista, que trataram todos os itens do *checklist* como pontos de discussão, com ênfase nos sete tópicos de discrepancia apontados no relatório referente à fase de Especificação do processo. Como pode ser visto na Figura 5.5, o relatório de discrepancias da fase de Implementação está vazio, porque não foram encontradas evidencias de que o modelo e a execução estavam sendo aplicados de forma diferente.

VProInsp – Validação de Processo de Desenvolvimento de Software por Inspeção	
Relatório de Discrepâncias	
Taxonomia de Defeito:	
<i>Omissão:</i>	
<ul style="list-style-type: none"> ✓ Quando um dos elementos (artefato, atividade, tarefa, etapa) necessários ao processo não foram definidos. ✓ Quando faltam seções de especificação de como os artefatos, atividades, tarefa e etapas serão definidos. ✓ Quando falta definição de termos utilizados no processo. 	
<i>Ambigüidade:</i>	
<ul style="list-style-type: none"> ✓ Quando a forma como os elementos do processo ou suas responsabilidades foram definidos dificultam ou impossibilitam seu uso. ✓ Quando elementos possuem o mesmo nome, mas responsabilidades diferentes (Homônimo); 	
<i>Inconsistência:</i>	
<ul style="list-style-type: none"> ✓ Quando elementos descritos possuem mesma responsabilidade, mas nomes distintos (Sinônimo) ✓ Quando um elemento presente na execução do processo não foi definido no modelo formal ✓ Quando a representação não condiz com a semântica estabelecida pela abordagem de documentação. ✓ Quando as responsabilidades dos stakeholders não foram definidas. 	
<i>Outro:</i>	
Quando o defeito não se encaixa nas categorias acima.	
ID Inspetor.....: 00001	Inspetor.: Sérgio Gontijo do Nascimento
Observações:	
Não foram encontradas discrepancia para o a fase de Implementação ao executar o checklist com os 19 itens.	
Quantidade total de discrepancias levantadas: 0	
Quantidade total de discrepancias aceitas.....: 0	

Figura 5.5 – Relatório de Discrepância preenchido com os dados do estudo de caso

A Figura 5.6 mostra o relatório de discrepancia da etapa de especificação do processo de desenvolvimento abordado pelo estudo de caso. Nota-se que sete discrepancias foram encontradas, sendo que cinco foram classificadas como omissão e duas classificadas como inconsistência.

<p style="text-align: center;">VProCInsp – Validação de Processo de Desenvolvimento de Software por Inspeção</p> <p style="text-align: center;">Relatório de Discrepâncias</p>		
<p>Taxonomia de Defeito:</p> <p><i>Omissão:</i></p> <ul style="list-style-type: none"> ✓ Quando um dos elementos (artefato, atividade, tarefa, etapa) necessários ao processo não foram definidos. ✓ Quando faltam seções de especificação de como os artefatos, atividades, tarefa e etapas serão definidos. ✓ Quando falta definição de termos utilizados no processo. <p><i>Ambigüidade:</i></p> <ul style="list-style-type: none"> ✓ Quando a forma como os elementos do processo ou suas responsabilidades foram definidos dificultam ou impossibilitam seu uso. ✓ Quando elementos possuem o mesmo nome, mas responsabilidades diferentes (Homônimo); <p><i>Inconsistência:</i></p> <ul style="list-style-type: none"> ✓ Quando elementos descritos possuem mesma responsabilidade, mas nomes distintos (Sinônimo) ✓ Quando um elemento presente na execução do processo não foi definido no modelo formal ✓ Quando a representação não condiz com a semântica estabelecida pela abordagem de documentação. ✓ Quando as responsabilidades dos stakeholders não foram definidas. <p><i>Outro:</i></p> <ul style="list-style-type: none"> ✓ Quando o defeito não se encaixa nas categorias acima. 		
ID Inspetor.....: 00002	Inspetor..: Túllio Mendes Rodrigues	
Nº Discrepância: 01	ID Checklist.: 001	ID Item do Checklist.: 02
<p>Descrição:</p> <p><i>Não foram encontrados diagramas UML para os dez requisitos inspecionados. Logo, não foram usados os documentos expostos no item como entrada.</i></p>		
<p>Classificação da Discrepância:</p> <p><input checked="" type="checkbox"/> Omissão <input type="checkbox"/> Ambigüidade <input type="checkbox"/> Inconsistência <input type="checkbox"/> Outros</p> <p>Discrepância aceita: <input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não – Motivo:</p>		
Nº Discrepância: 02	ID Checklist.: 001	ID Item do Checklist.: 07
<p>Descrição:</p> <p><i>Não foram encontrados diagramas UML para os dez requisitos inspecionados. Logo, não foram produzidos documentos de caso de uso e outros diagramas como saída.</i></p>		
<p>Classificação da Discrepância:</p> <p><input checked="" type="checkbox"/> Omissão <input type="checkbox"/> Ambigüidade <input type="checkbox"/> Inconsistência <input type="checkbox"/> Outros</p> <p>Discrepância aceita: <input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não – Motivo:</p>		
Nº Discrepância: 03	ID Checklist.: 001	ID Item do Checklist.: 12
<p>Descrição:</p> <p><i>Não foram criados diagramas de caso de uso para os requisitos. Logo não havia arquivos para serem atualizados no SourceSafe.</i></p>		
<p>Classificação da Discrepância:</p> <p><input checked="" type="checkbox"/> Omissão <input type="checkbox"/> Ambigüidade <input type="checkbox"/> Inconsistência <input type="checkbox"/> Outros</p> <p>Discrepância aceita: <input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não – Motivo:</p>		
Nº Discrepância: 04	ID Checklist.: 001	ID Item do Checklist.: 21
<p>Descrição:</p> <p><i>Não foram criados diagramas de caso de uso. Logo os templates não foram utilizados.</i></p>		
<p>Classificação da Discrepância:</p>		

<input checked="" type="checkbox"/> Omissão	<input type="checkbox"/> Ambigüidade	<input type="checkbox"/> Inconsistência	<input type="checkbox"/> Outros		
Discrepância aceita: <input checked="" type="checkbox"/> Sim		Não – Motivo:			
Nº Discrepância: 05		ID Checklist.: 001	ID Item do Checklist.: 24		
Descrição:					
<i>Não existem diagramas UML para esses requisitos no SourceSafe. Logo o desenvolvedor não criou nenhum diagrama.</i>					
Classificação da Discrepância:					
<input checked="" type="checkbox"/> Omissão	<input type="checkbox"/> Ambigüidade	<input type="checkbox"/> Inconsistência	<input type="checkbox"/> Outros		
Discrepância aceita: <input checked="" type="checkbox"/> Sim		Não – Motivo:			
Nº Discrepância: 06		ID Checklist.: 001	ID Item do Checklist.: 26		
Descrição:					
<i>Todas as solicitações de criação de script foram feitas pelo próprio desenvolvedor.</i>					
Classificação da Discrepância:					
<input type="checkbox"/> Omissão	<input type="checkbox"/> Ambigüidade	<input checked="" type="checkbox"/> Inconsistência	<input type="checkbox"/> Outros		
Discrepância aceita: <input checked="" type="checkbox"/> Sim		Não – Motivo:			
Nº Discrepância: 07		ID Checklist.: 001	ID Item do Checklist.: Inexistente		
Descrição:					
<i>Ao inspecionar os artefatos para responder o item 26 o inspetor relatou a seguinte discrepância: A solicitação de criação de script é feita na etapa de Implementação e não na Etapa de Especificação.</i>					
<i>Evidência: A data de abertura de ocorrência para os scripts eram posteriores ao apontamento de horas na fase de implementação. Ou seja, as datas de abertura de solicitação eram feitas depois do encerramento da fase de Especificação</i>					
Classificação da Discrepância:					
<input type="checkbox"/> Omissão	<input type="checkbox"/> Ambigüidade	<input checked="" type="checkbox"/> Inconsistência	<input type="checkbox"/> Outros		
Discrepância aceita: <input checked="" type="checkbox"/> Sim		Não – Motivo:			
Observações:					
<i>A discrepância 07 não existia no checklist, mas foi uma discrepância levantada a partir do item 26 e que deve ser verificada.</i>					
Quantidade total de discrepâncias levantadas: 7					
Quantidade total de discrepâncias aceitas.....: 7					

Figura 5.6 – Relatório de Discrepância preenchido com os dados do estudo de caso

Examinando a Figura 5.6 acima, perceber-se que todas as discrepâncias foram aceitas nessa fase de Coleção. Mas, essas informações podem ser alteradas na fase de discriminação porque nessa fase os itens serão alvo de debates, e caso fique acordado que a discrepância não procede, o Moderador remarca o item sob debate como uma discrepância não aceita.

5.5 Análise dos Dados e Resultados Obtidos

A análise feita nessa seção considera os dados produzidos pela abordagem do VProclnsp no estudo de caso, as observações do pesquisador e opiniões dos envolvidos no estudo de caso. Para melhor compreensão, a análise é dividida em: análise dos resultados e análise da utilização da abordagem.

5.5.1 Análise dos resultados da abordagem

Através dos dados presentes no relatório de discrepância levantados no estudo de caso, foram geradas tabelas com o intuito de transformar dados em informações úteis. Essas tabelas são compostas por linhas que representam os dados de cada item pertencente aos *checklists*. Em cada linha existem colunas com as possíveis respostas (Sim, Não e Não Aplicável – NA), do item do *checklist*.

Essas colunas são preenchidas com a quantidade de respostas “Sim”, “Não” ou “NA” para os dez requisitos inspecionados. A Resposta Esperado (RE) é outra coluna pertencente às tabelas. Ela é importante, pois representa a resposta que o modelo formal exige. Logo, se as respostas encontradas pelo inspetor são iguais às respostas esperadas, a execução do processo segue o modelo formal.

É importante ressaltar dois pontos nessa análise: (1) alguns itens dos *checklists* não são aplicáveis a todos os requisitos analisados e, por isso, não podem ser considerados discrepâncias. Por exemplo, no item 04 (“*O script e a ocorrência foram encaminhados para o Analista responsável?*”) seis requisitos não apresentaram a necessidade de criação de script, logo essa tarefa não deveria ser executada. Portanto, os requisitos foram classificados como Não Aplicáveis; (2) a discrepância é analisada de acordo com o número de requisitos que não obedeceram à RE, desconsiderando as respostas NA.

Os dados foram analisados em dois momentos distintos. A Tabela 5.2 mostra o primeiro instante dessa análise, onde os resultados foram apresentados sem as discussões proposta pela fase de discriminação do VProclnsp apresentada

na Seção 4.2.1.5, ou seja, a análise foi feita a partir das discrepâncias aceitas somente pelo moderador, sem levar em consideração as opiniões do autor, dos inspetores e dos analistas. Esta tabela é dividida em Tabela 5.2-A e 5.2-B que representam respectivamente os dados da etapa de especificação e implementação. Estas tabelas são importantes, pois, se pode comparar de forma simples o que era esperado com o que realmente foi executado no processo de desenvolvimento de software.

