UMA ABORDAGEM COM APOIO FERRAMENTAL PARA APLICAÇÃO DE TÉCNICAS DE LEITURA BASEADA EM PERSPECTIVA

Luís Felipe Santos Silva

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

Aprovada por:

Prof. Guilherme Horta Travassos, D.Sc.

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

Prof. José Carlos Maldonado, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

JULHO DE 2004

SILVA, LUIS FELIPE SANTOS

Uma Abordagem com Apoio Ferramental para Aplicação de Técnicas de Leitura Baseada em Perspectiva [Rio de Janeiro] 2004

VII, 112 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2004)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Inspeções de Software
2. Requisitos de Software

I. COPPE/UFRJ II. Título (série)

Aos meus pais.

Agradecimentos

Aos meus pais, por me apoiarem integralmente na realização deste trabalho. Eu amo vocês!

À Käthe, por compreender as minhas ausências e inquietudes neste período. Obrigado por você existir em minha vida!

Ao professor Guilherme Travassos, por me ensinar a fazer ciência e mostrar que, nesta área, a crítica é realmente a medida da amizade.

Aos professores Ana Regina e José Carlos Maldonado, por aceitarem participar de minha banca.

A toda a equipe de Engenharia de Software Experimental: Ana, Arilo, Christina, Edgar, Gladys, Hélio, Kali, Léo, Marcão, Paula, Rafael, Rodrigo, Sômulo, Tayana e Wladmir.

A todo o grupo do projeto COPPETEC na Embratel: Blaschek, Fernanda, Jeann, Jonice, Mário, Marluce, Renata, Tomás, Valdino e Wanderson.

Ao CNPq, pelo apoio financeiro.

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

UMA ABORDAGEM COM APOIO FERRAMENTAL PARA APLICAÇÃO DE TÉCNICAS DE LEITURA BASEADA EM PERSPECTIVA

Luís Felipe Santos Silva

Julho/2004

Orientador: Guilherme Horta Travassos

Programa: Engenharia de Sistemas e Computação

Inspeções de software têm sido sistematicamente avaliadas e há evidências experimentais dos benefícios de sua realização. A inspeção de documentos de requisitos de software é especialmente importante por permitir a identificação de defeitos logo no início do processo de desenvolvimento, quando os custos envolvidos com sua correção são menores. A identificação de defeitos pode ser feita de forma *ad hoc* ou com uma técnica específica como, por exemplo, a leitura baseada em perspectiva (PBR).

Diversos estudos indicaram a eficácia desta técnica na identificação de defeitos. Entretanto, estes mesmos estudos revelaram aspectos negativos envolvidos com a aplicação manual da técnica. De forma a minimizar estes aspectos, esta tese apresenta uma proposta de apoio ferramental para PBR. Esta ferramenta também coleta métricas de forma não intrusiva para permitir a verificação do grau de fidelidade com que a técnica é aplicada. Para avaliar a viabilidade desta proposta, dois estudos experimentais foram conduzidos. Foi observado que a ferramenta possui estabilidade de funcionamento que permite justificar sua utilização em estudos e processos de software mais complexos.

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

AN APPROACH WITH TOOL SUPPORT FOR THE APPLICATION OF PERSPECTIVE BASED READING TECHNIQUE

Luís Felipe Santos Silva

July/2004

Advisor: Guilherme Horta Travassos

Department: Computer Science and System Engineering

Software inspections have been systematically evaluated and there are empirical evidences of its benefits. Inspecting requirements documents is especially important because it enables defect removal very early in the development process, when it is cheaper to be corrected. Defect identification in an inspection process can be accomplished in an ad-hoc fashion or using a specific technique such as Perspective-Based Reading (PBR).

Several studies indicated that PBR is an effective approach to find defects. However, these same studies revealed negative aspects involved with the manual application of the technique. In order to minimize these aspects, this dissertation presents a proposal of tool support for the application of PBR. This tool also collects metrics in an unobtrusive way in order to verify if the technique is faithfully applied. To evaluate the tool feasibility, two experimental studies were conducted. It was observed that the tool's functionalities are stable, justifying its use in more complex studies and software processes.

Sumário

Capítulo 1 - Introdução

Neste capítulo são apresentadas as motivações para a realização deste trabalho, além de seus objetivos e a forma como esta tese está organizada.

1.1 – Motivação

Num recente artigo sobre uma grande rede de lojas do varejo de eletrodomésticos e móveis (BLECHER, 2004) é citado que um sistema de informação introduzido em 1995 deveria permitir uma economia de quatro milhões de reais com a geração automática de carnês. Surpreendentemente, a utilização de tal sistema provocou um aumento acentuado na inadimplência. A razão do insucesso deveu-se a um detalhe: o tamanho do carnê. Pelo fato de não caber no bolso, os carnês eram esquecidos em gavetas e a data de vencimento passava despercebida pelos clientes. Detalhes como esse, que não são capturados durante a análise do sistema a ser construído, constituem defeitos de especificação.

A qualidade de um software não pode ser imposta depois que o produto estiver finalizado, constituindo um aspecto que deve ser tratado simultaneamente ao processo de desenvolvimento (MALDONADO E FABBRI, 2001). Como este processo envolve a participação humana, mesmo as melhores tecnologias não serão capazes de evitar a ocorrência de defeitos (LAITENBERGER e DEBAUD, 2000), que ocorrem justamente na transformação das informações pelas diferentes fases do processo de desenvolvimento (TRAVASSOS *et al.*, 2001).

Parece clara, portanto, a necessidade de se identificar estes defeitos tão logo eles sejam inseridos nos diferentes artefatos produzidos durante o processo de desenvolvimento. Neste contexto, inspeções de software têm como propósito reduzir o número de defeitos transmitidos de uma fase para outra do desenvolvimento (YOUNESSI, 2002).

Inspeção é um processo rigoroso, formal e com papéis bem definidos que pode ser aplicado em todas as fases do processo e em diferentes artefatos, incluindo requisitos, projeto de alto nível, projeto detalhado, código e planos de testes. A identificação de defeitos pode ser realizada de forma *ad hoc*, com a utilização de *checklists* ou com a adoção de uma técnica específica.

Dentre essas técnicas, as técnicas de leitura baseada em perspectiva (PBR) foram criadas com o objetivo de aprimorar a identificação de defeitos em requisitos de software escritos em linguagem natural. Este suposto aprimoramento se dá através da divisão de responsabilidades entre os inspetores, que assumem uma perspectiva específica em relação ao artefato em inspeção. As perspectivas definidas pela técnica são: testador, projetista e usuário do sistema.

PBR tem sido submetida a diversos estudos experimentais (BASILI *et al.*, 1996c; CIOLKOWSKI *et al.*, 1997; SORUMGARD, 1997; SHULL, 1998; REGNELL *et al.*, 2000; LANUBILE e VISAGGIO, 2000; NETO *et al.*, 2001; HALLING *et al.*, 2001), havendo evidências de sua maior eficiência em relação às demais técnicas. Segundo Boehm e Basili (2001), inspeções de requisitos que se utilizem das técnicas de leitura baseadas em perspectiva encontrariam, em média, 35% mais defeitos do que inspeções *ad hoc*. Por outro lado, estes mesmos autores reconhecem que a aplicação da técnica pode requerer um esforço 30% maior.

Em todos estes estudos as técnicas têm sido aplicadas manualmente. Nesta situação, reconhecemos uma série de tarefas que, se apoiadas por computador, poderiam ser feitas com menor esforço. Em razão disto, esta tese apresenta uma ferramenta de apoio à aplicação de PBR para a perspectiva do usuário, estabelecendo a hipótese de que sua utilização permitiria a aplicação da técnica em um tempo menor do que quando a técnica é aplicada manualmente.

1.2 – Objetivos

O objetivo deste trabalho é prover um apoio ferramental para a aplicação das técnicas de leitura baseada em perspectiva na inspeção de requisitos de software escritos em linguagem natural. Além de estabelecer a hipótese da redução do tempo de inspeção, a utilização desta ferramenta permitiria aos pesquisadores observar, de forma não intrusiva, como se comportam os inspetores na aplicação da técnica.

As técnicas de leitura foram adaptadas para uso industrial e treinamento em diversas instituições: Allianz Life Insurance, Bosch Telecom, Robert Bosch GmbH (SHULL *et al.*, 2001b), CPqD (FREITAS *et al.*, 2004), RioSoft.

Entretanto, a transferência de tecnologia para a indústria não é uma tarefa trivial. Para avaliar a viabilidade desta proposta e reduzir o risco da possibilidade de transferir uma tecnologia ainda imatura, foi adotada neste trabalho a metodologia proposta por SHULL *et al.* (2001a). Esta metodologia propõe uma série de

questionamentos que devem ser realizados na análise de novos processos ou técnicas, sugerindo os tipos de estudo experimentais mais adequados para obter as respostas a estes questionamentos.

De forma a evitar que esforços sejam potencialmente despendidos em vão, foi decidido, também, limitar o escopo deste apoio ferramental para apenas uma das perspectivas definidas por PBR. A decisão de estender a ferramenta proposta nesta tese só será tomada caso hajam evidências experimentais dos seus possíveis benefícios. Neste contexto é que se insere a metodologia citada acima.

É importante ressaltar que este trabalho é parte de uma infra-estrutura de apoio ao processo de inspeção de software denominada ISPIS (KALINOWSKI *et al.*, 2004), que pode ser utilizada por equipes geograficamente distribuídas para inspecionar os diferentes artefatos produzidos durante o ciclo de vida de software. ISPIS foi construída de forma a servir como arcabouço em uma arquitetura extensível de apoio à inspeção de software através de ferramentas independentes, como a descrita nesta tese. Além destas, no contexto de inspeção de software, uma ferramenta de apoio à aplicação das técnicas de leitura orientadas a objeto (REIS e TRAVASSOS, 2003) está sendo desenvolvida no âmbito do Grupo de Engenharia de Software Experimental da COPPE/UFRJ.

1.3 – Organização

Além desta introdução, esta tese está dividida em outros cinco capítulos.

No capítulo a seguir são apresentadas algumas das técnicas de verificação de software propostas na literatura científica, com ênfase na conceituação de inspeções de software.

A técnica de leitura baseada em perspectiva é descrita no capítulo 3, com os fatores que motivaram a sua criação, sua aplicação na inspeção de requisitos de software, os estudos experimentais relacionados e as adaptações feitas para a inspeção de outros artefatos de software.

No capítulo 4 é descrita a proposta de apoio computadorizado para PBR, o objetivo principal deste trabalho. São apresentados os requisitos que levaram à sua criação, a arquitetura proposta e as funcionalidades da ferramenta.

No capítulo 5 são apresentados os estudos experimentais conduzidos para avaliar a viabilidade da ferramenta.

A conclusão desta tese é feita no capítulo 6, com a apresentação das considerações finais do trabalho. São relatadas, não só as suas contribuições, mas também suas limitações e perspectivas para trabalhos futuros.

Capítulo 2 - Técnicas de Verificação de Software

Neste capítulo são apresentadas algumas das técnicas de verificação de software, com foco especial nas atividades de inspeção.

2.1 – Introdução

Podemos considerar o desenvolvimento de software como um conjunto de atividades que, progressivamente, transformam necessidades do usuário, representadas pelos requisitos do software, em representações computacionais. Temos, então, os requisitos que descrevem o sistema sendo transformados em projetos de alto e/ou baixo nível, que continuam a representar as funcionalidades esperadas, só que com mais rigor e formalismo. A transformação do projeto em código segue o mesmo raciocínio (TRAVASSOS *et al.*, 2001).

Este processo de transformação de requisitos em código envolve intensamente a participação dos desenvolvedores de software, sendo, portanto, fortemente dependente da atividade humana. Isto nos leva à situação onde, mesmo as melhores tecnologias de desenvolvimento, não serão capazes de evitar a ocorrência de defeitos (LAITENBERGER e DEBAUD, 2000), que ocorrem justamente na transformação das informações pelas diferentes fases do processo (TRAVASSOS *et al.*, 2001).

Para ilustrar esta situação podemos observar a Figura 2.1. Nesta figura, diferentes possibilidades de transformação são consideradas juntamente com suas conseqüências.

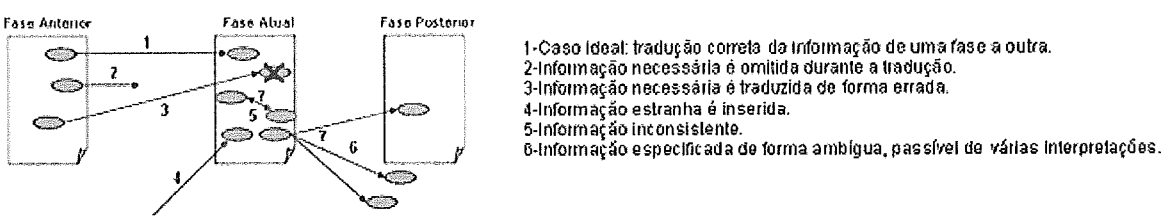


Figura 2.1 – Transformação da Informação no Desenvolvimento de Software (Adaptado de TRAVASSOS *et al.*, 2001)

A ocorrência de defeitos constitui, portanto, um aspecto inevitável do desenvolvimento do software (SHULL *et al.*, 2000). Neste sentido, as atividades de verificação e validação, inseridas no processo de desenvolvimento do software, buscam determinar se os artefatos produzidos estão adequados, atendem às necessidades do usuário e não contêm defeitos. Para atingir seu objetivo, uma atividade típica de verificação e validação deve focar nos artefatos produzidos durante o ciclo de vida do software, certificando-se de que estes artefatos estão corretos por si só (verificação) e descrevem exatamente o sistema que deve ser produzido (validação) (TRAVASSOS *et al.*, 2001).

Atividades de verificação e validação são conhecidas como atividades de garantia de qualidade. Vale destacar que como a qualidade não pode ser imposta depois que um produto estiver finalizado, ela constitui um aspecto que deve ser tratado simultaneamente ao processo de desenvolvimento do software (MALDONADO e FABRI, 2001). Neste sentido, e de acordo com a norma ISO/IEC 12207 (ISO, 1995), os processos de verificação e validação são considerados processos de apoio ao ciclo de vida e como tal devem ser planejados, gerenciados e aprimorados.

Na próxima seção deste capítulo, com o intuito de buscar a uniformização da terminologia utilizada nesta tese, serão descritos os termos aqui adotados e seus respectivos conceitos associados. Na seção 2.3 serão descritos alguns métodos para a identificação de defeitos e na seção subseqüente serão descritas as inspeções de software.

2.2 – Definição dos Conceitos

Pelo fato de encontramos na literatura terminologias inconsistentes, faz-se necessário definir os termos que serão utilizados neste trabalho. Consideramos defeito qualquer deficiência num artefato de software.

Ainda que o termo defeito seja genericamente utilizado, é importante ter em mente que sua interpretação dependerá do contexto em que ele for utilizado. As definições a seguir seguem a terminologia padrão para Engenharia de Software do IEEE (IEEE 610.12, 1990):

- Erro: ato incorreto cometido por um indivíduo ao tentar entender uma determinada informação, resolver um problema ou utilizar um método ou uma ferramenta.

- Defeito: é uma manifestação concreta de um erro num artefato de software. Um erro pode causar diversos defeitos.
- Falha: é o comportamento operacional do software diferente do esperado pelo usuário. Uma falha pode ter sido causada por diversos defeitos e alguns defeitos podem nunca causar uma falha.

Em alguns momentos o termo discrepância será utilizado. Este termo refere-se a um suposto defeito encontrado, por exemplo, através de um processo de inspeção, onde nem todo suposto defeito identificado constituirá um defeito de fato do artefato. Uma discrepância poderá ser considerada um defeito de fato e, quando isto não acontecer, teremos o chamado falso positivo.

Esta terminologia padrão do IEEE não é amplamente utilizada. Em algumas organizações o termo defeito foi substituído por caso[1]. Os desenvolvedores sentiram-se mais confortáveis, já que o termo defeito parecia colocar os autores dos artefatos sendo avaliados na defensiva, limitando a efetividade da atividade de verificação e validação, neste caso em particular, inspeção de código (FRANZ *et al.*, 1994).

É importante também estabelecer, no âmbito do desenvolvimento de software, a diferenciação entre técnica, método e tecnologia. Neste trabalho, da mesma forma que em (NETO *et al.*, 2001), será utilizada a terminologia proposta por (PORTER *et al.*, 1995):

- Técnica: série de passos cuja aplicação deve produzir um efeito desejado.
- Método: procedimento gerencial que define as técnicas que devem ser utilizadas sob determinadas circunstâncias.
- Tecnologia: coleção de técnicas e métodos.

Assim, por exemplo, inspeção de software é um método, com critérios de entrada e saída bem definidos, que pode contar com a Leitura Baseada em Perspectiva (PBR) como uma técnica para apoiar a identificação de defeitos.

2.3 – Métodos para a Identificação de Defeitos

Podemos classificar os métodos para identificação de defeitos em três grandes grupos: (1) aqueles que não envolvem a execução do software, (2) os que executam o software indiretamente e (3) os que requerem que o software esteja em operação.

Esta classificação não é ortogonal, ou seja, um determinado método pode se enquadrar em mais de uma classificação. Medição seria um exemplo de método que

[1] Do termo em inglês *issue*.

pode envolver ou não o software em execução. Com ela, as características estruturais do artefato construído ou as características dinâmicas do software em execução são confrontadas com as características desejadas, auxiliando a identificação de defeitos. Prototipagem e simulação, que virtualmente antecipam a execução real do software, seriam exemplos de métodos de execução indireta. Entre os métodos que não envolvem a execução do software podemos citar as revisões de software. O método de execução direta mais difundido é o teste de software.

2.3.1 – Medição

Medição é um método que busca avaliar a qualidade de um artefato através de suas características estruturais. O desafio é encontrar métricas apropriadas e viáveis para o atributo de qualidade de interesse. Um exemplo desta dificuldade seria avaliar a capacidade de manter um software, um atributo que não pode ser medido diretamente (TRAVASSOS *et al.*, 2001).

Este tipo de atividade tanto pode ser realizado sem a necessidade de execução do software, quando as medidas estruturais do artefato serão obtidas, quanto com o software em execução, quando se estará tentando analisar o comportamento dinâmico do artefato.

O conjunto de métricas proposto por CHIDAMBER e KEMERER (1994) para projetos orientados a objetos é um exemplo que ilustra a possibilidade de se utilizar medição como um método de identificação de defeitos. Estas métricas podem identificar possíveis módulos no projeto do software onde será mais suscetível a ocorrência de defeitos (BASILI *et al.*, 1996b). Esta identificação é importante para permitir, por exemplo, o direcionamento dos esforços de teste, já que testar exaustivamente todas as partes de um sistema pode ter um custo proibitivo.

Como exemplo, uma hierarquia de classes muito extensa pode representar um risco, já que quanto mais profunda for a definição de uma classe na hierarquia, mais difícil será prever seu comportamento, em razão do provável maior número de métodos herdados. Isto aumenta a complexidade do projeto, potencializando a ocorrência de defeitos. No conjunto de métricas propostas por CHIDAMBER e KEMERER (1994), este aspecto é tratado pela métrica DIT (Profundidade de Herança) e, quanto maior for seu valor para uma classe, maior será a probabilidade de ocorrência de defeitos nesta classe.

Métricas têm sido úteis para (TRAVASSOS *et al.*, 2001):

- Avaliar a qualidade do software (exemplo: medindo confiabilidade);

- Entender o processo de desenvolvimento (exemplo: medindo o esforço despendido em cada atividade do processo);
- Identificar problemas no produto (exemplo: identificando módulos com complexidade excessiva);
- Melhorar soluções (exemplo: entendendo a eficiência de técnicas de projeto);
- Adquirir conhecimento (exemplo: medindo o tamanho do projeto).

2.3.2 – Teste

Teste é uma atividade de verificação e validação de software desempenhada perto do fim do ciclo de vida, ou mais precisamente, perto do fim de alguma iteração do processo de desenvolvimento (TRAVASSOS *et al.*, 2001).

Esta atividade consiste na análise dinâmica do produto de software com o objetivo de verificar a presença de defeitos e aumentar a confiança de que o produto esteja correto (MALDONADO e FABRI, 2001). O grande questionamento é quando um sistema terá qualidade suficiente para que não seja mais preciso testar. Esta decisão deve considerar a probabilidade de se encontrar mais defeitos através de testes, o custo da atividade em si, a probabilidade de os usuários encontrarem defeitos e o impacto destes defeitos (HUMPHREY, 1989).

A execução de estudos experimentais, através das técnicas de experimentação aplicadas à Engenharia de Software, tem ajudado a desmistificar a crença de que teste é a técnica mais eficiente na busca por defeitos (CIOLKOWSKI *et al.*, 2002b).

Entretanto, freqüentemente, os esforços de desenvolvimento têm contado apenas com atividades de teste como atividade de verificação e validação, o que faz com que os defeitos só sejam encontrados ao fim do ciclo de vida, quando são mais caros e difíceis de serem corrigidos. Um outro aspecto a se considerar é que, a confiança excessiva nos testes, alimenta nos desenvolvedores uma tendência a se dedicar menos à produção do projeto e do código, ao assumirem que os testes identificariam todas as falhas, e a partir destas, os defeitos seriam identificados e corrigidos (TRAVASSOS *et al.*, 2001).

O detalhamento desta técnica está fora do escopo desta tese. Maiores informações sobre o assunto podem ser obtidas em (BINDER, 1999), (KANER *et al.*, 1999) e (MALDONADO e FABBRI, 2001).

2.3.3 – Revisão de Software

De acordo com o padrão 1028 da IEEE (IEEE 1028, 1998), revisão de software é definida como uma avaliação dos artefatos de software, quando é chamada revisão técnica. Uma revisão gerencial ocorre quando o *status* de um projeto é comparado com o plano original do mesmo.

Segundo este mesmo padrão, o propósito de se realizar revisões técnicas é a avaliação dos produtos de software por uma equipe tecnicamente qualificada com o intuito de identificar discrepâncias.

Esta atividade proveria evidências para confirmar que:

- O produto de software é adequado às suas especificações;
- O produto de software adere aos padrões, planos e procedimentos aplicáveis ao projeto;
- Alterações no produto foram implementadas de forma apropriada e afetam somente as partes necessárias.

Diversos métodos têm sido propostos para a realização de revisões técnicas. WEINBERG e FREEMAN (1984) destacam que os métodos de revisão são diferenciados pelo seu grau de formalidade.

Revisões técnicas de software abrangem um grande espectro de atividades, desde a verificação individual e informal, até revisões mediadas por ambientes de trabalho cooperativo, com processos bem definidos (JOHNSON, 1998).

2.3.4 – Walkthrough

O objetivo de uma sessão de *walkthrough* é avaliar um artefato de software, identificando suas partes problemáticas. Estas partes problemáticas do artefato serão corrigidas após a sessão, pelo autor. Uma reunião de acompanhamento pode ser agendada para informar os revisores sobre as medidas tomadas (FAIRLEY, 1985).

Muitas vezes esta atividade é associada à revisão de código, mas pode ser realizada em qualquer outro artefato. Segundo o padrão IEEE 1028 (IEEE 1028, 1998), além da busca por defeitos, também devem ser consideradas implementações alternativas para o artefato em revisão.

Numa sessão de *walkthrough*, o artefato é apresentado pelo seu autor e avaliado por uma equipe de três a cinco revisores. Enquanto o autor explica o artefato de software, os revisores fazem questionamentos sobre determinados aspectos deste artefato (FAIRLEY, 1985). Os membros desta equipe de revisores podem incluir o líder do projeto, um representante do grupo de qualidade da organização e outros membros

do projeto. Clientes e usuários podem fazer parte desta equipe quando o artefato em revisão for um documento de requisitos ou um projeto de alto nível.

Uma descrição detalhada de como se conduzir *walkthroughs* pode ser encontrada em (YOURDON, 1989).

2.4 – Inspeções de Software

2.4.1 – Definição

Inspeção é um método rigoroso e formal realizado para examinar artefatos de software. Seu principal objetivo é identificar os defeitos resultantes dos erros cometidos pelos desenvolvedores. Inspeções podem ser aplicadas em todas as fases do desenvolvimento e em diferentes artefatos, incluindo requisitos, projeto de alto nível, projeto detalhado, código e procedimento de testes. O propósito de se conduzir inspeções formais é reduzir o número de defeitos transmitidos de uma fase de desenvolvimento para outra (YOUNESSI, 2002). A Figura 2.2 ilustra a possibilidade de se realizar inspeções nos diferentes artefatos de software.

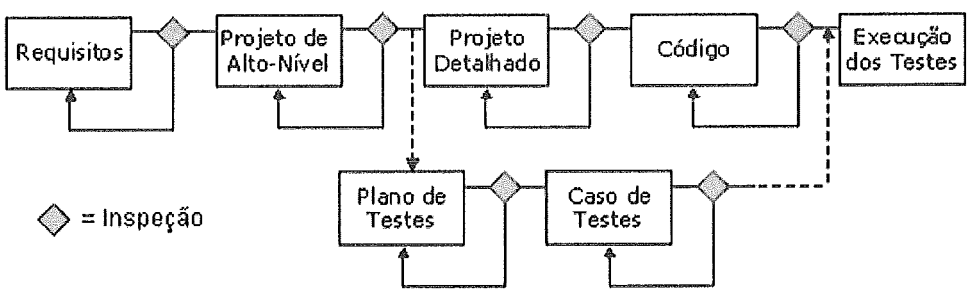


Figura 2.2 – Inspeções de Software nos Diferentes Artefatos- Adaptado de (ACKERMAN *et al.*, 1989)

Este método tem sido submetido a diversos estudos experimentais e os resultados indicam que sua realização permite a identificação de defeitos em artefatos de software tão logo estes artefatos sejam criados.

Inspeção envolve tarefas nas quais uma equipe de pessoas tecnicamente qualificadas determina se um dado artefato criado possui qualidade satisfatória. As possíveis deficiências de qualidade identificadas são subseqüentemente corrigidas. Desta forma, a atividade de inspeção contribui não só para a melhoria da qualidade do software, como também leva a ganhos significativos de prazos e custos (LAITENBERGER e DEBAUD, 2000). Como originalmente observado por FAGAN (1976), é menos custoso identificar e reparar defeitos na fase em que estes foram inseridos. Os custos associados ao teste, isolamento, correção e re-teste do software seriam muitos maiores do que o custo de se destinar um número razoável de

inspetores para inspecionar um artefato imediatamente após ele ter sido escrito (VOTTA JR, 1993).

Em um artigo comparando as diversas abordagens sobre inspeções de software encontradas na literatura[2], LAITENBERGER e DEBAUD (2000) mostraram que, conforme representado na Figura 2.3, inspeções em código são as mais comuns. Ainda assim, vale lembrar que os benefícios da atividade de inspeção serão maiores para os artefatos produzidos no início do ciclo de desenvolvimento.

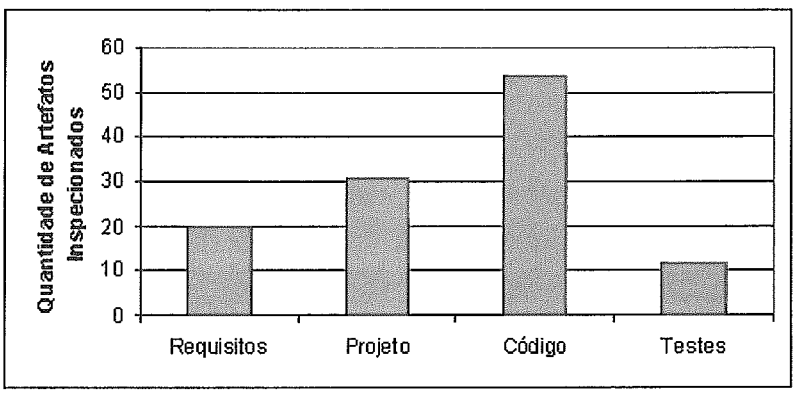


Figura 2.3 – Distribuição do Uso de Inspeção nos Diferentes Artefatos (Adaptado de LAITENBERGER e DEBAUD, 2000)

2.4.2 – Processo

No primeiro trabalho sobre inspeções de software, FAGAN (1976) propôs um processo de inspeção composto por cinco fases: Visão Geral, Preparação, Inspeção, Re-trabalho e Acompanhamento. Os objetivos de cada uma destas fases, e a sua forma de execução, são mostrados na Tabela 2-1:

Tabela 2-1 – Objetivos das Fases do Processo de Inspeção Proposto por FAGAN (1976)

Fase	Objetivos	Execução
Visão Geral	Definir os objetivos da inspeção e os papéis a serem desempenhados. Distribuir os documentos da inspeção (artefatos, procedimentos, etc.).	Em grupo
Preparação	Compreensão individual do artefato em inspeção.	Individual (Inspetores)
Inspeção	Identificar, classificar e registrar os defeitos. Não devem ser propostas soluções para os defeitos!	Em grupo
Re-trabalho	Corrigir os defeitos identificados.	Individual (Autor)
Acompanhamento	Garantir que os defeitos encontrados tenham sido corrigidos.	Individual (Moderador)

[2] Os autores analisaram 97 artigos e como alguns destes artigos tratavam de vários artefatos, o total de artefatos inspecionados foi 112.

Este processo se diferencia dos métodos tradicionais de revisão, como, por exemplo, *walkthroughs*, pela sua estruturação, formalidade e comprometimento com a identificação e correção de defeitos. É importante ressaltar que Fagan acredita que, ainda que erros flagrantes possam ser identificados durante a fase de preparação, a maior parte dos erros será identificada durante a reunião de inspeção.

Muitos dos trabalhos publicados sobre inspeção tratam das variações no processo de inspeção originalmente proposto por FAGAN (1976). Alguns autores propõem alterações na forma como os revisores se preparam para a reunião de inspeção ou mesmo na maneira como a reunião é conduzida.

GLASS (1999) acredita que as reuniões de inspeção (coleta de discrepâncias) tenderiam a retardar o progresso de um projeto em razão dos problemas de coordenação entre os revisores. Entretanto, os gerentes e desenvolvedores, de forma geral, acreditam que a sinergia provocada por estas reuniões aumentaria a quantidade de defeitos identificados. VOTTA JR (1993) teria observado na prática que esta sinergia não acontece e que as reuniões provocariam um aumento no tempo total de desenvolvimento do produto. O autor sugere, então, que o número de participantes nestes encontros seja minimizado, propondo a substituição das reuniões por encontros entre o autor do artefato e um ou dois representantes dos inspetores.

PORTER e JOHNSON (1997) analisaram esta questão ao conduzirem estudos experimentais controlados e projetados independentemente nas universidades de Maryland e do Havaí. Seu objetivo era avaliar o efeito das reuniões na efetividade do processo com relação à identificação de defeitos. Ambos os estudos indicaram não haver uma diferença substancial neste sentido. Entretanto, os autores observaram que o processo com reuniões de inspeção encontraram, ainda que em pequeno número, classes de defeitos não encontradas quando somente a inspeção individual foi realizada. Uma outra observação interessante foi a geração de um menor número de falso positivos quando as reuniões de inspeção ocorreram.

KITCHENHAM *et al.* (2002) destacam que, apesar do fato destes estudos indicarem não ocorrer a sinergia esperada nas reuniões de inspeção, estas só poderiam ser definitivamente abandonadas quando houver a certeza de que todos os demais benefícios do processo de inspeção não são afetados pela supressão desta fase.

Segundo SAUER *et al.* (2000), um processo de inspeção ruim pode impedir a realização correta da atividade de revisão. Entretanto, este mesmo autor destaca que

ter um processo excelente não será sinônimo de proficiência na identificação dos defeitos por parte dos inspetores.

As sub-seções seguintes descrevem as principais variações ao processo de inspeção originalmente proposto por FAGAN (1976).

2.4.2.1 – Processo de Inspeção de Humphrey

O processo de inspeção proposto por HUMPHREY (1989) é bastante semelhante ao processo proposto por FAGAN (1976), tendo inclusive as mesmas fases.

A diferença está na fase de preparação, quando os revisores devem identificar e registrar os defeitos, diferentemente do processo de FAGAN, onde a identificação de defeitos será, primordialmente, feita na reunião de inspeção. Os registros são consolidados numa lista única de defeitos que serão discutidos na reunião. A reunião é seguida pelas atividades típicas de pós-inspeção: Re-trabalho e Acompanhamento.

2.4.2.2 – Active Design Review –ADR (PARNAS e WEISS, 1987)

Parnas e Weiss citam que as abordagens para revisão de projetos de software seriam incompletas e superficiais. Os autores afirmam que os revisores não teriam uma responsabilidade bem definida e nem conheceriam os objetivos e restrições dos projetos em inspeção. Além disso, a interação entre os revisores e o autor do projeto seria limitada, já que as revisões são feitas em reuniões que envolvem todos os revisores, impedindo o exame detalhado de determinados pontos num projeto.

