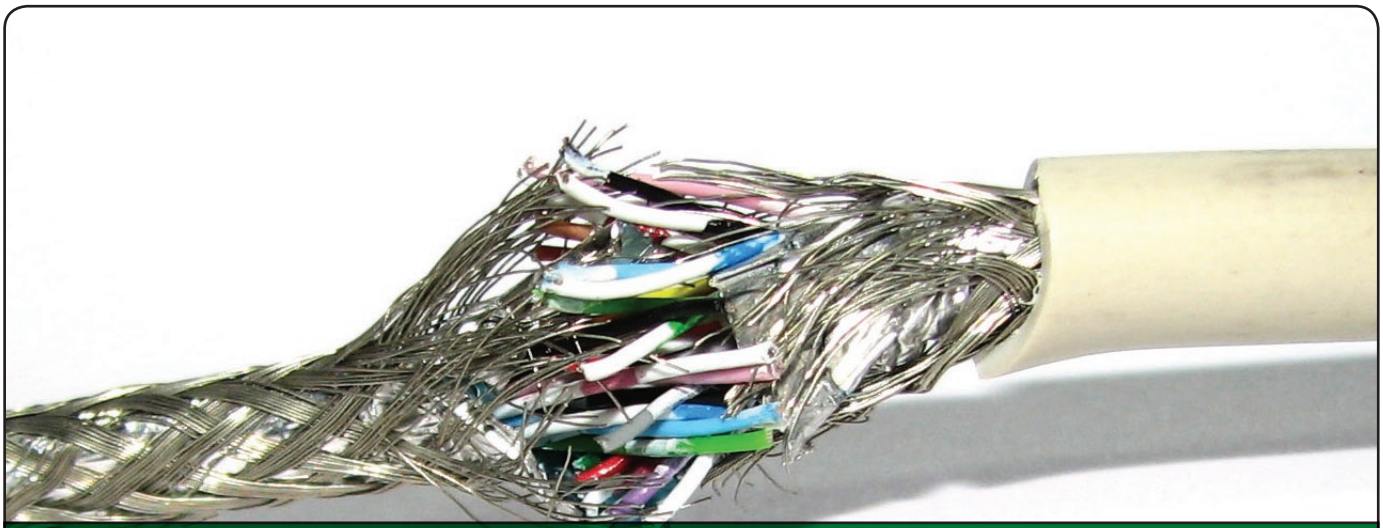

Introdução à Inspeção de Software

Aumento da qualidade através de verificações intermediárias

Marcos Kalinowski
mk@kalisoftware.com
Doutorando em Engenharia de Sistemas e Computação (COPPE/UFRJ). Mestre em Engenharia de Software (COPPE/UFRJ, 2004). Bacharel em Ciências da Computação (UFRJ, 2001). Diretor executivo e instrutor da Kali Software (www.kalisoftware.com), empresa de consultoria e treinamento em Engenharia de Software. Professor e pesquisador do curso de Ciência da Computação do Centro Universitário Metodista Bennett, ministrando disciplinas de Engenharia de Software. Consultor para implementação e avaliação do MPS.BR tendo colaborado na elaboração do guia geral e do guia de implementação deste modelo.

Rodrigo Oliveira Spínola
rodrigo@sqlmagazine.com.br
Doutorando em Engenharia de Sistemas e Computação (COPPE/UFRJ). Mestre em Engenharia de Software (COPPE/UFRJ, 2004). Bacharel em Ciências da Computação (UNIFACS, 2001). Diretor de operações e instrutor da Kali Software (www.kalisoftware.com), tendo ministrado cursos na área de Qualidade de Produtos e Processos de Software, Requisitos e Desenvolvimento Orientado a Objetos. Consultor para implementação do MPS.BR. Atua como Gerente de Projeto em projetos de consultoria na COPPE/UFRJ. É Editor Chefe da Engenharia de Software Magazine.

Na engenharia de software, assim como em outras disciplinas de engenharia, é necessário considerar variáveis como esforço, produtividade, tempo e custo de desenvolvimento. Essas variáveis são afetadas negativamente quando artefatos defeituosos são produzidos, devido ao retrabalho para corrigir defeitos. Sabe-se, ainda, que o custo do retrabalho para correção de defeitos aumenta na medida em que o processo de desenvolvimento progride. Desta forma, iniciativas devem ser realizadas no sentido de encontrar e corrigir defeitos tão logo sejam introduzidos. Uma abordagem que tem se mostrado eficiente e de baixo custo para encontrar defeitos, reduzindo o retrabalho e melhorando a qualidade dos produtos é a revisão dos artefatos produzidos ao longo do processo de desenvolvimento de software.

