Inspeções de Requisitos de Software Utilizando PBR e Apoio Ferramental

Luís Felipe Santos Silva Wladmir Araújo Chapetta Guilherme Horta Travassos
{lfelipe,wladmir, ght}@cos.ufrj.br

Programa de Engenharia de Sistemas e Computação
Universidade Federal do Rio de Janeiro - COPPE
Caixa Postal 68511 – CEP 21945-970 - Rio de Janeiro – Brasil

Resumo

A ocorrência de defeitos em artefatos de software é praticamente inevitável. É extremamente arriscado contar apenas com atividades de teste para identificar estes defeitos. Aspectos de qualidade devem ser tratados simultaneamente ao processo de desenvolvimento de software, já que não poderão ser impostos quando o produto estiver finalizado. Os custos associados ao teste, isolamento, correção e re-teste do software são maiores que o custo necessário para identificar os defeitos tão logo eles sejam inseridos nos artefatos produzidos ao longo do ciclo de desenvolvimento. Inspeções de software se propõem a reduzir o número de defeitos propagados de uma fase de desenvolvimento para outra.

Num processo de inspeção, a identificação de defeitos pode ser feita de forma *ad-hoc*, com a utilização de *checklists* ou com a adoção de uma técnica específica. As técnicas de leitura baseada em perspectiva (PBR) foram criadas para apoiar a identificação de defeitos em documentos de requisitos de software escritos em linguagem natural. PBR tem sido submetida a diversos estudos experimentais e as observações resultantes destes estudos nos motivaram a definir uma ferramenta que apoiasse sua aplicação. Nossa hipótese se relaciona à possibilidade da redução do tempo necessário para a inspeção. Um estudo de viabilidade da ferramenta realizado com estudantes de pós-graduação mostrou indícios desta possibilidade e da viabilidade de utilização desta ferramenta.

Palavras-Chave: Inspeções de Software, Técnicas de Leitura, Engenharia de Software Experimental.

Abstract

Artifact's defects occurrence is almost inevitable. To identify these defects counting just on testing activities is extremely risky. Quality issues should be addressed in every artifact produced throughout the software development process, since quality cannot be imposed when the software product is finished. The costs related to software testing, isolation, correction and retesting are higher than the costs related to the identification of defects as soon as they are inserted into the produced software artifacts. Software inspections aim to reduce the number of defects transmitted from one development phase to another.

Considering the software inspection process, defects identification can be accomplished by applying an *ad-hoc* fashion, through checklists or using a specific technique. Perspective-Based Reading (PBR) was created to support defects identification in software requirements documents written in natural language. PBR has been submitted to several empirical studies and the resulting observations motivated us to define a tool to support it, having the hypothesis that it would be able to reduce inspections total time. A feasibility study conducted with graduate students gave us some indication that our hypothesis may be true and tool usage is feasible.

Keywords: Software inspections, Reading Techniques, Experimental Software Engineering.

1. Introdução

Num recente artigo sobre uma grande rede de lojas do varejo de eletrodomésticos e móveis [1] é citado que um sistema de informação introduzido em 1995 permitiria uma economia de 4 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, 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 [2]. Como este processo envolve a participação humana, mesmo as melhores tecnologias não serão capazes de evitar a ocorrência de defeitos [3], que ocorrem justamente na transformação das informações pelas diferentes fases do processo de desenvolvimento [4].

Parece claro, 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 [5].

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 plano 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.

PBR tem sido submetida a diversos estudos experimentais, havendo evidências de sua maior eficiência em relação às demais técnicas. Segundo Boehm e Basili [6], inspeções de requisitos que se utilizem das técnicas de leitura baseadas em perspectiva encontrariam, em média, 35% mais defeitos que inspeções *ad-hoc*. Por outro lado, estes mesmos autores reconhecem que a aplicação da técnica requer 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. Neste artigo é apresentada uma ferramenta de apoio para aplicação de PBR para a perspectiva do usuário, estabelecendo a hipótese de que sua utilização permitiria a aplicação da técnica num tempo menor do que quando a técnica é aplicada manualmente. Num estudo experimental realizado com o intuito de verificar a viabilidade desta ferramenta, identificamos indícios que podem confirmar nossa hipótese.

Além desta introdução, este artigo está dividido em mais 4 seções. As técnicas de leitura baseada em perspectiva são descritas na seção 2. Na seção seguinte é apresentada a ferramenta de apoio a PBR citada acima e as observações experimentais resultantes de um estudo de viabilidade da ferramenta conduzido com estudantes de pós-graduação em Engenharia de Software da UFRJ. A seção 4 contextualiza a utilização da ferramenta num processo de desenvolvimento de software e a seção 5 conclui esse artigo enumerando os trabalhos futuros.