É proposta então a condução de revisões curtas, cada uma focando em diferentes tipos de defeitos e determinados aspectos num projeto de software. O processo proposto possui três fases: Visão Geral, Revisão e Discussão.

Na primeira fase, os projetistas apresentam o artefato e as responsabilidades de cada revisor são determinadas. Os revisores devem ser escolhidos de forma que haja uma correspondência entre o aspecto do projeto avaliado e a habilidade específica do revisor.

Na fase seguinte, os revisores individualmente inspecionam o projeto com a utilização de questionários, que supostamente definiriam as responsabilidades de cada revisor. Algumas das questões requerem que o revisor escreva trechos de código que traduzam a informação expressa no projeto do software. Além disso, os revisores têm que justificar a aceitação ou a rejeição das decisões de projeto sendo avaliadas.

Na fase de Discussão, em vez de ser realizada uma prolongada reunião envolvendo o autor do projeto e os diferentes revisores, cada revisor encontra-se individualmente com o autor do documento para analisar os resultados da inspeção. Em nenhum momento há uma reunião envolvendo toda a equipe. Havendo consenso em relação a um possível defeito, o próprio autor do artefato faz a correção.

2.4.2.3 – Inspeções N-Fold (MARTIN e TSAI, 1990)

Nesta abordagem, que é destinada à inspeção de documentos de requisitos para softwares críticos, "N" inspeções independentes acontecem paralelamente, sob a supervisão de um único moderador e com a utilização de um mesmo *checklist*. Os autores argumentam que o objetivo principal é a identificação dos defeitos que, provavelmente, não seriam identificados por uma única equipe.

Um estudo piloto foi realizado com 40 engenheiros de software divididos em 10 equipes independentes, que inspecionaram requisitos para um sistema em tempo real de controle de rodovias. Os resultados mostraram ter havido pouca sobreposição dos defeitos identificados pelas diferentes equipes. Na média, menos de 4 equipes identificaram um mesmo defeito. Além disso, na média, a quantidade de defeitos encontrados por uma única equipe foi pequena em comparação com o número total de defeitos, representando, então, uma cobertura de 27% (25/92). A cobertura obtida por todas as equipes em conjunto foi de 83% (77/92).

Obviamente, quanto maior o número de equipes maior será a quantidade de erros identificados. Entretanto, haverá um ponto em que a adição de mais equipes não contribuirá para um aumento significativo na cobertura de defeitos. Apesar de os autores afirmarem que a escolha do melhor valor para "N" dependerá do custo e da disponibilidade de equipes adicionais e do custo potencial de possíveis falhas no artefato em inspeção, eles não dão qualquer indicação de como esta análise deve ser feita.

2.4.2.4 – Inspeções por Fases (MYERS e KNIGHT, 1993)

Esta abordagem busca não só identificar defeitos, mas também avaliar características de qualidade como manutenibilidade, portabilidade e reutilizabilidade.

Uma Inspeção por Fases consiste em uma série de inspeções parciais coordenadas. Cada uma dessas inspeções parciais, denominada fase, tem por objetivo garantir a presença de uma única propriedade ou um pequeno conjunto de propriedades no artefato em inspeção.

Os objetivos deste tipo de inspeção são: (1) a possibilidade da aplicação rigorosa de forma que os resultados sejam repetíveis; (2) a capacidade de adaptação para que sirva a outras funções além da identificação de erros; (3) apoio computacional abrangente para que os recursos humanos sejam utilizados somente nas atividades necessárias e (4) a utilização máxima dos recursos disponíveis.

Os autores desta abordagem acreditam também que a dependência do esforço humano limita a efetividade do processo de inspeção. Assim, propõem suplementar o processo com recursos computacionais que permitiriam a utilização mais eficiente dos recursos humanos e uma cobertura maior dos artefatos a serem revisados.

O apoio computacional é fornecido com a ferramenta InspeQ. Esta ferramenta facilitaria o registro e o acesso às informações além de assegurar que o processo proposto seja seguido na prática. Isto é feito monitorando-se a atribuição de inspetores por fases, os arquivos associados, e a ordenação das fases.

A abordagem prevê a realização de inspeções por um único ou por vários revisores. Inspeções com um único revisor contam com um *checklist* para que sejam verificadas propriedades independentes da aplicação, como, por exemplo, conformidade com as normas de codificação. Nas inspeções com múltiplos revisores, cada revisor conta com um *checklist* específico para o domínio da aplicação em questão. Os revisores inspecionam o artefato individualmente e depois se reúnem para discutir os resultados obtidos.

2.4.2.5 – Inspeções Assíncronas: FTArm (JOHNSON, 1994)

FTArm é o acrônimo em inglês para o termo "Método de Revisão Técnica Formal Assíncrona". Seu principal objetivo é fornecer apoio computacional para a execução das inspeções de forma que não só os supostos problemas com a realização manual desta atividade (preparação insuficiente, dominação do moderador, conflitos pessoais, atividades burocráticas, digressão de reuniões, etc.) sejam superados como também dados sejam coletados para permitir o aprimoramento do processo. Esta não é uma abordagem específica a um determinado artefato ou a uma fase de desenvolvimento.

O processo proposto por JOHNSON (1994) é composto por 6 fases: Configuração, Orientação, Revisão Particular, Revisão Pública, Consolidação e Reunião.

Na fase de Configuração a equipe de inspeção é definida e o artefato sendo revisado é organizado numa estrutura de hipertexto.

A fase seguinte depende da complexidade do artefato e da familiaridade dos revisores com o mesmo. Assim, uma explicação geral sobre este artefato pode ser feita tanto por uma simples mensagem eletrônica como até por uma reunião presencial.

A inspeção 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.

Um relatório condensado dos resultados da Revisão Pública é gerado pelo moderador na fase de Consolidação. Se não houver consenso, uma última reunião síncrona é realizada para que os itens pendentes sejam decididos através do voto ou da decisão unilateral do moderador.

2.4.2.6 – Gerenciamento Ativo de Inspeções - AIM (HALLING *et al.*, 2002)

Segundo HALLING *et al.*, o aprimoramento das inspeções depende de um processo de gerência sistemático baseado em dados apropriados. A coleta destes dados em inspeções manuais (baseadas em papel) tem limitações, como o alto custo, a ineficiência e a interferência nas atividades realizadas durante o processo.

A automatização poderia ajudar tanto aos pesquisadores na monitoração de experimentos controlados quanto aos profissionais da área, que poderiam usar a infra-estrutura para a gerência de inspeções e coleta transparente de dados.

O Gerenciamento Ativo de Inspeções (AIM) é um conceito para aprimorar a gerência de inspeções, baseado na coleta, monitoramento e análise contínua dos dados sobre inspeções.

Em contraste com os demais processos de inspeção, na fase de planejamento, o moderador conta com dados para aprimorar o processo a ser seguido. Outros dados permitiram que, mesmo com o processo em andamento, atitudes de curto prazo, se necessárias e possíveis, sejam tomadas.

Uma representação desta abordagem é feita na Figura 2.4 e suas etapas são descritas a seguir:

- Planejamento: o objetivo desta fase é semelhante aos objetivos dos processos de inspeção: seleção dos inspetores e da técnica de identificação de defeitos, realização ou não de reuniões, etc. Entretanto, as decisões são tomadas com base na avaliação de inspeções passadas e em informações sobre o processo de desenvolvimento e o projeto em questão.
- Monitoramento do Processo Corrente: a coleta de dados é feita em tempo real para permitir o monitoramento do processo de inspeção. Estes dados permitem a alteração de determinados parâmetros como restrições de tempo e inclusão/exclusão de inspetores.
- Avaliação Pós Processo: os resultados finais do processo realizado são avaliados para identificar pontos potenciais de melhoria. O foco é entender a relação entre processo de inspeção, inspetores e artefato inspecionado.
- Melhoria do Processo: as informações coletadas podem permitir não só a melhoria do processo de inspeção em si como também do próprio processo de desenvolvimento de software. O objetivo desta fase é otimizar aspectos como treinamento dos inspetores e técnicas de identificação de defeitos.

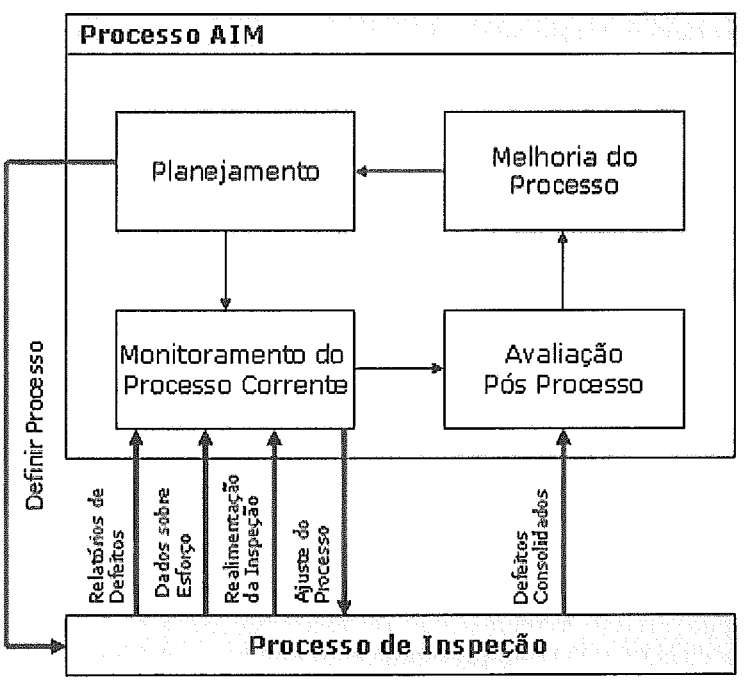


Figura 2.4 – Gerenciamento Ativo de Inspeções – Adaptado de (HALLING *et al.*, 2002)

HALLING *et al.* reconhecem a necessidade de avaliar experimentalmente o conceito proposto, deixando claro que isto será um trabalho futuro.

2.4.2.7 – Processo de (SAUER *et al.*, 2000)

Tendo por base estudos experimentais sobre inspeções de software, SAUER *et al.* (2000) propuseram uma reorganização do processo de inspeção clássico proposto por Fagan. Os autores, basicamente, substituem as fases de "Preparação" e "Inspeção" do processo de Fagan por três novas fases seqüenciais.

A primeira, "Detecção", propõe não só o entendimento do artefato (objetivo da fase de "Preparação" do processo de Fagan) como também a identificação de defeitos. As duas fases restantes, "Coleta" e "Discriminação" buscam substituir a necessidade de reuniões de inspeção. Durante a "Coleta", os defeitos identificados individualmente por cada inspetor são consolidados para que na "Discriminação" os falsos positivos sejam removidos.

2.4.2.8 – Síntese das Propostas

As propostas descritas nas seções anteriores defendem alguns aspectos que julgamos fundamentais para o processo de inspeção.

PARNAS e WEISS (1987) destacam que os inspetores devem focar em diferentes tipos de defeitos e em determinados aspectos de um projeto de software, definindo assim a responsabilidade de cada um. Um outro ponto interessante dessa proposta, e que será observada quando as técnicas de leitura baseada em perspectiva forem definidas no capítulo seguinte, é o fato dos inspetores assumirem uma postura mais ativa com relação à inspeção. Na proposta de PARNAS e WEISS, os inspetores, além de revisarem o projeto, devem também escrever trechos de código que traduzam a informação expressa no projeto. HUMPHREY (1989) defende que cada inspetor individualmente identifique e registre os defeitos antes das reuniões de inspeções.

MYERS e KNIGHT (1993) defendem o uso abrangente de apoio computacional, facilitando o registro e o acesso às informações. Além disso, a própria monitoração do processo é feita com o uso de ferramentas. JOHNSON (1994) também defende o uso deste apoio computacional não só para contornar os supostos problemas com a realização manual das inspeções como também para coletar dados para permitir o aprimoramento do processo de inspeção em si. Este aspecto também é destacado na proposta de HALLING *et al.* (2002).

2.4.2.9 – Processo utilizado neste Trabalho

Conforme visto nas seções anteriores, muitos dos trabalhos na área de inspeção focam apenas em aspectos organizacionais, como o número de

participantes, a estrutura e freqüências das reuniões de inspeção, negligenciando muitas vezes o aspecto técnico (SHULL *et al.*, 2000). Presume-se que os revisores saibam como ler o documento, focando nos pontos importantes e identificando os problemas. O problema é que isto acontece num processo lento, na base de tentativa e erro (CIOLKOWSKI *et al.*, 2002a).

Consideramos que a fase de identificação de defeitos é a mais importante do processo de inspeção, sendo uma tarefa individual, quando os revisores lêem um documento de software e aplicam alguma técnica (formal ou informal) para descobrir os defeitos. Assim sendo, adotaremos neste trabalho o processo de inspeção definido em (TRAVASSOS *et al.*, 2001) e representado na Figura 2.5, onde o foco é a identificação de defeitos. Este processo é composto pelas seguintes fases:

- Planejamento: o escopo, o artefato, o cronograma e os desenvolvedores que participarão da inspeção são decididos. As informações relevantes e os materiais são distribuídos para cada inspetor e as responsabilidades de cada um definidas.
- Identificação: os inspetores individualmente revisam o artefato, identificando defeitos e outras questões de qualidade.
- Coleta: os inspetores se reúnem para discutir o artefato e os problemas a ele associados que possam existir. Uma lista definitiva é elaborada pela equipe e entregue ao autor do artefato.
- Correção: o autor atualiza o artefato de acordo com as indicações definidas na reunião. FAGAN (1976) destaca que se mais de 5% do artefato tiver sido corrigido, deve ser realizada uma nova inspeção naquele artefato.

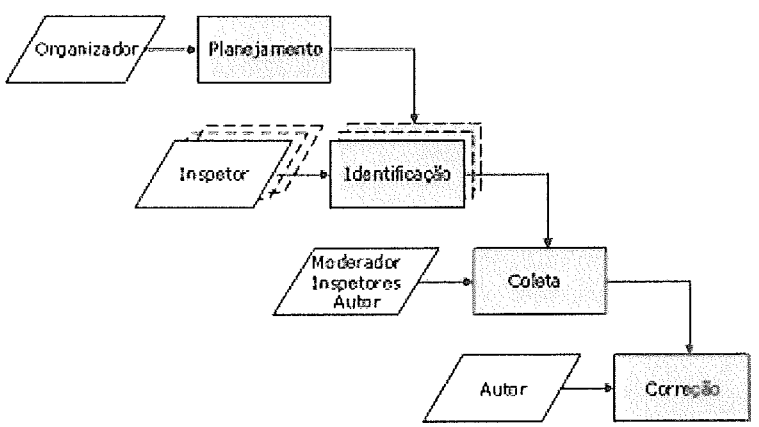


Figura 2.5 – Processo de Inspeção (TRAVASSOS *et al.*, 2001)

Neste processo, temos também os seguintes papéis:

- Organizador: planeja todas as atividades de inspeção.
- Inspetor: responsável por identificar defeitos no artefato em inspeção. A princípio, todos os membros do tipo de inspeção podem atuar como inspetores.
- Moderador: deve garantir que os procedimentos estipulados estão sendo seguidos e que cada membro da equipe assume as responsabilidades de seu papel. FAGAN (1976) afirma que o moderador é o elemento chave para o sucesso de uma inspeção. Para preservar a objetividade e aumentar a integridade da inspeção, normalmente é vantajoso utilizar um moderador de um outro projeto. O moderador deve gerenciar a equipe de inspeção e oferecer liderança. Para obtenção de melhores resultados, este moderador deveria receber um treinamento específico, ainda que este seja breve.
- Autor: produziu o artefato em inspeção e é responsável por corrigir os defeitos na fase de correção. Um autor não deve atuar como moderador.

2.4.3 – Identificação de Defeitos

As técnicas de identificação de defeitos podem variar desde procedimentos intuitivos e não sistemáticos como *ad hoc* ou *checklists* até procedimentos explícitos e altamente sistemáticos como provas formais de correção (PORTER *et al.*, 1995).

Checklist e *ad hoc* são as técnicas mais convencionais (CHENG e JEFFERY, 1996). Entretanto, estas técnicas forçam os revisores a confiar em procedimentos não sistemáticos para buscar uma grande variedade de defeitos (PORTER *et al.*, 1995).

A forma mais simples de se inspecionar um documento de requisitos é *ad hoc*, que não possui um procedimento formal ou sistemático e é baseado, largamente, na capacidade, competência e experiência do indivíduo (CHEN *et al.*, 2002). *Ad hoc* não oferece nenhum apoio já que o documento é simplesmente entregue aos inspetores sem qualquer sugestão de como proceder ou quais informações checar (LAITENBERGER *et al.*, 1997).

Checklist representa uma técnica similar à *ad hoc*, onde os itens do *checklist* tentam capturar importantes lições aprendidas de inspeções prévias em um ambiente ou domínio de aplicação (PORTER *et al.*, 1995). Este método oferece apoio na forma de questionários, que os inspetores devem responder ao inspecionarem o documento. Entretanto, estas questões são normalmente genéricas e nem sempre adaptadas ao ambiente específico onde se realiza a inspeção (LAITENBERGER *et al.*, 1997).

Poderíamos, neste ponto, argumentar que *checklist* é um método sistemático

porque ajudaria na definição das responsabilidades de cada revisor, sugerindo formas de identificar os defeitos. Entretanto, a generalidade dessas questões e a falta de estratégias concretas para responder a estas questões fariam deste um procedimento não sistemático (PORTER *et al.*, 1995). Um exemplo de *checklist* para inspeção de requisitos de software pode ser encontrado no Apêndice A.

Pesquisas revelam alguns problemas com estas duas técnicas (CHENG e JEFFERY, 1996):

- Os revisores são sobrecarregados com informações excessivas;
- Os revisores ignoram suposições e detalhes do projeto;
- As responsabilidades são ambíguas;
- Não há um caminho claro pelo qual começar a identificação de defeitos (em razão das anteriores);
- Interações limitadas entre a equipe de revisores;
- Há participação de revisores sem habilidade;
- Ausência de um procedimento de revisão sistemático e ausência de um conjunto preparado de questões a serem perguntadas.

Estudos experimentais têm demonstrado que a escolha do método de identificação de defeitos afeta, significativamente, o desempenho dos inspetores na inspeção (PORTER *et al.*, 1995).

Ainda que os revisores (que tipicamente são desenvolvedores) saibam escrever documentos de software, pouco treinamento lhes é dado na leitura (análise) destes documentos. Revisores que normalmente contam com técnicas *ad hoc*, sem um procedimento bem definido, aprendem basicamente com a experiência na execução da tarefa (SHULL *et al.*, 2000).

Uma alternativa a inspeções *ad hoc* ou com *checklists* é a utilização das chamadas técnicas de leitura. Uma técnica de leitura de software deve oferecer uma maneira sistemática e bem definida de inspecionar um artefato de software. Este formalismo garante que este procedimento possa ser seguido e repetido, possibilitando a melhoria contínua do processo de revisão.

No capítulo 3 deste trabalho serão descritas estas técnicas de leitura, dando ênfase às técnicas de leitura baseada em perspectiva (PBR) para inspeção de documentos de requisitos de software.

2.4.4 – Benefícios

Há muito pouca pesquisa avaliativa sobre as tecnologias de desenvolvimento de software (métodos estruturados, orientação a objetos, ferramentas CASE, etc.) e as que existem tendem a ser equivocadas, mostrando benefícios modestos, no melhor caso. Por outro lado, estudos experimentais sobre inspeção são bem comuns (GLASS, 1999).

Inspeções tenderiam a remover mais defeitos do que qualquer outra tecnologia e o fariam a menor custo (custo por erro seria menor) (GLASS, 1999). Tem sido mostrado que tecnologias de inspeção podem levar a taxas de 50% a 90% de identificação de defeitos (LAITENBERGER & DEBAUD, 2000).

TRAVASSOS *et al.* (2001) relatam uma série de benefícios resultantes da execução do processo de inspeção:

- Inspeções podem ser aplicadas assim que um artefato tenha sido escrito;
- Por serem normalmente executadas em equipe, inspeções são uma forma de se disseminar proficiência técnica ao se avaliar aspectos positivos e negativos de um artefato.
- Inspeções criam uma tendência de se ter artefatos cada vez mais compreensíveis.

KELLY *et al.* (1992), a partir de 203 inspeções em 5 projetos diferentes durante 3 anos no *Jet Propulsion Laboratory* da NASA, reportaram uma diminuição do tempo médio para a correção de defeitos a partir do uso de inspeções. Enquanto eram necessárias, em média, de 5 a 17 horas para identificar e corrigir um defeito com a execução de testes formais, com o uso de inspeções, este tempo médio caiu para 0,5 horas. Isto se deveu pela necessidade de se identificar e rastrear os defeitos em todos os documentos associados, corrigí-los e então, refazer o teste.

FRANZ *et al.* (1994) relatam que o retorno do investimento em inspeções seria três vezes maior que o retorno do investimento em testes.

Estes mesmos autores afirmam que, além de encontrar defeitos, que ajudam na economia de tempo e re-trabalho, o processo de inspeção possui outros benefícios intangíveis:

- Apóiam o treinamento da equipe envolvida, que se torna mais familiarizada com o sistema e confiante que ele atenderá às necessidades dos usuários;

- Atua como um excelente fórum onde serão identificadas forças e fraquezas. As forças podem ser difundidas como melhores práticas de desenvolvimento dentro da organização e as fraquezas podem ser combatidas coletivamente.

2.4.5 – Dificuldades

Ainda que inspeções tendam a remover mais defeitos que qualquer outro método, ainda assim demandam esforços e custos consideráveis (GLASS, 1999).

Apesar dos benefícios da inspeção, suas vantagens não seriam bem percebidas pelos gerentes (CIOLKOWSKI *et al.*, 2002a). O processo de inspeção tem encontrado dificuldades na sua implantação na prática. Primeiramente ele requer um investimento financeiro e de tempo para ser introduzido. Ainda que este investimento seja razoável se compararmos com os possíveis benefícios, pode haver uma certa relutância na aplicação dos recursos necessários, especialmente durante um projeto que já esteja atrasado (como a maioria o está). Um outro fator é a imagem "*low-tech*" que inspeção tem, que é contrária aos ambientes de desenvolvimento atuais, saturados de diversos tipos de tecnologias. Uma outra questão é a confusão que é feita entre inspeções, revisões informais e *walkthroughs*. Se um destes métodos já tiver sido utilizado, o convencimento da superioridade das inspeções pode ser difícil (MACDONALD *et al.*, 1995).

Ainda que trabalhos como o de CARVER (2003) tenham tentado abordar a influência de alguns fatores técnicos, não podemos afirmar que haja um completo domínio sobre como estes fatores (tamanho da equipe, grau de experiência dos revisores, etc.) influenciam na efetividade das atividades de revisão de forma geral.

LAITENBERGER & DEBAUD (2000) citam que as seguintes dúvidas existem com relação aos papéis e tamanho de uma equipe de inspeção: (1) Quais papéis estão envolvidos numa inspeção? (2) Quantas pessoas devem assumir cada papel? (3) Como selecionar as pessoas para cada papel? Estes mesmos autores assumem que não há uma resposta definitiva para o número de pessoas assumindo cada papel, já que ela depende do tipo de artefato e do ambiente no qual as inspeções acontecem.

Em relação à escolha das pessoas para cada papel, FAGAN (1976) afirma que os principais candidatos são as pessoas envolvidas com o desenvolvimento do produto. O grande questionamento passa a ser o nível de experiência destes desenvolvedores. Ainda que os mais experientes tendam a ser os melhores

inspetores, os menos experientes são os que mais aprenderiam ao participarem do processo de inspeção.

Segundo SAUER *et al.* (2000), parece haver consenso, por exemplo, quanto ao fato do desempenho individual de um revisor ser aprimorado quando lhe é fornecido treinamento. Entretanto, não sabemos exatamente o que faz um revisor tornar-se um perito. Logo, a qualidade deste treinamento pode ser questionada. Um outro questionamento refere-se ao tamanho ideal de uma equipe de revisão. Estes e outros fatores contribuem para que haja relutância, em nível gerencial, para a adoção do processo de inspeção.

Por depender da atividade humana, outros fatores não técnicos influenciam o processo de inspeção. Segundo TRAVASSOS *et al.* (2002), algumas das dificuldades seriam:

- A existência de revisores com diferentes níveis de proficiência;
- O fato de revisores se entediarem com artefatos extensos;
- O fato de revisores terem uma visão particular da importância de determinados aspectos num artefato;
- A existência de influências políticas ou pessoais.

2.4.6 – Melhores Práticas

FRANZ *et al.* (1994) fazem uma série de observações que podem ser entendidas como lições aprendidas a partir da execução de alguns processos de inspeção de código. Uma adaptação destas observações para inspeção de qualquer artefato de software é feita a seguir:

a) Vale a pena reservar um tempo para inspeções. Conseguir a aprovação gerencial pode ser difícil, especialmente quando se está sob intensa pressão de tempo. Ao assumir o compromisso de se entregar um produto com qualidade, deve-se colocar atividades de inspeção como um esforço necessário para atingir tal fim;
b) Deve-se ter uma reunião de acompanhamento adequada de forma a garantir que todos os defeitos identificados tenham sido corrigidos;
c) Padronizações devem ser estabelecidas, de forma que um tempo mínimo seja gasto com questionamentos sobre estilo, já que o foco deve ser a correção das informações contidas no artefato.
d) A atitude é um ponto chave para inspeções efetivas. Ninguém escreve artefatos de software isentos de defeitos, ainda que muitos

desenvolvedores acreditem que o façam. Os autores devem assumir que cometem erros e ter a postura de querer que os inspetores encontrem os defeitos resultantes destes erros. Os inspetores, por sua vez, podem destruir todo o processo se forem excessivamente críticos. Ao serem construtivos, mantêm o ego do autor intacto. Talvez a melhor maneira de "educar" as pessoas seja mostrar a elas que os papéis podem ser invertidos. Assim, ter um processo de inspeção para a maior parte dos projetos parece ser mais efetivo do que realizar esta atividade aleatoriamente;

e) Não deve haver a presença de gerentes nesse processo. Isto ocorrendo causa a tendência, no autor, de permanecer na defensiva, ou ainda recusar-se a prosseguir com o processo.

HUMPHREY (1989) recomenda também que não seja estabelecido um grupo específico de inspetores numa organização. O autor argumenta que, ao rever materiais idênticos ou similares por várias vezes, o inspetor poderia perder a capacidade de identificar os problemas, diminuindo a efetividade do processo.

Num trabalho recente, CARVER (2003) investigou fatores que influenciam o desempenho de um inspetor. Estes fatores estão relacionados basicamente ao nível e ao tipo de experiência do inspetor com o desenvolvimento de software. As hipóteses estabelecidas a partir dos diversos estudos experimentais conduzidos podem servir como uma guia para o planejamento de inspeções. Conhecendo os objetivos particulares de uma determinada inspeção e tendo a caracterização dos possíveis inspetores envolvidos, estas hipóteses auxiliam na escolha dos inspetores mais adequados.

2.5 – Conclusão

As atividades de verificação e validação não devem ser vistas como atividades concorrentes. Na verdade, estas atividades são complementares e uma combinação destas atividades pode auxiliar na produção de software com qualidade, dentro dos prazos e custos esperados.

Teste tem sido a atividade mais difundida, mas incluir outras atividades de verificação e validação pode ajudar a encontrar, mais cedo, defeitos no sistema em desenvolvimento. Por exemplo, inspeções de requisitos podem ajudar a identificar problemas na maneira como o sistema tratará as necessidades do usuário antes que um projeto não apropriado seja construído. Por sua vez, inspeções em projeto podem

identificar problemas na arquitetura do sistema antes que um esforço significativo seja investido na produção do código, que terá que ser refeito se problemas forem detectados mais tarde (TRAVASSOS *et al.*, 2001).

O próximo capítulo relata as técnicas de leitura de software, que se apresentam como uma alternativa às inspeções *ad hoc* ou com *checklists*. Como visto neste capítulo, estas técnicas não sistemáticas apresentam uma série de problemas.

No capítulo 3 será enfatizada a técnica de leitura baseada em perspectiva (PBR), para que a proposta deste trabalho, descrita no capítulo 4, possa ser plenamente compreendida.

Capítulo 3 - Técnicas de Leitura Baseada em Perspectiva (PBR)

Neste capítulo, as técnicas de leitura são contextualizadas de forma geral. As técnicas de leitura baseada em perspectiva são definidas e avaliadas com ênfase na inspeção de requisitos de software.

3.1 – Introdução

Muita pesquisa tem sido realizada com o intuito de melhorar, de forma sistemática, a qualidade do software. Como visto no capítulo anterior, inspeção é uma abordagem que busca atingir este fim, ao permitir a identificação de defeitos o mais cedo possível em artefatos de software, através de um processo bem definido que deve ser seguido por uma equipe de revisores tecnicamente qualificados para analisar um artefato.

No processo de inspeção, a fase de identificação de defeitos é a mais importante (SHULL *et al.*, 2000). Não existe consenso em relação à natureza dessa atividade. Alguns autores acreditam que é uma atividade a ser realizada primordialmente em grupo. Como conseqüência, conforme visto no capítulo anterior, diversos trabalhos tratam principalmente dos aspectos organizacionais de uma inspeção. Temos então variações para o processo proposto por FAGAN (1976).

De forma geral, métodos como *walkthroughs*, revisões e inspeções não tentam aprimorar a identificação de defeitos em artefatos de software alterando a maneira como os desenvolvedores compreendem o artefato, mas sim alterando a forma como eles utilizam esta compreensão, como, por exemplo, explicando este entendimento para outros desenvolvedores ou avaliando determinadas características a partir do entendimento adquirido (SHULL, 1998).

Em nível individual, nenhum destes métodos, quando realizados de forma *ad hoc*, provê um direcionamento em como os revisores podem efetivamente identificar defeitos. Tais métodos simplesmente assumem que um documento pode ser revisto efetivamente, ignorando o processo individual de compreensão (BASILI *et al.*, 1996a).

Ainda que a compreensão do documento seja uma atividade chave para a identificação de defeitos, existem poucas técnicas ou ferramentas para apoiá-la

(LAITENBERGER *et al.*, 1997). Além disso, a maior parte dos trabalhos sobre inspeção simplesmente assume que a revisão individual dos artefatos é feita de forma satisfatória (SHULL, 1998). Esta constatação e as deficiências das técnicas existentes levaram à criação das chamadas técnicas de leitura de software.

3.2 – Técnicas de Leitura

3.2.1 – Definição

Técnicas de leitura tratam do processo individual de compreensão. Ainda que o método (por exemplo, inspeção de software) nas quais as técnicas estejam inseridas possa requerer que os desenvolvedores trabalhem em conjunto, a compreensão de algum aspecto do artefato ainda será uma tarefa individual (SHULL, 1998).

No contexto de inspeções de software, uma técnica de leitura é um procedimento que guia individualmente os inspetores no entendimento de um artefato de software. Estas técnicas provêem uma maneira sistemática e bem definida de se inspecionar um artefato, auxiliando na busca por defeitos. Ao realimentarem o processo de inspeção, permitem também sua melhoria (BASILI *et al.*, 2000).

Um outro objetivo destas técnicas é determinar que cada revisor concentre-se em aspectos diferentes do artefato de forma que cada revisor tenha um conjunto distinto e bem definido de responsabilidades (CIOLKOWSKI *et al.*, 2002a).

O foco das técnicas de leitura é em como as pessoas adquirem uma compreensão de alto-nível sobre algum artefato, suficiente para entender quais partes deste documento são importantes para a execução de uma determinada tarefa, e como as pessoas compreendem estas informações em relação ao problema sendo tratado (SHULL, 1998).

A técnica deve dar um direcionamento ao desenvolvedor. O grau deste direcionamento depende do objetivo da tarefa a ser desempenhada, podendo variar de um procedimento passo a passo específico, até um conjunto de questionamentos, nos quais o desenvolvedor deva focar.

3.2.2 – Taxonomia para as Técnicas de Leitura

Artefatos de software normalmente requerem compreensão, revisão e modificação contínuas durante o ciclo de vida do desenvolvimento. A análise individual dos artefatos é então uma atividade chave em muitas tarefas na Engenharia de Software: verificação e validação, manutenção, evolução e reuso.

O uso efetivo de um artefato requer que os desenvolvedores realizem tarefas de análise e construção (BASILI *et al.*, 1996a). Com a análise do artefato, busca-se avaliar suas características de qualidade. Técnicas que apóiam esta tarefa auxiliarão o desenvolvedor tanto na compreensão da natureza e da estrutura do produto o qual o artefato em análise representa, como na identificação de possíveis defeitos do próprio artefato. Por sua vez, as tarefas de construção se apóiam no uso de artefatos existentes para a construção de um novo artefato. Técnicas de construção são importantes para a compreensão dos objetivos do sistema como um todo.