Inspeção de software é um tipo particular de revisão que pode ser aplicado a todos os artefatos de software e possui um processo de detecção de defeitos rigoroso e bem definido. A **Figura 1** ilustra a possibilidade de se realizar inspeções nos diferentes artefatos de software.

De forma resumida, o processo tradicional de inspeção envolve o planejamento da inspeção, indivíduos revisando um determinado artefato, um encontro em equipe para discutir e registrar os defeitos, a passagem dos defeitos para o autor do artefato para que possam ser corrigidos e uma avaliação final sobre a necessidade de uma nova inspeção.

A importância de inspeções na redução do retrabalho e na garantia da qualidade de software está bem documentada na literatura e é discutida em maiores neste artigo. A seção seguinte apresenta a definição de alguns conceitos utilizados ao longo deste trabalho. Veremos ainda neste artigo alguns benefícios de se realizar inspeções em artefatos produzidos ao longo do processo de desenvolvimento de software são descritos; conceitos sobre técnicas de leitura para detecção de defeitos em artefatos de software;

o processo de inspeção de software e suas características, e; o estado atual da utilização de inspeções na prática e do suporte ferramental existente.

Definição dos Conceitos

O termo defeito muitas vezes é utilizado de forma genérica. No entanto, é importante ter em mente que sua interpretação dependerá do contexto em que ele for utilizado. Defeitos encontrados através de revisões estarão relacionados às faltas no artefato sendo revisado. Quando um defeito se manifesta através de atividades de teste, por sua vez, estaremos lidando com uma falha no software. Estas definições seguem a terminologia padrão para Engenharia de Software do IEEE (IEEE 610.12, 1990):

- **Erro:** É um defeito cometido por um indivíduo ao tentar entender uma determinada informação, resolver um problema ou utilizar um método ou uma ferramenta.
- **Defeito (ou Falta):** É uma manifestação concreta de um erro num artefato de software. Um erro pode resultar em diversos defeitos.
- **Falha:** É o comportamento operacional do software diferente do esperado pelo usuário. Uma falha pode ter sido causada por diversas faltas e algumas faltas podem nunca causar uma falha.

Em alguns momentos o termo discrepância será utilizado. Este termo refere-se a um suposto defeito encontrado. No nosso contexto, uma discrepância poderá ser considerada um defeito de fato ou um chamado falso positivo.

Para obter uma classificação para os defeitos encontrados nas revisões (as faltas), partimos do fato de que todos os artefatos gerados durante o desenvolvimento de software utilizam como base o documento de requisitos ou artefatos gerados a partir deste. Desta forma, as classes de defeito seriam os tipos de defeito presentes em documentos de requisitos acrescidos dos tipos de defeitos introduzidos pela transformação de artefatos ao longo do desenvolvimento de software.

Um padrão IEEE (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 que a falta de qualquer um destes atributos constituiria um tipo de defeito. Assim, a seguinte taxonomia foi definida:

- **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 solicitas 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.

É importante ressaltar que estas classes genéricas de defeitos podem ser divididas em classes mais específicas, dependendo da necessidade. Além disto, esta classificação não pretende ser definitiva e cada organização pode acrescentar mais tipos de defeito de acordo com suas necessidades.

Benefícios da Aplicação de Inspeções de Software

Esforço, Produtividade, Tempo e Custo

O esforço gasto por organizações de software com retrabalho pode variar em média entre 40% e 50% do esforço total do desenvolvimento de um projeto. Uma estimativa da distribuição do retrabalho pelas atividades de desenvolvimento de software está ilustrada na **Figura 2**.

Analisando a **Figura 2**, é possível verificar que o retrabalho tende a aumentar na medida em que o desenvolvimento progride. Uma das razões para isto é o aumento no esforço para corrigir defeitos nas atividades finais do processo de desenvolvimento. Através da análise de 63 projetos, BOEHM (1981) apresenta o custo relativo da correção de defeitos encontrados em cada uma das atividades de desenvolvimento (**Figura 3**).