2. Técnica de Leitura Baseada em Perspectiva (PBR)

Especificações de requisitos servem de base para o projeto, implementação e teste do software, sendo utilizadas pelos diversos participantes do processo (patrocinadores, usuários, projetistas, programadores, testadores, etc). Assim, defeitos em requisitos, ao serem propagados pelas demais fases podem causar uma série de problemas, desde o desperdício de recursos, fruto do re-trabalho necessário para a sua correção, até o não atendimento das

necessidades do usuário do produto. Identificar estes defeitos pode constituir um fator crítico de sucesso para um projeto.

As abordagens para a identificação de defeitos em documentos de requisitos de software têm focado em duas áreas principais [7]:

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

A inspeção de software envolve tarefas nas quais uma equipe de indivíduos tecnicamente qualificados determina se um dado artefato criado possui qualidade satisfatória. As possíveis deficiências de qualidade detectadas 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 [3]. Como originalmente observado por Fagan [8], 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 muito maiores do que o custo de se destinar um número razoável de inspetores para inspecionar um artefato imediatamente após ele ter sido escrito [9].

2.1 Técnicas de Leitura

Apesar da compreensão de um artefato ser uma atividade chave para a identificação de defeitos, existem poucas técnicas ou ferramentas para apoiá-la [10]. Além disso, a maior parte dos trabalhos sobre inspeção simplesmente assume que a revisão individual dos artefatos é feita de forma satisfatória [11]. Estas constatações e as deficiências das técnicas existentes levaram à criação das chamadas técnicas de leitura de software.

No contexto de inspeções de software, uma técnica de leitura é um procedimento que guia individualmente um inspetor 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.

As técnicas de leitura focam em ajudar os indivíduos a adquirir uma compreensão de alto-nível sobre algum artefato de software. Esta compreensão deve ser suficiente para permitir aos indivíduos identificar tanto as informações importantes para a execução de uma determinada tarefa, como a relação destas informações com o problema de que se está tratando [11].

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 a um conjunto de questionamentos, nos quais o desenvolvedor deva focar.

2.2 Técnica de Leitura Baseada em perspectiva – PBR

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 [11]. Assim, identificaríamos três diferentes "consumidores" deste artefato: projetistas, testadores e usuários do sistema.

PBR inicialmente sugere tarefas que levam à construção de esboços de artefatos de alto-nível, que são abstrações do documento sendo 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). Estas tarefas devem ajudar na percepção de quais informações no documento de requisitos são importantes para a identificação de defeitos.

Seguindo esta linha de raciocínio, foi identificado 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 casos de testes (utilizando particionamento equivalente) e um conjunto de casos de uso. Esta idéia está representada na figura 1. Como cada um destes artefatos é tratado separadamente no ciclo de vida do software, cada inspetor deve individualmente lidar com cada uma dessas abstrações.

Enquanto esboça o artefato de alto-nível correspondente à perspectiva adotada, o inspetor responde alguns questionamentos que o guiarão na identificação dos possíveis problemas no documento de requisitos [11]. 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 que estas perguntas sejam respondidas também não conterão as informações esperadas pelo seu futuro "consumidor".

Figura 1 – Diferentes Perspectivas para Inspeção de Documentos de Requisitos de Software

Um outro aspecto tratado pela técnica refere-se aos possíveis tipos de defeitos em requisitos de software. De forma ampla, podemos pensar em dois tipos principais de defeitos:

- Omissão: quando informações importantes são deixadas de lado;
- Comissão: quando uma informação errada é introduzida no documento.

O fato do documento de requisitos ser escrito em linguagem natural pode trazer uma série de ambigüidades. Permitindo múltiplas interpretações, os artefatos produzidos a partir dele podem não contemplar as características que o software deve possuir. Requisitos podem ainda não expressar de forma correta o conhecimento que se tem sobre o domínio da aplicação, ou ainda, representar funcionalidades fora do escopo do sistema a ser desenvolvido. Há também a possibilidade de determinados requisitos serem contraditórios, deixando o documento inconsistente.

Assim, PBR propõe que os seguintes tipos de defeitos sejam tratados: omissão, ambigüidade, informação incorreta, informação estranha e inconsistência. Estes tipos de defeitos são representações dos problemas encontrados em requisitos de software e formam a base para a derivação das questões a serem respondidas pelos inspetores [10]. Entretanto, esta taxonomia não pretende ser definitiva e cada organização pode acrescentar mais tipos de acordo com suas necessidades [12].

2.3 PBR - Perspectiva do Usuário

Com o intuito de facilitar a compreensão da técnica definida anteriormente, esta seção irá brevemente relatar 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.

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. *PBR Tool* - Apoio Ferramental para PBR na Perspectiva do Usuário

As diversas observações resultantes dos estudos experimentais aos quais a técnica foi submetida nos motivaram a definir uma ferramenta que apoiasse a aplicação de PBR, na perspectiva do usuário, para documento de requisitos escritos em linguagem natural.

A decisão de estender a ferramenta para as demais perspectivas será tomada caso haja evidências experimentais dos possíveis benefícios que este apoio ferramental pode trazer.