Tabela 5.2 – Informações levantadas pelo VProclsp antes da fase de Discriminação

Itens do Checklist	RE	Sim	Não	NA
1	Sim	10	0	0
2	Sim	0	10	0
3	Sim	10	0	0
4	Sim	4	0	6
5	Sim	4	0	6
6	Sim	10	0	0
7	Sim	0	10	0
8	Sim	10	0	0
9	Sim	4	0	6
10	Sim	4	0	6
11	Sim	10	0	0
12	Sim	0	10	0
13	Sim	10	0	0
14	Sim	4	0	6
15	Sim	10	0	0
16	Sim	10	0	0
17	Sim	10	0	0
18	Sim	10	0	0
19	Sim	10	0	0
20	Sim	10	0	0
21	Sim	0	10	0
22	Sim	10	0	0
23	Sim	10	0	0
24	Sim	0	10	0
25	Sim	10	0	0
26	Sim	0	4	6
27	Sim	4	0	6
28	Sim	0	4	6

Tabela A – Dados levantados na etapa de Especificação

Itens do Checklist	RE	Sim	Não	NA
1	Sim	10	0	0
2	Sim	10	0	0
3	Sim	10	0	0
4	Sim	10	0	0
5	Sim	10	0	0
6	Sim	10	0	0
7	Sim	10	0	0
8	Sim	10	0	0
9	Sim	10	0	0
10	Sim	10	0	0
11	Sim	10	0	0
12	Sim	10	0	0
13	Sim	10	0	0
14	Sim	10	0	0
15	Sim	10	0	0
16	Sim	10	0	0
17	Sim	10	0	0
18	Sim	10	0	0
19	Sim	10	0	0

Tabela B – Dados levantados na etapa de Implementação

A Tabela 5.2-A e 5.2-B são compostas por linhas representando as respostas obtidas dos 28 itens do *checklist* da etapa de especificação e dos 19 itens do *checklist* da etapa de implementação. Estas tabelas possuem coincidentemente a resposta “Sim” para todas as suas linhas da coluna RE. É dito coincidente, pois, como proposto pelo VProclsp na Seção 4.2.1.5, a RE é informada pelo Moderador na fase de Planejamento durante a atividade de

criação dos *checklists*, e neste estudo de caso, o Moderador atribui o valor “Sim” para todos os itens levando em consideração o processo de desenvolvimento da TOTVS S.A.

A Tabela 5.2-A mostra que os itens 2, 7, 12, 21 e 24 tinham como resposta esperada para os dez requisitos, o “Sim”. No entanto, nenhum desses itens obteve tal resposta, mostrando que os itens não são executados pelos *stackholders* e aparentemente pode indicar uma atividade que não é válida no processo. Os itens 26 e 28 também têm como resposta esperada o “Sim”, mas em ambos, quatro requisitos não obedeceram a essa resposta. Os outros seis não são aplicáveis, sendo assim desconsiderados. Portanto, essa tabela indica que os sete itens do *checklist* (2, 7, 12, 21, 24, 26 e 28) não estão sendo executados apresentando uma discrepância entre o modelo e a prática do processo.

A Tabela 5.2-B mostra que a execução do processo segue o modelo formal proposto, pois os 19 itens do *checklist* têm a mesma resposta que a coluna RE. Logo, pode-se concluir que a execução está seguindo exatamente o que o modelo propõe, não existindo assim discrepâncias nessa etapa.

Como o estudo de caso não apresentou uma descrição aprofundada sobre o processo de desenvolvimento da TOTVS S.A., a Tabela 5.3 explica os itens da Tabela 5.2-A que apresentam discrepâncias para facilitar o entendimento da análise dos resultados.

Tabela 5.3 – Descrição dos itens com discrepâncias do checklist de especificação.

Nº do Item	Descrição do Item	Explicação do item
2	A aplicação Delphi e o RM Especifica contendo todos os requisitos do produto serviram como entrada para criar/atualizar casos de uso e/ou outros diagramas	Como a empresa está migrando seus produtos de Delphi para Dot Net, o processo orienta os desenvolvedores a fazer uso do código fonte em Delphi e também de um software chamado RM Especifica, que centraliza os documentos de modelagem para criar/atualizar os diagramas e casos de uso.
7	Foi gerada a especificação dos casos de uso ou outros diagramas?	A fase de especificação de softwares é fundamental para os processos de desenvolvimento modernos, sabendo da importância desta fase, a TOTVS exige que os diagramas e casos de uso sejam criados quando necessários.
12	Ao final da atividade de criar/atualizar casos de uso e/ou outros diagramas, os documentos de saída foram atualizados no Source Safe?	A empresa usa o software da Microsoft chamado <i>SourceSafe</i> para fazer o controle de versões dos documentos. Logo, os documentos de casos de uso e diagramas devem ser salvos neste software.
21	Os templates disponíveis em SS\RM.NET\Souce\RM.Templates\RM.Docs\E_UC - Caso de Uso.doc foram utilizados ao criar/atualizar casos de uso e/ou outros diagramas?	O processo da TOTVS fornece documentos padrões para a especificação das implementações. Logo, os desenvolvedores devem seguir estes padrões, que ficam disponíveis no <i>SourceSafe</i> .
24	O Desenvolvedor foi responsável por criar/atualizar casos de uso e/ou outros diagramas?	Fica a cargo das pessoas que assumem o papel de Desenvolvedor criar/atualizar os diagramas e caso de uso.
26	Analista Responsável foi o responsável por solicitar ao criação/atualização do script?	A pessoa que assume o papel de Analista Responsável tem a função de gerenciar uma equipe de desenvolvedores, e uma das atribuições do Analista Responsável é fazer a solicitação para a equipe de banco de dados, quando os desenvolvedores especificam que é necessário criar ou alterar a base de dado dos software criados pela empresa. Estas criações/alterações na base de dados são delicadas e, por isso, devem ser feitas pelo Analista Responsável.
28	A solicitação de criação de script é feita na etapa de Especificação	As solicitações de criação ou alteração na base de dados feita pelo Analista Responsável para a equipe de banco de dados da empresa devem ser feitas na fase de especificação

Passado a primeira fase, as tabelas 5.2 A e B e todos os documentos produzidos pelo estudo de caso foram analisados em uma reunião. Nesta reunião proposta pela fase de discriminação do VProclnsp, os itens foram debatidos e os participantes chegaram a um consenso. Ficou acordado que os itens 2, 7, 12, 21 e 24 que são relacionados com os diagramas UML não são considerados discrepâncias. Nessa reunião, o Autor alegou que esses itens relacionados à UML são considerados opcionais pelos *desenvolvedores*, porque somente os documentos de regra de negócio gerados na fase de

entendimento foram suficientes para adquirir o conhecimento do problema. Outra questão apontada pela equipe, é que na empresa ainda não existe um modelo padrão de especificação utilizando linguagem UML para representar aplicações em três camadas. Sem esse padrão as equipes ficam confusas quanto ao nível de detalhes que os diagramas devem atingir. Para resolver esse problema um padrão está sendo proposto e as equipes serão treinadas. Com essas ações o Autor acredita que os diagramas UML serão mais utilizados. Nesse momento, esses cinco itens da fase de especificação tiveram suas respostas no *checklist* alteradas de “Não” para “NA”.

Os itens 26 e 28 foram considerados como discrepâncias. O primeiro item se refere à obrigatoriedade da atividade de solicitação de script ser feita pelo “Analista Responsável” da equipe. Os desenvolvedores declararam que essa atividade é desnecessária, pois eles acabam tendo mais domínio do problema do que o Analista Responsável, o qual concorda com essa discrepância, pois esse é um cargo com muitos afazeres como análise de *bug*, solicitações de implementação, planejamento da versão, acompanhamento de cronograma, entre outras.

O item 28 trata da obrigatoriedade de fazer a solicitação de script na etapa de especificação. Os desenvolvedores alegam que muitas vezes não podem esperar pela criação do script para que seja feita a implementação e, em outras vezes, a especificação não deixou claro se será realmente necessário a alteração na base de dados, sendo que existe um custo para que a base de dados seja alterada para o estado anterior. É importante lembrar que a maioria dos processos de desenvolvimento hoje são incrementais e iterativos. Ou seja, a especificação pode ser alterada várias vezes até que o problema seja finalmente implementado.

Ao final da etapa de discriminação, o relatório de discrepância e a Tabela 5.2-A foram atualizados. A nova tabela pode ser vista abaixo (Tabela 5.4), onde fica claro que as discrepâncias ocorreram somente nos itens 26 e 28. O item 26 serviu de confirmação para uma alteração da fase de Especificação que, coincidentemente, estava sendo feita pela equipe de engenharia de software da empresa. A alteração consistia justamente em retirar da responsabilidade do

analista Responsável a função de solicitação de criação dos *scripts* de banco de dados.

O item 28 foi considerado como uma questão nova e interessante, que pode ser revista no modelo do processo de desenvolvimento. Mas, obviamente precisa de mais estudos e outras opiniões, até mesmo pelo porte da empresa, que possuem aproximadamente 200 pessoas envolvidas diretamente com o processo de desenvolvimento.

Tabela 5.4 – Informações levantadas pelo VProclnsp na etapa de Especificação na fase de Discriminação

Itens do Checklist	RE	Sim	Não	NA
1	Sim	10	0	0
2	Sim	0	0	10
3	Sim	10	0	0
4	Sim	4	0	6
5	Sim	4	0	6
6	Sim	10	0	0
7	Sim	0	0	10
8	Sim	10	0	0
9	Sim	4	0	6
10	Sim	4	0	6
11	Sim	10	0	0
12	Sim	0	0	10
13	Sim	10	0	0
14	Sim	4	0	6
15	Sim	10	0	0
16	Sim	10	0	0
17	Sim	10	0	0
18	Sim	10	0	0
19	Sim	10	0	0
20	Sim	10	0	0
21	Sim	0	0	10
22	Sim	10	0	0
23	Sim	10	0	0
24	Sim	0	0	10
25	Sim	10	0	0
26	Sim	0	4	6
27	Sim	4	0	6
28	Sim	0	4	6

5.5.2 Análise da utilização da abordagem

Em paralelo a utilização e a análise dos resultados do VProclsp, foi realizada uma avaliação da qualidade da técnica proposta considerando dados coletados por observação e por questionários, nos quais, os Inspetores e o Autor responderam questões relacionadas à utilização da técnica.

Por observação, presenciou-se a necessidade de criação de dois campos de informação e uma coluna para os itens do *checklist*. Os campos: “comentários” e “evidência” mostraram-se importantes na etapa de discriminação, deixando os dados mais coerentes e confiáveis. A criação da coluna “Não aplicável” foi fundamental para representar o cenário real da inspeção na etapa de análise e comparação. Conseqüentemente, esses dois campos e a coluna foram integrados ao modelo padrão do *checklist* proposto pelo VProclsp.

A facilidade de inspecionar o processo através de *checklists* foi apontada como vantajosa, pois, segundo os entrevistados, deixa a inspeção automática e simples, sendo caracterizada pelos participantes como um passo a passo, ou seja, os inspetores são guiados pelos *checklists*. Foi ressaltada, também, a boa organização que as etapas propostas pelo VProclsp proporciona.

Pontos negativos, quanto à dificuldade de manipulação dos vários documentos não informatizados e quanto à manutenção da versão digital, feita em um editor de texto foram constatados, esses dados podiam ser facilmente alterados ou mesmo apagados. A ausência de um *feedback* online sobre o andamento da inspeção e um controle de usuários sobre as informações e documentos também foram citados como desvantagens.

5.6 Considerações em relação à hipótese do estudo

Com base na análise dos dados da abordagem apresentada na seção 5.5.1, percebe-se que o VProclsp foi capaz de apontar duas discrepâncias entre a execução do modelo e o modelo formal do processo de desenvolvimento sob estudo. A própria equipe de engenharia de software da empresa afirmou que os resultados são relevantes e que já estavam estudando a alteração no modelo formal do processo para a questão relativa ao item 26 do *checklist*, e o

estudo confirmou essa necessidade. A discrepância no item 28 também foi considerada uma descoberta importante e que será alvo de estudos.

Contudo, as considerações dos itens 26 e 28 podem falsear a hipótese nula formuladas no estudo de caso. O VProclnsp foi capaz de confirmar uma discrepância já conhecida, mas não documentada pela equipe de engenharia de software, e levantar uma nova discrepância em um processo que é utilizado há anos e é considerado validado mesmo que de forma ad-hoc. Logo, o VProclnsp mostrou-se capaz de levantar discrepâncias entre o modelo de processo e a execução do mesmo.