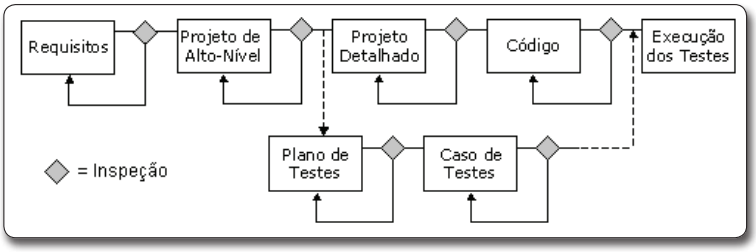

Figura 1. Inspeções de Software nos Diferentes Artefatos.

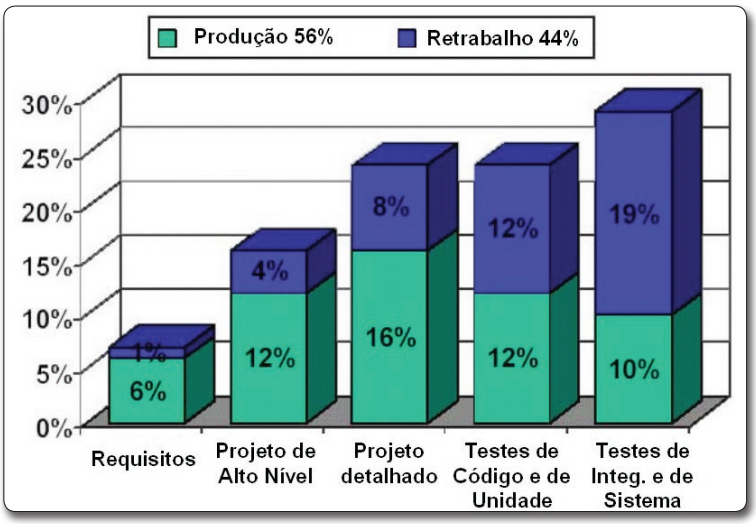

Figura 2. Distribuição do retrabalho pelas atividades de desenvolvimento de software. Adaptado de (WHEELER et al., 1996).

Assim, um dos maiores benefícios de se utilizar inspeções de software é a detecção de defeitos nas fases iniciais do processo de desenvolvimento de software, facilitando a correção destes defeitos com menor esforço e custo. Desta forma, de acordo com (JONES, 1991), o esforço com retrabalho é reduzido em média para 10% a 20% do esforço total de desenvolvimento. Esta redução no retrabalho pode implicar em melhorias significativas para a produtividade de software. De acordo com (BOEHM et al., 2000) a maior redução de esforço é gerada pela melhoria de (1) maturidade de processos de software, (2) arquiteturas de software e (3) gerência de riscos é proveniente da redução do retrabalho. Resultados experimentais mostram como este benefício pode afetar as variáveis esforço, produtividade, tempo e custo, mencionadas na seção anterior:

- Esforço. O departamento de desenvolvimento da Ericsson em Oslo, Noruega, calculou uma redução bruta do esforço total de desenvolvimento em 20% aplicando inspeções. Além disto, resultados de estudos mostram que a introdução de inspeções de projeto pode reduzir o esforço com retrabalho em 44%.
- Produtividade. De acordo com (GILB e GRAHAM, 1993), inspeções aumentam a produtividade de 30% a 50%;
- Tempo. De acordo com (GILB e GRAHAM, 1993), inspeções reduzem o tempo de desenvolvimento de 10% a 30%;
- Custo. Resultados de estudos mostram que a introdução de inspeções de código pode reduzir os custos de implementação de projetos em 39%.

Uma estimativa de (WHEELER et al., 1996) sobre o custo de desenvolvimento quando inspeções são aplicadas, quando comparado ao desenvolvimento sem utilizar inspeções, se encontra na **Figura 4**. Esta estimativa representa bem os benefícios apresentados nesta seção. É possível observar que utilizando inspeções se obtém um aumento na produtividade, já que projetos são concluídos em menos tempo e envolvem menos gastos.

Qualidade de Software

De acordo com Pressman (2001), qualidade de software é a conformidade a: (1) requisitos funcionais e não funcionais que têm sido explicitamente declarados, (2) padrões de desenvolvimento que tenham sido claramente documentados e (3) características implicitamente esperadas de todo software a ser desenvolvido. De forma resumida, qualidade consiste de um conjunto de requisitos e de um produto ou serviço que esteja em conformidade com estes requisitos e, por esta razão, atenda completamente às necessidades dos clientes. Entre as atividades que podem ser utilizadas para verificar a qualidade de software se encontram:

- Revisões de software;
- Testes;
- Padrões e procedimentos formais;
- Controle de mudanças;
- Métricas de software;
- Procedimentos para coleção e disseminação de informações.