Um mesmo documento pode ser o foco de diferentes tarefas. Um documento de requisitos, por exemplo, pode ser criado (atividade de construção) e revisado (atividade de análise) antes que a fase de projeto se inicie. Este mesmo documento pode ser reutilizado em um outro projeto, no qual o sistema sendo criado tenha funcionalidades similares.

Com o intuito de facilitar a contextualização de uma técnica de leitura, SHULL (1998) desenvolveu uma taxonomia, representada na Figura 3.1 abaixo:

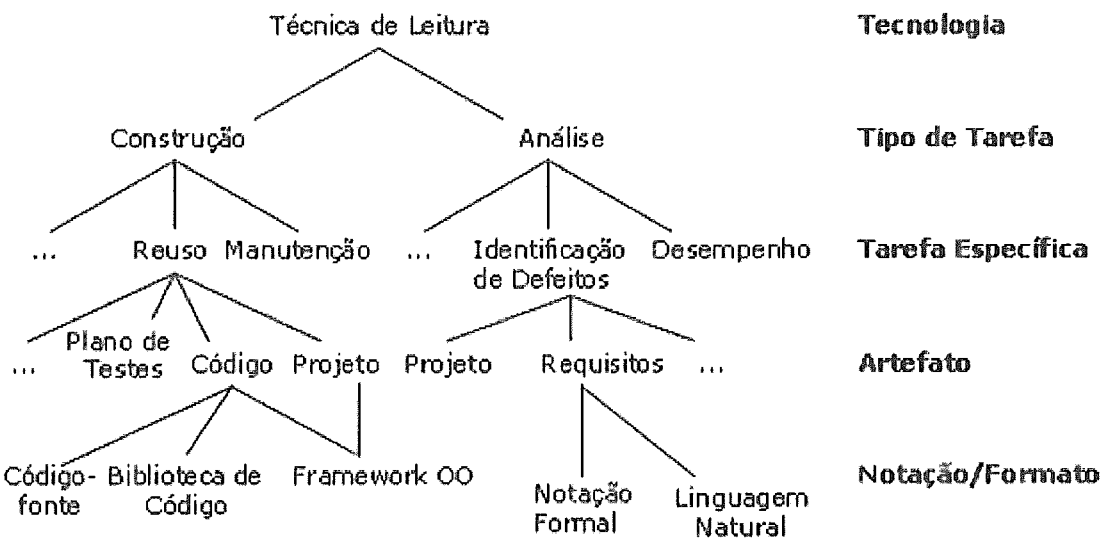


Figura 3.1 – Taxonomia para as Técnicas de Leitura – Adaptado de (SHULL, 1998)

Cada nível da taxonomia representa uma especialização do problema tratado pela tecnologia em questão, de acordo com os atributos mostrados no lado direito da figura. Num nível mais alto, todas as tarefas são classificadas como tarefas de construção ou de análise. A seguir, estas tarefas são agrupadas de acordo com o objetivo específico que ela busca alcançar. Estas tarefas específicas são sugeridas a partir das características desejadas para o software em desenvolvimento. Por

exemplo, para atingir-se correção no sistema, a identificação de defeitos deve ser realizada, e, para obter-se eficiência, é necessário avaliar o desempenho.

Algumas destas tarefas podem ser aplicadas de diferentes formas nas diferentes fases do desenvolvimento. Um exemplo seria a identificação de defeitos, que se dá de forma diferente para projeto ou requisitos. Além disso, algumas tarefas utilizam-se de documentos de múltiplas fases do processo. Um *framework* OO seria um artefato que, presumidamente, apoiaria o reuso tanto de projeto quanto de código. Por fim, o último nível da taxonomia trata da notação específica na qual o artefato está representado. Requisitos, por exemplo, poderiam ser escritos em linguagem natural ou com uma notação formal.

Para exemplificar o uso desta taxonomia, podemos citar uma família de técnicas de leitura orientadas a objetos, as chamadas OORTs (TRAVASSOS *et al*,2002). Estas técnicas (Tecnologia) buscam auxiliar a análise (Tipo de Tarefa) e a identificação de defeitos (Tarefa Específica) de diagramas de projeto orientado a objetos (Artefato) representados com a notação definida pela UML (Notação/Formato).

3.2.3 – Processo de Criação das Técnicas de Leitura

A palavra "leitura" foi escolhida para enfatizar as similaridades com o processo mental que as pessoas utilizam quando tentam entender um texto; uma técnica de leitura é um processo que o desenvolvedor pode seguir para tentar entender artefatos de software (SHULL, 1998).

O processo de criação das técnicas de leitura baseou-se em metodologia científica, especificamente no QIP (*Quality Improvement Paradigm*), que tem sido utilizado na indústria para aprimorar processos de desenvolvimento de software (BASILI, 1985). De forma resumida, este método sugere que observações sobre o mundo real devem ser feitas para que hipóteses possam ser geradas. Coletando e analisando os dados verificam-se ou não as hipóteses estabelecidas.

No desenvolvimento de software, estas hipóteses normalmente estão relacionadas ao fato de que alguma mudança no processo de desenvolvimento possa ter um efeito benéfico (por exemplo, reduzindo custo e/ou esforço). Faz-se então a alteração no processo para verificar se os resultados na prática apóiam a hipótese formulada.

No contexto do processo de criação de uma técnica de leitura, é observada a forma como um determinado artefato é utilizado. O resultado desta observação é encapsulado em dois modelos: (1) um modelo com as informações no artefato que são

importantes para a realização de uma determinada tarefa (abstrações da informação) e (2) um modelo do processo pelo qual a tarefa é alcançada (utilização da informação).

Uma técnica de leitura é gerada através do uso de cada um dos modelos, sugerindo um procedimento para execução de uma tarefa. Este é composto pelo procedimento de abstração, que apóia o desenvolvedor no entendimento da abstração, e pelo procedimento de utilização, que dá um direcionamento de como utilizar a abstração para a realização da tarefa. Uma representação desta idéia é feita na Figura 3.2.

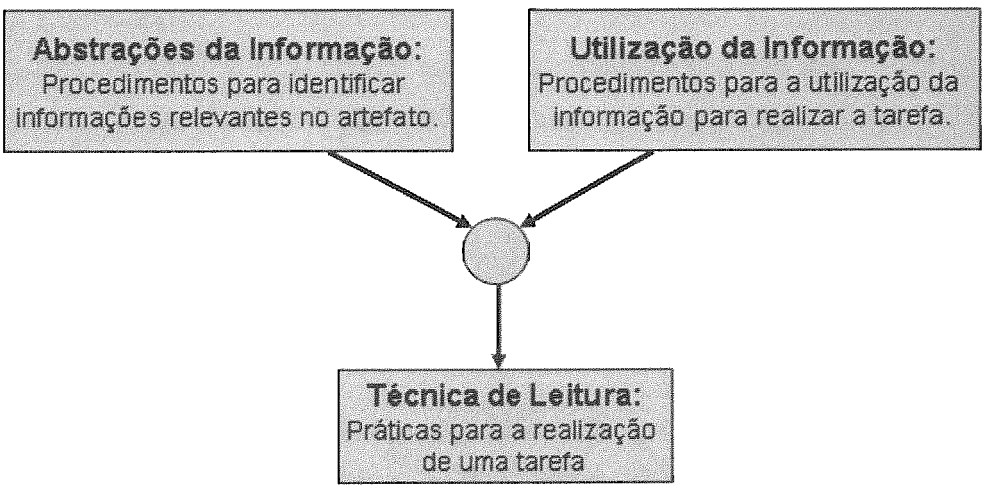


Figura 3.2 – Construção de uma Técnica de Leitura - Adaptado de (SHULL, 1998)

3.3 – Técnicas de Leitura Baseada em Perspectiva (PBR)

3.3.1 – Contextualização

As especificações de requisitos formam a base para o projeto e implementação do software, por isso elas devem ser completas, corretas, consistentes e não ambíguas. Isto não ocorrendo, defeitos serão propagados pelas demais fases do ciclo de vida do desenvolvimento do software (CHEN *et al.*, 2002).

As técnicas de identificação de defeitos em documentos de requisitos de software têm focado em duas áreas principais (CHENG e JEFFERY, 1996):

- Evitar erros: através do uso de linguagens formais de especificação.
- Identificar defeitos: através da realização de revisões como inspeções formais e *walkthroughs*.

Seguindo a taxonomia para as técnicas de leitura definida na seção 3.2.2, as técnicas de leitura baseada em perspectiva (PBR) foram moldadas para a análise (tipo de tarefa), especificamente para a identificação de defeitos (tarefa específica) em documentos de especificação de requisitos de software (artefato) escrito em linguagem natural (notação/formato) (SHULL, 1998). A Figura 3.3 representa esta contextualização, seguindo a taxonomia para as técnicas de leitura baseada em perspectiva.

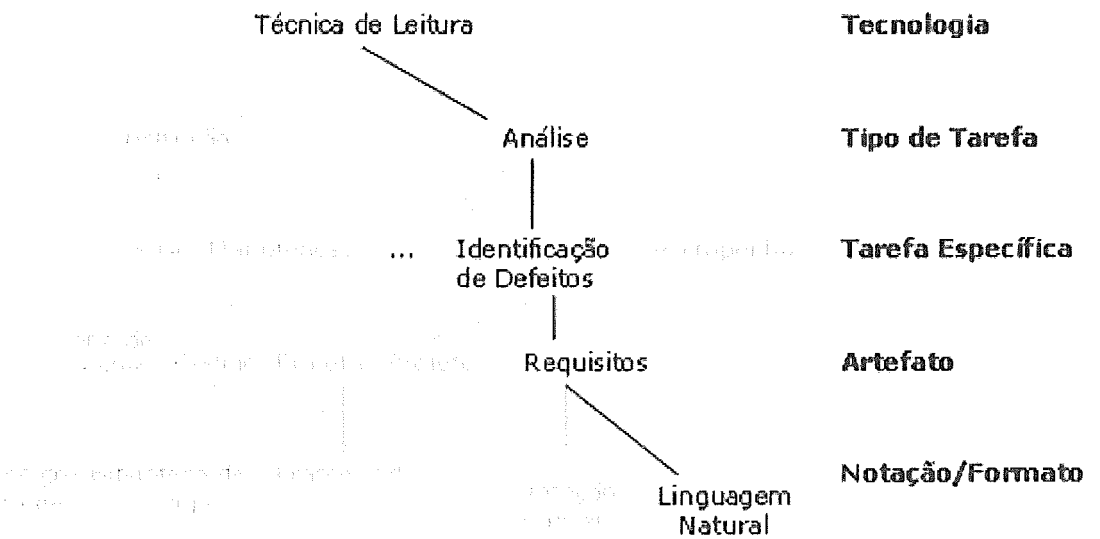


Figura 3.3 – Representação de PBR Segundo a Taxonomia das Técnicas de Leitura

3.3.2 – Documento de Requisitos de Software

O processo de criação dos requisitos envolve todas as atividades relacionadas à identificação e documentação das necessidades do usuário. A identificação da essência do problema não é uma tarefa trivial já que os problemas, as questões e as preocupações são elementos cognitivos, ou seja, eles estão na mente das diferentes pessoas envolvidas (YOUNESSI, 2002).

Na grande maioria dos processos de desenvolvimento de software, o documento de especificação de requisitos é o primeiro artefato tangível a ser produzido. Sua função principal é descrever todas as características e as funções que o software a ser desenvolvido deve possuir, atuando como um contrato entre o cliente e o desenvolvedor.

Este documento servirá de base para todas as etapas subseqüentes do desenvolvimento, sendo utilizado pelos diversos participantes do processo (patrocinadores, usuários, projetistas, programadores, testadores, etc). Uma representação simplificada desta constatação é mostrada na Figura 3.4.

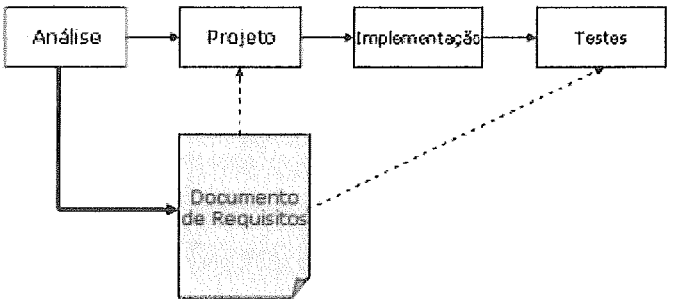


Figura 3.4 – Utilização do Documento de Requisitos em uma Iteração do Ciclo de Desenvolvimento

De forma a facilitar seu entendimento por todos estes participantes, ele normalmente é escrito em linguagem natural. É sabido que isto pode trazer uma série de ambigüidades ao seu conteúdo. Permitindo múltiplas interpretações, os modelos produzidos a partir dele podem não contemplar as características que o software deve possuir. Além disso, a própria identificação dos requisitos pode não ser feita de forma correta. Requisitos podem ser omitidos ou, ainda, não expressar corretamente o conhecimento que se tem sobre o domínio da aplicação. Há também a possibilidade de determinados requisitos contradizerem outros, deixando o documento inconsistente.

Na Tabela 3.1, temos os tipos de defeitos mais cometidos na identificação e documentação dos requisitos de software. Estes dados não são recentes, mas a prática nos tem mostrado que esta distribuição de defeitos ainda prevalece.

Tabela 3.1 - Tipos de Defeitos Encontrados em Requisitos - Adaptado de (DAVIS, 1990)

Tipo de Defeito	Percentagem (%)
Fato Incorreto	40
Omissão	31
Inconsistência	13
Ambigüidade	5
Localização Incorreta	2

Ainda que a possibilidade de defeitos persista por todo o processo de desenvolvimento do software, os problemas resultantes de defeitos durante a fase de requisitos são os mais preponderantes (YOUNESSI, 2002).

A ocorrência destes defeitos pode ter diversas conseqüências para um projeto:

- As estimativas de esforço e prazo podem deixar de fazer sentido já que seriam feitas com base em requisitos que não retratam as características desejadas para o software;

- *Desperdício de recursos ocorreria na medida em que seriam dedicados* esforços para implementação de funcionalidades incorretas ou desnecessárias;
- O produto final pode não atender as necessidades do usuário.

Assim, identificar e documentar corretamente os requisitos de um software é um fator crítico de sucesso para um projeto.

Encontramos na literatura científica diversas diretrizes para atingir este objetivo. O padrão IEEE 830 (IEEE 830,1998) recomenda diversos preceitos neste sentido. Ele define características de qualidade para um documento de especificação de requisitos bem como recomendações de como atingi-las. Estas características de qualidade com uma breve definição estão listadas na tabela abaixo:

Tabela 3.2 – Características de Qualidade para Documentos de Especificação de Requisitos – Adaptado de (IEEE 830, 1998)

Característica de Qualidade	Definição
Correto	Reflete somente a necessidade do usuário.
Não Ambíguo	Passível de apenas uma interpretação.
Completo	Reflete todas as necessidades do usuário.
Consistente	Não possui requisitos conflitantes.
Classificável	Cada requisito possui um indicativo de seu grau de importância ou estabilidade.
Verificável	Usa termos concretos e quantidades mensuráveis.
Modificável	Sua estrutura e estilo permitem que mudanças sejam feitas de forma consistente.
Rastreável	A origem dos requisitos é clara e é possível referenciá-los.

Ainda que diversas diretrizes sejam seguidas, seria no mínimo arriscado assumir que o documento não conterá defeitos. BOEHM e BASILI (2001) realçam que *corrigir defeitos após a entrega do produto pode ser até cem vezes mais caro* que corrigi-los nas primeiras fases do desenvolvimento (em projetos menores esta relação parece ser de 5:1). Além disso, em projetos recentes de software, teríamos um esforço de re-trabalho entre 40% e 50% do esforço total.

Neste contexto, realizar inspeções no documento de Especificação de Requisitos é uma alternativa muito interessante para identificar seus defeitos e assim minimizar o re-trabalho e aumentar a qualidade final do produto desenvolvido.

3.3.3 – Processo de Criação de PBR

Conforme definido na seção 3.2.3, uma técnica de leitura é criada a partir de um modelo de abstração da informação, que salienta quais informações num determinado

artefato são importantes para a realização de uma tarefa, e um modelo de utilização da informação, que orienta na execução desta tarefa.

Um documento de requisitos bem escrito deve ser capaz de apoiar os diferentes usos que este pode ter: servir de base para o desenvolvimento do projeto do sistema, permitir a criação de casos de teste apropriados às funcionalidades envolvidas e garantir que o sistema como um todo tenha as funcionalidades esperadas pelos clientes e usuários (SHULL, 1998). Assim, identificaríamos três diferentes "consumidores" deste artefato: projetistas, testadores e usuários do sistema.

Cada um destes diferentes "consumidores" serviu de base para a criação do modelo de abstração da informação para a técnica, que trata de quais informações no documento de requisitos são importantes para a identificação de defeitos. Como exemplo, um testador deve produzir um plano de testes baseado nas informações contidas nos requisitos. Se um plano de testes não puder ser criado, os requisitos estariam insuficientes já que o documento não poderia ser utilizado durante a fase de testes quando o plano de testes seria desenvolvido.

Seguindo esta linha de raciocínio foi decidido que um documento de requisitos deve apoiar a construção de três artefatos (cada um importante para cada um dos "consumidores" identificados): um projeto de alto-nível (utilizando análise estruturada), um conjunto de caso de testes (utilizando particionamento equivalente) e um conjunto de casos de uso. Esta idéia está representada na Figura 3.5. Como cada um destes artefatos é tratado separadamente no ciclo de vida do software, cada revisor deve individualmente lidar com uma dessas abstrações (SHULL, 1998).

Figura 3.5 – Diferentes Expectativas Definindo as Perspectivas para Inspeção de Documentos de Requisitos de Software

O segundo modelo, que orienta a utilização da informação contida no artefato, foi criado a partir dos possíveis defeitos existentes em requisitos de software. De forma

ampla, podemos pensar em dois tipos principais de defeitos: omissão, quando informações importantes são deixadas de lado, e comissão, quando informações estranhas, incorretas, ambíguas ou inconsistentes são inseridas no documento. Assim, um conjunto de questões foi formulado para auxiliar o inspetor na identificação dos diferentes tipos de defeitos.

Uma representação da utilização destes dois modelos para a criação da técnica de leitura baseada em perspectiva para inspeção de requisitos é mostrada na Figura 3.6.

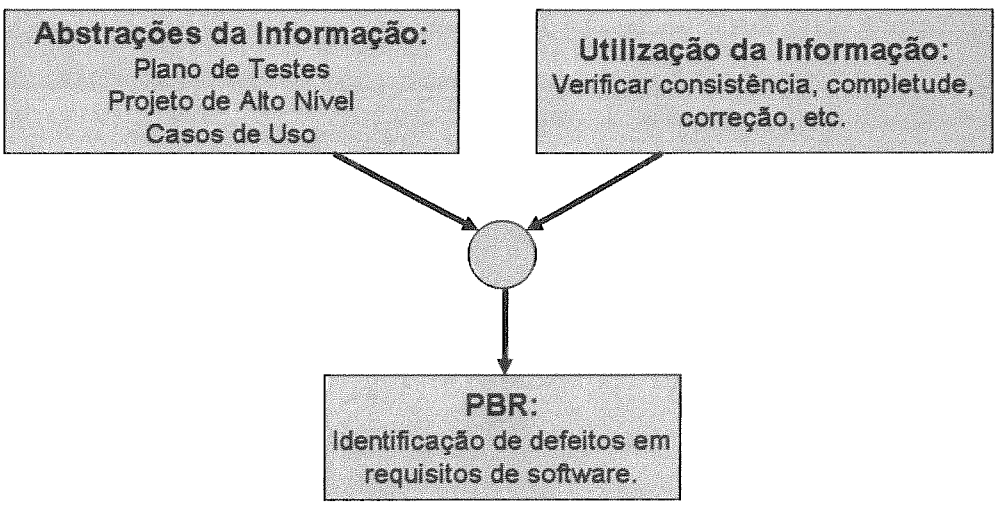


Figura 3.6 – Construção de PBR – Adaptado de (SHULL, 1998)

A técnica combina os dois modelos. Inicialmente, ela sugere atividades que levam a construção dos artefatos de alto-nível, que são abstrações do documento inspecionado. Estas atividades estão tipicamente relacionadas às atividades desempenhadas por um desenvolvedor assumindo um determinado papel no processo de desenvolvimento de software (ex: testadores construindo casos de testes). Enquanto cria este artefato de alto-nível, o inspetor responde algumas questões que o guiarão na identificação dos possíveis problemas no documento de requisitos (SHULL, 1998). As classes de defeitos são representações desses problemas e formam a base para a derivação das questões (LAITENBERGER *et al.*, 1997).

O objetivo desse procedimento é identificar se as informações contidas no documento permitem que uma representação correta do sistema seja feita. Os requisitos que não possibilitam as respostas a estas perguntas também não conterão as informações esperadas pelo seu futuro "consumidor". Quando isto acontece, uma discrepância é identificada e deve, então, ser categorizada.

PBR sintetiza alguma das idéias apresentadas superficialmente em outros trabalhos sobre inspeção de software (LAITENBERGER e ATKINSON, 1999). FAGAN (1976) já citava que trechos de código deveriam ser inspecionados por um testador, enquanto que FOWLER (1986) sugere que cada inspetor assuma um ponto de vista particular ao inspecionar um artefato e GRADEN *et al.* (1986) afirmam que cada inspetor deveria identificar a responsabilidade assumida.

3.3.4 – Classificação dos Defeitos

Para definir uma classificação de defeitos para requisitos de software foi necessário buscar um entendimento sobre como os desenvolvedores identificam defeitos nestes requisitos. O padrão IEEE 830 (IEEE 830, 1998), que recomenda práticas para especificação de requisitos de software, define atributos de qualidade que um documento de requisitos deve possuir. Foi considerado então que a falta de qualquer um destes atributos constituiria um defeito (SHULL, 1998). Com isso, a seguinte taxonomia foi definida por SHULL e sua descrição está baseada em (TRAVASSOS, 2001):

- 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.

Os autores da técnica afirmam que estes tipos de defeito não são ortogonais, isto é, um dado defeito poderia pertencer a mais de um tipo (SHULL, 1998). Além disso, esta classificação não pretende ser definitiva e cada organização pode

acrescentar mais tipos de defeito de acordo com suas necessidades (SHULL *et al.*, 2000).

3.3.5 – Exemplo de Utilização

Com o intuito de facilitar a compreensão da técnica definida anteriormente, será brevemente relatado nesta seção as tarefas propostas pela técnica para a perspectiva do usuário. Nesta perspectiva, a abstração do documento de requisitos estará representada pelo conjunto de casos de uso do sistema. A técnica completa para esta perspectiva está descrita no Apêndice B.

PBR inicialmente direciona a identificação dos participantes (atores) do sistema numa primeira leitura do documento. A técnica define o que são estes participantes e fornece dicas, através de perguntas, de como encontrá-los no documento. Um exemplo de uma dica seria: *Quais são os sistemas externos ou grupos de usuários que recebem informação do sistema?*

Tendo encontrado os participantes do sistema, o inspetor deve responder a uma série de perguntas que o auxiliarão a encontrar possíveis defeitos no documento de requisitos. Como dito anteriormente, estas perguntas são baseadas em algum tipo de defeito. Uma pergunta que auxiliaria na identificação de possíveis defeitos de ambigüidade seria: *Um mesmo participante está sendo descrito por múltiplos termos nos requisitos?*

A seguir, devem ser identificadas as funcionalidades do sistema, definidas pela técnica como as atividades explicitamente descritas nos requisitos que o sistema deve executar. Os casos de uso também devem ser identificados, descritos e relacionados às funcionalidades identificadas anteriormente. Na execução manual da técnica, é fornecida uma série de formulários para o registro destas informações.

Seguindo o mesmo princípio adotado na identificação dos atores, perguntas sobre a tarefa desempenhada são feitas para a identificação de defeitos. Para identificar defeitos de omissão, um exemplo de pergunta seria: *As condições de início para cada caso de uso estão especificadas num nível de detalhe apropriado?*

A última tarefa proposta é a associação entre os participantes e os casos de uso. Novamente são feitas perguntas que auxiliarão na identificação dos defeitos. Um exemplo seria: *"Participantes necessários foram omitidos? (existem casos de uso que requerem informação que não possa ser obtida de alguma fonte descritas nos requisitos?)".*

A dinâmica da técnica é semelhante para todas as demais perspectivas, possibilitando a inclusão não só de novos tipos de defeitos como também até de novas perspectivas, se for o caso.

3.3.6 – Características

SHULL *et al.* (2000) afirmam que a utilização de diferentes perspectivas quando PBR é aplicada, traria uma série de benefícios para o processo de inspeção, mais especificamente para a fase de identificação de defeitos. Estes benefícios diferenciariam PBR, para inspeção de requisitos de software, em relação às demais técnicas de identificação de defeitos citadas anteriormente nesta tese: *ad hoc* e *checklist*.

- Sistematização: a identificação explícita dos diferentes usos dos requisitos dá aos revisores um procedimento bem definido para ler o documento, verificando se estes usos são alcançáveis.
- Foco: PBR auxilia os revisores a se concentrarem em determinados tipos de defeitos, ao invés de escrutinar o documento de requisitos por todos os tipos de defeito. Esta característica evita a duplicação de esforço entre os membros de uma equipe de inspeção.
- Orientação a objetivos e adaptação: as perspectivas devem refletir os usos que são feitos dos requisitos. Cada perspectiva fornece um procedimento específico e este pode ser adaptado para as necessidades da organização.
- Treinamento: como PBR conta com um procedimento definido e não apenas com a própria experiência do revisor para identificar defeitos, novos revisores podem ser treinados na utilização da técnica.

Para permitir a comparação entre as técnicas de identificação de defeitos, as características citadas acima foram consideradas e estão representadas na Tabela 3.3:

Tabela 3.3 - Características das Técnicas de Identificação de Defeitos – Adaptado de (SHULL et al., 2000)

Características	Ad hoc	Checklist	PBR
Linguagem: Em que linguagem ou notação os documentos devem estar escritos?	Qualquer	Qualquer	Natural
Sistematização: Os passos específicos do processo de revisão individual são definidos?	Não	Parcial-mente	Sim
Foco: Cada revisor deve focar em diferentes aspectos do documento?	Não	Não	Sim
Aprimoramento Contínuo: A partir da realimentação dos revisores, aspectos da técnica podem ser melhorados?	Não	Parcial-mente	Sim
Adaptação: A técnica pode ser adaptada para projetos ou organizações específicas?	Não	Sim	Sim
Treinamento: A técnica permite e necessita de treinamento?	Não	Parcial-mente	Sim

3.3.7 – Hipóteses: Possíveis Benefícios

PBR auxiliaria os revisores a responderem duas importantes questões sobre os requisitos que eles estão inspecionando: (1) Quais informações devem ser verificadas nos requisitos? (2) Como identificar defeitos nessas informações? (SHULL *et al.*, 2000).

A maior parte das técnicas de inspeção não auxilia o revisor em focar aspectos particulares do documento. Elas tratam todas as informações como sendo igualmente importantes já que representam funcionalidades e restrições reais do sistema a ser desenvolvido. PBR tem a premissa que as diferentes informações contidas no documento terão graus de importância variáveis de acordo com o uso que é feito do documento, ou seja, qual atividade do ciclo de desenvolvimento será apoiada pelo documento sob análise. Assim, PBR sugere um conjunto de revisões individuais, cada uma sob um ponto de vista particular, que em conjunto cobririam todos os aspectos relevantes de um documento.

PORTER *et al.* (1995) acreditam que abordagens de identificação de defeitos que dêem aos revisores responsabilidades específicas e não sobrepostas irão resultar em inspeções mais eficientes. Ao requerer que cada revisor assuma uma perspectiva específica em relação ao documento inspecionado, PBR ajudaria na busca por uma certeza que o documento de requisitos contem uma qualidade suficiente para servir de base para as fases de desenvolvimento subseqüentes (BASILI *et al.*, 2000). Foi feita também a hipótese de que o revisor seria mais cuidadoso, ou seja, mais atento a

detalhes do documento, pelo fato de ele(a) ter que se concentrar em um único aspecto ao invés do documento como um todo (SHULL, 1998).

Os autores da técnica supuseram que a união das diferentes perspectivas proveria uma cobertura extensiva do documento, ainda que cada revisor seja responsável por uma visão mais estreita do mesmo. Este último fato é que levaria a uma análise mais profunda dos potenciais erros no documento (SHULL, 1998).

3.3.8 – Estudos Experimentais

As técnicas de leitura baseada em perspectiva têm sido submetidas a diversos estudos experimentais. Algumas evidências e observações experimentais destes estudos serviram de motivação para a proposta de apoio ferramental a ser descrita nesta tese. Em razão disso, serão relatados nas seções seguintes os principais estudos envolvendo as técnicas de leitura baseada em perspectiva e seus resultados.

→ NASA (BASILI *et al.*, 1996c):

Este primeiro estudo de PBR fez a comparação entre a técnica e a identificação *ad hoc* de defeitos num ambiente industrial. Realizou-se, inicialmente, um estudo piloto com 12 desenvolvedores e um segundo estudo contou com 13 desenvolvedores. Dois documentos de requisitos de domínios genéricos (terminais bancários ATM, 17 páginas, e sistema de controle de estacionamento PGCS, 16 páginas) e dois documentos do domínio particular da organização (dinâmica de vôo) compuseram a instrumentação do estudo. Os principais resultados foram:

- Desenvolvedores aplicando PBR nos documentos genéricos identificaram significantemente mais defeitos que os que inspecionaram de forma *ad hoc*;
- Desenvolvedores aplicando PBR nos documentos do domínio da NASA identificaram mais defeitos que os que inspecionaram de forma *ad hoc*, ainda que esta diferença não tenha sido significativa;
- Equipes virtuais (composição aleatória dos desenvolvedores) identificaram significantemente mais defeitos que as equipes de desenvolvedores que inspecionaram de forma *ad hoc*, tanto nos documentos genéricos quanto nos específicos ao domínio da NASA;
- Desenvolvedores mais experientes não foram os que identificaram mais defeitos.

→ KAISERSLAUTERN (CIOLKOWSKI *et al.*, 1997):

Este estudo comparou PBR com a identificação *ad hoc* de defeitos na universidade de Kaiserslautern, na Alemanha, utilizando os documentos genéricos do

estudo da NASA (ATM e PGCS). Uma primeira rodada do estudo contou com 25 estudantes e a segunda com 26 estudantes. Os resultados mostraram que:

- Desenvolvedores aplicando PBR identificaram significantemente mais defeitos que os que inspecionaram de forma *ad hoc*;
- Equipes virtuais (composição aleatória dos desenvolvedores) identificaram significantemente mais defeitos que as equipes de desenvolvedores que inspecionaram de forma *ad hoc*;
- Nenhuma perspectiva identificou de forma significativa mais tipos de defeitos específicos do que outra;

→ TRONDHEIM (SORUMGARD, 1997):

Este estudo, realizado na universidade de Trondheim, Noruega, comparou PBR com uma segunda versão da técnica (PBR2), na qual os revisores recebiam instruções mais detalhadas em como aplicar a técnica. Os documentos utilizados no estudo, que contou com 48 estudantes, foram os do domínio genérico (ATM e PGCS). Foi evidenciado que:

- Desenvolvedores aplicando PBR2 não identificaram significantemente mais defeitos que os que inspecionaram com a versão original da técnica;
- Desenvolvedores aplicando PBR2 levaram significantemente mais tempo para inspecionar os documentos;
- Desenvolvedores aplicando PBR2 relataram muito menos defeitos;
- A eficiência dos indivíduos que utilizaram PBR2 foi significantemente menor.

→ MARYLAND (SHULL, 1998):

Contando com 66 estudantes da universidade de Maryland, este estudo também utilizou os documentos dos sistemas ATM e PGCS. Os resultados mostraram que:

- Desenvolvedores aplicando PBR identificaram significantemente mais defeitos que os que inspecionaram de forma *ad hoc*;
- Entre os desenvolvedores com muita experiência, os que utilizaram PBR não identificaram significantemente mais defeitos que os que inspecionaram de forma *ad hoc*;

- Entre os demais desenvolvedores (pouca ou média experiência), os que utilizaram PBR identificaram significantemente mais defeitos que os que inspecionaram de forma *ad hoc*;
- A produtividade (custo eficiência) dos desenvolvedores que utilizaram PBR foi menor.