Esta ferramenta se propõe a:

a) Fornecer um guia para a execução das tarefas especificadas pela técnica

A metáfora da interface gráfica de *PBR Tool* foi inspirada nas ferramentas CASE desenvolvidas para a estação TABA [13]. 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. A figura 2 mostra essa organização do *menu*.

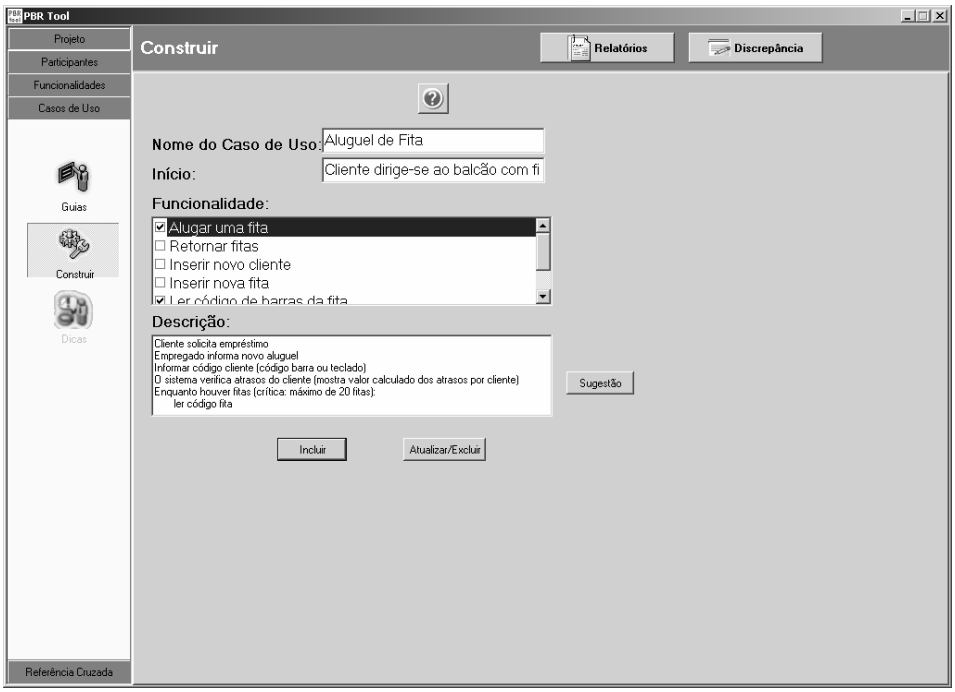


Figura 2 – Metáfora de Interface de *PBR Tool*

Para guiar o inspetor, *PBR Tool* exige que o mesmo execute cada uma das tarefas na ordem originalmente proposta pela técnica. Graficamente, isso é representado através de ícones com imagens indistintas, 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 inspetor tentar iniciar uma tarefa nesta condição, será exibida uma mensagem avisando sobre 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 que este ícone representa pode ser iniciada.

Seria ingênuo afirmar que esta funcionalidade garantirá que os inspetores seguirão fielmente a técnica. Sua intenção é, na verdade, tentar aprimorar o entendimento da técnica pelo inspetor. A forma como PBR *Tool* verifica o quão fiel a técnica foi aplicada será descrita no item "d".

b) Facilitar o 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.

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. A figura a seguir representa como este relato é feito em *PBR Tool.*