Em relação à hipótese que norteia essa dissertação, o estudo mostrou evidências que ela pode ser atingida. O VProclnsp foi capaz de executar a validação de um processo de forma simples, sem a preocupação com um amplo treinamento e sem a utilização de grande quantidade de recursos humano e financeiro.

O VProclnsp pode ser considerado adaptável, uma vez que os documentos foram modificados para atender as necessidades dos inspetores e autores e, o modo em que a validação foi aplicada (somente em duas etapas do processo durante uma iteração) atendendo as exigências da gerência sobre o controle de acesso de documento.

Não foram executadas operações que comprovariam a capacidade de extensão do VProclnsp, mas o estudo de caso apontou a possibilidade de se aplicar outras técnicas como a Validação de Face ou o uso de técnicas formais para medir e exibir os dados coletados pelos *checklists*.

Assim, o estudo de caso para avaliar o VProclnsp mostrou resultados preliminares satisfatórios e dão indícios que a técnica de validação proposta consegue comparar de forma simples, o modelo formal e a execução do processo de desenvolvimento. Além disto, o estudo de caso mostrou a viabilidade de ser utilizada em ambientes reais. Dessa forma, organizações podem realizar a validação de processos de maneira mais sistematizada e menos subjetiva.

CAPÍTULO 6

CONCLUSÕES E CONSIDERAÇÕES FINAIS

Neste capítulo são apresentadas as conclusões deste trabalho, relatando as suas contribuições, limitações e perspectivas futuras

6.1 Resumo

Com o aumento da demanda e da complexidade dos sistemas, os processos de desenvolvimento de software vêm sendo considerados fundamentais para que um produto de qualidade seja obtido. Atualmente, empresas de todos os portes e instituições científicas fazem uso dos benefícios adquiridos pelo uso desses processos.

Contudo, os vários processos de desenvolvimento vêm sendo customizados e outros propostos para atender as necessidades de cada meio produtor de software, como é o caso, respectivamente, do processo da RM Sistemas e do PESC. Isto faz surgir a necessidade de validar os processos para garantir que são tangíveis de serem seguidos, descartando burocracias desnecessárias ou constatando a ausência de uma orientação mais metódica.

Durante a pesquisa que resultou nessa dissertação foram identificadas, através de uma revisão sistemática, algumas técnicas de validação de processos descritas na literatura. Com a caracterização dessas técnicas, foi possível identificar seus benefícios e limitações, e foi sugerida uma técnica de validação que permite minimizar as limitações e, suprir a ausência constatada de técnicas de validação de processos bem documentadas e genéricas.

A técnica proposta, intitulada de VProclnsp, é caracterizada pela simplicidade de uso, sem deixar de ser metódica, com fases, papéis, atividades e

documentos pré-estabelecidos. O VProclnsp tem o objetivo de inspecionar como os envolvidos na criação de software estão executando o modelo formal do processo. Para isso, a técnica proposta faz uso de *checklists* que são configurados de acordo com cada processo de desenvolvimento.

A técnica sugerida foi submetida a um estudo de caso para averiguar suas atividades, papéis e documentos propostos, além de analisar a sua capacidade de produzir informações para validar ou não um processo.

A realização dessa pesquisa se destaca pela forma como foi desempenhada, utilizando conceitos oriundos de Verificação & Validação de Software, para validar processos, seguindo a mesma linha de raciocínio adotada por outros autores, que visam comparar o modelo formal do processo com a execução desse modelo.

Com isso, foi proposta uma técnica que visa comparar o modelo com a execução baseada em inspeção de software, ou seja, conceitos e técnicas amplamente utilizados em software, agora, foram utilizados em processos de desenvolvimento de software, essa técnica é chamada de VProclnsp.

O VProclnsp foi desenvolvido visando ser:

- Genérico e Adaptável – para que a técnica possa ser aplicada na avaliação de diferentes artefatos do processo de desenvolvimento, ficando independente, portanto, da abordagem utilizada para criá-los;
- Simples – para que não seja necessária a execução de elaboradas atividades para sua aplicação, a necessidade de tipos de conhecimento específicos ou a alocação de grandes quantidades de recursos;
- Extensível – para que seja facilmente modificável e permita ser aplicada com outras técnicas de validação de processos de desenvolvimento.

Ao término deste trabalho, pode-se concluir que:

- (a) Comparar o modelo formal com a execução do processo pode ser uma boa estratégia para validar um processo. Atividades não existentes no modelo formal, mas freqüentemente executadas, ou que são propostas, e nunca utilizadas pelos *stakeholders*, podem ser facilmente percebidas e mostram evidencias que algo não está correto.
- (b) A utilização de conceitos de V&V e, principalmente, da técnica de Inspeção de Software torna a validação de processos metódica, pois os envolvidos sabem exatamente o que devem realizar em cada etapa estabelecida sem comprometer a simplicidade.
- (c) Para aplicar a técnica, a empresa deve sentir sua necessidade e todos os envolvidos na validação devem estar empenhados em descobrir discrepâncias e não levar a inspeção para o lado pessoal, pois não é a intenção apontar quem não está seguindo o modelo formal e sim, apontar por que o modelo não está sendo seguido.
- (d) A técnica proposta pode ficar dependente da estrutura da empresa para alcançar bons resultados, pois a empresa ou instituição científica devem possibilitar que os inspetores analisem os artefatos, que para o VProclnsp podem ser documentos de especificação, logs de aplicativos, caixa de emails, programas de controle de fluxo de atividades, atas de reuniões, entre outros. Logo, se a empresa não armazena ou disponibiliza tais informações a credibilidade da técnica pode ser reduzida, pois os dados seriam coletados somente através de entrevistas.

Apesar de mostrar indícios que comparar o modelo formal com a execução do processo pode ser uma boa estratégia para validar processos, várias questões ainda podem ser abordadas. Isto se deve ao fato que validar um produto, processo ou fenômeno é conceitualmente complexo e se refere a muitas questões subjetivas.

- (e) Um processo pode não ser seguido e produzir um resultado satisfatório, ou a prática e o processo podem ser coincidentes, mas o produto final

não corresponder às expectativas. Logo, percebe-se a necessidade de abordar outras variáveis no processo de validação, como analisar a satisfação dos desenvolvedores, usuários e da gerência da empresa produtora de software.

- (f) A Eficiência pode ser outro ponto interessante de análise, aonde recurso e tempo envolvido durante o processo são analisados se são compatíveis com o nível de desempenho esperado.

6.2 Contribuições

Com base no que foi realizado durante essa pesquisa, pode-se identificar as seguintes contribuições oferecidas à comunidade produtora de software:

- **Identificação das técnicas de validação de processo.** Através da realização de uma revisão sistemática nas principais fontes de publicações disponíveis, foram levantados trabalhos de diferentes autores que aplicavam diferentes técnicas para avaliar e validar um processo. Foi apresentada, também, uma diferenciação entre avaliar e validar um processo.
- **Identificação das técnicas de Verificação e Validação de Software.** Através da realização de uma revisão sistemática nas principais fontes de publicações disponíveis, foram levantados conceitos e técnicas de V&V, principalmente, técnicas informais.
- **Proposta de uma técnica de Validação por Inspeção.** Fazendo uso de conceitos, diretrizes e técnicas consolidadas de V&V de software, o VProclnsp foi proposto, tornando a tarefa de validar um processo de desenvolvimento, metódica, com fases, papéis e atividades bem definidas, sem deixar de ser simples, adaptável e extensível.
- **Planejamento e execução do Estudo de caso visando a avaliação da técnica em um ambiente real.** Além da criação da abordagem, um

estudo experimental foi planejado e executado para avaliar a viabilidade e a eficiência da técnica proposta. Esse estudo serviu para transferir essa tecnologia do meio acadêmico para um ambiente real e mostrou fortes evidências de sua viabilidade e eficiência, contribuindo com informações relevantes para a empresa TOTVS S.A.

6.3 Limitações

Ao analisar a técnica proposta foram identificadas algumas limitações. Uma delas é que o estudo realizado forneceu somente resultados preliminares sobre o processo validado. Isso ocorreu por não ser possível realizar um estudo mais amplo na empresa, englobando todas as etapas, um maior número de produtos e requisitos.

Seria importante aplicar a técnica proposta em outras empresas, com diferentes processos para que os resultados fossem considerados definitivos

6.4 Trabalhos Futuros

Entre os trabalhos futuros sugeridos no intuito de dar continuidade a esta pesquisa podemos citar:

- A realização de outros estudos de caso, aplicando o VProInsp em outras empresas ou mesmo no meio acadêmico, buscando resultados definitivos, uma vez que o estudo de caso apresentado nessa pesquisa foi direcionado somente ao amadurecimento da técnica proposta.
- A pesquisa pode ser ampliada aplicando outras técnicas de validação como, por exemplo, a validação de fase ou técnicas formais (uso matemático) para auxiliar o uso do *checklist* e a apresentação dos

resultados, ponderando os valores encontrados e comprovando, assim, a extensibilidade do VProclnsp.

- Outra contribuição importante é a construção de uma ferramenta computacional de apoio a todas as fases, atividades e documentos propostos pelo VProclnsp. Esta ferramenta pode controlar o fluxo das atividades, limitar permissões de acesso a documentos de acordo com o perfil de cada usuário e, armazenar informações sobre os participantes e processos validados criando um banco de dados de validação para possíveis consultas.
- Verificar a aplicabilidade de outras técnicas de V&V ou das normas da ISO 9126, ISO 12207 ou do modelo CMMI para auxiliar na busca por análises da satisfação e eficiência do processo de desenvolvimento de software a ser validado.

REFERÊNCIAS BIBLIOGRÁFICAS

ACKERMAN, A., BUCHWALD, L., LEWSKI, F., Software Inspections: An Effective Verification Process, IEEE Software, Vol. 6 No. 3 p. 31-37, 1989.

ASTELS D., MILLER G., NOVAK M., eXtreme Programming: Guia prático, Campus, Rio de Janeiro, 2002.

AMARAL, E.A.G. Empacotamento de Experimentos em Engenharia de Software, Dissertação de Mestrado, Programa de Engenharia de Sistemas e Computação – COOPE/RJ, 2003.

BALCI Osman. Validation, Verification, and Testing Techniques throughout the Life Cycle of a Simulation Study. Proceedings of the 26 th Winter Simulation Conference, Blacksburg, Virginia. p 215-220, 1994

BALCI Osman. Principles and Techniques of Simulation Validation, Verification, and Testing. Proceedings of the 27 th Winter Simulation Conference, Arlington, Virginia. p 147-154, 1995.

BARCELOS, R.F., TRAVASSOS, G.H., Avaliando documentos arquiteturais através de um checklist baseado em atributos de qualidade. In proceedings of Workshop de Teses e Dissertações de Engenharia de Software (WTES) – SBES, Uberlândia, MG, 2005.

BANKS, J., GERSTEIN, D., SEARLES, S. P., Modeling processes, validation, and verification of complex simulations: A survey, Methodology and Validation, Simulations Series, vol. 19 No. 1. The Society of Computer Simulation, 13 – 18, 1988

BECK K., Extreme Programming Explained. Boston, MA: Addison-Wesley, 2000.

BIFFL, S., GROSSMANN, W., 2001, Evaluating the accuracy of objective estimation models based on inspection data from multiple inspection cycles. ACM/IEEE ICSE, Toronto, Canada, IEEE Comp. Soc. Press, 2001.

BOEHM B., Software Engineering Economics, 1st edition. Prentice-Hall, 1981.

BORGES E.P., PAULA, W.P. Um modelo de medição para processos de desenvolvimento de software, SIMPROS, 2003.

EJIOGU, Lem O. Five principles for the formal validation of models of software metrics. ACM SIGPLAN Notices, vol 28, No 8, 1993.