→ LUND (REGNELL *et al.*, 2000):

Realizado na universidade de Lund, na Suécia, este estudo utilizou exatamente os mesmos instrumentos utilizados no estudo da NASA (BASILI *et al.*, 1996c). Entretanto, em vez de comparar PBR com uma outra técnica de identificação de defeitos, tentou-se obter um melhor entendimento do desempenho das diferentes perspectivas. Na verdade, buscou-se saber se as perspectivas encontrariam defeitos diferentes e se uma determinada perspectiva teria um desempenho superior às outras. O estudo contou com 30 estudantes de mestrado.

Esta análise levou em conta a efetividade, isto é, qual a razão entre a quantidade de defeitos encontrados e a quantidade de defeitos existentes e também a eficiência, que mostra quantos defeitos são encontrados por unidade de tempo.

Os resultados mostraram não haver, nos aspectos analisados, uma diferença significativa entre as perspectivas. Eles evidenciaram que:

- Não há uma diferença significativa entre as perspectivas com relação à efetividade e eficiência;
- Não há uma diferença significativa entre as perspectivas com relação ao grau de cobertura dos defeitos, ou seja, as diferentes perspectivas não levariam a identificação de diferentes defeitos como pré-suposto pela técnica.

Não parece ter havido qualquer alteração substancial na técnica ou nos instrumentos que pudessem explicar estes resultados. Na verdade, os trabalhos anteriores não buscaram fazer uma analise semelhante a esta, mas sim comparar PBR com outras técnicas de identificação de defeitos.

Os próprios autores reconhecem que um único estudo não seria suficiente para tornar estas conclusões definitivas. Realizar uma análise semelhante nos dados de experimentos anteriores e conduzir novas repetições deste estudo em outros contextos trará mais evidências para os questionamentos propostos por este estudo.

→ BARI (LANUBILE e VISAGGIO, 2000):

Este estudo, realizado na universidade de Bari, na Itália, comparou uma suposta versão melhorada de PBR com a identificação de defeitos *ad hoc* e com *checklists*. A primeira rodada do estudo, que comparou PBR a *ad hoc*, contou com 114 estudantes de graduação enquanto que a segunda contou com 109 estudantes. Os documentos inspecionados foram o ATM e o PGCS, utilizados em estudos anteriores. Os resultados do estudo não corroboraram com os resultados obtidos na maior parte dos estudos envolvendo PBR.

A análise do desempenho das equipes de inspeção revelou que:

- A técnica de identificação de defeitos utilizada não teve um efeito significativo. Todas as técnicas (*ad hoc*, *checklist* e PBR) levaram a um desempenho semelhante.

Com relação ao desempenho individual, foi evidenciado que:

- A inspeção feita de forma *ad hoc* teve um desempenho melhor que a inspeção utilizando PBR.
- A inspeção feita utilizando *checklists* e PBR apresentaram desempenhos semelhantes.

É importante ressaltar que os autores não mencionaram como PBR teria sido supostamente aprimorada. Eles utilizaram as perspectivas do usuário e do projetista, mas substituíram a perspectiva do testador por uma nova perspectiva, a do analista OO. Nesta perspectiva, o modelo de alto nível a ser criado seria um diagrama de classes. Logo, ainda que se esteja adotando um outro paradigma, esta nova perspectiva estaria intimamente relacionada à perspectiva do projetista, tendo grande possibilidade de sobreposição de responsabilidades com a mesma. Além disso, a supressão da perspectiva do testador provavelmente comprometeu a cobertura de defeitos dos documentos.

Um outro ponto interessante foi a análise dos questionários de acompanhamento do estudo revelar que apenas 20% dos estudantes seguiram fielmente a técnica de leitura baseada em perspectiva. Os próprios autores reconhecem que esta foi uma grande ameaça a validade do estudo. Além disso, o fato de os pesquisadores vincularem o desempenho dos estudantes no estudo a uma nota "bônus" no curso o qual estes participavam, ainda que a participação tenha sido "voluntária", é no mínimo questionável.

Como conseqüência, parece claro que os resultados deste estudo não apoiariam os resultados previamente obtidos em outros estudos.

→ USP São Carlos (NETO *et al.*, 2001):

Este estudo, realizado na Universidade de São Paulo e contando com 18 estudantes de graduação, repetiu o estudo descrito por BASILI *et al.* (1996c), obtendo um resultado semelhante. Foram inspecionados os mesmo documentos: PGCS e ATM. As evidências foram:

- PBR apresentou um desempenho superior a *checklist*.
- No documento ATM, *checklist* obteve uma cobertura de 46% (17/37) enquanto que PBR obteve 62% (23/37). Além disso, em média, os inspetores que utilizando PBR identificaram 3,72 defeitos/hora enquanto que os que utilizaram *checklist* identificaram 2,13 defeitos/hora.
- No documento PGCS, *checklist* obteve uma cobertura de 53% (20/32) e uma eficiência média de 3,53 defeitos/hora. Já PBR obteve 50% (16/32) de cobertura e uma eficiência média de 4,25 defeitos/hora.
- Cada perspectiva identificou determinados defeitos que as demais não identificaram, indicando a complementaridade entre as perspectivas.
- Ambas as técnicas tiveram um desempenho em relação a cobertura dos defeitos proporcionalmente superior com o documento ATM, ainda que este documento contenha requisitos funcionais mais complexos. Uma possível explicação é o maior conhecimento deste domínio por parte dos inspetores, conforme relatado nos questionários de acompanhamento.

É importante mencionar que os inspetores utilizaram os instrumentos do estudo na língua inglesa. Ainda que os próprios inspetores tenham declarado serem capazes de ler e entender nesta língua, isto pode constituir um fator de confusão no estudo.

→ VIENA (HALLING *et al.*, 2001):

HALLING *et al.* (2001) compararam a identificação de defeitos em requisitos com PBR a inspeção com *checklists*. Entretanto, ainda que ressaltem terem utilizado apenas uma "notação diferente", o documento de requisitos inspecionado contava com diagramas de análise OO, modelos de dados, etc. Como resultado, os procedimentos da técnica tiveram que ser adaptados. Esta adaptação de PBR para permitir a inspeção dos diagramas de análise foi baseada nas chamadas técnicas de leitura orientadas a objetos (TRAVASSOS *et al.*, 1999).

Os estudo envolveu estudantes de graduação da Universidade Técnica de Viena inspecionando documentos de requisitos do domínio de gerência administrativa, sendo executado em duas rodadas, contando com 169 estudantes na primeira rodada e 177 na segunda.

O fato de a técnica ter sido adaptada inviabiliza qualquer comparação com estudos anteriores com a técnica original. Ainda assim, os resultados evidenciaram que:

- Em média, inspetores utilizando *checklists* encontraram mais defeitos que os que utilizaram PBR.
- A identificação de defeitos utilizando *checklist* é mais dependente da qualificação do inspetor.
- Inspetores numa determinada perspectiva são mais efetivos no foco proposto pelo cenário (tipos de defeitos) do que os que utilizaram *checklist* ou uma outra perspectiva com um foco diferente.

É interessante notar que os autores também realizaram uma análise por seções dos documentos inspecionados. O desempenho dos inspetores que utilizaram *checklist* foi melhor como um todo, mas em seções específicas dos documentos, a utilização das diferentes perspectivas foi vantajosa: a perspectiva do usuário teve o melhor desempenho para as regras de negócio, a perspectiva do projetista para os modelos de análise OO e a perspectiva do testador para os modelos de dados.

3.3.9 – Dificuldades

As principais dificuldades da utilização da técnica de leitura baseada em perspectiva estão relacionadas à necessidade de criação dos modelos de alto nível.

Este trabalho adicional representa a maior restrição a PBR: o esforço adicional que o uso da técnica demanda é estimado em 30% ou mais, dependendo do documento revisado (SHULL *et al.*, 2000).

Entretanto, não é intenção da técnica a duplicação do trabalho. Este esforço adicional será diluído pelas demais fases do desenvolvimento já que as representações criadas durante a inspeção devem servir de base para a criação dos modelos mais detalhados nas fases subseqüentes do desenvolvimento. Assim, os modelos de casos de uso iniciais produzidos durante a inspeção de um documento de requisitos podem, e devem, ser utilizados para a criação dos casos de uso mais

detalhados que serão produzidos quando o documento de requisitos já tiver sido devidamente inspecionado e apresentando uma qualidade satisfatória.

Um outro ponto importante a se destacar é a experiência necessária para a criação destes modelos de alto nível. O objetivo da criação destes modelos é que os desenvolvedores obtenham melhor entendimento do documento em inspeção. Entretanto, conforme pudemos notar num estudo de observação realizado no segundo semestre de 2003 na COPPE/UFRJ com estudantes de graduação, desenvolvedores que não possuem experiência na criação destes modelos levam um tempo excessivo para criá-los e têm uma tendência a se dedicar mais a criação destes modelos do que a buscar defeitos no documento em si (Este estudo será detalhado no capítulo 5). A solução mais plausível para esta dificuldade parece ser fornecer treinamento a este tipo de desenvolvedor antes de envolvê-los em processos de inspeção de software. No outro extremo, alguns estudos mostraram que os revisores mais experientes algumas vezes ignoram a técnica e utilizam heurísticas próprias. Os desenvolvedores que já tenham participado de inspeções em diversos outros projetos já teriam desenvolvido uma abordagem pessoal, não se beneficiando do uso de PBR (SHULL *et al.*, 2000).

3.3.10 – Outras Aplicações de PBR

3.3.10.1 – PBR para Inspeção de Código

Baseados nos resultados da aplicação de PBR para inspeção de requisitos de software, LAITENBERGER e DEBAUD (1997) acreditaram que PBR poderia ser adaptada e aplicada em inspeções de código. A Figura 3.7 contextualiza este estudo seguindo a taxonomia definida na seção 3.2.2.

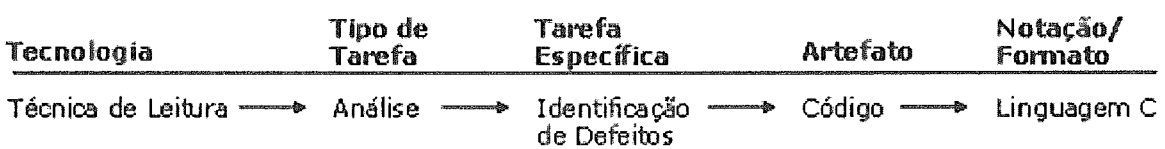


Figura 3.7 – PBR para Inspeção de Código

Para identificar as possíveis perspectivas, os autores analisaram o processo de desenvolvimento da Robert Bosch GmbH, uma organização multinacional alemã. Foram identificadas as seguintes perspectivas: Analista (checa a consistência entre código, projeto e especificações), Testador de Módulos (verifica a correção interna dos módulos) e Testador de Integração (verifica as interfaces entre os módulos).

Tendo identificado as perspectivas, foi desenvolvido um cenário operacional para cada uma delas. As atividades desempenhadas pelo desenvolvedor que assumisse uma dada perspectiva foram documentadas, de forma que pudessem ser seguidas como um procedimento passo a passo. Para apoiar a identificação de defeitos, foram formulados questionamentos a partir da classificação de defeitos existentes na própria organização. Como pode ser visto na Figura 3.8, os autores também estimaram quais perspectivas encontrariam os diferentes tipos de defeitos.

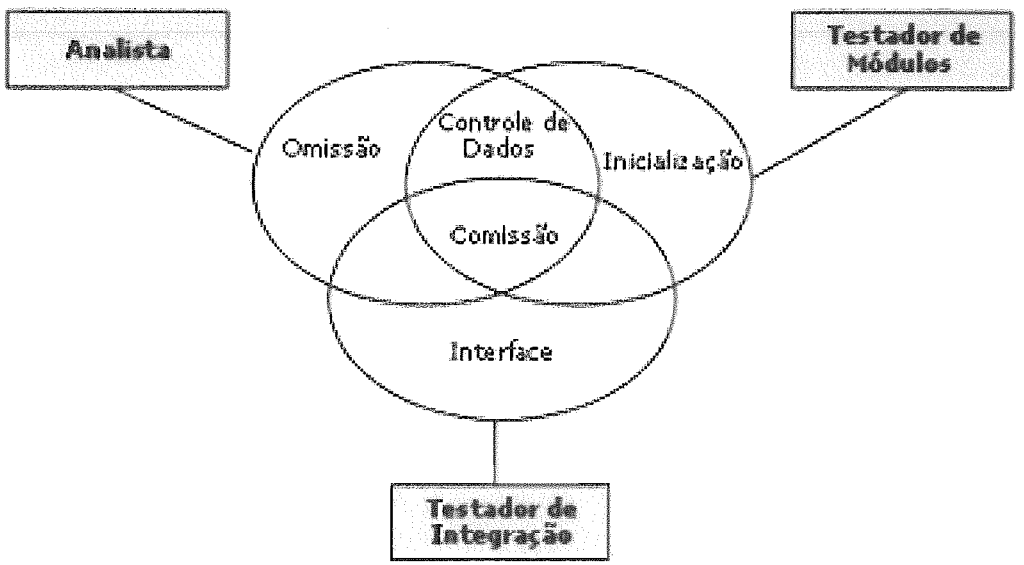


Figura 3.8 – Classes de Defeito por Perspectiva – Adaptado de (LAITENBERGER e DEBAUD, 1997)

Foi realizado um estudo com 11 desenvolvedores profissionais da Bosch. Na média, estes desenvolvedores tinham 19 meses de experiência na linguagem C (utilizada nos artefatos em inspeção), 19,7 meses de experiência no domínio de aplicação, pouca ou nenhuma experiência com inspeção de software. O objetivo do estudo não foi comparar PBR com uma outra técnica, mas sim avaliar a viabilidade de utilizar esta técnica para a inspeção de código.

Os pesquisadores buscavam respostas para as seguintes perguntas:

- (1) Os resultados individuais são afetados pelas diferentes perspectivas e pelo domínio da aplicação?
- (2) A identificação de defeitos é uma tarefa individual ou coletiva?
- (3) Qual o grau de sobreposição entre os defeitos encontrados pelas diferentes perspectivas?

Os resultados do estudo revelaram que:

- (1) A aplicação de PBR e o tipo de documento tiveram influência na identificação individual de defeitos. Não houve diferenças significativas com respeito à identificação de defeitos entre inspetores com diferentes níveis de experiência na linguagem de programação.
- (2) A quantidade de defeitos encontrados durante as reuniões foi muito menor do que a quantidade de defeitos encontrados individualmente. Este resultado confirmou a hipótese dos pesquisadores de que a identificação de defeitos é muito mais uma tarefa individual do que coletiva.
- (3) A sobreposição de defeitos entre os inspetores utilizando diferentes perspectivas foi pequena. Os resultados mostraram que não houve uma perspectiva supérflua, ou seja, cada perspectiva contribuiu efetivamente para a identificação dos defeitos.

Os autores têm plena consciência da necessidade de repetição deste estudo e das ameaças a sua validade. Em pesquisas futuras eles buscarão identificar quais são as perspectivas necessárias para se obter uma cobertura total do documento.

3.3.10.2 – PBR para Inspeção de Artefatos de Projeto Orientado a Objetos

Encontramos na literatura dois trabalhos que, independentemente, adaptaram PBR para inspeção de artefatos de projeto O.O. modelados com a UML: LAITENBERGER e ATKINSON (1999) e SABALIAUSKAITE *et al.* (2002).

A contextualização destes trabalhos, seguindo a taxonomia definida na seção 3.2.2, está representada na Figura 3.9.

Tecnologia	Tipo de Tarefa	Tarefa Específica	Artefato	Notação/ Formato
Técnica de Leitura →	Análise →	Identificação de Defeitos →	Projeto OO →	UML

Figura 3.9 – PBR para Inspeção de Projeto OO

LAITENBERGER e ATKINSON (1999) destacam que o termo defeito tem sido utilizado como uma anomalia num artefato de software e que esta interpretação limitaria o processo de inspeção por considerar correção como o único fator de qualidade. Para ilustrar esta idéia, citam o fato do mantenedor de um sistema estar interessando não somente na correção de um projeto, mas também em quão bem as boas práticas de projeto foram adotadas, e quão fácil seria realizar alguma modificação. Nesta visão, um artefato pode estar semântica e sintaticamente correto, mas pode ser considerado defeituoso se for de difícil manutenção.

Neste artigo os autores identificaram e descreveram as responsabilidades para 4 perspectivas diferentes: engenheiro de requisitos, projetista, testador e mantenedor. Entretanto, não foi relatado como os defeitos deveriam ser classificados.

Num segundo artigo, estes mesmos autores (LAITENGERBER *et al.,* 2000) descrevem um estudo experimental controlado, com 18 desenvolvedores profissionais, para comparar esta instanciação de PBR com *checklists* para projetos O.O. modelados em UML. Surpreendentemente, e sem qualquer justificativa, a perspectiva do engenheiro de requisitos foi suprimida. As classes de defeitos utilizadas continuaram sendo omitidas.

Para compreender as responsabilidades de cada uma das perspectivas identificadas é necessário destacar que os modelos de projeto inspecionados contaram não só com os diagramas tradicionais da UML como também com esquemas de operações, que definem as propriedades funcionais das operações do sistema.

Um inspetor assumindo a perspectiva do projetista estará interessado na correção entre os modelos de projeto e os esquemas. Desvios entre estes dois modelos seriam defeitos em potencial. O testador identifica as diferentes operações a partir dos modelos de projeto e formula casos de teste que mostrariam o comportamento de cada operação. Na perspectiva do implementador, deve-se verificar que toda informação requerida para a implementação do sistema pode ser extraída dos modelos de projeto. Isto envolve não só a verificação da completude dos modelos como também a viabilidade do projeto.

Os resultados indicaram que, em média, PBR teve uma cobertura de 58% (41% superior a *checklist*) e um custo médio por defeito de 56 minutos por defeito (58% menor que *checklist*).

O grande valor deste trabalho está no fato de ser uma primeira abordagem para adaptar PBR para inspeção de projetos O.O.

O trabalho de SABALIAUSKAITE *et al.* (2002) compara a sua adaptação de PBR para projeto O.O. com *checklists*, num experimento controlado com 59 estudantes. As perspectivas adotadas foram a do usuário, do projetista e do implementador. A classificação dos defeitos utilizada foi uma sumarização da classificação proposta por TRAVASSOS *et al.* (1998). Os defeitos foram classificados como sintáticos (omissão e informação estranha), semânticos (fato incorreto e ambigüidade) ou de inconsistência. Entretanto, não é justificada a escolha das perspectivas, as responsabilidades de cada uma delas e o por quê da classificação de

defeitos utilizada. Em razão destes motivos, decidimos não relatar os resultados deste estudo. Seu registro neste trabalho é muito mais uma questão de evidenciar sua existência do que a nossa concordância de sua relevância para a área.

3.3.10.3 – PBR para Inspeção de Usabilidade

ZHANG *et al.* (1999) propuseram a utilização de PBR para a inspeção de usabilidade. A Figura 3.10 representa uma contextualização desta proposta.

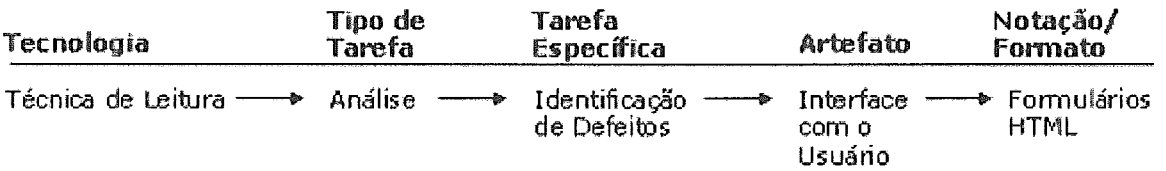


Figura 3.10 – PBR para Inspeção de Usabilidade

Inspeções de usabilidade diferem-se de outros métodos de avaliação de usabilidade, como, por exemplo, teste de usabilidade. Nestes últimos, os problemas de usabilidade são encontrados através da observação de usuários interagindo com uma determinada interface enquanto que na inspeção de usabilidade os defeitos serão encontrados a partir da proficiência do inspetor e da técnica que ele estiver utilizando.

A avaliação heurística, que tem sido o método mais popular para inspeção de usabilidade, envolve uma série de revisores examinando uma interface com o usuário e julgando sua conformidade com os princípios de usabilidade reconhecidos (as chamadas heurísticas). Fica claro que a efetividade da inspeção estará intimamente relacionada à proficiência do inspetor.

A partir destas observações, ZHANG *et al.* (1999) propuseram que PBR fosse utilizada para a inspeção de usabilidade, tendo dois diferenciais fundamentais: (1) cada inspetor deve ter foco e responsabilidades específicos e (2) um procedimento bem definido para cada perspectiva deve ser dado. Cada perspectiva representaria um determinado ponto de vista, contando com uma lista de questões que representariam os aspectos de usabilidade a serem checados e um procedimento a ser seguido durante a inspeção. Os pesquisadores assumiram a hipótese que perspectivas distintas focariam a atenção do inspetor em subconjuntos específicos de aspectos de usabilidade. Para isso, estas perspectivas deveriam ser o mais mutuamente exclusivas possíveis.

As perspectivas foram definidas a partir de dois questionamentos: (1) O usuário sabe como atingir o objetivo proposto? (2) O usuário executa a tarefa proposta corretamente?

Uma resposta negativa para a pergunta (2) leva a existência de uma perspectiva para tratamento de erros. A resposta positiva para a pergunta (1) define a perspectiva do proficiente e o contrário a perspectiva do novato. A caracterização destas perspectivas é mostrada na Tabela 3.4 a seguir:

Tabela 3.4 - Perspectivas para Inspeção de Usabilidade com PBR

Perspectiva	Característica	Objetivo
Novato	O conhecimento e a experiência do usuário não auxiliam o usuário a utilizar o sistema para atingir seu objetivo.	Inspetores devem pensar como os usuários que não estão familiarizados com a interface, devendo receber uma orientação desta em como executar corretamente as ações, perceber as respostas do sistema e entender os resultados das ações.
Proficiente	O usuário sabe como utilizar o sistema e o tenta fazer de forma eficiente, ou tenta atingir objetivos mais sofisticados.	Inspetores devem pensar como os usuários familiarizados com a interface e examiná-la com respeito à eficiência, flexibilidade e consistência no apoio a execução das ações. São verificados aspectos como existência de atalhos, valores default, organização da informação na tela, etc.
Tratamento de Erros	O usuário tem um problema como efeito de uma ação previamente executada, precisando contornar este problema.	Inspetores primeiramente produzem uma lista com os possíveis erros para cada tarefa a ser desempenhada. A seguir, para cada possível erro, o inspetor deve checar se a interface provê mensagens informativas e minimiza a conseqüência deste erro.

Vale ressaltar que outras perspectivas poderiam ter sido utilizadas, especialmente para casos particulares de determinados tipos de aplicações.

Os pesquisadores conduziram um experimento controlado, com 24 desenvolvedores profissionais, comparando a técnica proposta com a avaliação heurística na inspeção de interfaces para aplicativos baseados na Internet. Eles definiram a hipótese que tanto no nível individual quanto no coletivo (combinação das perspectivas) os inspetores identificariam uma maior porcentagem dos erros de usabilidade.

Os resultados mostraram que:

- As perspectivas do novato e de tratamento de erros identificaram significantemente mais defeitos que a avaliação heurística (de 30% a 200% a mais);

- A combinação das perspectivas resultou em mais erros encontrados (em média, 30% a mais).
- A perspectiva do proficiente não identificou significativamente mais defeitos que a avaliação heurística.

Uma provável explicação para este último resultado é o fato de os inspetores que assumiram esta perspectiva serem proficientes tanto no domínio da aplicação quanto na utilização da interface. Assim, eles identificaram muitos dos problemas sem nem mesmo utilizarem a técnica proposta.

3.4 – Conclusão

As técnicas de leitura baseada em perspectiva têm sido sistematicamente avaliadas através de estudos experimentais. Neste capítulo, foram relatados vários desses estudos, que em sua grande maioria evidenciam a superioridade de PBR com relação a outras técnicas de identificação de defeitos, como *ad hoc* e *checklist*. Diversas observações feitas nestes estudos serviram como fonte de requisitos para a proposta de apoio ferramental da técnica a ser descrita no próximo capítulo.

É importante relatar também que em vários dos estudos envolvendo a técnica, principalmente os que não evidenciaram sua superioridade, é dado somente um mínimo de informações sobre o contexto no qual os experimentos foram realizados. Percebe-se, em alguns artigos e teses, que o relato destes estudos é feito de forma superficial, como se eles fossem repetições fiéis do estudo original (BASILI *et al.*, 1996c), quando, na verdade, na maior parte das vezes, envolve adaptações radicais da técnica.

Ainda assim, todos estes estudos envolvendo PBR têm contribuído de forma significativa para um acúmulo de conhecimento na área de experimentação em Engenharia de Software. Esta área, comparada com outras como Medicina e Psicologia, carece, ainda neste momento, de maturidade.

Capítulo 4 - PBR Tool - Apoio Ferramental para Técnicas de Leitura Baseada em Perspectiva

Neste capítulo são apresentadas as motivações para a proposta de apoio ferramental para PBR. São descritos o escopo, os requisitos, a arquitetura e as funcionalidades da ferramenta.

4.1 – Motivação

Como vimos nos capítulos anteriores, inspeções de software têm sido sistematicamente avaliadas e há indícios experimentais de sua eficiência, ou seja, elas ajudariam a identificar, com custos justificáveis, defeitos em artefatos de software produzidos ao longo do ciclo de vida do mesmo.

Entretanto, como observado por MACDONALD *et al.* (1995), o processo de inspeção de software, por natureza, requer um certo esforço. As diferentes fases do processo envolvem uma série de atividades como a preparação e distribuição de materiais, definição dos inspetores e seus papéis, consolidação das discrepâncias identificadas, entre outras.

Esta constatação motivou o surgimento de algumas propostas de apoio computadorizado ao processo de inspeção como um todo, envolvendo principalmente apoio ao planejamento da inspeção e controle do processo de inspeção em si: ISPIS (Kalinowski *et al.*, 2004), Assist (MACDONALD, 1997), InspectA (MURPHY e MILLER, 1997) e Hypercode (Perpich *et al.*, 1997).

Muitas destas propostas têm sido experimentalmente avaliadas e há evidências do aumento da eficiência do processo de inspeção quando este é apoiado por ferramentas computacionais.

Por acreditarmos que a fase de identificação de defeitos é a mais importante do processo de inspeção, e sendo ela também composta por uma série de atividades clericais, é natural vislumbrarmos a possibilidade de se ter um apoio específico a esta fase do processo.

Existem algumas abordagens na literatura para fornecer apoio computacional na identificação de defeitos quando *checklists* são utilizados: Scrutiny (GINTELL *et al.*, 1995), PAE (BELLI e CRISAN, 1996), CSI/CAIS (MASHAYEKHI *et al.*, 1993). Vale

destacar que estas ferramentas, como a maior parte das abordagens propostas na literatura, foram desenvolvidas para a inspeção de código.

PORTER *et al.* (1995) citam que a verificação de um documento de requisitos é uma tarefa que, freqüentemente, é realizada de forma manual. MACDONALD *et al.* (1995) observaram que automatizar a identificação de defeitos em requisitos de software é muito mais difícil do que fazê-lo em código. Neste último caso, por exemplo, declarações inalcançáveis e possíveis conflitos de tipo podem ser automaticamente identificados em código C através de ferramentas como o *lint* do UNIX.

Nesta tese é proposta uma ferramenta de apoio para a identificação de defeitos em requisitos de software através da utilização de PBR para a perspectiva do usuário.

4.2 – Determinação do Escopo

Pelo fato da maior parte dos documentos de especificação de requisitos de software ser escrita em linguagem natural, a automatização completa da atividade de identificação de defeitos num processo de inspeção torna-se uma tarefa extremamente complexa, já que muito destes defeitos são encontrados pelo inspetor através do seu conhecimento do domínio. Entretanto, é legítimo pensar numa ferramenta que apóie o inspetor na execução desta atividade (SILVA e TRAVASSOS, 2003).

SHULL (1998) destaca que os profissionais da área de software encontram uma série de dificuldades no momento de escolher as ferramentas de apoio mais adequadas ao desenvolvimento de software. Algumas razões para esta constatação seriam o fato destas ferramentas:

- Serem muito difíceis de integrar na forma padrão de executar as tarefas rotineiras em uma organização;
- Não darem a devida atenção às necessidades dos usuários;
- Requererem que o desenvolvedor invista muito esforço em técnicas ou tecnologias que não são efetivas num ambiente particular;
- Não terem seus limites de efetividade conhecidos.

Este mesmo autor ressalta que esta situação poderia ser melhorada se o desenvolvimento de processos e a escolha de ferramentas se baseassem na avaliação experimental, levando-se em conta particularidades de cada organização.

As observações feitas anteriormente foram levadas em conta para definir o escopo deste trabalho. A estratégia adotada então foi de prover um apoio ferramental para a identificação de defeitos com PBR apenas para a perspectiva do usuário. Esta

estratégia foi tomada para que a decisão de estender a ferramenta para outras perspectivas seja tomada a partir de uma fundamentação científica, ou seja, fruto de evidências experimentais. No capítulo 5 serão apresentados os estudos experimentais conduzidos para avaliar a ferramenta descrita neste capítulo.

4.3 – Requisitos

Os requisitos para a ferramenta foram identificados a partir das seguintes fontes:

- Adaptação de funcionalidades oferecidas por outras abordagens, identificadas na literatura para prover apoio ferramental para o processo de inspeção como um todo ou para uma das fases do processo em particular;
- A experiência do próprio autor desta tese (e de seus colegas de mestrado e doutorado) como usuário da técnica;
- A experiência do orientador deste trabalho na condução de diversos estudos experimentais envolvendo a técnica;
- Críticas, sugestões e troca de experiências entre os membros da equipe do Projeto READERS[3] em *workshops* realizados na Universidade Federal do Rio de Janeiro e na Universidade de São Paulo em São Carlos.

Vale ressaltar que a versão da ferramenta a ser apresentada neste capítulo é fruto também das sugestões dos participantes dos estudos experimentais conduzidos para avaliar a viabilidade da ferramenta.

PBR Tool, na visão do inspetor, possui duas características principais:

a) fornecer um guia para a execução das tarefas especificadas pela técnica para a perspectiva do usuário;

b) facilitar o relato das discrepâncias identificadas.

Uma outra característica da ferramenta é permitir aos pesquisadores avaliar, através da observação não intrusiva, se a técnica PBR é aplicada fielmente (SILVA e TRAVASSOS, 2004).

Dessa forma, são listados abaixo os requisitos funcionais e não funcionais para a ferramenta.

- ***Requisitos Funcionais***
    - **Requisito RF1**

[3] Projeto que envolveu pesquisadores da Universidade de Maryland, USP São Carlos, COPPE/UFRJ, Instituto Fraunhofer de Maryland, UFSCar e UNIFACS na pesquisa de temas relacionados à complementaridade entre técnicas de inspeção (leitura) e testes de software.

A ferramenta deve permitir a identificação do projeto e do inspetor numa sessão de inspeção.

- **Requisito RF2**

A ferramenta deve permitir a abertura e visualização de documentos de texto nos formatos TXT e RTF. A ferramenta deve permitir ao usuário registrar o rótulo utilizado para identificar requisitos funcionais e não-funcionais para que o conteúdo desses requisitos possa ser visualizado individualmente.

- **Requisito RF3**

A ferramenta deve permitir a criação, a alteração e a exclusão dos elementos associados à criação de casos de uso. Na ferramenta, esses conceitos são oriundos da definição da técnica PBR e são: participante, funcionalidade, caso de uso (nome, início e descrição).

- **Requisito RF4**

A ferramenta deve permitir a identificação direta dos conceitos descritos no requisito anterior a partir do texto visualizado pelo usuário.

- **Requisito RF5**

A ferramenta deve permitir o relato, a alteração e a exclusão de discrepâncias. São informações de uma discrepância: um ou mais requisitos (funcionais ou não funcionais), uma classificação (omissão, ambigüidade, informação estranha, informação incorreta, inconsistência ou outros), uma localização (linha ou seção) e uma descrição. Além dessas informações, a ferramenta deve também registrar a tarefa que está sendo executada quando uma discrepância é identificada.

- **Requisito RF6**

A ferramenta deve oferecer ajuda na execução de qualquer uma das tarefas. Na identificação de discrepâncias, essa ajuda deve ser específica à classificação dos defeitos.

- **Requisito RF7**

A ferramenta deve garantir que uma determinada tarefa da técnica só seja executada quando as tarefas anteriores a esta tenham sido pelo menos

iniciadas. A ferramenta deve registrar o tempo e a ordenação de execução de cada uma dessas tarefas.

- **Requisito RF7**

  A ferramenta deve permitir a geração de um relatório de casos de uso, contendo todos os conceitos a ele associados. Este relatório deve ser visualizado em navegador de internet comum (ex: Internet Explorer).