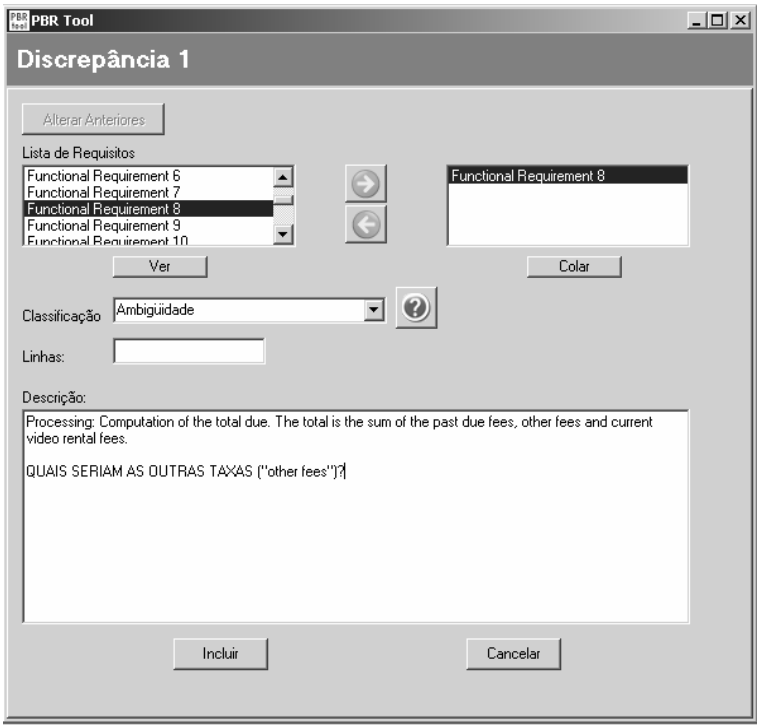
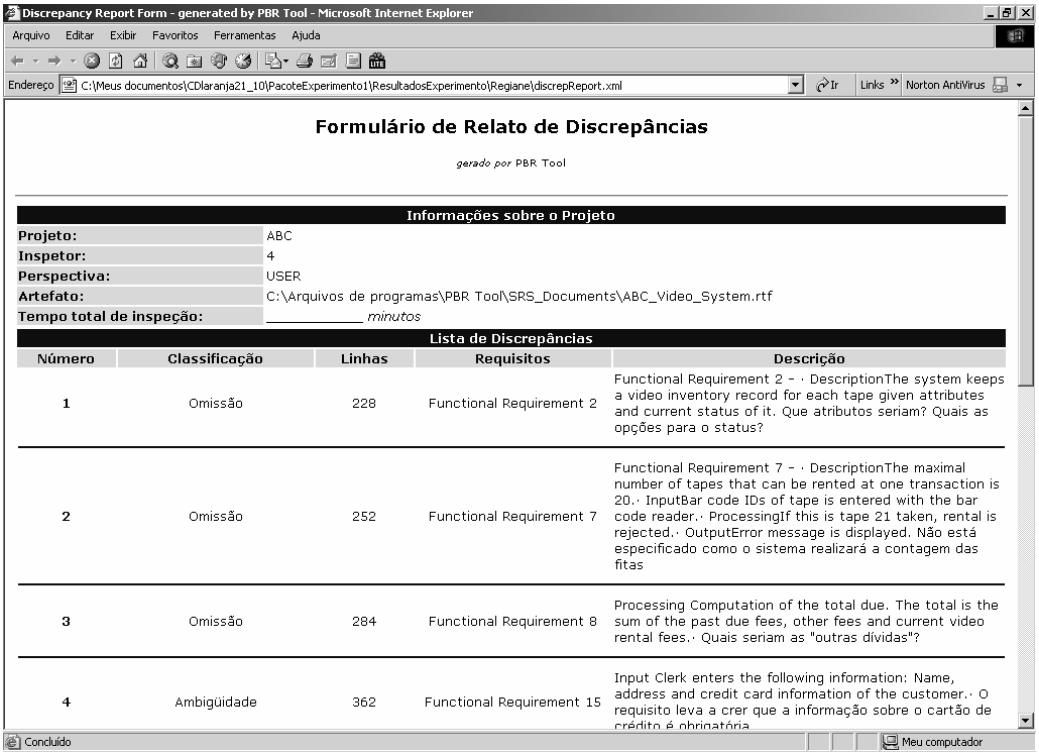


Figura 3 – Identificação de Discrepâncias em *PBR Tool*

Esperamos que esta funcionalidade possibilite que justificativas mais claras sejam dadas para uma determinada discrepância identificada. As discrepâncias identificadas por um inspetor são consolidadas num relatório, como o mostrado na figura 4 abaixo.



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

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 de 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.· ProcessingIf 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 dívidas"?
4	Ambigüidade	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

Figura 4 – Relatório de Discrepâncias

c) Fornecer uma 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.

Estas observações motivaram a possibilidade de se oferecer uma ajuda mais direcionada, com o intuito de reduzir o tempo gasto em atividades não diretamente relacionadas à identificação de defeitos. 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. Como exemplo, na figura 3 acima, este ícone permitirá ao inspetor ter acesso imediato às explicações e aos exemplos de determinado tipo de defeito, ao invés de ter que recorrer ao material de referência da técnica como um todo.

d) Verificar a conformidade da aplicação da técnica

Alguns autores acreditam que a verificação da fidelidade na aplicação da técnica possa ser feita a partir da simples análise dos produtos gerados [14]. Discordamos destes autores pelo fato de termos observado duas situações discrepantes nos diversos estudos envolvendo PBR conduzidos em nossa instituição.

Alguns inspetores se preocupam excessivamente com a elaboração de casos de uso detalhados, esquecendo-se de que o foco da aplicação da técnica é a identificação de defeitos. Existem também um outro tipo de inspetor, aquele que inicialmente busca os defeitos seguindo heurísticas próprias, ou seja, de forma *ad hoc*, e então produz os modelos de alto nível, sem a associação desta atividade com a identificação de defeitos.

Lanubile e Visagio [15], num estudo em que, comparando diferentes técnicas de identificação de defeitos, avaliaram a conformidade da aplicação de PBR através de questionários, descobrindo que apenas 20% dos indivíduos realmente seguiram a técnica de forma fiel.

A ocorrência destas situações pode invalidar quaisquer conclusões acerca de um estudo experimental, já que sua validade interna, ou seja, a relação causal entre o tratamento adotado e o resultado obtido, pode ser comprometida caso o processo, ou a técnica aplicada na prática pelos indivíduos, não seja a mesma que o processo em estudo [16].