CHMURA, L. J., WICINSKI, T. J., and NORCIO, A. F. Evaluating software design processes by analyzing change data over time. IEEE Transactions on Software Engineering. vol. 16, p. 729–740, 1990

COOK, Jonathan E., WOLF, Alexander L. Software Validation: Quantitatively Measuring the Correspondence of a Process to a Model. ACM Transactions on Software Engineering and Methodology. vol 8 No 2, p. 147-176, 1999.

COOK, C., VISCONTI, M. Issues in Software Process Assessment and Validation. Proceedings 21st ACM Computer Science Conference, Indianapolis IN, 1993.

FAGAN, M.E. Design and code inspection to reduce Errors in Program Development, IBM Systems Journal, v. 15, n. 3, pp 182-211, 1976

FERREIRA, B.; MOITA, G. F. Avaliação de técnicas para validação em processos de desenvolvimento de software. In: VIII Simpósio de Mecânica Computacional, Belo Horizonte: PUC Minas, vol. 1. p. 1-15, 2008 a

FERREIRA, B.; MOITA, G. F. The evaluation of different validation techniques for software development process. In: 8th World Congress on Computational

Mechanics WCCM8, Veneza. Proceedings of the 8th World Congress on Computational Mechanics, v. 1. p. 1-2, 2008 b.

FGV, Site da Fundação Getúlio Vargas acessado em 25/08/2008
http://www.fgv.br/fgvportal/principal/idx_materia.asp?str_chave=11269&sessao=2,
2008

GARG, P. K., Process Modeling of Software Reuse, 1992. In Proceedings of the 5th International Workshop on Institutionalizing Software Reuse, Palo Alto.

GILB, T., GRAHAM, D. Software Inspecion. Addison-Wesley, 1993.

HUMPHREY, W. S. A Discipline for Software Engineering. Addison-Wesley. Reading-MA, 1995.

IEEE, IEEE Recommended Practice for Software Requirements Specifications – Description, Standart 830, IEEE Press. 1998.

JACOBSON, I.; RUMBAUGH, J.; BOOCH, G. The Unified Software Development Process. Addison-Wesley, Reading – MA, 1999.

JOHN R. C., STEVE M. E. Verification and Validation in a Rapid Software Development Process, NASA Software IV&V Facility, 2000.

JOHNSON, P. An Instrumented Approach to Improving Software Quality through Formal Technical Review, in proc. 16th International Conference on Sotware Engineering, Sorrento, Italy, 1994.

KNEPELL, P. L., ARANGNO, D. C., Simulation validation: A confidence assessment methodology, IEEE Computer Society Simulation, 1993.

KALINOWSKI, M. Infra-Estrutura Computacional de Apoio ao Processo de Inspeção de Software, Dissertação de Mestrado, Programa de Engenharia de Sistemas e Computação – COPPE/UFRJ, Rio de Janeiro, 2004

KRUSKAL, J. B. An overview of sequence comparison. In Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison. In Proceedings of SIAM 1983 - Society for Industrial and Applied Mathematics. vol 25 No 2 p 201-237, 1983.

LAITENBERGER, O., ATKINSON, C. Generalized Perspective Based Inspection to handle Object Oriented Development Artifacts, Proceddings of ICSE 99, p. 494-503., 1998

LARMAN, C. Utilizando UML e Padrões: uma introdução à análise e ao projeto orientado a objetos e ao Processo Unificado. Trad. Luiz Augusto Meirelles Salgado e João Tortello. 2 ed. Porto Alegre: Bookman, 2004.

LANUBILE, F., MALLARDO, T., CALEFATO, F., Tool support for Geographically Dispersed Inspection Teams, Software Process Improvement and Practice, Vol. 8: p.217-231 (DOI: 10.1002/spip.184)., 2003.

MELLO, A.M.V. Processo de Desenvolvimento de Software: Uma abordagem para Melhoria Contínua da Qualidade.2004. Disponível em http://santafe.ipt.br/tede/tde_busca/arquivo.php?codArquivo=226 Acesso em: 28 de julho de 2008.

MOOR, A., DELUGACH H. Software Process Validation: Comparing Process and Practice Models. In Contributions to ICCS 2005, Kassel, Germany. p 533-540, 2005

NAS - System Engineering Manual – versão 3.1 fonte: http://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/operations/sysengsaf/seman/SEM3.1/Section%204.14%20v3.pdf acessado em 30/03/2008, 2006

OBERKAMPF, W.L., TRUCANO, T.G., Verification and validation benchmarks, Nuclear Eng. Design, Validation and Uncertainty Estimation Department, Sandia National Laboratories, vol 238 p. 716-743, 2007.

PAULA, W. P. Engenharia de Software – Fundamentos, Métodos e Padrões. 2^a ed. LTC Editora. Rio de Janeiro - RJ, 2003.

PEREIRA Jr, M. Concepção de um Processo Específico para Software Científico. Dissertação de Mestrado. Centro Federal de Educação Tecnológica de Minas Gerais – CEFET-MG. Belo Horizonte, 2007.

PERRY, William. Effective Methods for Software Testing. John Wiley & Sons, 1995.

PRESSMAN, R. S. Engenharia de Software. 6^a ed. Rio de Janeiro: McGraw-Hill, 2006.

PURRI, M. C. M. S; PEREIRA Jr, M.,; MOITA, G. F. PESC – Processo de Desenvolvimento Específico para Software Científico: Propostas Iniciais. XXVII CILAMCE – Iberian Latin American Congress on Computational Methods in Engineering Belém-PA, 2006

PURRI, M. C. M. S. Estudo e Propostas Iniciais para a Definição de um Processo de Desenvolvimento para Software Científico. Dissertação de Mestrado. Centro Federal de Educação Tecnológica de Minas Gerais – CEFET-MG. Belo Horizonte, 2006.

RAMOS, C., ZITA A., VALE, L., SANTOS, J., Aplicações Inteligentes em Centros de Controlo: Verificação e Validação, Jornadas Luso-Espanholas de Engenharia Electrotécnica, Salamanca, Spain, July 1997

SARGENT, Robert G., Verification, Validation, and Accreditation of Simulations Model, Proc. Of the 2000 Winter Simulation Conference. p. 50-59, 2000

SARGENT, Robert G., Some Approaches and Paradigms for Verification and Validation Simulations Model, Proc. Of the 2001 Winter Simulation Conference. p. 50-59, 2001.

SAUER, C., JEFFERY, D.R., LAND, L., YETTON, P. The Effectiveness of Software Development Technical Review: A Behaviorally Motivated Program of Research, IEEE Transactions on Software Engineering 26, 1-14, 2000.

SHULL, F. Developing Techniques for Using Software Documents: A Series of Empirical Studies, Ph.D. thesis, University of Maryland, College Park, 1998.

SOMMERVILLE, I., Engenharia de Software. 8^a ed. São Paulo: Addison Wesley, 2007.

SVAHNBERG, M., A study on agreement between participants in an architecture assessment. In proceedings of the International Symposium on Empirical Software Engineering – ISESE, p 61-70, 2003.

TRAVASSOS G. H., KALINOWSKI, M, SPÍNOLA R. O., Introdução à Inspeção de Software – Aumento da qualidade através de verificações intermediárias, Engenharia de Software Magazine, Ano 1, 1^a edição, 2007.

TICHY, W.F. Should Computer Scientists Experiment More? IEEE Software, Vol. 31, No. 5, p. 32-39, 1998.

WALLIN, C. Verification and Validation of Software Components and Components Based Software Systems. - Based Software Systems. Artech House Publishers, 2002. fonte: http://www.idt.mdh.se/cbse-book/extended-reports/05_Extended_Report.pdf acessado em 01/11/2007.

WHEELER, D.A., BRYKEZYNSKI, B., MEESON, R.N., Software Inspection: Na Insdustry Best Practice, IEEE Computer Society, 1996.

WOHLIN, C., RUNESON, P., HOST, M. OHLSSON, M.C., REGNELL, B., WESSLEN, A. Experimentation in Software Engineering: an Introduction, 2000.

WIKIPEDIA A enciclopédia livre. Disponível em: <http://www.wikipedia.com.br>. Acessado em 05/06/2008.

Apêndice A

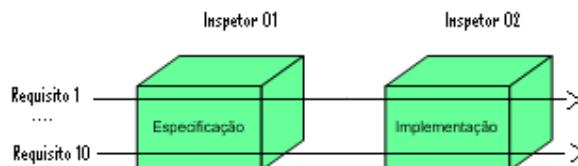
Documentos do VProcInsp Aplicados no Estudo de Caso

Planejamento

VProcInsp – Validação de Processo de Desenvolvimento de Software por Inspeção Documento de Definições do Planejamento Moderador...: Bruno Ferreira Autor(es)...: Isabela Iunes de Oliveira Sandra Cristina de Andrade Soares Data de Início: 30/06/2008 Data de Término prevista: 04/07/2008 Nome do Processo de Desenvolvimento: Processo de Desenvolvimento RM – V. 2.0 Descrição do Processo de Desenvolvimento:											
Características: <ul style="list-style-type: none"> - Baseado no Processo Unificado - Iterativo e Incremental - Poucos artefatos obrigatórios - Modelagem com UML - Criação de produtos orientado a objetos em duas e três camadas 											
Aplicar inspeção no final das (os): <input checked="" type="checkbox"/> Etapas <input checked="" type="checkbox"/> Iterações <input type="checkbox"/> Projeto <input type="checkbox"/> Outros											
Inspecionadores Selecionados: <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 70%;">1) Túlio Mendes Rodrigues</td> <td style="width: 30%;">Checklist entregue <input checked="" type="checkbox"/></td> </tr> <tr> <td>2) Sérgio Gontijo do Nascimento</td> <td>Checklist entregue <input checked="" type="checkbox"/></td> </tr> <tr> <td>3) Clique aqui para digitar texto.</td> <td>Checklist entregue <input type="checkbox"/></td> </tr> <tr> <td>4) Clique aqui para digitar texto.</td> <td>Checklist entregue <input type="checkbox"/></td> </tr> <tr> <td>5) Clique aqui para digitar texto.</td> <td>Checklist entregue <input type="checkbox"/></td> </tr> </table>		1) Túlio Mendes Rodrigues	Checklist entregue <input checked="" type="checkbox"/>	2) Sérgio Gontijo do Nascimento	Checklist entregue <input checked="" type="checkbox"/>	3) Clique aqui para digitar texto.	Checklist entregue <input type="checkbox"/>	4) Clique aqui para digitar texto.	Checklist entregue <input type="checkbox"/>	5) Clique aqui para digitar texto.	Checklist entregue <input type="checkbox"/>
1) Túlio Mendes Rodrigues	Checklist entregue <input checked="" type="checkbox"/>										
2) Sérgio Gontijo do Nascimento	Checklist entregue <input checked="" type="checkbox"/>										
3) Clique aqui para digitar texto.	Checklist entregue <input type="checkbox"/>										
4) Clique aqui para digitar texto.	Checklist entregue <input type="checkbox"/>										
5) Clique aqui para digitar texto.	Checklist entregue <input type="checkbox"/>										
Apresentação Realizada <input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não Data da Realização: 01/07/2008											
Observações: <i>As inspeções serão realizadas em duas etapas em uma iteração do processo.</i>											

Abaixo são listados os 10 requisitos que foram inspecionados durante a execução do processo de desenvolvimento.