- **Requisito RF8**

  A ferramenta deve permitir a geração de um relatório de discrepâncias. Este relatório deve ser visualizado em navegador de internet comum (ex: Internet Explorer).

- **Requisito RF9**

  A ferramenta deve registrar, além das informações explicitamente informadas pelo usuário, o tempo total de inspeção e o número de acesso ao recurso de ajuda da ferramenta.

- ***Requisitos Não-Funcionais***
  - **Requisito RNF1**

    A ferramenta deve garantir a persistência e recuperação de todos os documentos manuseados (discrepâncias e elementos do modelo de casos de uso). Essa persistência deve ser feito em arquivos no formato XML.

  - **Requisito RNF2**

    A ferramenta deve estar disponível para todas as versões dos sistemas operacionais Windows.

4.4 – Arquitetura e Projeto

A ferramenta desenvolvida para apoiar a aplicação de PBR para a perspectiva do usuário é formada por 4 módulos principais que viabilizam a execução da técnica. Outros 3 módulos têm como papel auxiliar os pesquisadores no entendimento de como a técnica é realmente aplicada. Uma representação destes componentes é feita na Figura 4.1 e, em seguida, são descritas as responsabilidades de cada um deles.

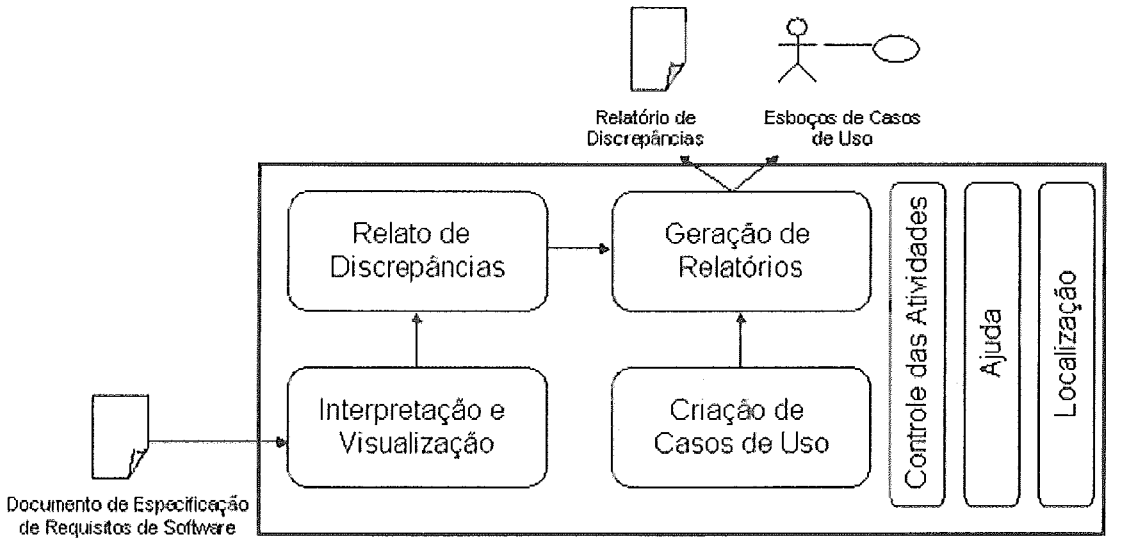


Figura 4.1 - Arquitetura da Ferramenta

- **Interpretação e Visualização**: componente que permite a discriminação (requisitos funcionais e não funcionais) do conteúdo do documento de especificação. É responsável também pela exibição e impressão do documento.
- **Relato de Discrepâncias**: componente responsável por permitir o relato das discrepâncias identificadas.
- **Criação de Casos de Uso**: componente que permite a identificação de participantes, funcionalidades e casos de uso, bem como a determinação do relacionamento entre estes conceitos.
- **Geração de Relatórios**: componente que, a partir dos dados persistidos em XML, utiliza a tecnologia XSLT para gerar relatórios visualizáveis em qualquer navegador (*browser*) de internet.
- **Controle das Atividades**: componente que garante que as atividades executadas pelo inspetor respeitam a ordenação das atividades propostas originalmente pela técnica. Responsável também por coletar, de maneira não intrusiva, métricas relativas à execução da técnica.
- **Ajuda**: componente que busca auxiliar os inspetores na execução de qualquer uma das atividades propostas pela técnica.
- **Localização**: componente que permite a instanciação da ferramenta em diferentes linguagens.

A seguir, na Figura 4.2, é representado o modelo do domínio do problema sendo considerado, mostrando os conceitos essenciais tratados pela ferramenta.

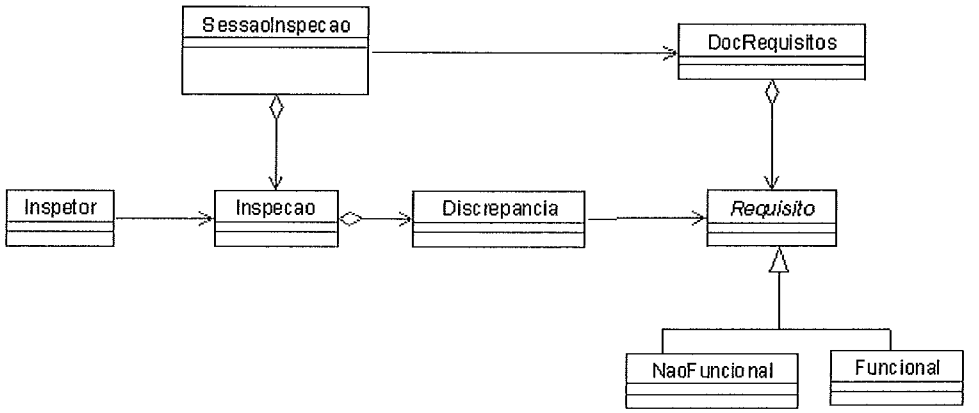


Figura 4.2 - Modelo de Análise

4.5 – Desenvolvimento da Ferramenta

O desenvolvimento desta ferramenta seguiu um processo iterativo e incremental. Na primeira iteração foram desenvolvidos os seguintes módulos: Interpretação e Visualização, Geração de Relatórios e Relato de Discrepâncias. Neste momento, as funcionalidades da ferramenta buscavam justamente facilitar o relato de discrepâncias feito, na versão original da técnica, de forma manual.

Em seguida, percebeu-se a possibilidade da ferramenta atuar como um guia para a aplicação da técnica. Como conseqüência, foram desenvolvidos os módulos de Criação de Casos de Uso e Controle das Atividades. No fim desta iteração, a viabilidade da ferramenta foi avaliada experimentalmente.

A realização de estudos experimentais motivou uma nova iteração, na qual o módulo de Controle das Atividades foi aprimorado. Ao fim desta iteração, a ferramenta passou a coletar também o tempo e a ordem de execução das tarefas, medida fundamental para permitir a verificação da conformidade da aplicação da técnica.

É importante destacar que a ferramenta foi desenvolvida na linguagem C++, utilizando como ambiente de desenvolvimento o Borland C++ Builder, versão 6.0.

4.6 – Solução

4.6.1 – Metáfora da Interface Gráfica com o Usuário

A metáfora da interface gráfica de PBR Tool foi inspirada nas ferramentas CASE desenvolvidas para a estação TABA (ROCHA *et al.*, 1990), que é um meta-

ambiente capaz de gerar ambientes de desenvolvimento de software adaptados aos diferentes processos de desenvolvimento.

Em PBR Tool, as tarefas a serem desempenhadas pelo inspetor são representadas por ícones localizados num *menu* lateral. Para efeito de organização, este *menu* foi dividido em 5 abas: Projeto, Participantes, Funcionalidades, Casos de Uso e Referência Cruzada. Esta organização está baseada na distribuição das tarefas que compõem a técnica em quatro atividades, além de uma atividade inicial de identificação do inspetor e do documento a ser inspecionado (Projeto). A Figura 4.3 mostra esta organização do menu.

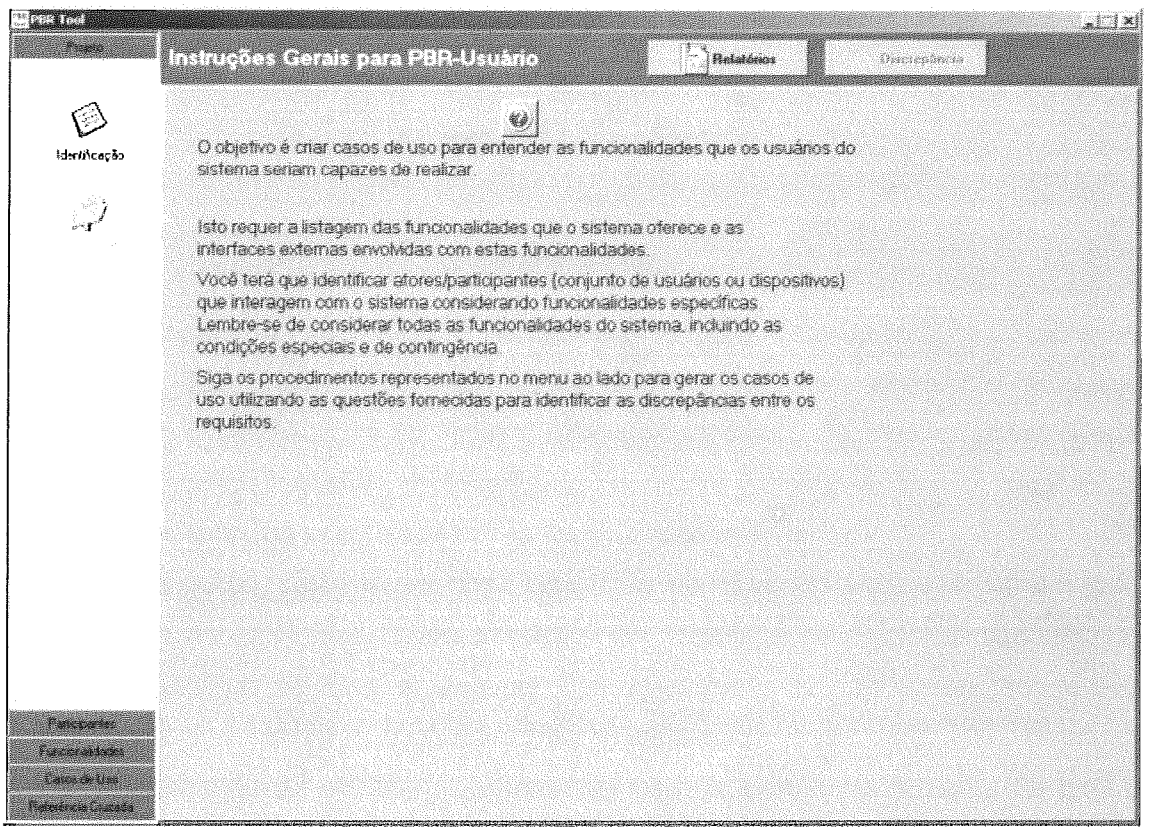


Figura 4.3 - Organização do Menu

Para atender a primeira característica desejada para a ferramenta que é guiar o inspetor, PBR Tool exige que o mesmo execute cada uma das tarefas na ordem originalmente proposta pela técnica. Assim, por exemplo, o inspetor não teria como identificar os participantes de um caso de uso, sem antes acessar as instruções especificadas pela técnica para identificá-los no documento de requisitos.

Graficamente isto é representado através de ícones com imagens nebulosas, ou seja, as tarefas representadas por ícones deste tipo não poderão ser executadas

até que as atividades anteriores tenham sido pelo menos iniciadas. Se o usuário tentar iniciar uma tarefa nesta condição, ele receberá uma mensagem avisando a impossibilidade de fazê-lo.

Quando todas as tarefas anteriores a uma tarefa particular tenham sido ao menos iniciadas, o ícone desta tarefa em particular passa a ter uma imagem nítida, indicando que a tarefa por ele representada pode ser iniciada.

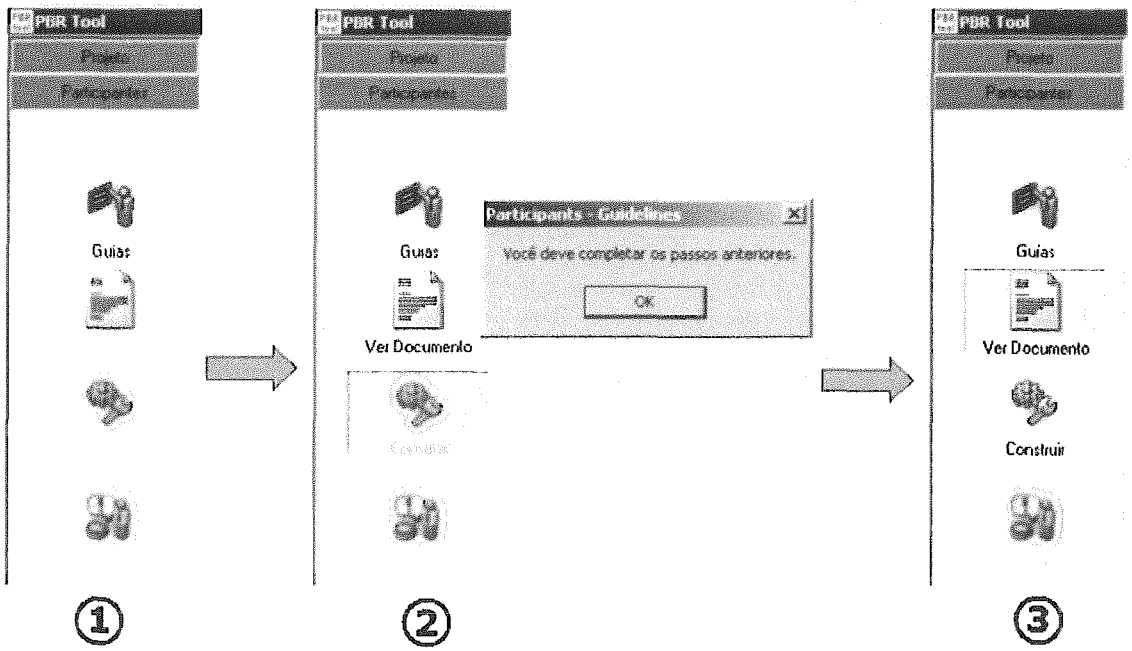


Figura 4.4 - Dinâmica do Menu

A dinâmica do *menu* citada nos parágrafos anteriores é representada na Figura 4.4. O ícone "Guias" representa as instruções especificadas pela técnica para identificar os participantes dos casos de uso no documento de requisitos.

No menu ①, este ícone possui uma imagem não nebulosa, indicando que todas as tarefas anteriores (representadas por ícones agrupadas na aba "Projeto", não visíveis nesta figura) foram pelo menos iniciadas. Os demais ícones possuem uma imagem nebulosa, indicando que as próximas tarefas só poderão ser executadas caso as instruções para identificar os participantes sejam acessadas. Quando isto é feito, o ícone que representa a tarefa seguinte (acesso ao documento de requisitos) deixa de ter uma imagem nebulosa. Isto está representado no menu ②. Este mesmo menu mostra uma situação não permitida. Antes de ler o documento de requisitos, o inspetor tenta identificar os participantes (tarefa representada pelo ícone "Construir"). Como essa ação contraria a ordenação das tarefas proposta originalmente pela técnica, uma mensagem indicando impossibilidade de execução é exibida.

Quando o documento de requisitos for acessado, o ícone "Construir" passa a ter uma imagem nítida, indicando que os participantes podem então ser identificados. Isto está representado no menu ③. Este princípio é adotado como um recurso para reforçar a aplicação correta da técnica.

4.6.2 – Elaboração dos Casos de Uso

Conforme visto nos capítulos anteriores, os produtos gerados quando a identificação de defeitos é realizada com PBR são uma lista de discrepâncias e uma representação de alto nível do sistema. Quando a perspectiva do usuário é utilizada, esta representação é composta pelos casos de uso de alto nível do sistema.

Assim, de uma maneira simplista, poderíamos resumir a técnica como um conjunto de instruções para a criação destes artefatos de alto nível, acompanhadas de uma série de dicas para auxiliar na identificação de defeitos no documento de requisitos utilizado como base para a criação dos artefatos.

A criação dos casos de uso se dará com a realização de diversas tarefas: identificação dos participantes (atores), identificação das funcionalidades, descrição dos casos de uso e associação dos participantes e funcionalidades com os casos de uso descritos. A técnica fornece uma série de dicas que permeiam estas diferentes tarefas para auxiliar na identificação de defeitos.

Em PBR Tool, cada uma destas tarefas está individualmente representada por um ícone na sua aba correspondente, indicando a ordem em que elas devem ser executadas. Ou seja, deve-se executar fielmente as tarefas na ordem originalmente proposta pela técnica, conforme explicado na seção anterior.

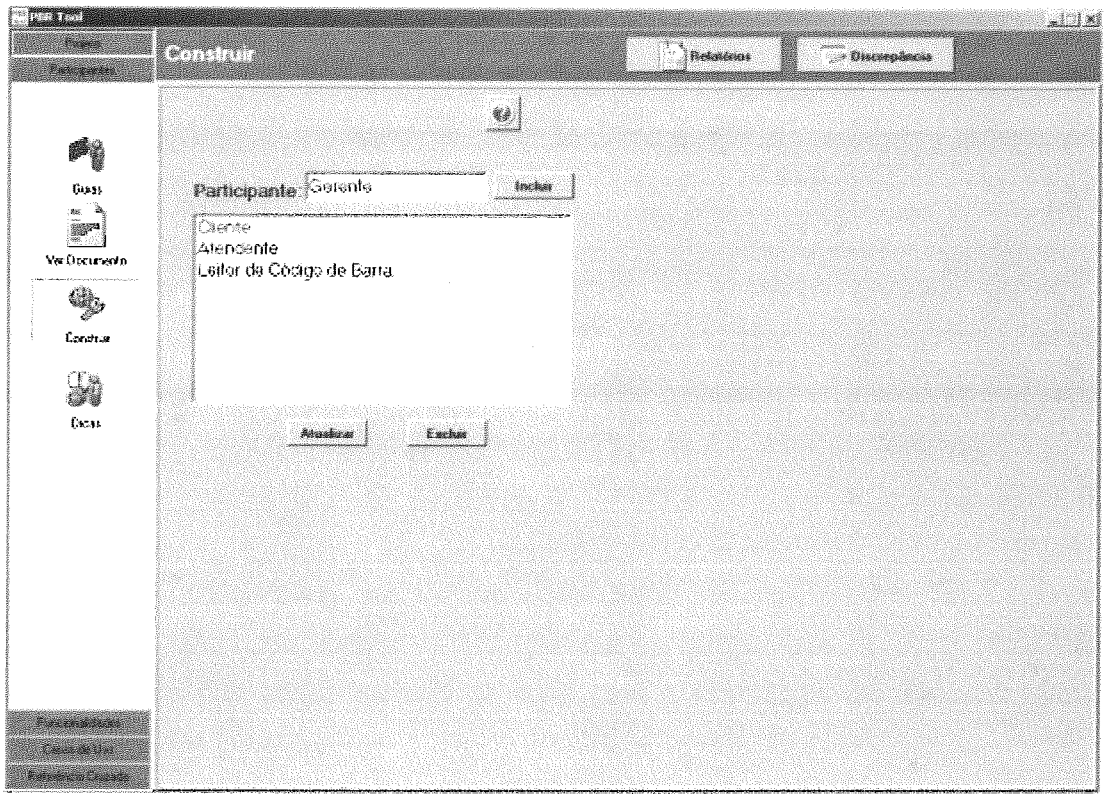


Figura 4.5 - Identificação dos Participantes

Desta forma, após ler tanto o documento de requisitos numa primeira vez e as instruções de como identificar os participantes dos casos de uso, o inspetor pode registrar os participantes identificados, como mostrado na Figura 4.5. Deve-se lembrar que após a execução desta tarefa existem dicas que podem auxiliar na identificação de defeitos no documento de requisitos.

A identificação das funcionalidades segue a mesma idéia, ou seja, o inspetor lê o documento de requisitos uma segunda vez e as instruções de como identificar as funcionalidades contidas no documento de requisitos. Seu registro, representado na Figura 4.6, é feito na ferramenta da mesma forma como é feito o registro dos participantes.

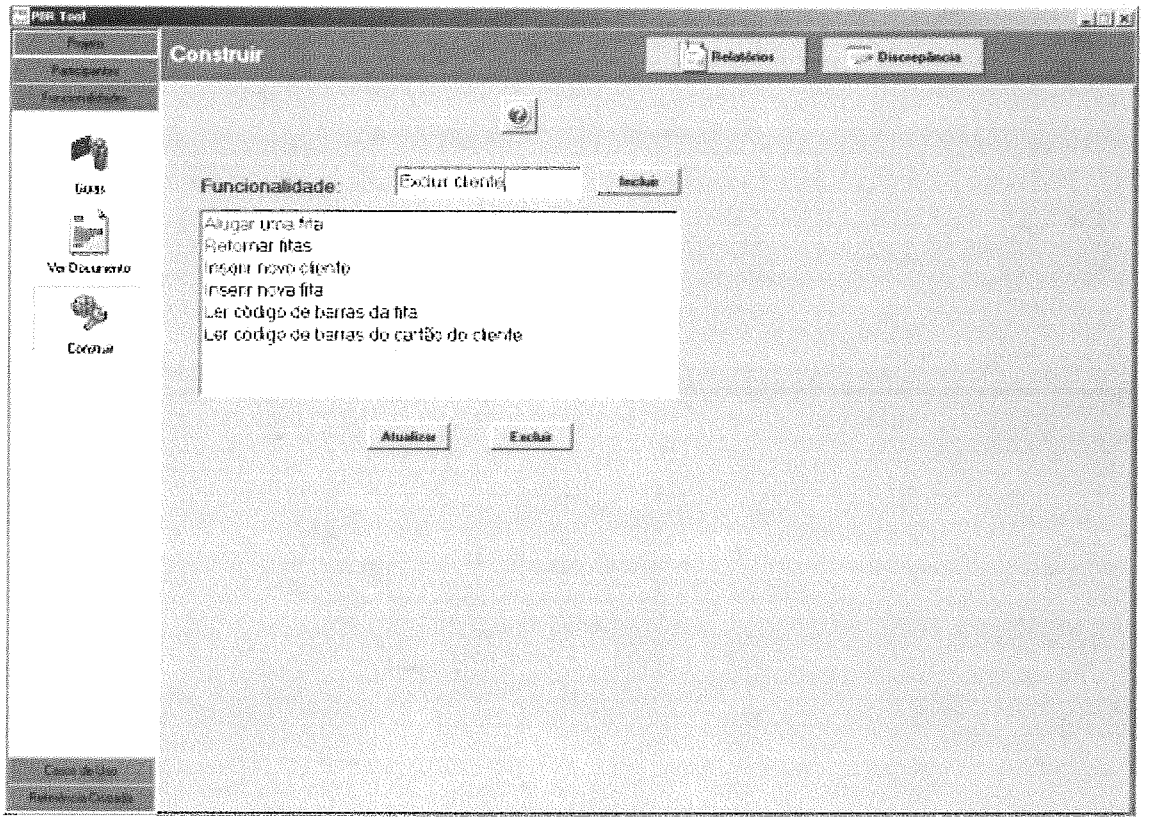


Figura 4.6 - Identificação das Funcionalidades

A tarefa seguinte envolve a identificação e o detalhamento dos casos de uso, além da associação entre estes casos de uso e as funcionalidades previamente identificadas. A forma como esta tarefa é feita na ferramenta é mostrada na figura a seguir.

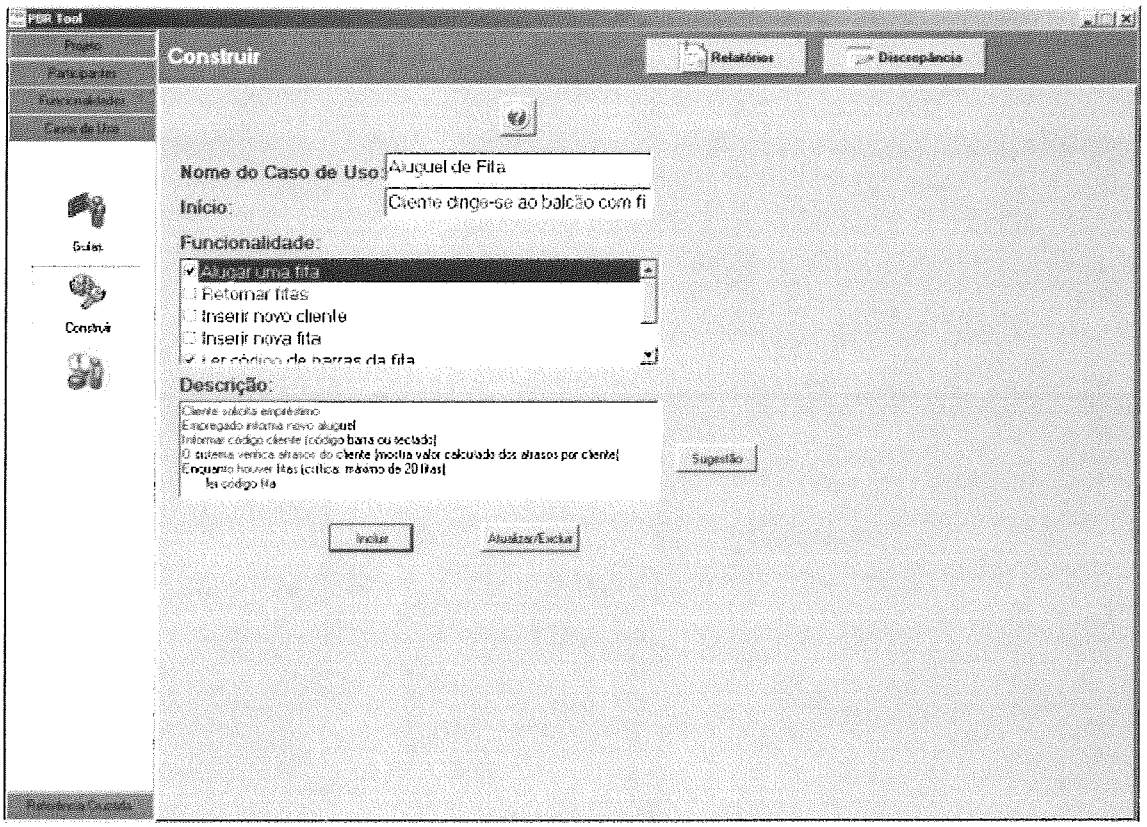


Figura 4.7 - Identificação e Descrição dos Casos de Uso

A forma como os casos de uso são descritos com a ferramenta difere substancialmente da forma como ela é sugerida quando a técnica é aplicada manualmente. Enquanto a técnica sugere a utilização de elementos gráficos para a descrição dos casos de uso (ANDERSSON e BERGSTRAND, 1995), a ferramenta propõe uma descrição textual simples, baseada nas ações dos participantes (atores) e a respectiva resposta do sistema. Esta decisão teve por base a observação experimental do fato de diversos indivíduos despenderem um tempo excessivo nesta descrição, enquanto que a técnica destaca que estes casos de uso devem representar apenas um esboço dos casos de uso reais do sistema descrito no documento.

A Figura 4.8 representa a última tarefa na criação dos casos de uso, que envolve a identificação dos participantes previamente identificados com cada caso de uso descrito.

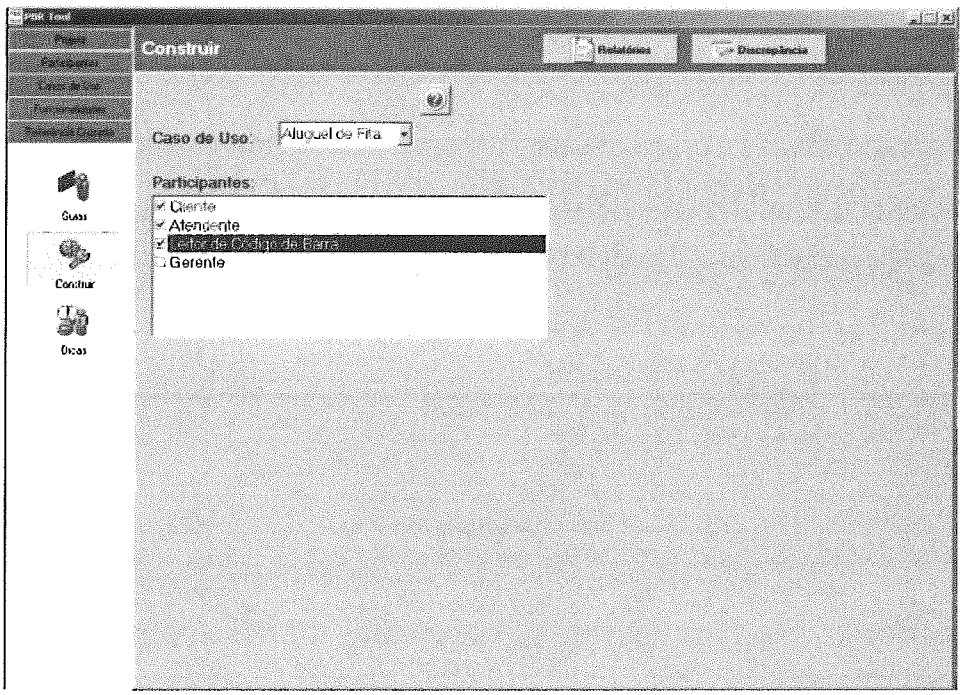
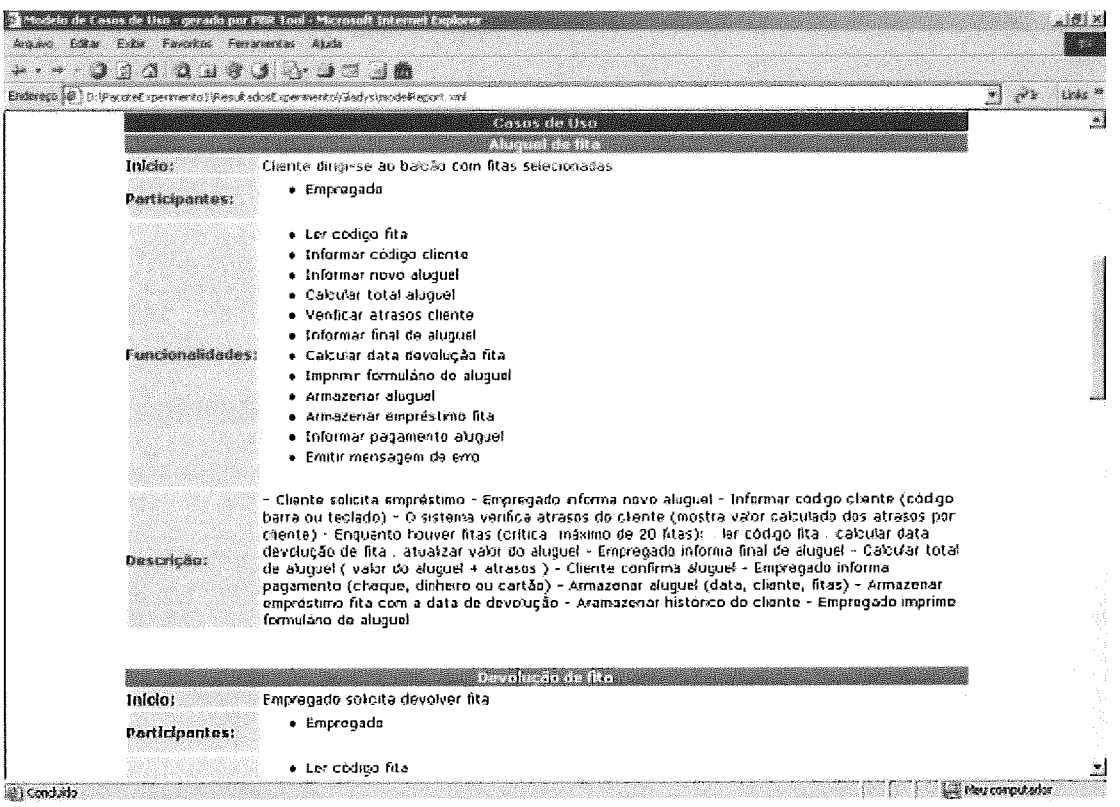


Figura 4.8 - Associação entre Participantes e Casos de Uso

Por fim, conforme observado na Figura 4.9, um relatório com os casos de uso identificados pode ser gerado.