Face às observações feitas acima, *PBR Tool* permite a verificação da conformidade da aplicação da técnica através da coleta de métricas de forma não obstrutiva.

A coleta dos dados feita pela ferramenta envolve não só as mesmas informações coletadas quando PBR é aplicada manualmente (descrição, classificação e requisitos associados às discrepâncias), como também envolve métricas relativas à aplicação da técnica em si: tempo de início e fim da inspeção, momento da identificação de uma discrepância, tarefa em execução quando uma discrepância é identificada e o tempo gasto em cada tarefa. Estas métricas, junto com as informações relativas a execução da técnica em si, são registradas em um arquivo XML. Um trecho deste arquivo, contendo as métricas relativas a uma discrepância identificada, é mostrado na figura 5 abaixo.

```
- <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 5 – Coleta de Métricas em *PBR Tool*

3.1 Estudo de Viabilidade

O estudo para avaliar a viabilidade de *PBR Tool* contou com 11 estudantes de pós-graduação em Engenharia de Software. Apenas um destes indivíduos reportou não ter experiência prévia no desenvolvimento de software. Todos eles tiveram uma única experiência anterior em inspeção de requisitos utilizando PBR na perspectiva do usuário, já que participaram de um estudo para analisar a influência de aspectos culturais na utilização da técnica.

Os indivíduos inspecionaram um documento de requisitos de 14 páginas, tratando de um sistema no domínio de vídeo locadoras. *PBR Tool* foi disponibilizada a estes indivíduos, que a instalaram em suas próprias máquinas.

A coleta de dados teve como principal objetivo verificar a viabilidade da ferramenta proposta. Entretanto, os inspetores finalizaram a inspeção num tempo médio de 147 minutos (tempo máximo de 220 minutos e mínimo de 80 minutos). Isto representou uma redução quando comparado tanto com o estudo anterior realizado como também com os estudos envolvendo PBR na nossa instituição. Como a cobertura de defeitos[1] entre as duas inspeções foi muito próxima, o custo-eficiência (nº de defeitos reais/tempo de inspeção) médio dos inspetores no estudo envolvendo a ferramenta foi superior.

Ainda que estes dados nos dêem um primeiro indício de que nossa hipótese possa ser verdadeira, é necessário avaliar os possíveis fatores de confusão: experiência anterior na aplicação da técnica e documentos de requisitos com tamanho/domínio diferentes.

4. Contextualização da Aplicação da Técnica num Processo de Desenvolvimento

Com o intuito de contextualizar o uso da ferramenta descrita neste artigo, utilizaremos o processo em cascata, proposto em [4] e representado pela figura 6 abaixo. Nela, as atividades do processo são representadas por retângulos, com as setas horizontais representando a seqüência das atividades e as circulares representando as inspeções, que devem tratar dos aspectos relacionados à qualidade dos produtos gerados durante o processo.

Ainda que não reflita a realidade da grande maioria dos processos utilizados na prática, consideramos este processo didático o suficiente para ilustrar como a ferramenta descrita neste artigo pode ser utilizada num processo de desenvolvimento real utilizando as técnicas de leitura baseadas em perspectiva para a inspeção de requisitos de software.

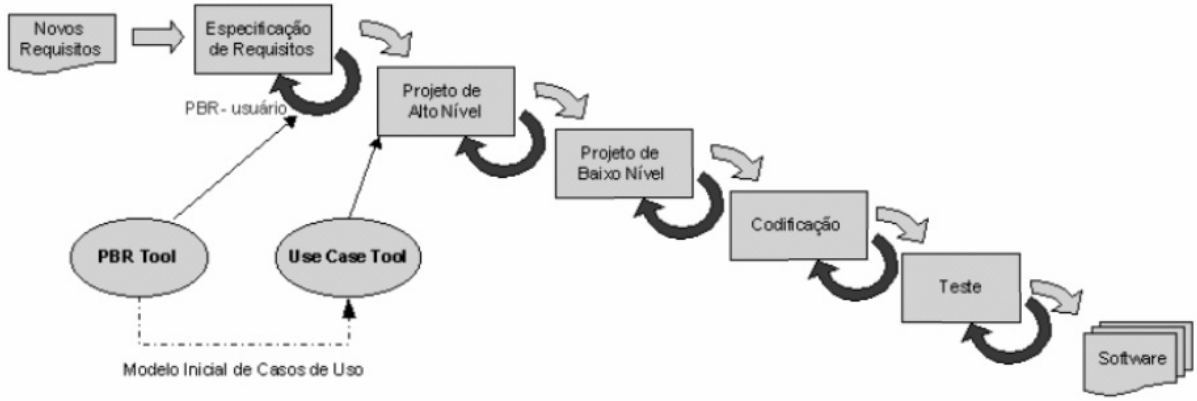


Figura 6 – Processo Didático de Desenvolvimento de Software

[1] Número total de defeitos reais identificados em relação à quantidade de defeitos existentes no documento.