- 01) 1287098 – Rateio Default de Itens de Contrato
- 02) 874346 – Centro de Custo
- 03) 790414 – Manter Contrato
- 04) 790435 – Implementar Associação Total ao item do contrato
- 05) 790436 – Implementar Associação parcial ao item do contrato
- 06) 790446 – Implementar Realinhamento de Preço
- 07) 1286910 – Calculo de Projeto – Calculo de Insumo com vigências
- 08) 1286913 – Calculo de Projeto – Calculo de Composições com vigências
- 09) 1286920 – Calculo de Projeto – Calculo de Tarefas com vigências
- 10) 1278552 – Calculo de Projeto – Calculo da Curva ABC e S utilizando Vigência



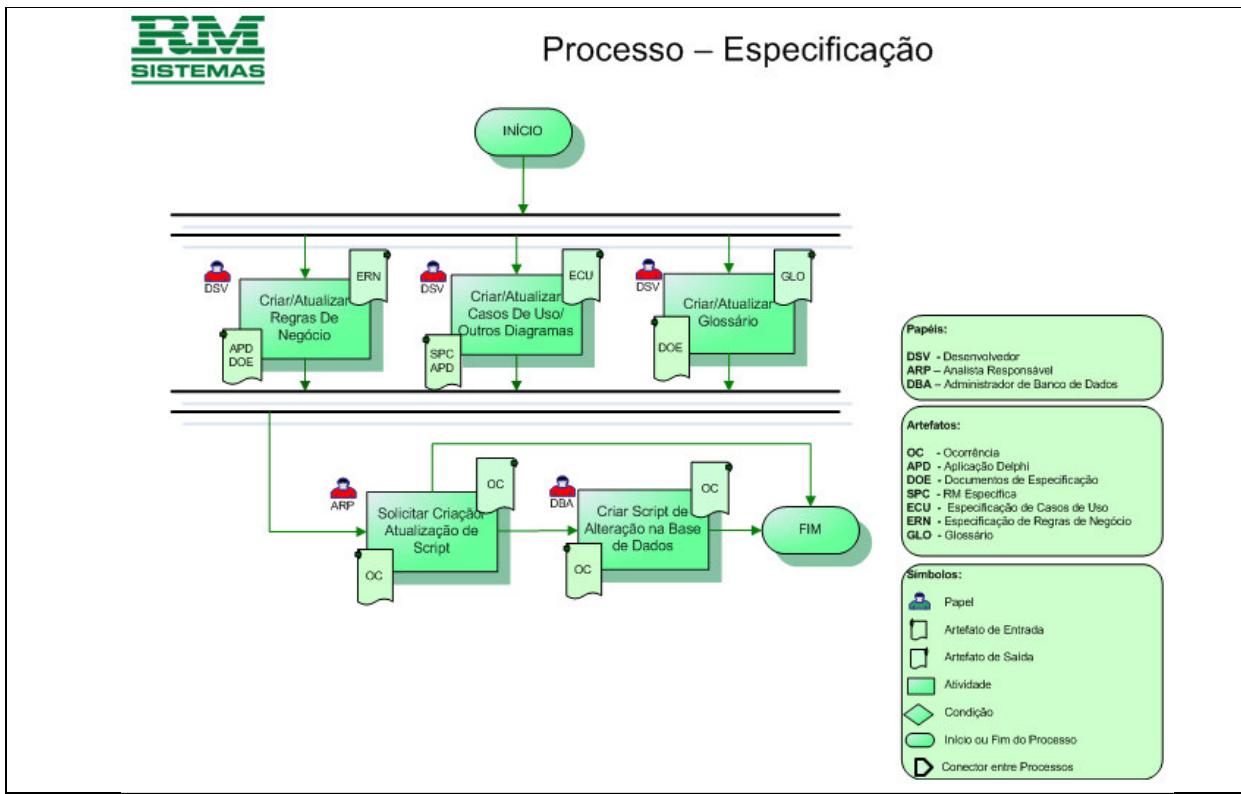
VProclsp – Validação de Processo de Desenvolvimento de Software por Inspeção				
Documento de Informações dos Inspetores				
ID: 00001	Inspetor.: Sergio Gontijo do Nascimento			
Nível de experiência com desenvolvimento de software				
<input type="checkbox"/> 1	<input type="checkbox"/> 2	<input type="checkbox"/> 3	<input type="checkbox"/> 4	<input checked="" type="checkbox"/> 5
Nível de experiência com o domínio do ambiente processo de desenvolvimento				
<input type="checkbox"/> 1	<input type="checkbox"/> 2	<input checked="" type="checkbox"/> 3	<input type="checkbox"/> 4	<input type="checkbox"/> 5
Nível de experiência com validação de processos				
<input checked="" type="checkbox"/> 1	<input type="checkbox"/> 2	<input type="checkbox"/> 3	<input type="checkbox"/> 4	<input type="checkbox"/> 5
1-Nenhum	2-Pouco	3-Razoável	4-Suficiente	5-Alto
Nome dos Processos de Desenvolvimento já validados pelo inspetor				
1)	<hr/>			
2)	<hr/>			
3)	<hr/>			
4)	<hr/>			
5)	<hr/>			
Estatística com o número de erros por categorias levantados pelo inspetor				
<hr/> Omissão	<hr/> Ambigüidade	<hr/> Inconsistência	<hr/> Outro	
Curriculum:				
<p><i>Analista de sistemas com mais de 15 anos de experiência no desenvolvimento de aplicações para ambiente Windows utilizando orientação a objetos, multicamadas, bancos de dados relacionais e ferramentas de modelagem de dados e processos. Experiência no uso de Delphi e Microsoft .NET. Formação acadêmica em Ciência da Computação pela PUC Minas e MBA pelo Ibme.</i></p>				
Observações:				

Documento de Informações dos Inspetores				
ID: 00002	Inspetor..: Túllio Mendes Rodrigues			
Nível de experiência com desenvolvimento de software				
<input type="checkbox"/> 1	<input type="checkbox"/> 2	<input type="checkbox"/> 3	<input type="checkbox"/> 4	<input checked="" type="checkbox"/> 5
Nível de experiência com o domínio do ambiente processo de desenvolvimento				
<input type="checkbox"/> 1	<input type="checkbox"/> 2	<input type="checkbox"/> 3	<input checked="" type="checkbox"/> 4	<input type="checkbox"/> 5
Nível de experiência com validação de processos				
<input checked="" type="checkbox"/> 1	<input type="checkbox"/> 2	<input type="checkbox"/> 3	<input type="checkbox"/> 4	<input type="checkbox"/> 5
1-Nenhum	2-Pouco	3-Razoável	4-Suficiente	5-Alto
Nome dos Processos de Desenvolvimento já validados pelo inspetor				
1)	<hr/>			
2)	<hr/>			
3)	<hr/>			
4)	<hr/>			
5)	<hr/>			
Estatística com o número de erros por categorias levantados pelo inspetor				
Omissão	Ambigüidade	Inconsistência	Outro	
Curriculum:				
<i>Desenvolvedor de sistemas com sete anos de experiência criando aplicações orientada a objetos utilizando Delphi e Dot Net. Experiência com manutenção em banco de dados relacionais e implantação de sistemas. Formação técnica em Processamento de dados e graduado em Gestão Financeira.</i>				
Observações:				

VProclnsp – Validação de Processo de Desenvolvimento de Software por Inspeção	
Documento de atribuições de Inspetores	
ID do Inspetor: 00001	Inspetor..: Sergio Gontijo do Nascimento
Lista com os identificadores dos checklists sob responsabilidade do inspetor.	
- 002 [responsável por todos os itens do checklist de Implementação]	
ID do Inspetor: 00002	Inspetor..: Túllio Mendes Rodrigues
Lista com os identificadores dos checklists sob responsabilidade do inspetor.	
- 001 [responsável por todos os itens do checklist de Especificação]	
Observações:	
Como serão validadas somente duas etapas, não foi necessário dividir os itens de um mesmo checklist entre os inspetores. Sendo que cada inspetor é responsável por todos os itens do seu checklist.	

Análise e Comparação

VProcInsp – Validação de Processo de Desenvolvimento de Software por Inspeção	
Checklist de Inspeção	
<p>*Instruções</p> <ul style="list-style-type: none"> ✓ Avalie as discrepâncias entre o modelo formal do processo e a execução do mesmo somente na fase de Especificação através dos itens de avaliação que compõem o checklist. ✓ Busque discrepâncias entre o processo e a execução do mesmo. Sem identificar possíveis erros de análises, modelagem, implementação ou testes. Se atente apenas às discrepâncias. ✓ Analise as discrepâncias nos requisitos listados abaixo. <ul style="list-style-type: none"> 01) 1287098 – Rateio Default de Itens de Contrato 02) 874346 – Centro de Custo 03) 790414 – Manter Contrato 04) 790435 – Implementar Associação Total ao item do contrato 05) 790436 – Implementar Associação parcial ao item do contrato 06) 790446 – Implementar Realinhamento de Preço 07) 1286910 – Calculo de Projeto – Calculo de Insumo com vigências 08) 1286913 – Calculo de Projeto – Calculo de Composições com vigências 09) 1286920 – Calculo de Projeto – Calculo de Tarefas com vigências 10) 1278552 – Calculo de Projeto – Calculo da Curva ABC e S utilizando Vigência ✓ Caso a sua resposta seja diferente da Resposta Esperada - RE (Não - N, Sim - S); uma discrepância entre o modelo formal e a execução foi descoberta. ✓ Nas colunas “Sim”, “Não” e NA (Não aplicável) de cada item do checklist devem ser preenchidas com valores inteiros entre zero e dez. Aonde zero significa que nenhum requisito obteve a resposta e dez significa que todos os dez requisitos obtiveram a resposta sim ou não ou NA. Atenção: A soma das três colunas deve ser igual a dez. ✓ O campo Comentário fica disponível caso o inspetor queira fazer algum comentário a respeito de um requisito ou alguma situação em que a discrepância precisa ser esclarecida. ✓ O campo Evidência tem a função de indicar a origem da inspeção que gerou uma discrepância ou não. Ou seja, informa o artefato que foi inspecionado. Esse campo pode assumir os valores: Ferramentas, Doc. Técnicos, Doc. Gestão, Logs, Ata de Reunião, Email, softwares utilizados no processo e outros. ✓ Caso haja dúvida em relação aos termos utilizados ou nos itens do checklist peça informações ao moderador. ✓ Leia as subseções Fase/Atividade, Principais Objetivos e Importância para ampliar o conhecimento sobre o contexto da inspeção. 	
*ID do CheckList..: 001	*Inspetor.....: Túllio Mendes Rodrigues
*Fase/Atividade..:	



*Principais Objetivos

Criar/Atualizar Regras de Negócio

- ✓ Especificar detalhadamente as regras de negócio contidas no processo que está sendo desenvolvido.

Criar/Atualizar Casos de Uso/Outros Diagramas

- ✓ Possibilitar o entendimento da funcionalidade do sistema em um nível mais voltado ao negócio a todos os envolvidos no projeto.

Criar/Atualizar Glossário

- ✓ Definir a terminologia específica do problema de domínio, explicando termos que possam ser desconhecidos ao leitor das descrições dos casos de uso ou outros documentos do projeto.

Solicitar Criação/Atualização de Script

- ✓ Escrever scripts com as alterações na base de dados necessárias para as implementações do Caso de Uso.

Criar Script de Alteração na Base de Dados

- ✓ Escrever scripts com as alterações na base de dados necessárias para as implementações do Caso de Uso.

*Importância

Criar/Atualizar Regras de Negócio

- ✓ Fonte de informação para entendimento das regras de negócio pertinentes a cada funcionalidade que está sendo especificada.

Criar/Atualizar Casos de Uso/Outros Diagramas

- ✓ Fonte de informação para realização dos casos de testes, futuros testes de funcionalidade e contribuição na documentação do produto. Fonte de consulta futura de novas implementações e análise de impacto.

Criar/Atualizar Glossário

- ✓ Documentar os termos utilizados na especificação.