A importância das revisões na garantia da qualidade é destacada pelo modelo CMMI, que exige a realização de revisões como uma prática específica do processo de verificação. Sabe-se ainda que inspeções de software, em particular, capturam em torno de 60% dos defeitos de artefatos (BOEHM e BASILI, 2001) o que deixa explícita a sua contribuição para a melhoria da qualidade.

Uma outra maneira de detectar defeitos em artefatos é através da aplicação de testes. No entanto, a aplicação de testes descobre apenas sintomas de problemas e, desta forma, pode ocasionar um trabalho refinado e custoso para detectar o defeito que causou o sintoma. BOEHM e BASILI (2001) ressaltam ainda que inspeções e testes capturam diferentes tipos de defeito e em diferentes momentos do processo de desenvolvimento de software.

Portanto, é interessante aplicar tanto inspeções quanto testes para detectar defeitos em artefatos de software.

Além disto, precedendo os testes com as inspeções, defeitos podem ser removidos nas fases iniciais do processo de desenvolvimento e os desenvolvedores terão uma visão mais ampla da complexidade do sistema. Esta visão os deixa mais bem preparados para o momento em que se confrontam com esta complexidade e com os defeitos que podem possivelmente surgir durante os testes.

Outros Benefícios

A aplicação de inspeções de forma bem planejada pode trazer diversos outros benefícios:

- Aprendizado. Inspetores experientes podem tentar detectar padrões de como os defeitos ocorrem e definir diretrizes que ajudem na detecção destes. Além disto, bons padrões de desenvolvimento podem ser observados durante a inspeção, sendo possível sua descrição como melhores práticas para a organização.
- Integração entre processos de detecção e de prevenção de defeitos. Saber onde e quando os defeitos ocorrem pode ajudar a estabelecer planos de contingência que

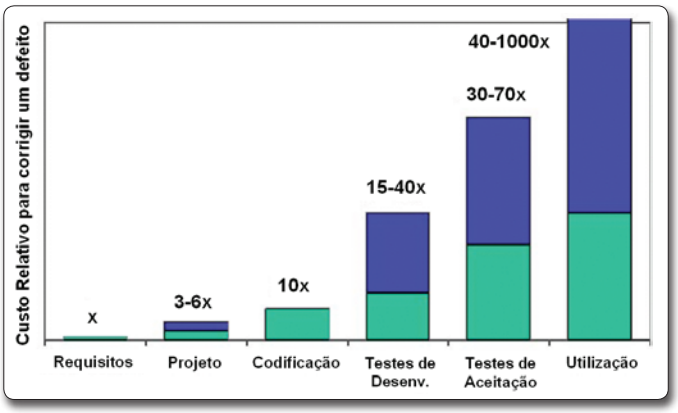

Figura 3. Custo relativo para corrigir um defeito. Adaptado de (BOEHM, 1981).

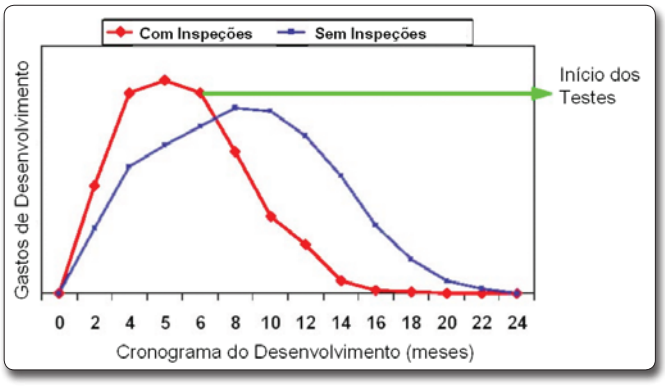

Figura 4. Estimativa dos gastos de desenvolvimento utilizando e não utilizando inspeções. Adaptado de (WHEELER et al., 1996).

evitem a sua ocorrência.

• Produtos mais inteligíveis. Os autores dos diversos artefatos, sabendo que estes serão inspecionados, passarão a produzir artefatos de uma forma que sua compreensão seja facilitada. A produção de artefatos mais inteligíveis, além de facilitar a inspeção, trará benefícios para as fases seguintes do processo de desenvolvimento, incluindo principalmente a fase de manutenção.