A partir do momento em que os requisitos do software são identificados e documentados, podemos, ainda que forma indireta, contribuir para o aumento da qualidade do produto final ao revisarmos estas especificações. Nesta hora é que se dá a aplicação das técnicas de leitura baseadas em perspectiva, podendo contar com o apoio da ferramenta *PBR Tool*.

O maior objetivo da realização da inspeção é se chegar numa lista de defeitos que devem ser corrigidos antes do início das próximas fases de desenvolvimento. O questionamento que se faz neste momento é, se o documento de requisitos terá, após a primeira inspeção, qualidade suficiente para que o processo de desenvolvimento possa prosseguir. Fagan [8] propõe a utilização de dados históricos ao recomendar que o artefato deve ser re-inspecionado se contiver 5% a mais de defeitos que a média das inspeções passadas. Outros autores como [17] e [18] sugerem modelos para estimar a eficiência de uma eventual re-inspeção.

4.1 *Use Case Tool* - Ferramenta para Detalhamento de Casos de Uso

Independentemente da decisão por uma nova inspeção, como citado anteriormente, a aplicação de PBR pode requerer um esforço 30% maior que as demais técnicas. Entretanto, PBR propõe ainda que os modelos de alto nível esboçados durante a inspeção devam ser utilizados como ponto de partida para a criação de artefatos mais detalhados nas fases de desenvolvimento subseqüentes, diluindo grande parte desse esforço na criação de artefatos definitivos do sistema[12]. Deste modo, com a utilização da *PBR Tool* e após o término das inspeções, teríamos um modelo inicial que, através do seu detalhamento, poderia ser aproveitado para construção de um modelo definitivo dos casos de uso do sistema.

Para permitir esse aproveitamento do modelo inicial de casos de uso produzidos por *PBR Tool*, construímos uma segunda ferramenta para detalhamento de modelos de casos de uso, *Use Case Tool* [19]. Na figura 6 ilustramos tal situação de uso conjunto das duas ferramentas no processo didático proposto. Deste modo, o modelo definitivo de casos de uso pode ser construído, tornando possível dar continuidade ao processo de desenvolvimento.

Visando fornecer um apoio efetivo para o detalhamento de casos de uso, tentamos capturar as informações que pudessem realmente refletir as expectativas do usuário em relação ao produto de software e que também fossem úteis no processo de desenvolvimento, preocupação esta não observada em muitas ferramentas que se limitam apenas a representar casos de uso através do uso de diagramas. A ferramenta *Use Case Tool* permite o detalhamento dos casos de uso a partir das propostas de [20] e [21], incluindo também conceitos de severidade e freqüência de utilização dos casos de uso por parte dos atores. Combinando estes conceitos, podemos priorizar casos de teste derivados a partir dos casos de uso [22]. Pode-se ainda utilizar estas informações para determinar a ordem de implementação de requisitos.

De forma resumida, com a utilização desta ferramenta deve ser possível:

- Descrever, editar e excluir atores, funcionalidades, casos de uso e diagramas UML de casos de uso, mantendo uma correlação consistente entre descrições de casos de uso e diagramas;

- Extrair diretamente da visualização do documento de requisitos os conceitos identificados (casos de uso, atores e funcionalidades);