Solicitar Criação/Atualização de Script

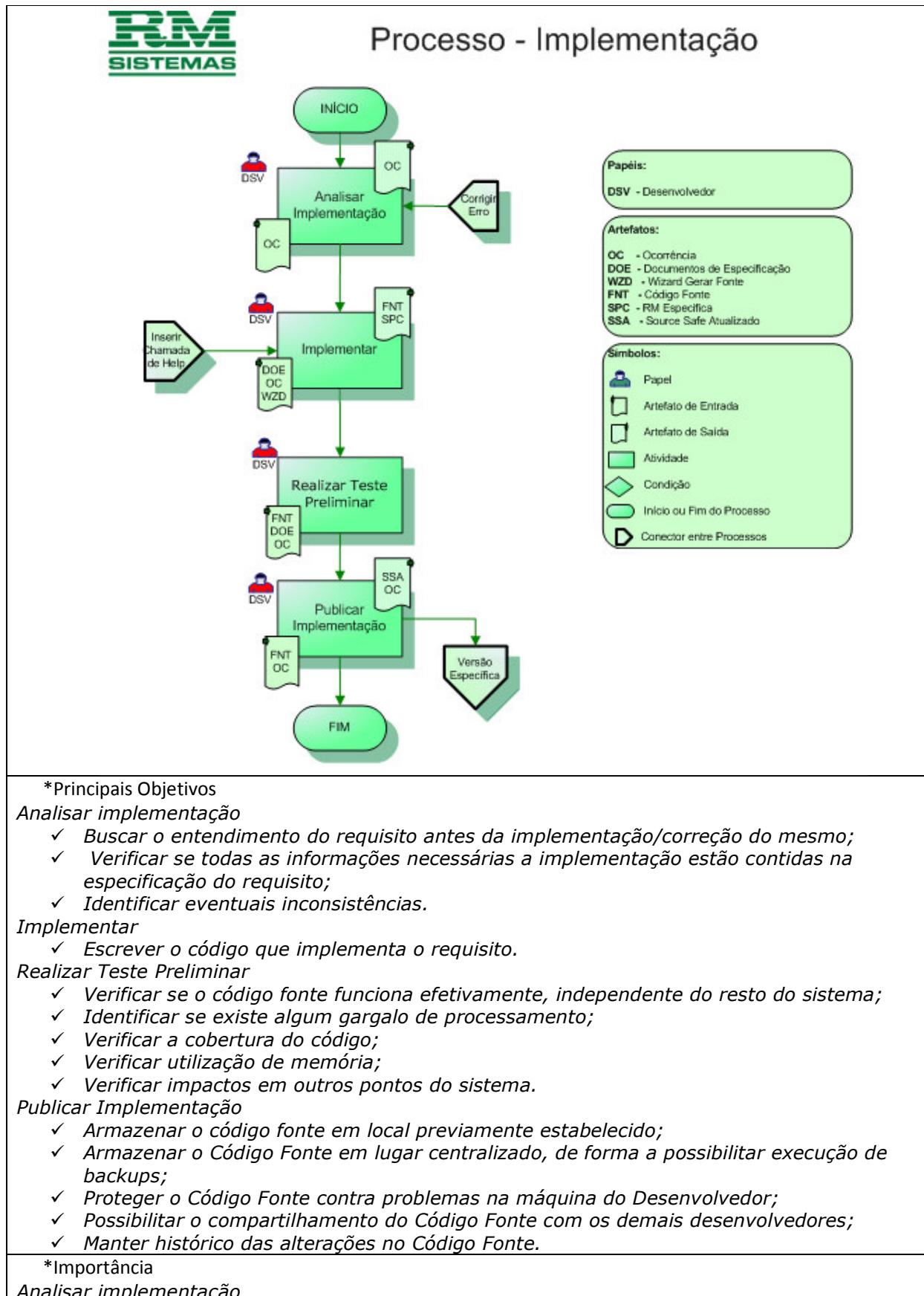
- ✓ Escrever scripts padronizados que serão enviados para os clientes, em forma de Conversor de base de dados em uma data pré-estabelecida, com o objetivo de ter a bases de dados de todos os clientes com a mesma estrutura.

<p><i>Criar Script de Alteração na Base de Dados</i></p> <ul style="list-style-type: none"> ✓ Escrever scripts padronizados que serão enviados para os clientes, em forma de Conversor de base de dados em uma data pré-estabelecida, com o objetivo de ter a bases de dados de todos os clientes com a mesma estrutura. 					
Entradas					
Nº	Descrição do Item de Avaliação	Sim	Não	NA	RE
01	<i>A Aplicação Delphi ou Documentos de Especificação ou outros documentos de especificação que por ventura já existam serviu como entrada para Criar/Atualizar Regras de Negócio?</i>	10	0	0	Sim
<i>Comentário: Os requisitos 01 ao 06 utilizaram o Delphi como documento de especificação. Os requisitos 07 até 10 utilizaram documentos de especificação já existentes.</i>		Evidência: Entrevista			
02	<i>A aplicação Delphi e o RM Específica contendo todos os requisitos do produto serviram como entrada para Criar/Atualizar Casos de Uso e/ou Outros Diagramas?</i>	0	10	0	Sim
<i>Comentário: Não foram encontrados diagramas UML para esses requisitos.</i>		Evidência: Entrevista e o RM Específica			
03	<i>O Documento de especificação serviu como entrada para Criar/Atualizar Glossário?</i>	10	0	0	Sim
<i>Comentário: O glossário foi construindo seguindo a especificação dos dez requisitos</i>		Evidência: Entrevista e o Glossário			
04	<i>Uma Ocorrência foi utilizada como entrada para Solicitar Criação/Alteração de Script?</i>	4	0	6	Sim
<i>Comentário: Os requisitos 01 ao 06 não necessitaram de alterações na Base de dados. Somente os requisitos 07, 08, 09 e 10 utilizaram scripts.</i>		Evidência: RM Agilis			
05	<i>Uma Ocorrência foi utilizada como entrada para a atividade de Criar Script de Alteração na Base de Dados?</i>	4	0	6	Sim
<i>Comentário: Os requisitos 01 ao 06 não necessitaram de alterações na Base de dados. Somente os requisitos 07, 08, 09 e 10 utilizaram scripts.</i>		Evidência: RM Agilis			
Saídas					
Nº	Descrição do Item de Avaliação	Sim	Não	NA	RE
06	<i>Foi criada a Especificação de Regras de Negócio?</i>	10	0	0	Sim
<i>Comentário: O documento ss\RM.Net\Source\PrjProjetos\Projeto\E_BR-Regra deNegocio.doc foi atualizado com as informações</i>		Evidência: Documento de Especificação			
07	<i>Foi gerada a Especificação dos Casos de Uso ou Outros Diagramas?</i>	0	10	0	Sim
<i>Comentário: Não foram encontrados diagramas UML para esses requisitos.</i>		Evidência: Documento de Especificação			
08	<i>Foi criado ou atualizado o Glossário?</i>	10	0	0	Sim
<i>Comentário: O documento ss\RM.Net\Source\PrjProjetos\Projeto\E_G-Glossário.doc foi atualizado com as informações</i>		Evidência: Documento de Especificação			
09	<i>Foi criada uma ocorrência para o DBA vinculada com a ocorrência de origem?</i>	4	0	6	Sim
<i>Comentário: Os requisitos 01 ao 06 não tiveram novas ocorrências vinculadas à ocorrência original. Somente os requisitos 07, 08, 09 e 10 utilizaram scripts e foram vinculadas.</i>		Evidência: RM Agilis			
10	<i>O script e a ocorrência foram encaminhados para o Analista responsável?</i>	4	0	6	Sim
<i>Comentário: No email do desenvolvedor continham os scripts recebidos para os requisitos 07 ao 10.</i>		Evidência: RM Agilis e MS Outlook			

Passos						
Nº	Descrição do Item de Avaliação	Sim	Não	NA	RE	
11	Ao final da Criação/Atualização das Regras de Negócio, os documentos de saída foram atualizados no Source Safe?	10	0	0	Sim	
	Comentário: Analisando a ultima versão do documento no SourceSafe pode-se verificar que o documento foi atualizado	Evidência: MS SourceSafe				
12	Ao final da atividade de Criar/Atualizar Casos de Uso e/ou Outros Diagramas, os documentos de saída foram atualizados no Source Safe?	0	10	0	Sim	
	Comentário: Não existem diagramas UML para esses requisitos no SourceSafe	Evidência: MS SourceSafe				
13	O check out/in foi executado no Glossário ao ser Criado/Atualizado ?	10	0	0	Sim	
	Comentário: Analisando a ultima versão do documento no SourceSafe pode-se verificar que o documento foi atualizado	Evidência: MS SourceSafe				
14	O DBA encaminhou o Script para o Analista Responsável e o arquivo ss/RMGlobalDados/RMConversor/RMConversor_[BancodeDados] foi atualizado ao Criar Script de Alteração na Base de Dados?	4	0	6	Sim	
	Comentário: Verificando o email e do desenvolvedor e no SourceSafe percebe-se que os scripts para os requisitos 07 ao 10 foram recebidos.	Evidência: MS SourceSafe e email				
Ferramentas						
Nº	Descrição do Item de Avaliação	Sim	Não	NA	RE	
15	Foi utilizado o MS Word, MS Visio ou Source Safe ao Criar/Atualizar Regras de Negócio?	10	0	0	Sim	
	Comentário: Software padrões e controlados pela equipe de Infra-Estrutura da empresa.	Evidência: Software Instalados e Extensão dos artefatos				
16	Foi utilizado o MS Visio, MS Word, MS Excel, VSS, RM Especifica ao Criar/Atualizar Casos de Uso e/ou Outros Diagramas?	10	0	0	Sim	
	Comentário: Software padrões e controlados pela equipe de Infra-Estrutura da empresa.	Evidência: Software Instalados e Extensão dos artefatos				
17	Foi utilizado o MS Word e o SourceSafe ao Criar/Atualizar o Glossário?	10	0	0	Sim	
	Comentário: Software padrões e controlados pela equipe de Infra-Estrutura da empresa.	Evidência: Software Instalados e Extensão dos artefatos				
18	Foi utilizado o RM Agilis para criar e vincular as ocorrências de entrada e saída ao Solicitar Criação/Alteração de Script?	10	0	0	Sim	
	Comentário: Software padrões e controlados pela equipe de Infra-Estrutura da empresa.	Evidência: Software Instalados e Extensão dos artefatos				
19	Foi utilizado o RM Agilis, SQL Server e Oracle ao Criar Script de Alteração na Base de Dados?	10	0	0	Sim	
	Comentário: Software padrões e controlados pela equipe de Infra-Estrutura da empresa.	Evidência: Software Instalados e Extensão dos artefatos				
Templates						
Nº	Descrição do Item de Avaliação	Sim	Não	NA	RE	
20	Os templates disponíveis em	10	0	0	Sim	