Casos de Uso

Aluguel de fita

Início: Cliente dirige-se ao balcão com fitas selecionadas

Participantes:
- Empregado

Funcionalidades:
- Ler código fita
- Informar código cliente
- Informar novo aluguel
- Calcular total aluguel
- Verificar atrasos cliente
- Informar final de aluguel
- Calcular data devolução fita
- Imprimir formulário do aluguel
- Armazenar aluguel
- Armazenar empréstimo fita
- Informar pagamento aluguel
- Emitir mensagem de erro

Descrição: - Cliente solicita empréstimo - Empregado informa novo aluguel - Informar codigo cliente (código barra ou teclado) - O sistema verifica atrasos do cliente (mostra valor calculado dos atrasos por cliente) - Enquanto houver fitas (crítica : máximo de 20 fitas): : ler código fita : calcular data devolução de fita : atualizar valor do aluguel - Empregado informa final de aluguel - Calcular total de aluguel (valor do aluguel + atrasos) - Cliente confirma aluguel - Empregado informa pagamento (cheque, dinheiro ou cartão) - Armazenar aluguel (data, cliente, fitas) - Armazenar empréstimo fita com a data de devolução - Armazenar historico do cliente - Empregado imprime formulário do aluguel

Devolução de fita

Início: Empregado solicita devolver fita

Participantes:
- Empregado

- Ler código fita

Figura 4.9 - Relatório de Casos de Uso

No contexto da equipe de Engenharia de Software Experimental da COPEE/UFRJ (http://www.cos.ufrj.br/~ese), foi desenvolvida, como um projeto de final de curso de graduação (CHAPETA, 2004), uma ferramenta para a criação detalhada de casos de uso com base nos conceitos de PBR para a perspectiva do usuário (participantes, funcionalidades, etc). Assim, a idéia defendida pela técnica de utilizar os artefatos esboçados durante a inspeção como base para a criação dos artefatos reais do sistema pode ser efetivamente posta em prática com a utilização destas duas ferramentas (SILVA *et al.*, 2004).

Esta idéia é representada na figura abaixo:

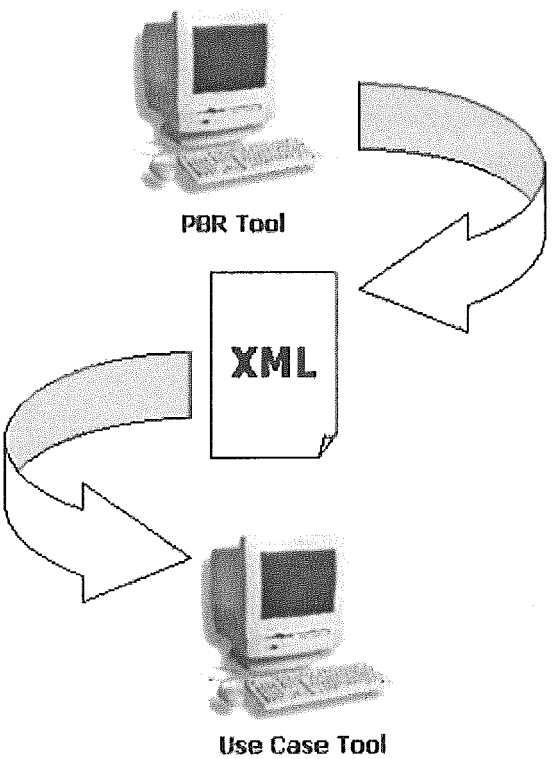


Figura 4.10 - Integração de Ferramentas

4.6.3 – Identificação das Discrepâncias

Uma outra característica da ferramenta é auxiliar o inspetor no relato das discrepâncias identificadas. A experiência na aplicação manual da técnica tem mostrado que, muitas vezes, os inspetores identificam pontos no documento de requisitos onde existem defeitos, mas não são capazes de expressar claramente o defeito existente. Um trecho de um formulário de relato de discrepâncias utilizado quando a técnica é aplicada manualmente é mostrado abaixo:

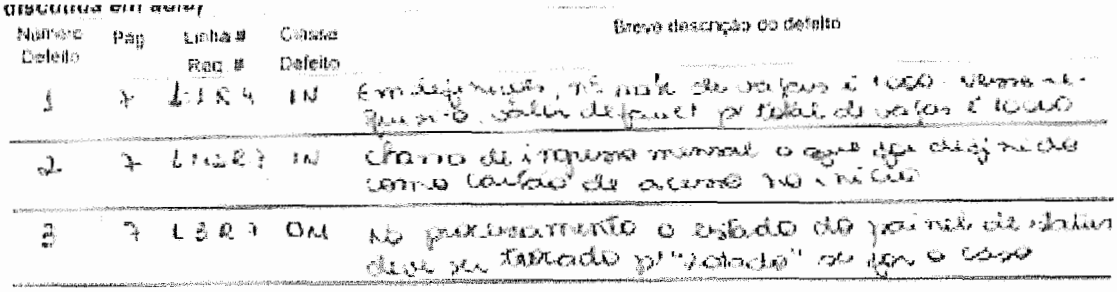


Figura 4.11 - Trecho de um Formulário de Relato de Discrepâncias

Conhecendo a formatação do documento de requisitos a ser inspecionado, a ferramenta permite, não só a visualização individual de cada um dos requisitos, como também a possibilidade de transferir seu conteúdo para a descrição da discrepância. O objetivo desta funcionalidade é possibilitar que justificativas mais claras sejam dadas para uma determinada discrepância identificada.

Assim, antes de indicar qual o arquivo que contém o documento de requisitos a ser inspecionado, o inspetor deve informar qual é o rótulo dado aos requisitos funcionais e não-funcionais, ou seja, como cada requisito é identificado no documento. Isto está representado na Figura 4.12 abaixo:

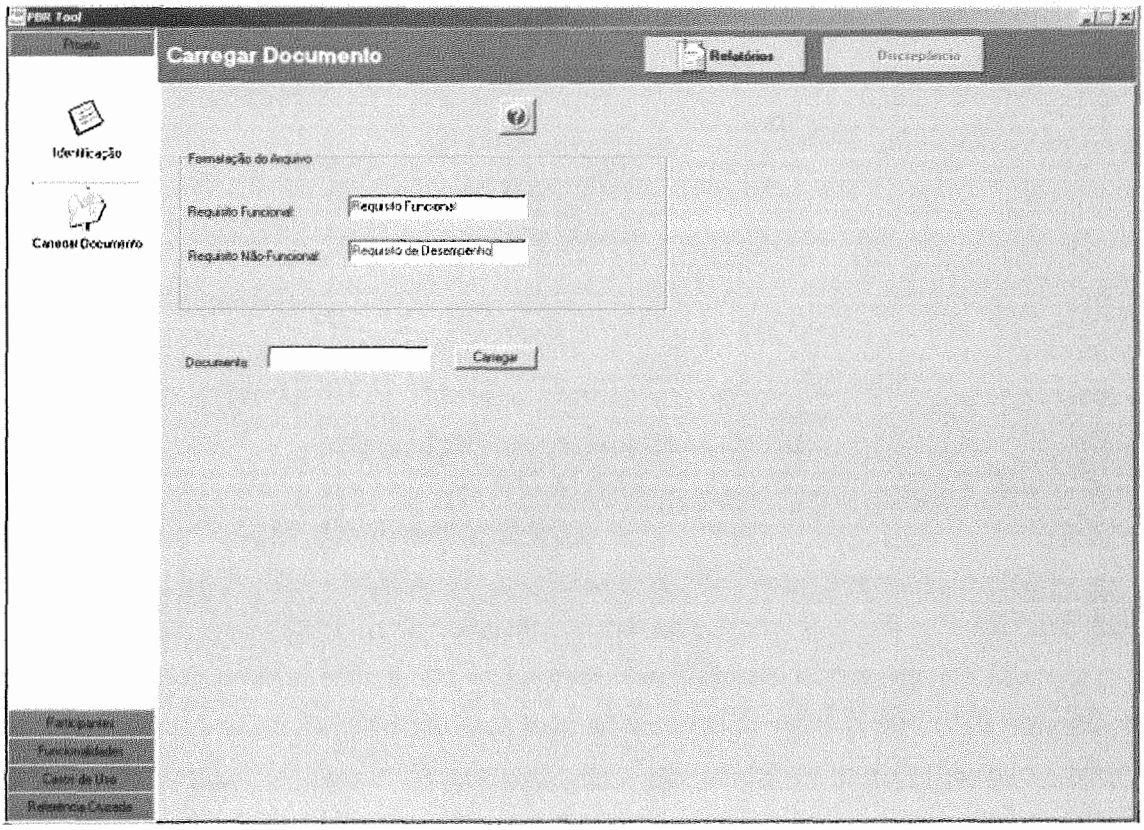


Figura 4.12 - Definição do Formato do Documento de Requisitos

Ao definir a formatação do documento de requisitos, o inspetor poderá, quando relatar uma discrepância, visualizar os requisitos individualmente (Figura 4.13),

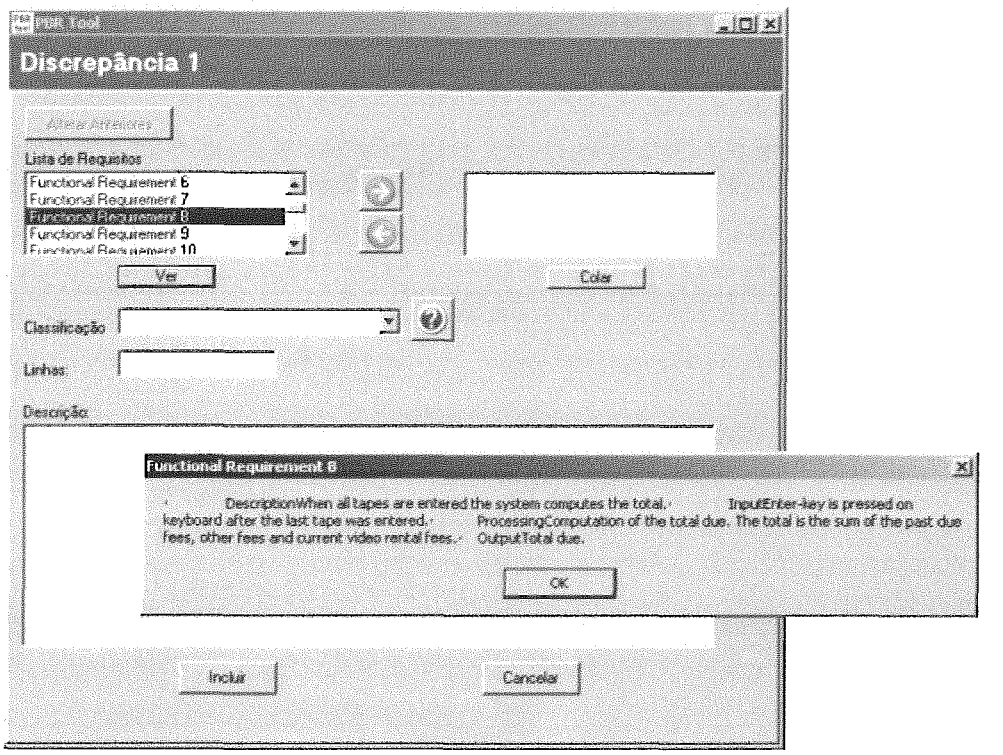


Figura 4.13 - Visualização Individualizada de um Requisito

ou ainda transferir seu conteúdo para a descrição da discrepância identificada. Esta última possibilidade é mostrada na Figura 4.14.

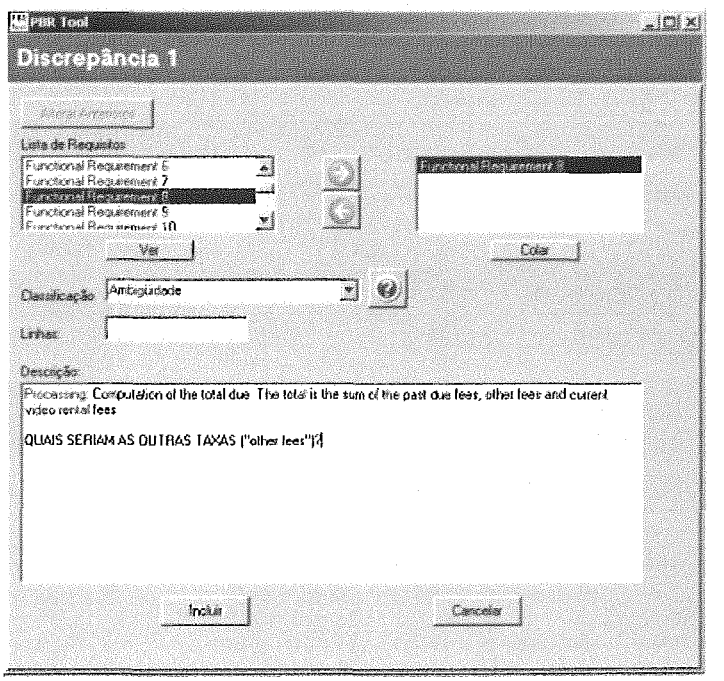
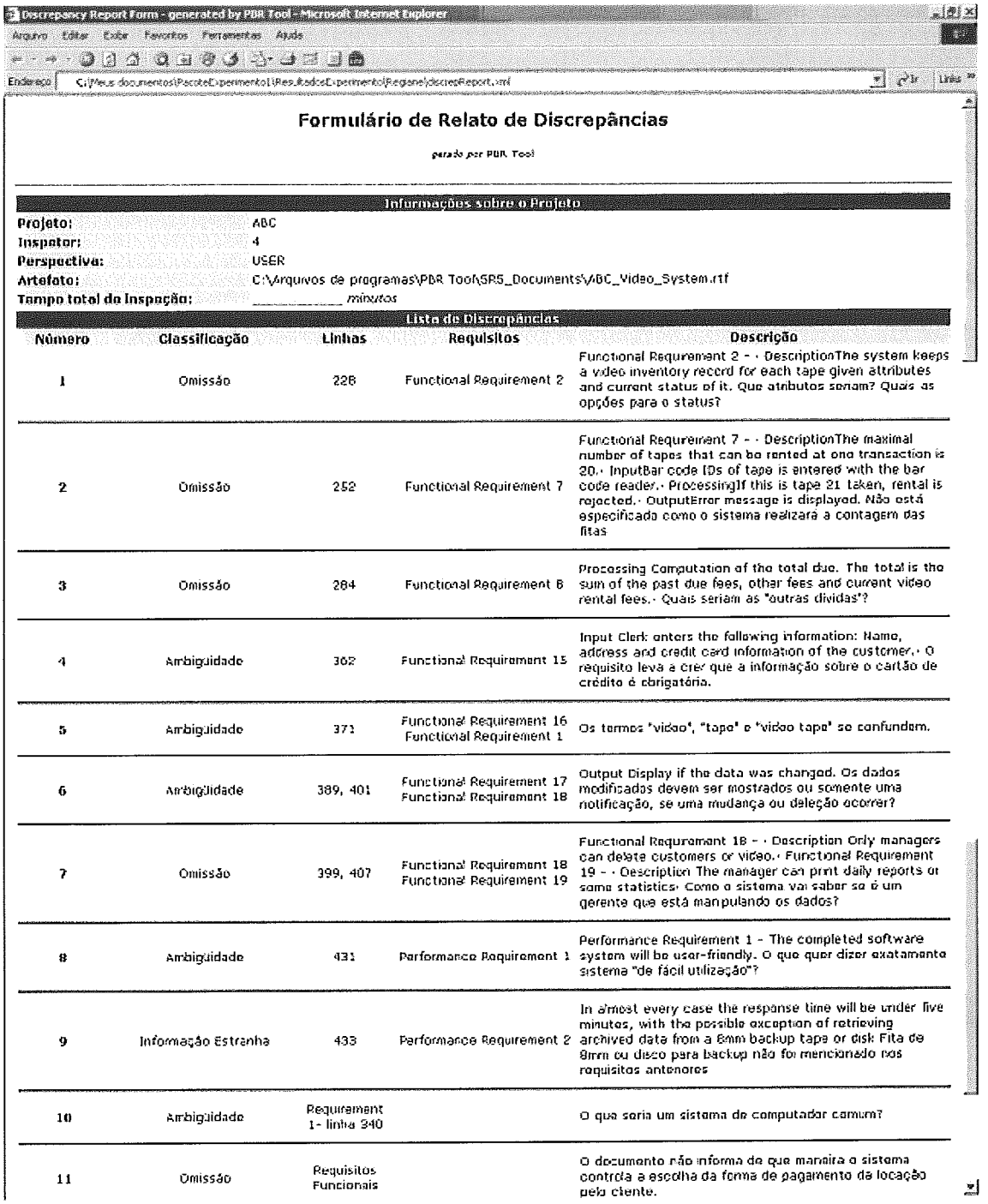


Figura 4.14 - Descrição de uma Discrepância Identificada

O produto mais importante da inspeção é o relatório com as discrepâncias identificadas. Este relatório contem informações que identificam o projeto, o inspetor e o documento inspecionado. Existe também um espaço para o inspetor, após imprimir o relatório, informar o tempo que levou para inspecionar o documento, apesar do fato desta informação já ser obtida de forma não intrusiva pela ferramenta (Figura 4.15).



Discrepancy Report Form - generated by PBR Tool - Microsoft Internet Explorer

Arquivo Editar Exibir Favoritos Ferramentas Ajuda

Endereço C:\Meus documentos\PacoteExperimento1\ResultadosExperimento\Regane\discrepReport.xml

Formulário de Relato de Discrepâncias

gerado por PBR Tool

Informações sobre o Projeto

Projeto:	ABC
Inspetor:	4
Perspectiva:	USER
Artefato:	C:\Arquivos de programas\PBR Tool\SRS_Documents\ABC_Video_System.rtf
Tempo total da Inspeção:	_____ *minutos*

Lista de Discrepâncias

Número	Classificação	Linhas	Requisitos	Descrição
1	Omissão	228	Functional Requirement 2	Functional Requirement 2 - · DescriptionThe system keeps a video inventory record for each tape given attributes and current status of it. Que atributos seriam? Quais as opções para o status?
2	Omissão	252	Functional Requirement 7	Functional Requirement 7 - · DescriptionThe maximal number of tapes that can be rented at one transaction is 20.· InputBar code IDs of tape is entered with the bar code reader.· Processing]If this is tape 21 taken, rental is rejected.· OutputError message is displayed. Não está especificado como o sistema realizará a contagem das fitas
3	Omissão	284	Functional Requirement 8	Processing Computation of the total due. The total is the sum of the past due fees, other fees and current video rental fees.· Quais seriam as "outras dividas"?
4	Ambiguidade	362	Functional Requirement 15	Input Clerk enters the following information: Name, address and credit card information of the customer.· O requisito leva a crer que a informação sobre o cartão de crédito é obrigatória.
5	Ambiguidade	371	Functional Requirement 16 Functional Requirement 1	Os termos "video", "tape" e "video tape" se confundem.
6	Ambiguidade	389, 401	Functional Requirement 17 Functional Requirement 18	Output Display if the data was changed. Os dados modificados devem ser mostrados ou somente uma notificação, se uma mudança ou deleção ocorrer?
7	Omissão	399, 407	Functional Requirement 18 Functional Requirement 19	Functional Requirement 18 - · Description Only managers can delete customers or video.· Functional Requirement 19 - · Description The manager can print daily reports or some statistics· Como o sistema vai saber se é um gerente que está manipulando os dados?
8	Ambiguidade	431	Performance Requirement 1	Performance Requirement 1 - The completed software system will be user-friendly. O que quer dizer exatamente sistema "de fácil utilização"?
9	Informação Estranha	433	Performance Requirement 2	In almost every case the response time will be under five minutes, with the possible exception of retrieving archived data from a 8mm backup tape or disk Fita de 8mm ou disco para backup não foi mencionado nos requisitos anteriores
10	Ambiguidade	Requirement 1- linha 340		O que seria um sistema de computador comum?
11	Omissão	Requisitos Funcionais		O documento não informa de que maneira o sistema controla a escolha da forma de pagamento da locação pelo cliente.

Figura 4.15 - Relatório de Discrepâncias

Não faz parte do escopo deste trabalho prover algum apoio para consolidar as discrepâncias relatadas pelos diversos inspetores, já que esta é uma atividade maçante, quando realizada manualmente. Na verdade, esta é uma das funcionalidades oferecidas pela infra-estrutura de apoio a inspeções ISPIS (KALINOWSKI *et al.*, 2004).

4.6.4 – Ajuda Direcionada

Uma observação recorrente por parte dos inspetores nos diversos estudos envolvendo a aplicação manual de PBR é a necessidade de se consultar diversas vezes o material de referência da técnica. Isto foi explicitado tanto informalmente nas entrevistas de avaliação feitas após os estudos quanto no próprio formulário de acompanhamento fornecido para avaliar a experiência na utilização da técnica, como mostrado na Figura 4.16.

> As classes de defeitos estavam bem definidas, mas eu senti um pouco de dificuldade em diferenciar as classes Fatos Incorretos e Inconsistência. Por várias vezes tive que voltar e ler novamente as suas definição para classificar os defeitos

Figura 4.16 - Formulário de Acompanhamento

Estas observações motivaram a possibilidade de se oferecer uma ajuda mais direcionada com o intuito de acelerar a inspeção. Em todas as telas que representam as tarefas a serem realizadas existe um ícone com um sinal de interrogação indicando a possibilidade de se obter ajuda em como realizar a tarefa.

Para o exemplo da ocorrência de uma dúvida sobre como classificar uma discrepância identificada, o inspetor tem acesso imediato às explicações e exemplos de determinado tipo de defeito, em vez de ter que recorrer ao material de referência da técnica como um todo. Isto é mostrado na Figura 4.17.

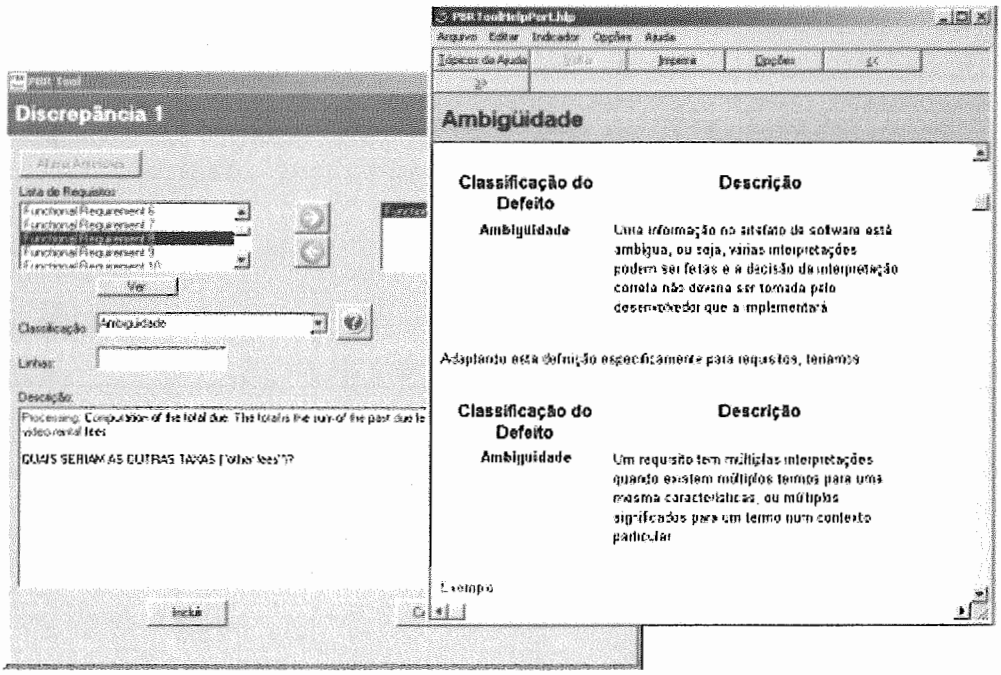


Figura 4.17 - Exemplo de Ajuda Direcionada

4.6.5 – Observação Não Intrusiva

Todas as características da ferramenta descritas nos itens anteriores buscam auxiliar o inspetor na execução das tarefas propostas pela técnica. Entretanto, a utilização de uma ferramenta deste tipo pode trazer benefícios aos próprios pesquisadores interessados no desenvolvimento da técnica.

Observamos, por exemplo, a possibilidade de inspeções que supostamente adotem PBR como técnica de identificação de defeitos possam na verdade ser feitas de forma *ad hoc*. Discordamos de LAITENBERGER *et al.* (2000) quando estes relatam que esta verificação de conformidade com relação à técnica possa ser feita a partir da simples análise dos produtos gerados. Temos observado duas situações recorrentes que refutam a hipótese deste autor:

- Inspetores que se preocupam excessivamente com a elaboração de casos de uso detalhados, esquecendo que o foco da aplicação da técnica é a identificação de defeitos.
- Inspetores que inicialmente buscam os defeitos seguindo heurísticas próprias, ou seja, de forma *ad hoc*, e então produzem os modelos de alto nível, sem a associação desta atividade com a identificação de defeitos.

4.6.5.1 – Conformidade do Processo

A validade interna de um experimento, ou seja, a relação causal entre o tratamento adotado e o resultado obtido, pode ser comprometida caso o processo (ou a técnica) aplicado na prática pelos indivíduos não for o mesmo que o processo em estudo (SORUMGARD, 1997).

Num estudo comparando diferentes técnicas de identificação de defeitos, LANUBILIE e VISAGIO (2000) avaliaram a conformidade do processo através de questionários de avaliação e descobriram que apenas 20% dos indivíduos realmente seguiram uma das técnicas propostas.

Entre os estudos envolvendo PBR descritos no capítulo 3 desta tese, a maior parte dos autores simplesmente assume que a técnica foi fielmente seguida. Dada às observações feitas anteriormente, seria no mínimo temeroso adotarmos a mesma postura. Ainda que a interface gráfica da ferramenta exija que o inspetor execute cada uma das tarefas na ordem originalmente proposta pela técnica, não podemos afirmar que esta característica garanta que o inspetor seguirá fielmente a técnica.

A princípio, a única forma de termos esta informação é realizar estudos observacionais, ou seja, efetivamente acompanhar os inspetores ao executarem a técnica. Entretanto, quando as pessoas estão cientes que são explicitamente avaliadas, seu comportamento tende a ser alterado. Mesmo que isto não seja feito de forma consciente, elas tendem a agradar os pesquisadores (DAVIDOFF, 1987). Este é o chamado efeito *Hawthorne*.

4.6.5.2 – Efeito Hawthorne

Numa clássica série de experimentos, ROETHLISBERGER e DICKSON (1939) investigaram possíveis maneiras de aumentar a produtividade dos trabalhadores da fábrica da Western Electric Company, em Chicago. Os pesquisadores manipularam tanto as influências físicas e ambientais do local de trabalho, como umidade e intensidade luminosa, como também aspectos psicológicos, como intervalos e duração da jornada de trabalho.

Em geral, a produtividade dos trabalhadores aumentou mesmo quando as condições de trabalho pioraram (DAVIDOFF, 1987). A conclusão deste estudo foi que a produtividade aumentou pelo fato de os trabalhadores estarem satisfeitos pelo simples fato de terem recebido a atenção dos pesquisadores. Desde então, o termo "efeito Hawthorne" tem sido usado para indicar a influência dos pesquisadores no desempenho dos indivíduos em observação.

4.6.5.3 – Abordagens para Verificar Conformidade do Processo

Os pesquisadores enfrentam então o seguinte dilema:

- Verificar a conformidade do processo, com a possibilidade do efeito Hawthorne distorcer os resultados obtidos, ou;
- Simplesmente não fazer esta verificação e como resultado ignorar se o processo em estudo foi realmente aplicado na prática.

Uma forma de verificar a conformidade do processo e evitar o efeito Hawthorne é conduzir observações não intrusivas, ou seja, minimizar ao máximo a influência do pesquisador durante a execução do experimento, no nosso caso, a inspeção de requisitos de software.

Conforme já visto, LANUBILIE e VISAGIO (2000) avaliaram a conformidade do processo através de formulários de acompanhamento, preenchidos após a realização do experimento. A ressalva a ser feita aqui é que os indivíduos podem ter tido um entendimento de tal forma equivocado do processo que as respostas obtidas através destes formulários podem não refletir a realidade. Existe por isso a necessidade de utilizar técnicas de elaboração de *surveys* para validar as respostas obtidas.

Num estudo para avaliar técnicas de inspeção de usabilidade, ZHANG *et al.* (1999) prepararam um ambiente denominado "laboratório cognitivo". As sessões de inspeção puderam ser observadas de forma não intrusiva em salas adjacentes ao laboratório, através de falsos espelhos. As sessões também foram gravadas em vídeo, com duas visões: a tela do computador, e a expressão facial junto com a movimentação da parte superior do corpo do inspetor. Apesar de ser uma abordagem interessante, os custos envolvidos e a artificialidade do local podem ser um obstáculo para a sua adoção.

Uma outra abordagem possível é o monitoramento remoto, possível de ser adotado quando o estudo envolver a utilização de ferramentas computacionais. NetSupport (http://www.netsupport-inc.com) e LanSchool (http://www.lanschool.com) são exemplos de softwares que permitiriam aos pesquisadores observar os indivíduos remotamente, em tempo real e com a possibilidade de gravação, como representado na Figura 4.18 abaixo.

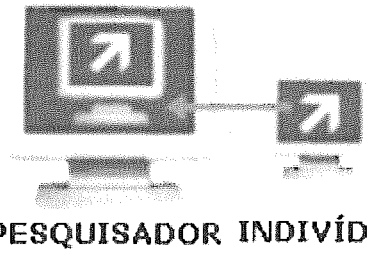


Figura 4.18 - Monitoramento Remoto (Adaptado de http://www.netsupport-inc.com)

A grande vantagem desta abordagem é a sua cobertura completa, permitindo a realização de diversas análises. Paradoxalmente, esta característica também representa uma desvantagem na medida em que a falta de foco pode demandar um tempo de análise considerável.

Uma outra abordagem envolve a coleta de métricas enquanto os indivíduos executam as tarefas. A ferramenta Hackystat (JOHNSON, 2001), por exemplo, coleta de forma implícita dados relativos ao processo através de componentes acoplados aos ambientes de desenvolvimento, de uma forma tal que os desenvolvedores não precisam interromper suas atividade para tal fim. Vale destacar que nesta ferramenta em particular, os usuários têm conhecimento que a coleta de métricas está sendo feita. Esta abordagem provê um foco maior que o monitoramento remoto, mas o grande fator de risco aqui é a escolha correta das métricas.

PBR Tool utiliza uma abordagem semelhante à proposta pela Hackystat, com a grande diferença da coleta de métricas não ser explícita para seu usuário. Este abordagem está descrita na seção seguinte.

4.6.5.4 – A Abordagem de PBR Tool para Verificar Conformidade do Processo

PBR Tool busca verificar a conformidade do processo através da coleta de métricas. Para evitar o efeito Hawthorne, esta coleta é feita de forma implícita, sem o conhecimento do inspetor. Esta é uma postura controversa, na medida em que envolve questões éticas.

A idéia dos estudos envolvendo a ferramenta é preservar o anonimato dos inspetores, que se identificam através de um número aleatoriamente escolhido. Os estudos realizados na COPPE/UFRJ envolvem estudantes cujas avaliações não estão atreladas ao seu desempenho, mas sim a simples participação nos estudos. Assim sendo, no nosso contexto de pesquisa, a coleta de métricas sem o conhecimento dos inspetores não fere qualquer de seus direitos.

A escolha destas métricas está fundamentada na observação experimental do fato de que alguns inspetores tenderem a negligenciar a técnica. Percebeu-se que inspetores mais experientes não enxergariam um benefício com a adoção de PBR. Nesses casos, os inspetores inicialmente buscam identificar os defeitos seguindo suas próprias heurísticas e então produzem os modelos de alto nível, sem relacionar esta atividade à identificação de defeitos.

Entre os menos experientes, observam-se situações em que inspetores preocupam-se excessivamente com a elaboração de casos de uso detalhados, esquecendo-se de que o foco deveria ser a identificação de defeitos.

Ambos os casos mascaram os resultados obtidos, já que a técnica não é aplicada de forma correta. Assim, para identificar esses casos sem a necessidade da observação intrusiva, PBR Tool coleta o tempo e ordem de execução de cada uma das tarefas propostas pela técnica, além do momento da identificação das discrepâncias. Com essas métricas seria possível identificar, por exemplo, um inspetor experiente que, após a leitura do documento de especificação de requisitos, identifica todas as discrepâncias e então executa os passos restantes da técnica sem identificar qualquer outra discrepância. Ou seja, estaríamos identificando um inspetor que realizou a inspeção de forma *ad hoc*. É importante destacar que o registro dessas métricas é feito em um arquivo XML.

(a) Tempo e ordem de execução de cada uma das tarefas:

Conforme citado antes, PBR Tool exige que o inspetor execute as tarefas propostas na ordem original da técnica. Entretanto, isto não é suficiente para afirmar que os inspetores a seguirão fielmente.