• Dados defeituosos ajudam a melhorar o processo de software do projeto. Analisando a causa dos defeitos encontrados é possível fazer ajustes no processo para evitar a ocorrência de defeitos deste mesmo tipo.

O Processo de Inspeção de Software

FAGAN (1976) desenvolveu o processo tradicional de inspeção de software, uma forma detalhada de se realizar uma revisão. Neste processo, existem seis atividades principais:

• Planejamento. Um usuário, desempenhando o papel de moderador da inspeção, define o contexto da inspeção (descrição da inspeção, técnica a ser utilizada na detecção de defeitos, documento a ser inspecionado, autor do documento, entre outros), seleciona os inspetores e distribui o material a ser inspecionado.

• Apresentação. Os autores dos artefatos a serem inspecionados apresentam as características destes. Esta fase pode ser omitida se os inspetores possuem conhecimento sobre o projeto e os artefatos que devem ser inspecionados.

• Preparação. Os inspetores estudam os artefatos individualmente, e eventualmente fazem anotações sobre estes produzindo uma lista de discrepâncias. O fornecimento de técnicas de leitura pode facilitar a execução desta tarefa.

• Reunião. Uma reunião em equipe ocorre, envolvendo o moderador, os inspetores e os autores do documento. Discrepâncias são discutidas, e classificadas como defeito ou falso positivos. A decisão final sobre a classificação de uma discrepância sendo discutida é do moderador. A solução dos defeitos não é discutida durante a reunião, que não deve exceder duas horas, uma vez que após este tempo a concentração e a capacidade de análise dos inspetores costuma reduzir drasticamente. No caso em que uma reunião precisar de mais de duas horas, é sugerido que o trabalho de inspeção continue no próximo dia.

• Retrabalho. O autor corrige os defeitos encontrados pelos inspetores e confirmados pelo moderador.

• Continuação. O material corrigido pelos autores é repassado para o moderador, que faz uma análise da inspeção como um todo e re-avalia a qualidade do artefato inspecionado. Ele tem a liberdade de decidir se uma nova inspeção deve ocorrer ou não.

A forma como estas atividades se relacionam no processo de inspeção está ilustrada na **Figura 5**. Note que a atividade de apresentação, opcional, não está representada na figura.

Entre as características deste processo, temos que ele pode ser aplicado a todos os artefatos produzidos ao longo do processo de desenvolvimento, permitindo a utilização de técnicas de leitura de artefatos específicos na atividade de preparação individual. Além disto, ele possui uma estrutura rígida, com aspectos colaborativos, onde papéis, atividades e os relacionamentos entre atividades estão bem definidos.

A Reorganização do Processo de Inspeção

Baseados em diversos estudos experimentais sobre inspeções de software SAUER et al., (2000) propuseram uma reorganização do processo tradicional de inspeção. Essa reorganização visa a adequação do processo tradicional a inspeções com reuniões assíncronas e equipes geograficamente distribuídas. Assim, mudanças para reduzir o custo e o tempo total para a realização deste tipo de inspeção foram introduzidas. Este projeto alternativo para inspeções de software mantém a estrutura rígida e os aspectos colaborativos do processo tradicional e consiste basicamente em substituir as atividades de preparação e de reunião do processo tradicional por três novas atividades seqüenciais: detecção de defeitos, coleção de defeitos e discriminação de defeitos. Estas atividades podem ser descritas da seguinte forma:

• Detecção de Defeitos. Cada um dos inspetores selecionados pelo moderador no planejamento realizará uma atividade de detecção de defeitos. A principal tarefa desta atividade consiste em buscar defeitos no documento a ser inspecionado e assim produzir uma lista de discrepâncias.

• Coleção de Defeitos. O moderador agrupa as listas de discrepâncias dos inspetores. Esta atividade envolve eliminar discrepâncias repetidas (encontradas por mais de um inspetor), mantendo só um registro para cada discrepância.