	<i>SS\RM.NET\Source\RM.Templates\RM.Docs\E_BR - RegrasNegocio.dot foram utilizados ao Criar/Atualizar Regras de Negócio?</i>								
	<i>Comentário: Os documentos preenchidos têm a mesma estrutura dos arquivos default dispostos no SourceSafe.</i>	<i>Evidência: Documento de especificação</i>							
21	<i>Os templates disponíveis em SS\RM.NET\Source\RM.Templates\RM.Docs\E_UC - Caso de Uso.dot foram utilizados ao Criar/Atualizar Casos de Uso e/ou Outros Diagramas?</i>	0	10	0	Sim				
	<i>Comentário: Não foram criados diagramas de caso de uso para os requisitos. Logo os templates não foram utilizados.</i>	<i>Evidência: Documento de especificação</i>							
22	<i>Os templates disponíveis em SS\RM.NET\Source\RM.Templates\RM.Docs\E_G - Glossario.dot foram utilizados ao Criar/Atualizar o Glossário?</i>	10	0	0	Sim				
	<i>Comentário: Os documentos preenchidos têm a mesma estrutura dos arquivos default dispostos no SourceSafe.</i>	<i>Evidência: Documento de especificação</i>							
Responsáveis									
Nº	Descrição do Item de Avaliação	Sim	Não	NA	RE				
23	<i>O Desenvolvedor foi responsável por Criar/Atualizar Regras de Negócio?</i>	10	0	0	Sim				
	<i>Comentário: Log de quem executou o check in do arquivo</i>	<i>Evidência: MS SourceSafe</i>							
24	<i>O Desenvolvedor foi responsável por Criar/Atualizar Casos de Uso e/ou Outros Diagramas?</i>	0	10	0	Sim				
	<i>Comentário: Não existem diagramas UML para esses requisitos no SourceSafe</i>	<i>Evidência: MS SourceSafe</i>							
25	<i>O Desenvolvedor foi responsável por Criar/Atualizar o Glossário?</i>	10	0	0	Sim				
	<i>Comentário: Log de quem executou o check in do arquivo</i>	<i>Evidência: MS SourceSafe</i>							
26	<i>Analista Responsável foi o responsável por Solicitar ao Criação/Atualização do Script?</i>	0	4	6	Sim				
	<i>Comentário: Informações retirada da visão de Repasses no RM Agilis</i>	<i>Evidência: RMAgilis</i>							
27	<i>O Administrador de Banco de Dados foi o responsável por Criar Script de Alteração na Base de Dados?</i>	4	0	6	Sim				
	<i>Comentário: Log de quem executou o check in do arquivo</i>	<i>Evidência: MS SourceSafe</i>							
Observações									
<i>Ao inspecionar os artefatos para responder o item 26 o inspetor relatou a seguinte discrepância: A solicitação de criação de script é feita na etapa de Implementação e não na Etapa de Especificação.</i>									
<i>Evidência: A data de abertura de ocorrência para os scripts eram posteriores ao apontamento de horas na fase de implementação. Ou seja, as datas de abertura de solicitação eram feitas depois do encerramento da fase de Especificação</i>									

VProcInsp – Validação de Processo de Desenvolvimento de Software por Inspeção	
Checklist de Inspeção	
<p>*Instruções</p> <ul style="list-style-type: none"> ✓ Avalie as <i>discrepâncias entre o modelo formal do processo e a execução do mesmo somente na fase de Especificação</i> através dos itens de avaliação que compõem o checklist. ✓ Busque <i>discrepâncias entre o processo e a execução do mesmo</i>. Sem identificar possíveis erros de análises, modelagem, implementação ou testes. Se atente apenas às <i>discrepâncias</i>. ✓ Analise as <i>discrepâncias nos requisitos listados abaixo</i>. <ul style="list-style-type: none"> 01) 1287098 – Rateio Default de Itens de Contrato 02) 874346 – Centro de Custo 03) 790414 – Manter Contrato 04) 790435 – Implementar Associação Total ao item do contrato 05) 790436 – Implementar Associação parcial ao item do contrato 06) 790446 – Implementar Realinhamento de Preço 07) 1286910 – Calculo de Projeto – Calculo de Insumo com vigências 08) 1286913 – Calculo de Projeto – Calculo de Composições com vigências 09) 1286920 – Calculo de Projeto – Calculo de Tarefas com vigências 10) 1278552 – Calculo de Projeto – Calculo da Curva ABC e S utilizando Vigência ✓ Caso a sua resposta seja diferente da Resposta Esperada - RE (Não - N, Sim - S); uma <i>discrepância entre o modelo formal e a execução</i> foi descoberta. ✓ Nas colunas “Sim”, “Não” e NA (Não aplicável) de cada item do checklist devem ser preenchidas com valores inteiros entre zero e dez. Aonde zero significa que nenhum requisito obteve a resposta e dez significa que todos os dez requisitos obtiveram a resposta sim ou não ou NA. Atenção: A soma das três colunas deve ser igual a dez. ✓ O campo <i>Comentário</i> fica disponível caso o inspetor queira fazer algum comentário a respeito de um requisito ou alguma situação em que a <i>discrepância</i> precisa ser esclarecida. ✓ O campo <i>Evidência</i> tem a função de indicar a origem da inspeção que gerou uma <i>discrepância</i> ou não. Ou seja, informa o artefato que foi inspecionado. Esse campo pode assumir os valores: Ferramentas, Doc. Técnicos, Doc. Gestão, Logs, Ata de Reunião, Email, softwares utilizados no processo e outros. ✓ Caso haja dúvida em relação aos termos utilizados ou nos itens do checklist peça informações ao moderador. ✓ Leia as subseções <i>Fase/Atividade, Principais Objetivos e Importância</i> para ampliar o conhecimento sobre o contexto da inspeção. 	
*ID do checklist.....: 002	*Inspetor.....: Sérgio Gontijo do Nascimento
*Fase/Atividade..:	



<ul style="list-style-type: none"> ✓ Conhecer o que precisa ser implementado/corrigido para saber qual ação tomar. <p>Implementar</p> <ul style="list-style-type: none"> ✓ Através desta atividade estamos concretizando uma implementação. <p>Realizar Teste Preliminar</p> <ul style="list-style-type: none"> ✓ Esta tarefa é importante, pois uma vez que este teste seja bem feito, quando o desenvolvedor encaminhar uma versão para o QSW homologar, a chance de existirem erros é menor, evitando assim possíveis retrabalhos. <p>Publicar Implementação</p> <ul style="list-style-type: none"> ✓ Depois do código fonte devidamente implementado (estiver compilando) deve-se disponibilizar o código fonte no SourceSafe. Esta tarefa é importante para proteger e compartilhar o código fonte. 						
Entradas						
Nº	Descrição do Item de Avaliação	Sim	Não	NA	RE	
01	Existe uma Ocorrência para a Análise de Implementação?	10	0	0	Sim	
Comentário: Verificado as ocorrências cadastradas						
02	Os Documentos de Especificação do Módulo, Ocorrência e Wizard de Geração de Código serviram como entrada para a Implementação?	10	0	0	Sim	
Comentário:						
03	O Código Fonte, Documento de Especificação e a Ocorrência serviram de entrada para os Testes Preliminares?	10	0	0	Sim	
Comentário:						
04	O Código Fonte e a Ocorrência serviram de entrada para Publicar a implementação?	10	0	0	Sim	
Comentário:						
Saídas						
Nº	Descrição do Item de Avaliação	Sim	Não	NA	RE	
05	A ocorrência foi analisada e classificada como Bug ou implementação e a prioridade foi definida na Análise de Implementação?	10	0	0	Sim	
Comentário: No momento da inspeção essas informações estavam cadastradas. Mas não se pode afirmar se na fase de Análise de Implementação isso foi feito.						
06	O Código Fonte foi criado e as Actions foram cadastradas no RM Específica na Implementação?	10	0	0	Sim	
Comentário: Análise realizada no código fonte e buscar realizadas no RM Específica foram as fontes de informação desse item.						
07	SourceSafe foi atualizado juntamente com o campo discussão de acordo com o padrão ao Publicar a Implementação?	10	0	0	Sim	
Comentário: No SourceSafe e no RM Agilis as informações foram comprovadas.						
Passos						
Nº	Descrição do Item de Avaliação	Sim	Não	NA	RE	
08	Antes da Implementação o Get no diretório Global e nos fontes do sistema foi realizado?	10	0	0	Sim	
Comentário: O log produzido pelo SourceSafe coincide com a data do primeiro apontamento de horas da fase de Implementação						
09	Depois da Implementação foi Verificado mensagens de Warnings?	10	0	0	Sim	

<i>Comentário: Foi requisitado ao desenvolver que compile o código fonte no Visual Studio com a ultima versão disponível no SourceSafe</i>		Evidência: MS Visual Studio			
10	Foi verificado se outros pontos do sistema não foram afetados nos Testes Preliminares?	10	0	0	Sim
<i>Comentário: Essa informação não pode ser verificável em logs e ou softwares utilizados..</i>		Evidência: Entrevista			
11	Foi inserido comentários ao executar o check in no SourceSafe?	10	0	0	Sim
<i>Comentário: Na opção "Details" do SourceSafe todas as ocorrências foram preenchidas com comentários.</i>		Evidência: MS SourceSafe			
Ferramentas					
Nº	Descrição do Item de Avaliação	Sim	Não	NA	RE
12	<i>Foi utilizado o RM Agilis e o RM Específica na Análise da Implementação?</i>	10	0	0	Sim
<i>Comentário: Os campos Solicitação, discussão entre outros são fundamentais segundo os entrevistados.</i>		Evidência: Entrevista			
13	<i>Foi utilizado o Visual SourceSafe, Visual Studio, MS Visio, MS Word, RM Agilis, Delphi na Implementação?</i>	10	0	0	Sim
<i>Comentário: Software padrões e controlados pela equipe de Infra-Estrutura da empresa.</i>		Evidência: Software Instalados e Extensão dos artefatos			
14	<i>Foi utilizado o Visual Studio.NET, MS Word, MS Visio, RM Agilis, Delphi nos Testes Preliminares?</i>	10	0	0	Sim
<i>Comentário: Software padrões e controlados pela equipe de Infra-Estrutura da empresa.</i>		Evidência: Software Instalados e Extensão dos artefatos			
15	<i>Foi utilizado o Visual Studio.NET, Visual SourceSafe, RM Agilis, Delphi para Publicar a Implementação?</i>	10	0	0	Sim
<i>Comentário: Software padrões e controlados pela equipe de Infra-Estrutura da empresa.</i>		Evidência: Software Instalados e Extensão dos artefatos			
Responsáveis					
Nº	Descrição do Item de Avaliação	Sim	Não	NA	RE
16	<i>O Desenvolvedor foi responsável por Analisar a Implementação?</i>	10	0	0	Sim
<i>Comentário: Foi verificado o Apontamento de Horas</i>		Evidência: RM Agilis			
17	<i>O Desenvolvedor foi responsável pela Implementação?</i>	10	0	0	Sim
<i>Comentário: Foi verificado o Apontamento de Horas</i>		Evidência: RM Agilis			
18	<i>O Desenvolvedor foi responsável pelos Testes Preliminares?</i>	10	0	0	Sim
<i>Comentário: Os usuário afirmaram que sempre testem as implementação.</i>		Evidência: Entrevista			
19	<i>O Desenvolvedor foi responsável por Publicar a Implementação?</i>	10	0	0	Sim
<i>Comentário: Os campos e o código fonte foi atualizado no SourceSafe produzindo logs do usuário.</i>		Evidência: SourceSafe e RM Agilis			
Observações					

VProInsp – Validação de Processo de Desenvolvimento de Software por Inspeção Relatório de Discrepâncias		
Taxonomia de Defeito:		
<i>Omissão:</i> <ul style="list-style-type: none"> ✓ Quando um dos elementos (artefato, atividade, tarefa, etapa) necessários ao processo não foram definidos. ✓ Quando faltam seções de especificação de como os artefatos, atividades, tarefa e etapas serão definidos. ✓ Quando falta definição de termos utilizados no processo. 		
<i>Ambigüidade:</i> <ul style="list-style-type: none"> ✓ Quando a forma como os elementos do processo ou suas responsabilidades foram definidos dificultam ou impossibilitam seu uso. ✓ Quando elementos possuem o mesmo nome, mas responsabilidades diferentes (Homônimo); 		
<i>Inconsistência:</i> <ul style="list-style-type: none"> ✓ Quando elementos descritos possuem mesma responsabilidade, mas nomes distintos (Sinônimo) ✓ Quando um elemento presente na execução do processo não foi definido no modelo formal ✓ Quando a representação não condiz com a semântica estabelecida pela abordagem de documentação. ✓ Quando as responsabilidades dos stakeholders não foram definidas. 		
<i>Outro:</i> <ul style="list-style-type: none"> ✓ Quando o defeito não se encaixa nas categorias acima. 		
ID Inspetor.....: 00002	Inspetor..: Túlio Mendes Rodrigues	
Nº Discrepância: 01	ID Checklist.: 001	ID Item do Checklist.: 02
Descrição: <i>Não foram encontrados diagramas UML para os dez requisitos inspecionados. Logo, não foram usados os documentos expostos no item como entrada.</i>		
Classificação da Discrepância: <input checked="" type="checkbox"/> Omissão <input type="checkbox"/> Ambigüidade <input type="checkbox"/> Inconsistência <input type="checkbox"/> Outros		
Discrepância aceita: <input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não – Motivo:		
Nº Discrepância: 02	ID Checklist.: 001	ID Item do Checklist.: 07
Descrição: <i>Não foram encontrados diagramas UML para os dez requisitos inspecionados. Logo, não foram produzidos documentos de caso de uso e outros diagramas como saída.</i>		
Classificação da Discrepância: <input checked="" type="checkbox"/> Omissão <input type="checkbox"/> Ambigüidade <input type="checkbox"/> Inconsistência <input type="checkbox"/> Outros		
Discrepância aceita: <input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não – Motivo:		
Nº Discrepância: 03	ID Checklist.: 001	ID Item do Checklist.: 12
Descrição: <i>Não foram criados diagramas de caso de uso para os requisitos. Logo não havia arquivos para serem atualizados no SourceSafe.</i>		
Classificação da Discrepância: <input checked="" type="checkbox"/> Omissão <input type="checkbox"/> Ambigüidade <input type="checkbox"/> Inconsistência <input type="checkbox"/> Outros		
Discrepância aceita: <input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não – Motivo:		
Nº Discrepância: 04	ID Checklist.: 001	ID Item do Checklist.: 21
Descrição:		