- Verificar a consistência entre os conceitos capturados, ou seja, verificar nos modelos construídos se há atores que não estão relacionados a nenhum caso de uso, casos de uso que não estão associados a nenhum ator, funcionalidades que não estão presentes em nenhum caso de uso, e;

- Verificar como as funcionalidades identificadas estão associadas aos atores participantes de um caso de uso.

4.2 Integração entre *PBR e Use Case Tool*

Ambas as ferramentas persistem os dados em arquivos XML. Ainda que a principal motivação para o desenvolvimento de *Use Case Tool* tenha sido permitir o detalhamento dos casos de uso esboçados em *PBR Tool*, a definição do formato dos arquivos produzidos por cada ferramenta foi propositadamente feita de forma separada. Esta decisão foi tomada para utilizar estas abordagens como um estudo de caso para o mecanismo de integração de ferramentas proposto em [23], que será o responsável por realizar as transformações estruturais e semânticas nos dados. Isto está representado na figura 7 a seguir.

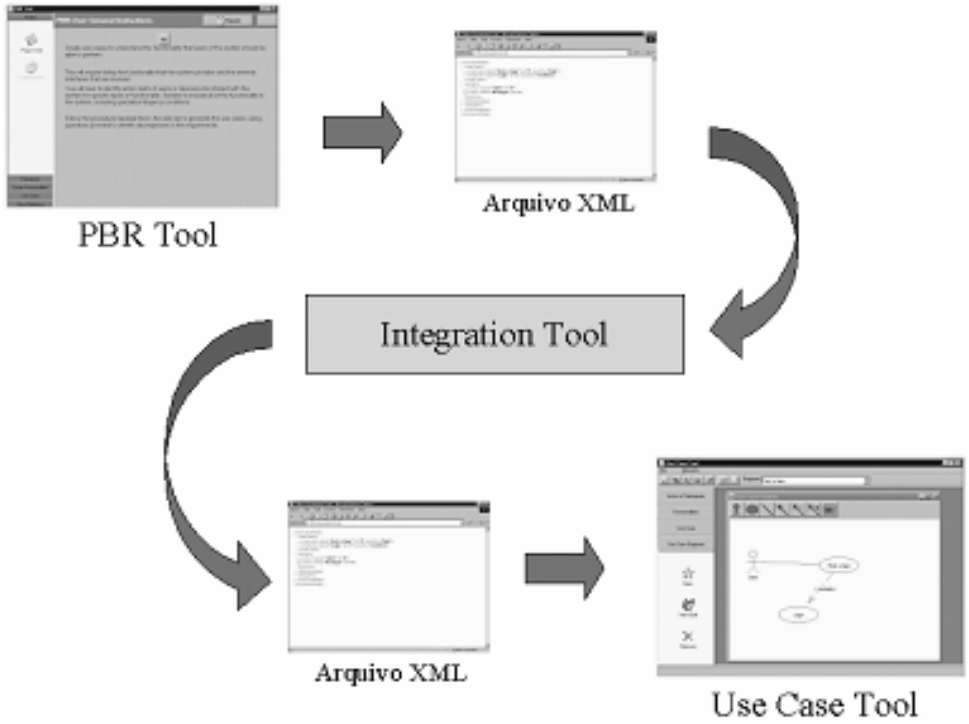


Figura 7 – Integração entre *PBR* e *Use Case Tool*

Por fim, para ilustrar tanto o resultado desta integração quanto a própria forma de detalhar casos de uso utilizada por *Use Case Tool*, apresentamos um exemplo de como o caso de uso esboçado durante a inspeção em *PBR Tool* (Figura 2) é detalhado em *Use Case Tool*.

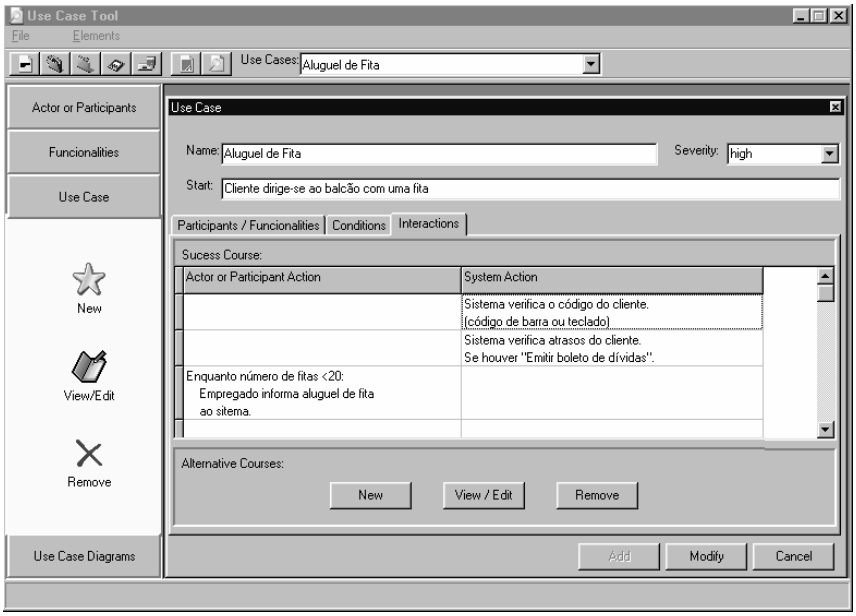


Figura 8 – Descrição de Casos de Uso em *Use Case Tool*

5. Conclusão

Neste artigo foram apresentados e discutidos alguns tópicos sobre inspeção, mais especificamente sobre a técnica de leitura PBR. Apresentamos o apoio ferramental para a aplicação da técnica de leitura PBR e para o detalhamento do modelo inicial de casos de uso obtidos com a aplicação da técnica. Ilustramos também o contexto de utilização conjunta dessas ferramentas inseridas em um processo de desenvolvimento didático.

Estas ferramentas estão disponíveis no contexto de uma infra-estrutura de suporte a inspeções de software [25], sendo integradas através da troca de dados em XML, seguindo a abordagem proposta em [23].

Em um futuro próximo, as duas ferramentas apresentadas neste artigo estarão integradas à estação TABA [13], um meta-ambiente capaz de gerar ambientes de desenvolvimento de software adaptados aos diferentes tipos de processos de desenvolvimento.

Os resultados experimentais do estudo realizado para verificar a viabilidade de *PBR Tool* [24] indicam ser interessante estender a ferramenta para as perspectivas do testador e do projetista. Para esta última, haverá a necessidade de estender a própria técnica, para considerar os artefatos da orientação a objetos.

Todas estas iniciativas têm como principal propósito reforçar a importância das atividades de Verificação e Validação para a qualidade do software. Sabemos que a transferência de tecnologia para a indústria não é uma tarefa trivial, sendo influenciada por diversos fatores como restrições de esforço, orçamento e a própria utilidade prática [26]. Em razão disso, todas estas propostas devem estar sempre, fortemente, baseadas em experimentação.

6. Agradecimentos

Os autores deste trabalho gostariam de agradecer ao CNPq e ao projeto NFS-CNPq Readers pelo apoio financeiro e à Equipe de Engenharia de Software Experimental da COPPE/UFRJ (http://www.cos.ufrj.br/~ese) pelas sugestões significativas. Agradecemos também aos alunos participantes dos estudos.

Referências

[1] BLECHER, N., 2004, “Máquina de Vender”, Revista Exame, Edição 811, ano 38 nº3, p.44-54.