Uma forma de tentar fazer esta verificação é inferir, a partir do tempo dedicado pelo inspetor em cada uma das tarefas, se o inspetor ignora completamente alguma destas tarefas.

Ao fazer a análise destes dados é fundamental levar em conta a experiência do inspetor com PBR, já que é natural aos inspetores mais experientes omitirem a leitura das orientações e das dicas fornecidas pela técnica. No nosso contexto de pesquisa, todos os estudantes são novatos na técnica, por isso esta análise é feita de forma direta. Um trecho deste registro é mostrado no exemplo abaixo:

```
- <eventList>
    <event order="1" length="00:01:15">Project Identification</event>
    <event order="2" length="00:00:15">Participants Guidelines</event>
    ...
  </eventList>
```

Figura 4.19 - Monitoramento das Atividades

(b) Tarefa Executada Quando uma Discrepância é Identificada:

Esta métrica nos permitiria identificar os casos em que os inspetores buscam identificar os defeitos seguindo suas próprias heurísticas, ou seja, os casos em que os defeitos identificados não tiveram qualquer relação com a aplicação da técnica.

Nestes casos, a maioria dos defeitos é encontrada após a primeira leitura do documento e a execução das tarefas subseqüentes tem uma contribuição pífia para a identificação dos defeitos.

É importante destacar que não há qualquer viés de "favorecimento" da técnica. O objetivo aqui é saber se o desempenho do inspetor (bom ou ruim) pode ser associado à execução da técnica.

Um exemplo de registro de discrepância é mostrado na Figura 4.20.

```
- <discrepancyList>
  - <discrepancy status="new">
      <classification>Informação Estranha</classification>
      <description>O cliente não tem interação com o sistema. Não precisa trazer o cartão pois o código pode ser entrado
        pelo teclado. O cliente precisa saber seu código? O sistema pode auxiliar.</description>
      <identTime>7/9/2003 10:40:28</identTime>
      <step>Use Cases - Hints</step>
      <line>414</line>
    </discrepancy>
```

Figura 4.20 - Registro de uma Discrepância

(c) Tempo total de inspeção:

Ainda que esta métrica não seja utilizada para verificar a conformidade da aplicação da técnica, ela é de extrema importância para o processo de inspeção.

Diversos critérios podem ser utilizados para determinar os desenvolvedores que farão parte de uma determinada equipe de inspetores, desde a cobertura média dos defeitos, ou seja, a quantidade média de defeitos reais que um inspetor encontra, até os tipos de defeitos mais encontrados pelos inspetores.

Um critério fundamental para esta escolha é o custo-eficiência, ou seja, a quantidade de defeitos encontrados por unidade de tempo. Neste contexto, é fundamental que a contagem deste tempo seja feita de forma uniforme e fidedigna para todos os inspetores.

PBR Tool registra, de forma implícita, o tempo de início e fim da inspeção (com eventuais intervalos) para obter o tempo total. Um exemplo deste registro é mostrado na Figura 4.21 a seguir:

```
<?xml version="1.0" ?>
- <form type="pbr">
    <perspective>USER</perspective>
    <project>ESE</project>
    <inspector>103137506</inspector>
    <start>7/9/2003 09:40:56</start>
    <artifact>C:\Program Files\PBR Tool\SRS_Documents\ABC_Video_System.rtf</artifact>
  + <discrepancyList>
  + <eventList>
    <PBR_help>0</PBR_help>
    <Tool_help>0</Tool_help>
    <end>7/9/2003 12:13:47</end>
    <totalTime>02:32:50</totalTime>
</form>
```

Figura 4.21 - Tempo Total de Inspeção

4.7 – Conclusão

Neste capítulo foi descrita a proposta de apoio ferramental para PBR, considerando desde os fatores que motivaram a sua concepção inicial, passando pela definição do escopo e arquitetura até a apresentação das funcionalidades.

Os resultados experimentais do estudo de viabilidade, a serem relatados no próximo capítulo, indicam ser interessante estender a ferramenta para as perspectivas do testador e do projetista. Para esta última perspectiva, haverá a necessidade de estender a própria técnica para considerar os artefatos da orientação a objetos.

Capítulo 5 - Estudos Experimentais sobre PBR Tool

Neste são descritos os estudos experimentais executados com o intuito de verificar a viabilidade da abordagem proposta neste trabalho. Também são descritos os objetivos e o planejamento de um estudo mais aprofundado a ser realizado como parte dos trabalhos futuros a este.

5.1 – Introdução

Temos visto uma grande discussão sobre os chamados transgênicos, alimentos modificados geneticamente. Entre os benefícios para a sociedade da utilização desta tecnologia temos o aumento da qualidade nutricional e da vida útil dos alimentos produzidos. Além disso, a maior resistência destes alimentos a pragas promete um aumento da produção, essencial para garantir as crescentes demandas mundiais. Entretanto, ainda não se tem completo domínio sobre as conseqüências da utilização destes alimentos. A falta de evidências científicas dá margem a que seu uso seja contestado por razões políticas, econômicas, éticas e até religiosas. Parece claro, portanto, a necessidade de novas tecnologias estarem sempre respaldadas no âmbito científico.

No contexto da Engenharia de Software, esta também é uma preocupação para um número crescente de pesquisadores. Estudos experimentais, que são basicamente uma observação sistemática do efeito de uma tecnologia em aplicações práticas, têm sido utilizados no direcionamento de pesquisas futuras ao revelar os problemas e as dificuldades que as pessoas encontram na prática das tecnologias avaliadas (CIOLKOWSKI *et al.*,2002a).

Especificamente nas tecnologias relacionadas à Verificação e Validação, a Engenharia de Software Experimental tem ajudado a desmentir a crença de que as atividades de teste constituem a única técnica na busca por defeitos em artefatos de software (CIOLKOWSKI *et al.*,2002b).

Nas seções seguintes serão apresentados os dados dos estudos experimentais conduzidos para avaliar a viabilidade de utilização da ferramenta de apoio para a aplicação da técnica de Leitura Baseada em Perspectiva proposta neste trabalho.

A estratégia utilizada neste trabalho foi de realizar estudos inicialmente num ambiente acadêmico para então realizá-los na indústria. Os experimentos de

laboratório são um passo necessário, já que reduzem o risco de transferência de uma tecnologia imatura. É importante lembrar que cada nova rodada de um experimento reduz a probabilidade de que os resultados possam dever-se a variações humanas ou erros experimentais (PORTER *et al*, 1995).

A seção seguinte apresenta a metodologia adotada para avaliar experimentalmente a ferramenta descrita neste trabalho. Os tipos de estudo, bem como os objetivos associados a sua execução, são detalhados. A seção 5.3 define o cronograma estabelecido e as seções posteriores relatam os estudos conduzidos.

5.2 – Metodologia Adotada

A utilidade prática de uma tecnologia pode ser influenciada por diversos fatores, como, por exemplo, restrições de esforço e orçamento. Em razão disso, a partir de lições aprendidas em múltiplos projetos de pesquisa, (SHULL *et al.*,2001) propuseram uma metodologia experimental iterativa para introduzir processos de software. Esta metodologia visa a isolar os fatores que possam afetar a tecnologia estudada, reduzindo a possibilidade de se transferir para a indústria uma tecnologia ainda imatura.

Os autores destacam que um novo processo (ou tecnologia) inserido num ambiente industrial pode não gerar as melhorias esperadas se o próprio processo contiver falhas, se este não for adequado ao ambiente industrial ou até se sua aplicação não for feita de forma correta. Uma metodologia iterativa permitiria um melhor entendimento, já que daria uma indicação mais precisa de possíveis pontos de falha.

Esta metodologia, representada na Figura 5.1, é composta por quatro etapas, que contemplam desde a fase conceitual, de definição da tecnologia, até sua transferência para a indústria. Os estudos sugeridos para cada uma das fases estão diretamente relacionados às variáveis que se desejam controlar e os riscos possíveis de serem assumidos na fase em questão.

A característica desta metodologia de controlar os riscos envolvidos com a execução dos estudos foi fundamental para a decisão de sua adoção neste trabalho. Além disso, esta metodologia foi utilizada com sucesso para avaliar as técnicas de leitura orientadas a objetos OORTs (TRAVASSOS *et al.*, 1999). Além disso, esta metodologia apresenta-se como uma alternativa para mostrar aos profissionais da área de software os possíveis benefícios das tecnologias desenvolvidas na academia.

Afinal, conforme observado por TICHY *et al.* (1995), 50% dos artigos mais relevantes em nossa área de pesquisa falharam em validar as propostas apresentadas.

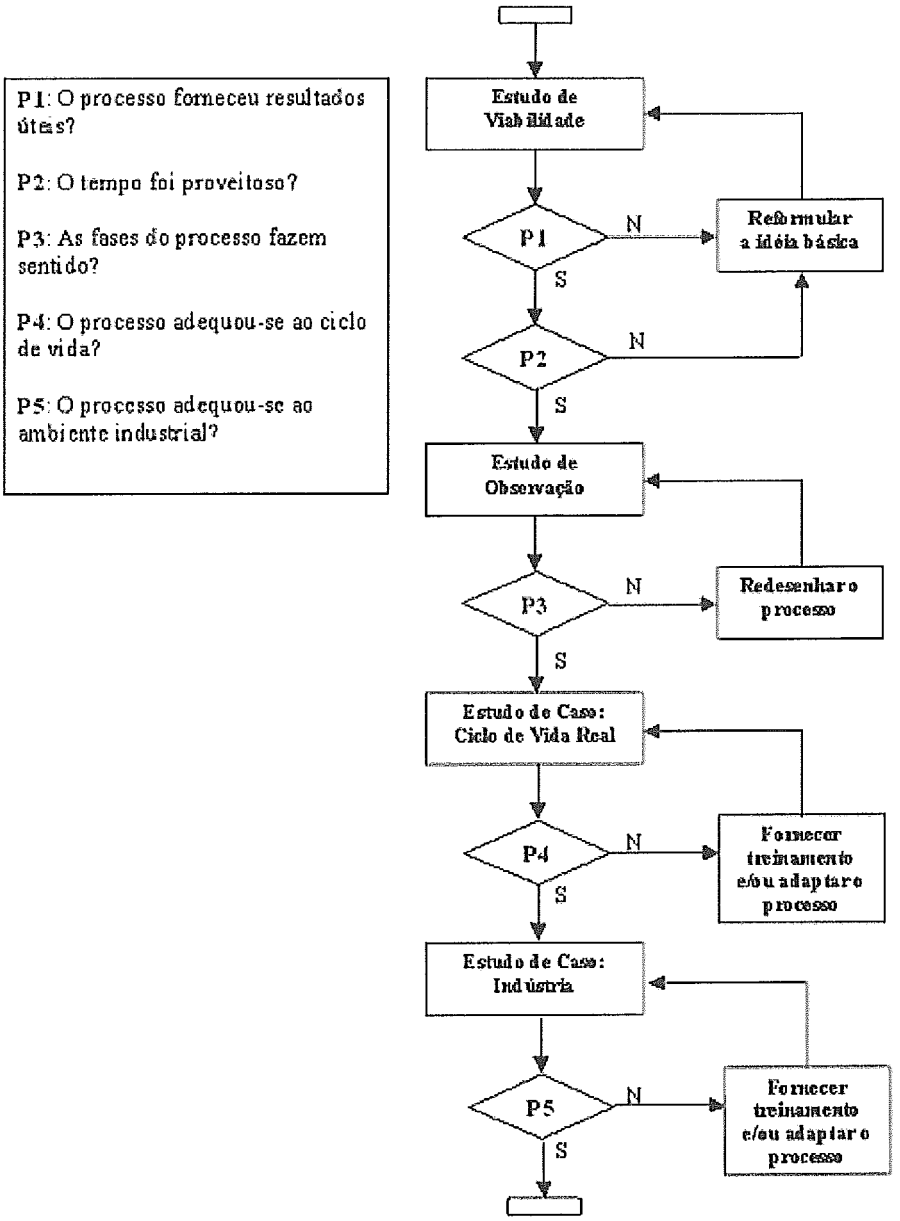


Figura 5.1 - Visão Geral da Metodologia Experimental

5.2.1 – Estudo de Viabilidade

Num estudo de viabilidade, ou pré-experimento, ainda que haja um planejamento e a coleta de dados siga um projeto experimental pré-definido, o controle total sobre todas as variáveis envolvidas é difícil de ser feito. Ao invés de se explorar os detalhes, o objetivo aqui é avaliar as questões fundamentais que motivaram a

criação desta tecnologia. Neste ponto da metodologia a preocupação é a geração das hipóteses.

Neste tipo de estudo busca-se avaliar se é exeqüível despender recursos para desenvolver a tecnologia estudada. (SHULL *et al.*, 2001) destacam que revisões nestas questões provocariam as maiores alterações na tecnologia, por isso elas são tratadas no início do processo de avaliação. Um exemplo deste tipo de estudo pode ser encontrado em (KITCHENHAM *et al.*, 2002).

5.2.2 – Estudo de Observação

Um estudo de observação se dá num ambiente no qual os indivíduos submetidos ao estudo desempenham alguma tarefa enquanto são observados pelos pesquisadores. O propósito deste estudo é coletar dados sobre como uma tarefa é executada. Desta forma, os pesquisadores podem adquirir uma compreensão mais refinada sobre como um novo processo é aplicado na prática, presenciando eventuais dificuldades que os indivíduos podem enfrentar.

A coleta de dados num estudo de observação pode se dar de duas formas: observacional e investigativa. Dados observacionais são coletados enquanto o processo está sendo executado, sem a interferência do pesquisador. Dados investigativos, por sua vez, são coletados ao término de determinado passo do processo, quando os indivíduos respondem a perguntas pré-definidas, abordando questões sobre as quais os indivíduos, normalmente, não pensariam a respeito. Um estudo de observação é relatado, por exemplo, em (TRAVASSOS *et al.*, 2000).

5.2.3 – Estudo de Caso: Ciclo de Vida Real

Atingir este ponto da metodologia pressupõe que haja indícios de que o processo sendo avaliado é realmente efetivo. Entretanto, pelo fato do processo ter sido usado apenas de forma isolada, não poderíamos dizer como seria seu comportamento num ciclo de vida real. Para ter acesso a essa informação, um estudo de caso deve ser realizado com o intuito de examinar este processo no contexto de um ciclo de vida de software.

Estudos de caso são custosos, já que os indivíduos devem ser treinados para superar a curva de aprendizagem. Por esta razão é que os estudos de caso foram introduzidos somente neste ponto. As etapas anteriores da metodologia buscaram identificar os possíveis problemas do processo em si, quando o processo em estudo foi analisado isoladamente. Da interação deste processo com outros aspectos de um ciclo de vida real podem decorrer outros problemas que, em função de toda a análise

feita anteriormente, seriam mais facilmente diferenciados e entendidos. É importante destacar que um determinado processo pode ser viável apenas em alguns ambientes particulares.

Em (HÖHN *et al.*, 2004) pode ser encontrado um relato recente da execução deste tipo de estudo.

5.2.4 – Estudo de Caso: Indústria

Nesta etapa é essencial que desenvolvedores profissionais sejam submetidos ao estudo, que se realizará num ambiente industrial. Todas as etapas anteriores da metodologia buscaram minimizar os riscos associados a este estudo já que os custos envolvidos são os mais altos, considerando desde os custos dos próprios desenvolvedores profissionais até os impactos negativos de eventuais falhas inesperadas no processo. Toda esta cautela tem por objetivo convencer parceiros industriais a participarem deste tipo de estudo. (CONRADI *et al.*, 2003) realizaram este tipo de estudo para avaliar, num ambiente industrial, as chamadas técnicas de leitura orientada a objetos, OORTs.

5.3 – Organização dos Estudos Experimentais

Seguindo a metodologia proposta por (SHULL *et al.*,2001) o primeiro passo executado foi a realização de um estudo de viabilidade, no primeiro semestre de 2003. A seguir, um estudo de observação foi conduzido no segundo semestre daquele ano. O terceiro estudo está previsto para o segundo semestre de 2004, sendo que seu planejamento já foi realizado. A realização do último estudo dependerá do parceiro na indústria, ainda não definido. Uma representação da aplicação desta metodologia para a avaliação da ferramenta de apoio a PBR é feita na Figura 5.2 a seguir:

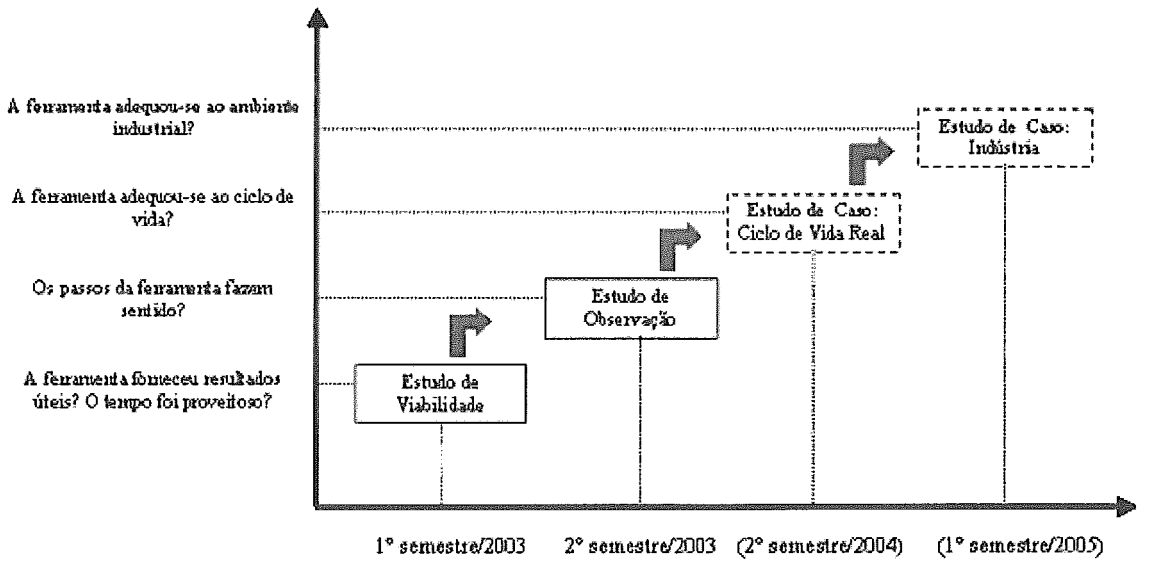


Figura 5.2 - Aplicação da Metodologia Proposta por (SHULL et al., 2001)

5.4 – Estudo de Viabilidade

Este primeiro estudo, conduzido durante o primeiro semestre de 2003, contou com 11 participantes, alunos de pós-graduação (3 alunos de doutorado e 8 de mestrado) da COPPE/UFRJ matriculados na disciplina Engenharia de Software Orientada a Objetos.

A representatividade da utilização de estudantes em estudos de Engenharia de Software ainda é debatida na comunidade, não havendo um consenso entre os pesquisadores. Estudos recentes indicam que as diferenças entre estudantes de pós-graduação e desenvolvedores profissionais são mínimas e que, sob certas condições, estudantes podem ser considerados uma amostra representativa dos desenvolvedores profissionais (HOST *et al.*, 2000) (RUNESON, 2000).

De qualquer forma, conforme destacado por (CARVER *et al.*, 2003), é importante que os estudantes tenham um benefício pedagógico com o estudo do qual participam e, para isso, o estudo tem que estar bem integrado aos objetivos do curso. Esta constatação representou um primeiro desafio: alinhar a data de realização do estudo com a programação do curso.

Na maior parte das vezes, a ementa de um curso é organizada de forma que os tópicos a serem abordados possuam um encadeamento lógico, uma seqüência de informações que favoreçam ao entendimento pelo aluno. Desta forma, a realização de

um estudo que aborde um dos tópicos de um curso deve respeitar esta cronologia. A conseqüência disso é uma limitação no prazo para a realização do estudo.

No presente caso, num curso de Engenharia de Software que aborda todas as fases de um processo de desenvolvimento, seria mais didático para os alunos tratar de inspeções de requisitos de software tão logo a atividade de análise de requisitos seja apresentada. O tempo de aula dedicado a este tópico determinou a data de realização do nosso estudo de viabilidade.

A superação deste desafio envolve um planejamento bem feito, condição fundamental para o sucesso de um estudo. Ainda assim, existe a dificuldade de se prever o nível de maturidade da pesquisa quando o estudo for realizado.

Um outro desafio, comum à nossa área de pesquisa, é a escassez de participantes disponíveis para os estudos conduzidos. Como conseqüência, existe a possibilidade de diferentes estudos serem realizados com os mesmos participantes, num curto intervalo de tempo. Este fato ocorreu nesta pesquisa, trazendo restrições na escolha dos instrumentos e do projeto experimental do estudo.

No nosso caso, um estudo anterior, também envolvendo a aplicação de PBR (neste caso, manual), foi conduzido com os mesmos estudantes para avaliar a influência de aspectos culturais e da língua nativa no desempenho dos inspetores. Os alunos, para inspecionar um documento de requisitos escritos na língua portuguesa, foram divididos em dois grupos: o primeiro grupo (PP) utilizou as técnicas de leitura PBR também escritas na língua portuguesa enquanto que o segundo (EP) utilizou as técnicas escritas em inglês. Em razão disso, para permitir uma eventual comparação de desempenho, os grupos definidos no estudo de viabilidade foram mantidos, assim como, obviamente, a caracterização dos inspetores.

Os estudantes concordaram em participar dos dois estudos ao assinarem um termo de consentimento, sendo caracterizados pelo nível de experiência nas atividades relacionadas a estes estudos, ou seja, desenvolver software, escrever, revisar e modificar requisitos de software, além de escrever e revisar casos de uso.

O resultado dessa caracterização é mostrado na Tabela 5.1:

Tabela 5.1 - Caracterização dos Inspetores no Estudo de Viabilidade

Id	Experiência					
	Desenvolvendo software	Escrevendo requisitos	Escrevendo casos de uso	Revisando requisitos	Revisando casos de uso	Modificando requisitos para manutenção
PP01	4	4	4	3	3	2
PP02	4	4	4	1	1	4
PP03	4	4	4	5	3	5
PP04	4	4	4	2	2	4
PP05	4	4	4	3	3	2
PP06	1	1	3	1	1	1
EP01	4	3	3	1	1	2
EP02	4	2	2	1	1	1
EP03	4	4	4	2	2	4
EP04	4	5	5	4	4	1
EP05	3	3	3	2	1	3

Experiência: (1) Nenhum (2) Estudei em aula ou em livro (3) Pratiquei em 1 projeto em sala de aula (4) Usei em 1 projeto na indústria (5) Usei em vários projetos na indústria

A técnica foi apresentada em uma aula de 180 minutos de duração e, ao contrário de muitos estudos anteriores envolvendo a técnica, não houve um exercício formal sobre a utilização da técnica, ainda que diversos exemplos de sua utilização tenham sido dados.

A ferramenta foi disponibilizada aos inspetores, que a instalaram em seus próprios computadores e a executaram sem qualquer acompanhamento dos condutores do estudo. O artefato inspecionado continha 14 páginas que descreviam requisitos para um sistema de gerenciamento de vídeo locadoras, contendo 16 defeitos reais conhecidos.

Além das informações qualitativas, obtidas através de um questionário e de uma reunião de acompanhamento, as principais variáveis de interesse neste estudo são a quantidade de defeitos reais identificados e o tempo total de inspeção. Estas variáveis permitem que sejam obtidos a cobertura dos defeitos (quantidade de defeitos identificados em relação à quantidade total de defeitos reais conhecidos) e o custo-eficiência (quantidade de defeitos identificados por unidade de tempo).

Os dados qualitativos permitiram identificar que todos os participantes do estudo concordaram com a viabilidade da ferramenta proposta, que foi capaz de auxiliá-los a "pensar de acordo com os conceitos de PBR". Mais importante do que isso foi a percepção, pelos participantes, da diminuição do esforço de preenchimento das informações (discrepâncias e casos de uso), em relação à aplicação manual da técnica.

Os participantes também identificaram algumas falhas na primeira versão da ferramenta e sugeriram possíveis melhorias. Estas considerações foram levadas em conta para o desenvolvimento da segunda versão da ferramenta, utilizada no estudo de observação subseqüente.

O resultado quantitativo deste estudo é detalhado na Tabela 5.2 a seguir:

Tabela 5.2 - Resultados Quantitativos do Estudo de Viabilidade

Id	Tempo (min)	Discrepâncias	Defeitos Reais	Cobertura (%)	Custo-Eficiência (defeitos/hora)
EP01	121	16	2	12,50	0,99
EP02	105	4	0	0	0
EP03	198	4	0	0	0
EP04	80	16	6	37,50	4,50
EP05	161	11	3	18,75	1,12
PP01	120	4	2	12,50	1,00
PP02	152	22	4	25,00	1,58
PP03	136	15	4	25,00	1,76
PP04	132	8	2	12,50	0,91
PP05	198	10	3	18,75	0,91
PP06	220	14	5	31,25	1,36

Para se ter uma visão geral dos dados, pode-se buscar a média de cada uma das variáveis para os dois grupos de estudo. Entretanto, é importante ressaltar que não faz sentido calcular a cobertura média dos defeitos, já que defeitos idênticos encontrados por vários inspetores são computados apenas uma vez.

A média das variáveis quantitativas, bem como o desvio padrão associado, para cada um dos grupos envolvidos neste estudo são exibidas na Tabela 5.3.

Tabela 5.3 – Média e Desvio Padrão das Variáveis Quantitativas

Grupo		Tempo (min)	Discrepâncias	Defeitos Reais	Cobertura (%)	Custo-Eficiência (defeitos/hora)
EP	x_m	133,00	10,20	2,20	13,75	1,32
	DP(x)	46,76	6,02	2,49	15,56	1,85
PP	x_m	159,67	12,17	3,33	20,83	1,25
	DP(x)	40,17	6,27	1,21	7,57	0,37

Ainda que os dados quantitativos não fossem o objeto principal deste estudo, eles demonstraram um forte indício da redução do tempo total de inspeção em comparação com a inspeção anterior, sem que isso diminuísse a cobertura de defeitos, resultando assim num custo-eficiência mais elevado.

Entretanto, alguns fatores podem causar confusão na análise dos dados:

(a) Os documentos de requisitos inspecionados tratavam de domínios diferentes;

(b) Os documentos eram de tamanhos (quantidades de páginas) e número de defeitos reais conhecidos diferentes, e;

(c) Os inspetores tinham uma experiência maior quando realizaram a segunda inspeção (estudo de viabilidade).

Por exemplo, considerando o fator de confusão descrito no item (a), um mesmo questionário foi utilizado para caracterizar a experiência dos inspetores em contextos diferentes. Lembrando que o domínio do documento inspecionado no estudo sobre a influência cultural tratava de postos de gasolina, obtivemos os resultados descritos na Tabela 5.4.

Tabela 5.4 - Experiência em Diferentes Contextos

Id	Experiência em Diferentes Contextos: Quanto você sabe sobre ...	
	Utilizar um posto de gasolina?	Utilizar uma vídeo locadora?
PP01	3	5
PP02	3	3
PP03	3	3
PP04	5	5
PP05	3	5
PP06	1	3
EP01	5	5
EP02	3	3
EP03	3	3
EP04	5	5
EP05	5	5

Experiência: (1) Eu não tenho familiaridade com a área. Eu nunca fiz isso. (2) Eu utilizo isto algumas vezes, mas não sou especialista. (3) Eu sou muito familiar com esta área. Eu me sentiria confortável fazendo isso.

Percebemos então que apenas três indivíduos (PP01, PP05 e PP06) relataram ter mais familiaridade com o domínio de uma vídeo-locadora, com todos os demais tendo a mesma familiaridade nos dois domínios. Assim, este fator teve uma influência pequena nos resultados.

Da mesma forma, os outros dois fatores de confusão podem ter influenciado, de certa maneira, os resultados individuais. Entretanto, mesmo considerando estes aspectos, não é objetivo dos estudos colocar à prova apenas a eficiência ou alguma característica individual dos participantes utilizando PBR, mas avaliar a utilização da ferramenta, tecnologia em questão, que apóia PBR por um grupo de desenvolvedores. Portanto, consideramos que os resultados são suficientes para estabelecer uma hipótese que deve ser testada em estudos com mais controle no futuro.

Este estudo nos forneceu respostas que indicam que é viável explorar esta abordagem, estabelecendo a hipótese de que a ferramenta reduz o tempo total de inspeção. É fundamental destacar que as observações feitas nesta seção não são obviamente conclusões definitivas sobre o papel da ferramenta de apoio à aplicação

de PBR, mas apenas conjecturas que devem ser exploradas com maior rigor em estudos futuros. A seção seguinte relatará a segunda etapa da aplicação da metodologia, o estudo de observação.

5.5 – Estudo de Observação

Este segundo estudo, conduzido no segundo semestre de 2003, envolveu 10 participantes, alunos de graduação em Engenharia Eletrônica e de Computação da UFRJ matriculados no curso de desenvolvimento de software orientado a objetos.

Os participantes foram aleatoriamente divididos em três grupos. Para atender às necessidades de três departamentos distintos da universidade, cada grupo ficou responsável pelo desenvolvimento de uma aplicação web: "Sistema de Gerenciamento de Equipamentos", "Sistema Gerenciador do Primeiro Emprego" e "Sistema de Workflow de Projetos de Engenharia". Todos os grupos seguiram o processo de desenvolvimento descrito em (TRAVASSOS *et al.*, 2001). Uma das características deste processo é a realização de inspeções ao longo do ciclo de vida do software.

Os estudantes declararam que estavam motivados para a realização deste estudo. Sete deles tinham experiência prévia no desenvolvimento de software e todos eles foram treinados em PBR para a perspectiva do usuário, utilizando a técnica manualmente num exercício em classe.

Os requisitos para os três sistemas descritos acima foram identificados a partir de entrevistas com os futuros usuários dos sistemas. Os grupos trocaram entre si os documentos de requisitos produzidos para que fossem inspecionados com o apoio da ferramenta. Para que a observação direta dos condutores do estudo fosse possível, a inspeção ocorreu no laboratório do grupo de Engenharia de Software da COPPE/UFRJ.

A sessão de inspeção foi agendada para um dia de aula normal, mas sem restrição de tempo de duração. Numa aula anterior, um treinamento de 45 minutos foi dado com o intuito de revisar os conceitos de PBR para a perspectiva do usuário e demonstrar a utilização da ferramenta de apoio.

Todas as equipes foram capazes de terminar a inspeção durante o tempo de aula (120 minutos), mas em razão dos diferentes tipos de sistemas e da inexistência de um suposto "gabarito", não faz sentido tentar inferir qualquer hipótese a respeito deste tempo. Os casos de uso elaborados durante a inspeção, bem como a lista de

discrepâncias identificadas, foram então distribuídos aos grupos para que pudessem dar prosseguimento ao desenvolvimento dos sistemas propostos.

Os dados qualitativos deste estudo corroboraram os resultados obtidos no estudo anterior. A utilização da ferramenta de apoio fez com que a aplicação de PBR se tornasse mais motivante para os inspetores. Além do aspecto de facilitar a realização das atividades clericais que envolvem a técnica, a ferramenta atuou como um "brinquedo tecnológico", aumentando o interesse dos inspetores. Os inspetores declararam que a ferramenta ajudou a aprimorar o conhecimento em relação à técnica.

Entretanto, diversos fatores podem ter influenciado os resultados do estudo. Como os indivíduos envolvidos no estudo eram alunos de graduação (portanto não tão maduros), e inspecionaram o documento produzido por seus colegas de classe, uma atmosfera de competição tomou conta do ambiente, apesar de o fato de suas avaliações independerem dos seus desempenhos como inspetores.

Além disso, o fato de os condutores do experimento terem explicitamente observado a inspeção pode ter feito com que, de forma consciente ou não, os indivíduos tentassem agradar aos pesquisadores. Esta constatação também representa um desafio: contornar o efeito Hawthorne (ver seção 4.5.5.2).

Em muitos casos, de acordo com os objetivos do estudo, a ocorrência deste efeito pode não ter impactos significativos para justificar o esforço de evitá-lo. Mesmo assim, é importante reconhecê-lo na avaliação dos dados obtidos como o estudo.

5.6 – Estudo de Caso: Ciclo de Vida Real

Este estudo ainda não foi conduzido, mas seu planejamento irá considerar o ciclo de vida nos mesmos moldes do estudo de observação relatado na seção anterior, adotando o processo descrito em (TRAVASSOS *et al.*, 2001).

Neste contexto, a ferramenta de apoio a PBR vem sendo utilizada numa infra-estrutura de apoio ao processo de inspeção de software para apoiar a inspeção de requisitos em projetos reais. O caso mais recente envolveu a inspeção do documento de requisitos para a ferramenta de apoio a aplicação das OORTs (REIS e TRAVASSOS, 2003), estando os resultados disponíveis em (KALINOWSKI *et al.*, 2004).