Não foram criados diagramas de caso de uso. Logo os templates não foram utilizados.		
Classificação da Discrepância:		
<input checked="" type="checkbox"/> Omissão <input type="checkbox"/> Ambigüidade <input type="checkbox"/> Inconsistência <input type="checkbox"/> Outros		
Discrepância aceita: <input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não – Motivo:		
Nº Discrepância: 05	ID Checklist.: 001	ID Item do Checklist.: 24
Descrição: <i>Não existem diagramas UML para esses requisitos no SourceSafe. Logo o desenvolvedor não criou nenhum diagrama.</i>		
Classificação da Discrepância:		
<input checked="" type="checkbox"/> Omissão <input type="checkbox"/> Ambigüidade <input type="checkbox"/> Inconsistência <input type="checkbox"/> Outros		
Discrepância aceita: <input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não – Motivo:		
Nº Discrepância: 06	ID Checklist.: 001	ID Item do Checklist.: 26
Descrição: Todas as solicitações de criação de script foram feitas pelo próprio desenvolvedor.		
Classificação da Discrepância:		
<input type="checkbox"/> Omissão <input type="checkbox"/> Ambigüidade <input checked="" type="checkbox"/> Inconsistência <input type="checkbox"/> Outros		
Discrepância aceita: <input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não – Motivo:		
Nº Discrepância: 07	ID Checklist.: 001	ID Item do Checklist.: Inexistente
Descrição: <i>Ao inspecionar os artefatos para responder o item 26 o inspetor relatou a seguinte discrepância: A solicitação de criação de script é feita na etapa de Implementação e não na Etapa de Especificação.</i>		
<i>Evidência: A data de abertura de ocorrência para os scrips eram posteriores ao apontamento de horas na fase de implementação. Ou seja, as datas de abertura de solicitação eram feitas depois do encerramento da fase de Especificação</i>		
Classificação da Discrepância:		
<input type="checkbox"/> Omissão <input type="checkbox"/> Ambigüidade <input checked="" type="checkbox"/> Inconsistência <input type="checkbox"/> Outros		
Discrepância aceita: <input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não – Motivo:		
Observações: <i>A discrepancia 07 não existia no checklist, mas foi uma discrepancia levantada a partir do item 26 e que deve ser verificada.</i>		
Quantidade total de discrepâncias levantadas: 7		
Quantidade total de discrepâncias aceitas.....: 7		

VProInsp – Validação de Processo de Desenvolvimento de Software por Inspeção Relatório de Discrepâncias	
Taxonomia de Defeito:	
<i>Omissão:</i> <ul style="list-style-type: none"> ✓ Quando um dos elementos (artefato, atividade, tarefa, etapa) necessários ao processo não foram definidos. ✓ Quando faltam seções de especificação de como os artefatos, atividades, tarefa e etapas serão definidos. ✓ Quando falta definição de termos utilizados no processo. 	
<i>Ambigüidade:</i> <ul style="list-style-type: none"> ✓ Quando a forma como os elementos do processo ou suas responsabilidades foram definidos dificultam ou impossibilitam seu uso. ✓ Quando elementos possuem o mesmo nome, mas responsabilidades diferentes (Homônimo); 	
<i>Inconsistência:</i> <ul style="list-style-type: none"> ✓ Quando elementos descritos possuem mesma responsabilidade, mas nomes distintos (Sinônimo) ✓ Quando um elemento presente na execução do processo não foi definido no modelo formal ✓ Quando a representação não condiz com a semântica estabelecida pela abordagem de documentação. ✓ Quando as responsabilidades dos stakeholders não foram definidas. 	
<i>Outro:</i> <i>Quando o defeito não se encaixa nas categorias acima.</i>	
ID Inspetor.....: 00001	Inspetor..: Sérgio Gontijo do Nascimento
Observações: Não foram encontradas discrepância para o a fase de Implementação ao executar o checklist com os 19 itens.	
Quantidade total de discrepâncias levantadas: 0 Quantidade total de discrepâncias aceitas.....: 0	

Apêndice B

Questionário de Avaliação Pós-Experimento

Por favor, responda o questionário abaixo, de forma a nos permitir o registro de sua opinião sobre a aplicação da abordagem para a validação de processo de desenvolvimento de software baseado na técnica proposta pelo VProclnsp

Nome: *Bruno Ferreira*

Papel: *Moderador*

- 1) Como você classifica o grau de dificuldade de utilização do VProclnsp?

Muito Fácil Fácil Mediana Difícil Muito Difícil
- 2) Na sua opinião, quais aspectos da técnica tornam sua aplicação fácil/difícil de usar?

A divisão do processo de validação em fases bem definidas e a utilização de checklists tornam a utilização fácil. Mas a ausência de um feedback online sobre o andamento da inspeção é uma informação difícil. Pois o moderador tem que procurar cada inspetor para saber como anda a execução do checklist

- 3) Na sua opinião, o VProclnsp se adequou a realidade da empresa e do processo validado?

Totalmente
 Satisfatório
 Atendeu as necessidades
 Poderia ser melhor
 Não
- 4) Como o VProclnsp o auxiliou a identificar discrepâncias entre a execução e o modelo do processo?

Negativamente. Os artefatos e fases atuaram como um obstáculo. O meu desempenho teria sido melhor se não tivesse utilizado.

Neutro. Acho que encontraria as mesmas discrepâncias caso não tivesse utilizado o VProclnsp.

Positivamente. Os artefatos e fases me auxiliou na detecção das discrepâncias. Talvez não tivesse detectado algumas discrepâncias caso não tivesse utilizado.
- 5) Na sua opinião, como a técnica poderia ser melhorada

Criar um software online, que informe quantos itens foram preenchidos e quantos faltam ser inspecionados para cada inspetor.

Questionário de Avaliação Pós-Experimento

Por favor, responda o questionário abaixo, de forma a nos permitir o registro de sua opinião sobre a aplicação da abordagem para a validação de processo de desenvolvimento de software baseado na técnica proposta pelo VProclnsp

Nome: *Isabela Iunes de Oliveira*

Papel: *Autor*

1) Como você classifica o grau de dificuldade de utilização do VProclnsp?

Muito Fácil Fácil Mediana Difícil Muito Difícil

2) Na sua opinião, quais aspectos da técnica tornam sua aplicação fácil/difícil de usar?

Fácil: A utilização de inspeção fazendo uso de checklist para verificar a discrepâncias do processo

Difícil: A manipulação de muitos documentos (papeis).

3) Na sua opinião, o VProclnsp se adequou a realidade da empresa e do processo validado?

Totalmente
 Satisfatório
 Atendeu as necessidades
 Poderia ser melhor
 Não

4) Como o VProclnsp o auxiliou a identificar discrepâncias entre a execução e o modelo do processo?

Negativamente. Os artefatos e fases atuaram como um obstáculo. O meu desempenho teria sido melhor se não tivesse utilizado.

Neutro. Acho que encontraria as mesmas discrepâncias caso não tivesse utilizado o VProclnsp.

Positivamente. Os artefatos e fases me auxiliou na detecção das discrepâncias. Talvez não tivesse detectado algumas discrepâncias caso não tivesse utilizado.

5) Na sua opinião, como a técnica poderia ser melhorada

Criação de um sistema para substituir os vários documentos e criar um controle de usuário para que cada participante tenha acesso a somente as informações que lhe convém.

Questionário de Avaliação Pós-Experimento

Por favor, responda o questionário abaixo, de forma a nos permitir o registro de sua opinião sobre a aplicação da abordagem para a validação de processo de desenvolvimento de software baseado na técnica proposta pelo VProclnsp

Nome: *Sergio Gontijo do Nascimento*

Papel: *Inspetor*

1) Como você classifica o grau de dificuldade de utilização do VProclnsp?

- Muito Fácil Fácil Mediana Difícil Muito Difícil

2) Na sua opinião, quais aspectos da técnica tornam sua aplicação fácil/difícil de usar?

Pode-se apontar como facilidade o uso do checklist, pois ele serve como um passo a passo. E como ponto negativo a utilização de documentos do Microsoft Word, pois pode-se apagar um item do checklist ou uma informação já preenchida acidentalmente.

3) Na sua opinião, o VProclnsp se adequou a realidade da empresa e do processo validado?

- Totalmente
 Satisfatório
 Atendeu as necessidades
 Poderia ser melhor
 Não

4) Como o VProclnsp o auxiliou a identificar discrepâncias entre a execução e o modelo do processo?

Negativamente. Os artefatos e fases atuaram como um obstáculo. O meu desempenho teria sido melhor se não tivesse utilizado.

Neutro. Acho que encontraria as mesmas discrepâncias caso não tivesse utilizado o VProclnsp.

Positivamente. Os artefatos e fases me auxiliou na detecção das discrepâncias. Talvez não tivesse detectado algumas discrepâncias caso não tivesse utilizado.

5) Na sua opinião, como a técnica poderia ser melhorada

Criar um software para auxiliar no uso do checklist durante a inspeção, ou utilizar o Microsoft Excel com algumas células travadas para proteger o conteúdo das planilhas.

Questionário de Avaliação Pós-Experimento

Por favor, responda o questionário abaixo, de forma a nos permitir o registro de sua opinião sobre a aplicação da abordagem para a validação de processo de desenvolvimento de software baseado na técnica proposta pelo VProclnsp

Nome: *Túllio Mendes Rodrigues*

Papel: *Inspecor*

1) Como você classifica o grau de dificuldade de utilização do VProclnsp?

- Muito Fácil Fácil Mediana Difícil Muito Difícil

2) Na sua opinião, quais aspectos da técnica tornam sua aplicação fácil/difícil de usar?

O uso do checklist auxiliou bastante. Mas os checklists poderiam ser informatizados.

3) Na sua opinião, o VProclnsp se adequou a realidade da empresa e do processo validado?

- Totalmente
 Satisfatório
 Atendeu as necessidades
 Poderia ser melhor
 Não

4) Como o VProclnsp o auxiliou a identificar discrepâncias entre a execução e o modelo do processo?

Negativamente. Os artefatos e fases atuaram como um obstáculo. O meu desempenho teria sido melhor se não tivesse utilizado.

Neutro. Acho que encontraria as mesmas discrepâncias caso não tivesse utilizado o VProclnsp.

Positivamente. Os artefatos e fases me auxiliou na detecção das discrepâncias. Talvez não tivesse detectado algumas discrepâncias caso não tivesse utilizado.

5) Na sua opinião, como a técnica poderia ser melhorada

Se a técnica tivesse um software para auxiliar todo o processo de validação, o trabalho seria minimizado. Logo, sugiro a criação de um software.