[2] MALDONADO, J.C., FABBRI, S.C.P.F., 2001, “Verificação e Validação de Software”, Capítulo 3, Seção 3.4, Qualidade de Software – Teoria e Prática, Prentice Hall.

[3] LAITENBERGER, O., DEBAUD, J.M., 2000, "An Encompassing Life Cycle Survey of Software Inspections”, The Journal of Systems and Software, v50, p.5-31

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

[5] YOUNESSI, H., 2002, "Object-Oriented Defect Management of Software”, Upper Saddle River, NJ, Prentice Hall.

[6] BOEHM, B. & BASILI, V., 2001, “Software Defect Reduction Top 10 List”, Janeiro, IEEE Software, pp. 135-137.

[7] CHENG, B., JEFFERY, R., 1996, “Comparing Inspection Strategies for Software Requirement Specifications”, Proceedings of 1996 Australian Conference on Software Engineering, pp. 203-211, July.

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

[9] VOTTA JR, L.G., 1993, “Does Every Inspection Need a Meeting?”, ACM Software Engineering Notes, v.18, n.5, pp107-114.

[10] LAITENBERGER, O., DEBAUD, J.M., 1997, "Perspective-Based Reading of Code Documents at Robert Bosch GmbH”, Information and Software Technology, Volume 39, pp781-791.

[11] 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.

[12] SHULL, F., RUS, I., BASILI, V., 2000, “How Perspective-Based Reading Can Improve Requirements Inspections”, IEEE Computer, Volume 33, Issue 7, pp73-79, July.

[13] VILLELA, K.; Travassos, G.H.; ROCHA, A.R. “Ambientes de Desenvolvimento de Software Orientados a Organização”, IDEAS'2001 - Workshop Ibero-americano de Ingeniería de Requisitos y Ambientes de Software; Jan Jose, Costa Rica, abril de 2001.

[14] LAITENBERGER, O., ATKINSON, C., SCHLICH, M., EL EMAM, K., 2000, "An Experimental Comparison of Reading Techniques for Defect Detection in UML Design Documents”, The Journal of Systems and Software, v53, Issue 2, p.183-204

[15] LANUBILE, F., VISAGIO, G., 2000, "Evaluating Defect Detection Techniques for Software Requirements Inspection”, ISERN Report n. 00-08.

[16] 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.

[17] BIFFL, S., GUTJAHR, W., 2002, "Using a Reliability Growth Model to Control Software Inspection", Empirical Software Engineering: An international journal; vol.7, pp. 257-284, Kluwer Academic Publishers.

[18] ADAMS, T., 1999, “A formula for the re-inspection decision”, Software Engineering Notes 24(3): 80.

[19] 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.

[20] ANDERSSON, M., & BERGSTRAND, J., 1997, *“*Formalizing Use Cases with Message Sequence Charts”, Master thesis, Department of Communication Systems at Lund Institute of Technology.

[21] GELPERIN, D.,2003,*“*Precise Use Cases*”*, LivesSpecs Software (http://livespecs.com).

[22] MCGREGOR,J., MAJOR, M.L., 2000, *“*Selecting Test Cases Based on User's Priorities*”*, Software Development Magazine (http://www.sd.magazine.com).

[23] SPINOLA, R. O., TRAVASSOS, G.H., 2003, “Uma Abordagem para Integração de Ferramentas”, VIII Workshop de Teses em Engenharia de Software – SBES 2003, vol. 1, pp.59-64, Manaus-AM.

[24] SILVA, L.F.S., TRAVASSOS, G.H., 2004, “Tool-Supported Unobtrusive Evaluation of Software Engineering Process Conformance”, Submetido para International Symposium on Empirical Software Engineering, ISESE 04.

[25] KALINOWSKI, M., SPINOLA, R. O., TRAVASSOS, G.H., 2004, “Infra-Estrutura Computacional para Apoio ao Processo de Inspeção de Software”, Simpósio Brasileiro de Qualidade de Software 2004.

[26] SHULL, F., CARVER, J., TRAVASSOS, G.H., 2001, "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), Vienna, Austria, Sept. 10-14,2001. p. 288-296.