Assim, foi possível identificar que a ferramenta de apoio a PBR já se encontra com maturidade suficiente para que possamos avaliá-la segundo uma perspectiva

industrial. Neste caso, a ferramenta será integrada a um ambiente de desenvolvimento de software instanciado a partir da Estação TABA (http://www.cos.ufrj.br/~taba) e distribuída para as organizações de software que, atualmente, participam do projeto de transferência de tecnologia em processo de desenvolvimento de software em convênio com a Riosoft (SOFTEX). Desta forma, será possível avaliar sua aplicabilidade em situações atuais de desenvolvimento e realizar os ajustes necessários, que esperamos serem mínimos.

5.7 – Conclusão

O objetivo deste capítulo foi relatar não somente os estudos realizados, mas também a experiência preliminar adquirida na avaliação de PBR Tool seguindo uma metodologia para a transferência de novos processos e tecnologias para a indústria.

O desenvolvimento de tecnologias de software apoiadas por experimentação tem demonstrado ser uma abordagem adequada, podendo dar um diferencial de qualidade e crédito às tecnologias que vêm sendo desenvolvidas na academia. Esta não é uma tarefa trivial, e grande parte de seu sucesso dependerá da cooperação entre as diferentes instituições. Visando isto, foram expostos neste capítulo os desafios encontrados e as medidas que julgadas adequadas para enfrentá-los. De uma forma resumida, esta experiência nos fez concluir que:

- O impacto da introdução de uma nova tecnologia de software deve ser iterativamente avaliado, ou seja, é ingênuo assumir que todos os problemas serão identificados e contornados em um só estudo.
- É essencial planejar os mecanismos de experimentação e os estudos experimentais a serem realizados. Eventuais falhas ocorridas durante um estudo experimental podem comprometer a sua execução.
- A necessidade de um planejamento bem feito é ainda maior quando enfrentamos uma realidade de escassez de indivíduos disponíveis e aptos a participar de nossos estudos experimentais.

Vale lembrar que a execução destes estudos motivou a criação da funcionalidade de monitoramento não intrusiva da aplicação da técnica para verificar o quão fiel ela é aplicada pelos inspetores.

Capítulo 6 - Considerações Finais

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

6.1 - Conclusões

Fundamentando-se no conceito que aspectos de qualidade devem ser tratados simultaneamente ao processo de desenvolvimento de software, este trabalho destacou uma proposta de apoio ferramental para as técnicas de leitura baseada em perspectiva (PBR), utilizadas para a identificação de defeitos em documentos de requisitos de software.

Foram relatados diversos estudos mostrando a eficiência dessas técnicas em determinadas circunstâncias. Entretanto, muitos dos estudos de PBR, que envolveram a aplicação manual da técnica, também revelaram aspectos negativos. Os requisitos para a ferramenta proposta neste trabalho foram definidos para contornar estes aspectos negativos. Assim, PBR Tool se propôs a:

- **Facilitar o relato das discrepâncias:**

A grande questão que se coloca quando a inspeção é feita de forma manual, com os envolvidos utilizando formulários em papel, é a motivação dos inspetores. Ainda que identifiquem corretamente os requisitos com defeitos, a utilização de formulários em papel inibe uma descrição mais apurada do defeito encontrado. Além disso, conforme observado por HALLING *et al.* (2002), inspeções manuais baseadas em papel têm limitações, como o alto custo, a ineficiência e a interferência nas atividades realizadas durante o processo.

Ao oferecer um apoio para o relato das discrepâncias, espera-se que PBR Tool contribua para o aumento da motivação dos inspetores.

- **Auxiliar na diferenciação dos tipos de defeito:**

Esta dificuldade, por parte dos inspetores, não é resultado de uma eventual má definição das classes de defeito. O fato é que o entendimento e a absorção da taxonomia de defeitos pelos inspetores requer um certo tempo. De qualquer forma, como resultado, esta dificuldade leva a inspeções mais demoradas. PBR Tool oferece, através de uma ajuda direcionada, explicações e exemplos sobre cada uma das

classes de defeitos. Espera-se com isso diminuir esse tempo de absorção e, conseqüentemente, o tempo total de inspeção.

- **Verificar, de forma não intrusiva, a conformidade da aplicação da técnica:**

Poucos estudos envolvendo PBR preocuparam-se em verificar se a técnica é fielmente aplicada. Conforme visto nos capítulo anteriores, podem haver distorções nos resultados obtidos quando essa observação é feita de forma intrusiva pelos pesquisadores.

Esse trabalho identificou e caracterizou algumas possíveis abordagens para permitir a verificação da conformidade de processos. PBR Tool adotou como solução a coleta de métricas de forma não intrusiva.

6.2 - Contribuições

A ferramenta em questão representa uma inovação do ponto de vista de apoio à inspeção, já que apóia especificamente a aplicação de uma técnica de identificação de defeitos reconhecidamente eficiente.

A avaliação desta ferramenta foi fortemente baseada em experimentação. Em se tratando de uma área de pesquisa relativamente nova no contexto da Engenharia de Software, este trabalho também buscou oferecer uma contribuição à área ao relatar a experiência preliminar adquirida na avaliação da ferramenta seguindo uma metodologia para a transferência de novos processos e tecnologias para a indústria. O desenvolvimento de tecnologias de software apoiadas em experimentação tem se demonstrado uma abordagem adequada, podendo dar um diferencial de qualidade e crédito às tecnologias que vêm sendo desenvolvidas na academia.

O desenvolvimento desta ferramenta também motivou uma discussão a respeito da conformidade da aplicação de processos e/ou técnicas. Temos observado uma tendência crescente entre os pesquisadores de basear suas propostas em experimentação. Se por um lado isso representa um sinal de maturidade na área, por outro pode representar uma ameaça muito grande, já que conclusões precipitadas podem ser tomadas. Isto ocorrerá quando as ameaças à validade destes experimentos não forem identificadas e isoladas. Entre essas ameaças, o grau de fidedignidade com que a técnica ou processo estudado é efetivamente aplicado deve ser avaliado, ainda que muitos pesquisadores ainda ignorem este aspecto.

Por eliminar a necessidade de manipulação de uma grande quantidade de documentos (formulários de relato de discrepâncias, documento de especificação de

requisitos, descrição textual da técnica, etc.), a ferramenta apresenta-se também como um recurso para facilitar a preparação e a execução de estudos envolvendo PBR. Como resultado, erros de compilação dos dados obtidos poderão ser minimizados, permitindo uma melhor meta análise.

Esta ferramenta está disponível no contexto de uma infra-estrutura de suporte a inspeções de software, sendo integrada com ferramentas complementares através da troca de dados em XML, seguindo a abordagem proposta em (SPINOLA e TRAVASSOS, 2003).

6.3 – Limitações

Este trabalho apresenta algumas limitações. Em relação à ferramenta, o fato de seu escopo estar limitado a apenas uma das perspectivas da técnica inibe uma das principais premissas de PBR, que é focar a responsabilidade de cada inspetor.

Uma outra limitação da ferramenta refere-se à manipulação do documento de requisitos. Ainda que a ferramenta seja capaz de ler documentos nos formatos TXT e RTF, funcionalidades de manipulação mais elaboradas poderiam ser oferecidas no caso de haver bibliotecas livres de manipulação de arquivos do Microsoft Word. Ferramentas para a gerência de requisitos como o Rational Requisite Pro, por exemplo, identificam tipos de requisitos a partir do estilo de texto utilizado no documento.

Por fim, uma outra limitação está relacionada aos estudos executados. A metodologia experimental iterativa para introduzir processos de software não foi completamente seguida e, como resultado, um estudo com maior controle sobre as variáveis envolvidas não foi realizado.

6.4 – Perspectivas Futuras

No curto-prazo, as próximas atividades de pesquisa relacionadas a este trabalho concentrar-se-ão na execução das etapas restantes da metodologia proposta por (SHULL *et al.*,2001) e adotada neste trabalho para a avaliação experimental da ferramenta aqui apresentada. Um estudo de caso envolvendo a ferramenta num ciclo de desenvolvimento real foi planejado e tem a sua execução prevista para o segundo semestre de 2004.

Este trabalho representa um primeiro passo para o desenvolvimento de uma ferramenta que apóie todas as perspectivas sugeridas pela técnica PBR. A decisão de estender a ferramenta para as demais perspectivas foi tomada com base em

evidências obtidas com a execução dos estudos. Os estudos já realizados, descritos no capítulo 5, forneceram indícios da viabilidade desta pesquisa, incentivando a sua continuidade.

REFERÊNCIAS BIBLIOGRÁFICAS

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

ANDERSSON, M. e BERGSTRAND, J., 1995, *Formalizing Use Cases with Message Sequence Charts*, Master Thesis, Department of Communication Systems, Lund Institute of Technology.

BASILI, V., 1985, "Quantitative Evaluation of Software Engineering Methodology", In: *Proceedings of the First Pan Pacific Computer Conference*, v.1, pp. 379-398, Melbourne, Australia.

BASILI, V., CALDIERA, F., LANUBILE, F., et al., 1996a, "Studies on Reading Techniques", In: *Proceedings of the Twenty-First Annual Software Engineering Workshop*, pp. 59-65, Maryland, December.

BASILI, V.R., BRIAND, L.C., MELO, W.L., 1996b, "A Validation of Object-Oriented Design Metrics as Quality Indicators", *IEEE Transactions on Software Engineering*, v.22, n.10, pp. 751-761.

BASILI, V.R., GREEN, S., LAITENBERGER, O., et al., 1996c, "The Empirical Investigation of Perspective-Based Reading", *Empirical Software Engineering: An International Journal*, v.1, n.2, pp. 133-164.

BASILI, V., LAITENBERGER, O., SHULL, F., RUS, I, 2000, "Improving Software Inspections by Using Reading Techniques", In: *Proceedings of the International Conference on Software Engineering (ICSE)*, Tutorial, June.

BELLI, F. e CRISAN, R., 1996, "Towards Automation of Checklist-Based Code Reviews", In: *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE 96)*, pp. 24-34.

BINDER, R., 1999, *Testing Object-Oriented Systems: Models, Patterns, and Tools*, 1 ed., Addison-Wesley Pub Co.

BLECHER, N., 2004, "Máquina de Vender", *Revista Exame*, Edição 811, ano 38, nº 3, pp. 44-54.

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

CARVER, J.C., 2003, *The Impact of Background and Experience on Software Inspections*, Tese de Doutorado, University of Maryland, Department of Computer Science, Maryland.

CARVER, J.C., JACCHERI, L., MORASCA, S., et al., 2003, "Issues Using Students in Empirical Studies in Software Engineering Education", In: *Proceedings of 2003 International Symposium on Software Metrics (METRICS 2003)*, pp 234-244, September.

CHAPETTA, W.A., 2004, *Ferramenta para Construção de Modelos de Casos de Uso*, Projeto Final de Curso, Departamento de Ciência da Computação, Universidade Federal do Rio de Janeiro.

CHEN, T.Y., POON, P.L., TANG, S.F., et al., 2002, "Towards a Problem-Driven Approach to Perspective-Based Reading", In: *Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering (HASE 2002)*, pp 221-229, October.

CHENG, B., JEFFERY, R., 1996, "Comparing Inspection Strategies for Software Requirement Specifications", In: *Proceedings of 1996 Australian Conference on Software Engineering (ASWEC 96)*, pp. 203-211, July.

CHIDAMBER, S.R., KEMERER, C.F., 1994, "A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*, v.20, n.6, pp. 476-493.

CHRISTIENSEN, M. e THAYER, R., 2001, *The Project Manager's Guide to Software Engineering's Best Practices*, IEEE Computer Society Press.

CIOLKOWSKI, M., DIFFERDING, C., LAITENBERGER, O., et al., 1997, *Empirical Investigation of Perspective-Based Reading: A Replicated Experiment*, International Software Engineering Research Network Report n. 97-13.

CIOLKOWSKI, M., SHULL, F. BIFFL, S., 2002a, "A Family of Experiments to Investigate the Influence of Context on the Effect of Inspection Techniques", In: *Proceedings of the 6th International Conference on Empirical Assesment in Software Engineering (EASE)*, pp. 48-60, April.

CIOLKOWSKI, M., LAITENBERGER, O., ROMBACH, D., et al., 2002b, "Software Inspections, Reviews and Walkthroughs", In: *Proceedings of the 24rd International Conference on Software Engineering*, pp. 641-642, May.

CONRADI, R., MOHAGHEGHI, P., ARIF, T., et al., 2003, "Object-Oriented Reading Techniques for Inspection of UML Models – An Industrial Experiment", In: *Proceedings of the European Conference on Object-Oriented Programming (ECOOP 03)*, pp. 483-500, July.

DAVIDOFF, L.L., 1987, *Introduction to Psychology*, McGraw-Hill.

DAVIS, A., 1990, *Software Requirements Analysis and Specificatio*", Prentice Hall. Ref: (CHRISTIENSEN e THAYER, 2001)

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

FAIRLEY, R.E., 1985, *Software Engineering Concepts*, McGraw-Hill Book Co.

FOWLER, P., 1986, "In-Process Inspection of Workproducts at AT&T", *AT&T Technical Journal*, v.65, n.2, pp. 102-112.

FRANZ, L.A., SHIH, J.C., 1994, "Estimating the Value of Inspections and Early Testing for Software Projects", *Hewlet Packard Journal,* pp. 60-67, December.

FREITAS, M.E., PAGLIUSO, P.B.B., VILLAS BOAS, A., et al., 2004, "Inspeção de documentos de requisitos baseado em técnica de leitura PBR: experiência prática no CPqD", In: *Anais do Simpósio Brasileiro de Qualidade de Software.*

GINTELL, J., HOUDE, M., MCKENNEY, R., 1995, "Lessons Learned by Building and Using Scrutiny, a Collaborative Software Inspection System", In: *Proceedings of the 7th International Workshop on Computer-Aided Software Engineering*, pp 350-357.

GLASS, R., 1999, "Inspections – Some Surprising Findings", *Communications of the ACM*, v. 42, n.4, pp 17-19, April.

GRADEN, M.E., HORSLEY, P.S., PINGEL, T.C., 1986, "The Effects of Software Inspections on a Major Telecommunication Project", *AT&T Technical Journal*, v.65, n.3, pp. 32-40, May/June.

HALLING, M., BIFFL, S., GRECHENIG, T., et al., 2001, "Using Reading Techniques to Focus Inspection Performance", In: *Proceedings of the 27th Euromicro Workshop on Software Process and Product Improvement*, pp. 248-257.

HALLING, M., BIFFL, S., GRÜNBACHER, P., 2002, "A Groupware-Supported Process for Active Inspection Management", In: *Proceedings of the 28th Euromicro Conference*, pp 251-258.

HOST, M., REGNELL, B., WOHLIN, C., 2000, "Using Students as Subjects – A Comparative Study of Students and Professionals in Lead-Impact Assessment", *Empirical Software Engineering*, v.5, n.3, pp 201-214.

HÖHN, E.N., MALDONADO, J.C., MENDONCA, M.G., et al., 2004, "PBR: Transferência de Tecnologia Baseada em Pacotes de Experimentação", In: Anais do Simpósio Brasileiro de Qualidade de Software.

HUMPHREY, W.S., 1989, *Managing the Software Process*, Addison-Wesley.

IEEE 610.12, 1990, *IEEE Standard Glossary of Software Engineering Terminology*, The Institute of Electrical and Electronics Engineers.

IEEE 830, 1998, *IEEE Recommended Practice for Software Requirements Specifications*, The Institute of Electrical and Electronics Engineers.

IEEE 1028, 1988, *IEEE Standard for Software Reviews and Audits*, The Institute of Electrical and Electronics Engineers.

ISO/IEC 12207:1995, 1997 NBR, *Tecnologia da Informação – Processos de Ciclo de Vida de Software.*

JOHNSON, P.M., 1994, "An Instrumented Approach to Improving Software Quality Through Formal Technical Review", In: *Proceedings of the 16th International Conference on Software Engineering*, pp 113-122.

JOHNSON, P.M., 1998, "Reengineering Inspection", *Communications of the ACM*, v.42, n2, pp 49-52.

JOHNSON, P.M., 2001, *Project Hackystat: Accelerating Adoption of Empirically Guided Software Development Through Non-Disruptive, Developer-Centric, In-Process Data Collection and Analysis*, University of Hawaii Technical Report 96822, Honolulu.

KALINOWSKI, M., SPINOLA, R.O., TRAVASSOS, G.H., 2004, "Infra-Estrutura Computacional para Apoio ao Processo de Inspeção de Software", In: Anais do Simpósio Brasileiro de Qualidade de Software.

KANER, C., FALK, J., NGUYEN, H. Q., 1999, *Testing Computer Software*, 2nd Edition, John Wiley & Sons.

KELLY, J.C., SHERIF, J.S., ISOPS, J., 1992, "An Analysis of Defects Densities Found During Software Inspections", *Journal of Systems and Software*, v.17, pp 111-117.

KITCHENHAM, B.A, PFLEEGER, S.L., PICKARD, L.M,, et al., 2002, "Preliminary Guidelines for Empirical Research in Software Engineering", *IEEE Transactions on Software Engineering*, pp 721-734.

LAITENBERGER, O., DEBAUD, J.M., 1997, "Perspective-Based Reading of Code Documents at Robert Bosch GmbH", *Information and Software Technology*, v.39, n.11, pp. 781-791.

LAITENBERGER, O., ATKINSON, C., 1999, "Generalizing Perspective-Based Inspection to Handle Object-Oriented Development Artifacts", In: *Proceedings of the 21st International Conference on Software Engineering*, pp. 494-503.

LAITENBERGER, O., DEBAUD, J.M., 2000, "An Encompassing Life Cycle Survey of Software Inspections", *The Journal of Systems and Software*, v.50, n.1, p.5-31.

LAITENBERGER, O., ATKINSON, C., SCHLICH, M., et al., 2000, "An Experimental Comparison of Reading Techniques for Defect Detection in UML Design Documents", The Journal of Systems and Software, v.53, n.2, pp.183-204.

LANUBILE, F., VISAGIO, G., 2000, "Evaluating Defect Detection Techniques for Software Requirements Inspection", *International Software Engineering Research Network Technical Report n.00-08.*

MACDONALD, F., MILLER, J., BROOKS, A., ROPER, M., WOOD, M., 1995, "A Review of Tool Support for Software Inspections", In: *Proceedings of the Seventh International Workshop on Computer-Aided Software Engineering*, pp 340-349, July.

MACDONALD, F., 1997, "Assist v1.1 User Manual", *EFoCS-22-96 Technical Report RR-96-199.*

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

MARTIN, J., TSAI, W.T., 1990, "N-Fold Inspection: A Requirements Analysis Technique", *Communications of the ACM*, v.33, n.2 , pp. 225-232.

MASHAYEKHI, V., DRAKE, J.M., TSAI, et al., 1993, "Distributed, Collaborative Software Inspection", *IEEE Software*, v.10, n.5, pp. 66-75.

MYERS, E.A., KNIGHT, J.C., 1993, "An Improved Software Inspection Technique and An Empirical Evaluation of Its Effectiveness", *Communications of the ACM*, v.36, n.11, pp. 50-61.

MURPHY, P., MILLER, J., 1997, "A Process for Asynchronous Software Inspection", In: *Proceedings of the 8th International Workshop on Software Technology and Engineering Practice*, pp. 96-104, IEEE Computer Press.

NETO, M.G.M., MALDONADO, J.C., FABBRI, et al., 2001, *Readers Project – Replication of Experiments: A Case Study Using Requirements Documents*, Relatório Técnico RT-NUPERC-2001-8/e, Universidade Salvador.

PARNAS, D.L., WEISS, D.M., 1987, "Active Design Reviews: Principles and Practice", *Journal of Systems and Software*, v.7, n.4, pp. 259-265.

PERPICH, J., PERRY, D., PORTER, A., et al., 1997, "Anywhere, Anytime Code Inspections: Using the Web to Remove Inspection Bottlenecks in Large-Scale Software Development", In: *Proceedings of the 19th International Conference on Software Engineering*, pp. 14-21.

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

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

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

ROETHLISBERGER, F.J. e DICKSON, W.J., 1939, *Management and the Worker*, Harvard University Press.

REGNELL, B., RUNESON, P., THELIN, T., 2000, "Are the Perspectives Really Different? – Further Experimentation on Scenario-Based Reading on Requirements", *Empirical Software Engineering: An International Journal*, v.5, n.4, pp. 331-356.

REIS, L.N.M., TRAVASSOS, G.H., 2003, "Apoio Automatizado à Configuração e Aplicação de OORTs", In: *Anais do 8º Workshop de Teses em Engenharia de Software*, pp. 35-40.

RUNESON, P., 2000, "Using Students as Experiment Subjects – An Analysis on Graduate and Freshmen Student Data", *Empirical Software Engineering*, v.5, n.3., pp. 201-214.

SABALIAUSKAITE, G., MATSUKAWA, F., KUSUMOTO, S., et al., 2002, "An Experimental Comparison of Checklist-Based Reading and Perspective-Based Reading for UML Design Document Inspection", In: *Proceedings of the International Symposium on Empirical Software Engineering*, pp. 148-157.

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

SHULL, F.J., 1998, *Developing Techniques for Using Software Documents: A Series of Empirical Studies*, Tese de Doutorado, University of Maryland, Department of Computer Science, Maryland.

SHULL, F., RUS, I., BASILI, V., 2000, "How Perspective-Based Reading Can Improve Requirements Inspections", *IEEE Computer*, v.33, n.7, pp. 73-79.

SHULL, F., CARVER, J., TRAVASSOS, G.H., 2001a, "An Empirical Methodology for Introducing Software Processes.", In: *Proceedings of the Joint 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9)*, pp. 288-296.

SHULL, F., RUS, I., BASILI, V., 2001b, " Improving Software Inspections by Using Reading Techniques", In: *Proceedings of the 23rd International Conference on Software Engineering*, pp. 726-727.

SHULL, F., BASILI, V., BOEHM, B., et al., M., 2002, "What We Have Learned About Fighting Defects", In: *Proceedings of the Eight IEEE Symposium on Software Metrics*, pp.249-259.

SILVA, L.F.S, TRAVASSOS, G.H., 2003, "Apoio Ferramental para Aplicação de Técnicas de Leitura Baseada em Perspectiva", In: *Anais do 8º Workshop de Teses em Engenharia de Software*, pp. 83-88.

SILVA, L.F.S., CHAPETTA, W.A., TRAVASSOS, G.H., 2004, "Inspeção de Requisitos de Software utilizando PBR e Apoio Ferramental", In: *Anais do Simpósio Brasileiro de Qualidade de Software*.

SILVA, L. F. S., TRAVASSOS, G. H., 2004, "Tool-Supported Unobtrusive Evaluation of Software Engineering Process Conformance". Aceito para publicação em *Proceedings of the International Symposium on Empirical Software Engineering*.

SORUMGARD, S., 1997, *Verification of Process Conformance in Empirical Studies of Software Development*, Tese de Doutorado, The Norwegian University of Science and Technology, Department of Computer and Information Science, Noruega.

TICHY, W.F.P., LUKOWICZ, L., PRECHELT, L., et al., 1995, "Experimental Evaluation in Computer Science: A Qualitative Study", *Journal of Systems and Software,* v.28, n.1, pp. 9-18.

TRAVASSOS, G.H., SHULL, F., FREDERICKS, M., et al., 1999, "Detecting Defects in Object-Oriented Designs: Using Reading Techniques to Increase Software Quality", In: *Proceeding of the Conference on Object-Oriented Programming Systems, Languages & Applications (OOPSLA),* pp.47-56.

TRAVASSOS, G.H., SHULL, F., CARVER, J., 2000, "A Family of Reading Techniques for OO Design Inspections", In: *Anais do Workshop de Qualidade de Software – Simpósio Brasileiro de Engenharia de Software*, pp.225-237.

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

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

VOTTA JR, L.G., 1993, "Does Every Inspection Need a Meeting?", *ACM Software Engineering Notes*, v.18, n.5, pp107-114.

YOUNESSI, H., 2002, *Object-Oriented Defect Management of Software*, Upper Saddle River, NJ, Prentice Hall.

YOURDON, E., 1989, *Revisões Estruturadas (Structured Walkthroughs)*, Editora Campus, Rio de Janeiro.

WEINBERG, G.M., FREEDMAN, D.P., 1984, "Reviews, Walk-through and Inspections", *IEEE Transactions on Software Engineering*, v.10, n.5, pp. 68-72.

ZHANG, Z., BASILI, V., SHNEIDERMAN, B., 1999, "Perspective-Based Usability Inspection: An Empirical Validation of Efficacy", In: *Proceedings of the Usability Professionals Association*, pp. 281-282.

APÊNDICE A – *Checklist* para Inspeção de Requisitos

1. Os requisitos exibem a distinção clara entre funções e dados?
2. Os requisitos definem todas as informações a ser apresentada aos usuários?
3. Os requisitos descrevem as respostas do sistema ao usuário devido à condições de erro?
4. Cada requisito está descrito claramente, é conciso e está apresentado de forma não ambígua?
5. O requisito é testável?
6. Existem requisitos implícitos ou ambíguos?
7. Existem requisitos conflitantes?
8. Existem áreas não tratadas no documento de requisitos que precisam ser consideradas?
9. Os requisitos de desempenho (tais como tempo de resposta, armazenamento de dados, etc.) foram definidos?
10. Se os requisitos envolvem a descrição de processos de tomada de decisão complexos, eles estão descritos de forma a facilitar sua compreensão (i.e., tabelas de decisão, árvores de decisão, etc.)?
11. Os requisitos para executar as atualizações do software foram especificados?
12. Existem requisitos que contêm algum nível desnecessário de detalhe do projeto?
13. As restrições de tempo-real foram especificadas em detalhe suficiente?
14. A precisão e acurácia dos cálculos foram especificadas?
15. É possível desenvolver um completo conjunto de casos de teste baseado apenas na informação contida na especificação de requisitos? Se não, que informação está faltando?
16. As restrições e dependências foram claramente descritas?
17. O documento contém realmente toda a informação prometida em sua introdução?

APÊNDICE B – PBR: Perspectiva do Usuário

Técnicas de Leitura para Construção de Casos de Uso

O objetivo é criar casos de uso para entender as funcionalidades que usuários do sistema seriam capazes de realizar. Isto requer a listagem das funcionalidades que o sistema oferece e as interfaces externas envolvidas com estas funcionalidades. Você terá que identificar atores/participantes (conjunto de usuários ou dispositivos) que interagem com o sistema considerando funcionalidades específicas. Lembre-se de considerar todas as funcionalidades do sistema, incluindo as condições especiais e de contingência. Siga os procedimentos abaixo para gerar os casos de uso utilizando as questões fornecidas para identificar as discrepâncias entre os requisitos:

Entrada: Um conjunto de novos requisitos

Saída: Os casos de uso do novo sistema
Uma lista contendo os defeitos que devem ser corrigidos para o novo sistema

1) Leia todos os requisitos uma vez, identificando os participantes envolvidos.

   a) Identifique os participantes nos novos requisitos. Participantes são os outros sistemas e usuários que interagem com o sistema descrito nos requisitos - isto é, eles participam das funcionalidades do sistema enviando mensagens para o sistema e/ou recebendo dele informações e instruções. Liste estes participantes no formulário A. Você pode usar as seguintes questões para ajudá-lo a identificar os participantes:
      - Quais grupos de usuários utilizam o sistema para executar suas tarefas?
      - Que grupos de usuários são necessários pelo sistema para executar suas funções? Estas funções podem ser as principais, ou mesmo as secundárias tais como manutenção e administração.
      - Quais são os sistemas externos que utilizam o sistema visando executar as tarefas?
      - Quais são os sistemas externos que são gerenciados ou de outra forma utilizados pelo sistema visando executar suas tarefas?
      - Quais são os grupos de usuários ou sistemas externos que enviam informação para o sistema?
      - Quais são os sistemas externos ou grupos de usuários que recebem informação do sistema?

   Q1.1 Um mesmo participante está sendo descrito por múltiplos termos nos requisitos?
   Q1.2 A descrição de como o sistema interage com um participante está inconsistente com a descrição do participante? Os requisitos estão confusos ou inconsistentes sobre esta interação? Esta situação está omitindo alguma parte importante da funcionalidade como um todo?
   Q1.3 Existem participantes necessários que foram omitidos? Isto é, o sistema precisa interagir com outro sistema, ou pedaço de hardware, ou um tipo de usuário que não está descrito?
   Q1.4 Existe algum sistema externo ou classe de "usuários" descrito nos requisitos que formalmente não interage com o sistema?

2) Leia todos os requisitos uma segunda vez, identificando as funções do produto.

   a) Identifique o conjunto de funcionalidades que o sistema deve ser capaz de executar. Isto é, que atividades os requisitos explicitamente descrevem que o sistema deve executar? (E.g. apresentar um registro do banco de dados, imprimir um relatório, criar e apresentar um gráfico) Registre a funcionalidade no formulário A.
   b) Agora considere como um usuário do sistema o visualizará. Ele ou ela provavelmente não está preocupado com as atividades individuais que o sistema executa, mas ao invés disto pensa em como pode usar o sistema para alcançar algum objetivo (e.g. adicionar informação em alguma base de dados, calcular um pagamento, visualizar a situaçao atual de uma conta). Estes objetivos do usuário serão definidos como os casos de uso para o sistema.
   c) Para cada usuário, utilize uma nova cópia do formulário B. Decida qual das atividades descritas no formulário A estão envolvidas neste caso de uso e anote-as no formulário B. Descreva então, do ponto de vista do usuário, que passos são necessários para alcançar o objetivo e utilize setas para identificar o fluxo de controle do sistema ("Primeiro esta funcionalidade acontece, então esta..."). Utilize setas duplas para representar o momento onde o fluxo de controle pode se dividir. Lembre-se de incluir funcionalidades excepcionais no caso de uso. Verifique os requisitos funcionais apropriados para assegurar que você tem os detalhes corretos do processamento e fluxo de controle.
   d) Para cada caso de uso que você criar, lembre-se de relacionar que classe(s) de participante(s) utilizariam a funcionalidade (as classes de participantes já devem estar registradas na lista de participantes do formulário A), bem como as ações que iniciam a funcionalidade (e.g. "selecionando a opção 2 do menu principal" deve disparar o caso de uso para apagar os registros das base de dados) Existem espaços específicos para registrar estas informações no formulário B. Finalmente, dê ao caso de uso um nome descritivo que represente a funcionalidade que ele descreve e associe a ele um número para referência futura.

   Q2.1 As condições de início para cada caso de uso estão especificadas num nível de detalhe apropriado?
   Q2.2 As classe(s) de participantes que utilizam a funcionalidade estão descritas e são corretas e apropriadas?
   Q2.3 Existe alguma funcionalidade do sistema que deveria ser incluída no caso de uso mas que está descrita com detalhe insuficiente ou omitida dos requisitos?
   Q2.4 O sistema está suficientemente descrito de forma que você possa entender que atividades são necessárias para o usuário executar o objetivo do caso de uso? Esta combinação de atividades faz sentido, baseado na descrição geral do sistema e em seu conhecimento do domínio? A descrição permite mais de uma interpretação de como o sistema alcança seu objetivo?
   Q2.5 Os requisitos estão omitindo casos de uso que você sente serem necessários, de acordo com seu conhecimento do domínio ou descrição geral?

3) Confira os participantes com todos os casos de uso que estejam envolvidos. (lembre-se que se dois participantes estão envolvidos em todos os mesmos casos de uso, eles podem representar um único ator e poderiam ser combinados) Utilize a coluna "Envolvido em que caso de uso?" do formulário A para conferir os participantes com os casos de uso apropriados.

Q3.1 Está claro nos requisitos quais participantes estão envolvidos em quais casos de uso?

Q3.2 Baseado nos requisitos gerais e em seu conhecimento do domínio, cada participante foi relacionado a todos os casos de uso relevantes?

Q3.3 Os participantes envolvidos nos casos de uso estão incompatíveis com a descrição do participante?

Q3.4 Participantes necessários foram omitidos? (e.g. existem casos de uso que requerem informação que não pode ser obtida de alguma das fontes descritas nos requisitos)?

Agradecimentos

Estas diretrizes estão baseadas nos seguintes trabalhos:

Andersson, Michael, and Johan Bergstrand. (1995) *Formalizing Use Cases with Message Sequence Charts*. Masters Thesis for the Department of Communication Systems at Lund Institute of Technology.

Jacobson, Ivar, *et al.* (1992) *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley Publishing Company.

E no trabalho de Curt Zimmerman e Guilherme H. Travassos durante sua estada junto ao grupo de Engenharia de Software Experimental (Empirical Software Engineering Group).