• Discriminação de Defeitos. O modera-

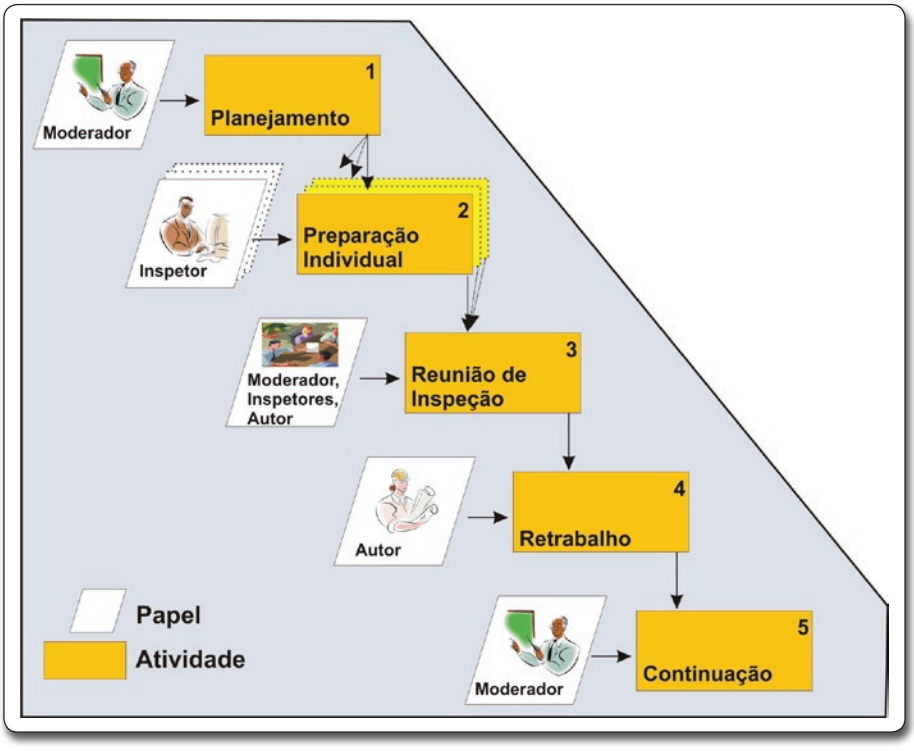

Figura 5. Processo de inspeção de software, conforme definido em (FAGAN, 1976).

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

A forma como as atividades se relacionam na reorganização do processo de inspeção de SAUER et al. (2000) está ilustrada na **Figura 6**.

Este processo permite a utilização de um número grande de inspetores em paralelo para a detecção de defeitos e, assim, aumentar a probabilidade de se encontrar defeitos difíceis de serem encontrados. Um grande número de inspetores tem um efeito no custo, mas não no tempo de detecção de defeitos e não implicará em problemas de coordenação, afinal discrepâncias encontradas por mais de um inspetor são encaminhadas diretamente para a atividade de retrabalho e as discrepâncias encontradas por apenas um inspetor não precisam ser discutidas necessariamente por todos os inspetores.

Estado Atual: Prática de Revisões em Organizações de Software e Suporte Ferramental Existente

Ao longo dos anos, muito conhecimento tem sido produzido na área de inspeções de software. Incluindo variantes do processo tradicional de inspeção, técnicas de estimativa do número de defeitos de documentos e da cobertura de inspeções, técnicas de leitura de documentos visando aumentar o número de defeitos encontrados por inspetores, diretrizes para pontos de tomada de decisão do processo de inspeção, dentre outras. Parte deste conhecimento tem sido avaliado em estudos experimentais e se mostrado adequado.

No entanto, muito deste conhecimento não tem sido utilizado na prática por organizações de software ao realizarem suas inspeções, e é negligenciado na maioria das propostas de apoio ferramental ao processo de inspeção de software.

Esta seção apresenta o estado da prática de revisões de software e o ferramental de apoio atualmente existente.

Prática de Revisões em Organizações de Software

Em um artigo comparando as diversas abordagens sobre inspeções de software encontradas na literatura, LAITENBERGER e DEBAUD (2000) mostraram que, conforme representado na **Figura 7**, inspeções em código são as mais comuns. No entanto, sabe-se que os benefícios de inspeções são maiores para os artefatos produzidos no início do ciclo de desenvolvimento.

Um survey foi realizado em 2002 visando avaliar o estado da prática de revisões de software (CIOLKOWSKI et al., 2003). De acordo com seus resultados, embora muitas organizações de software realizem revisões, a forma como as revisões são realizadas ainda é pouco sistematizada, pouco conhecimento da área de inspeções de software é utilizado e assim o verdadeiro potencial das revisões raramente é explorado. Este argumento é baseado em três observações sobre as organizações participantes do survey:

- (1) Revisões raramente cobrem sistematicamente as fases de desenvolvimento de software. Cerca de 40% das organizações responderam realizar inspeções em requisitos e 30% inspeções em código.
- (2) Muitas vezes a atividade de detecção de defeitos não é realizada sistematica-

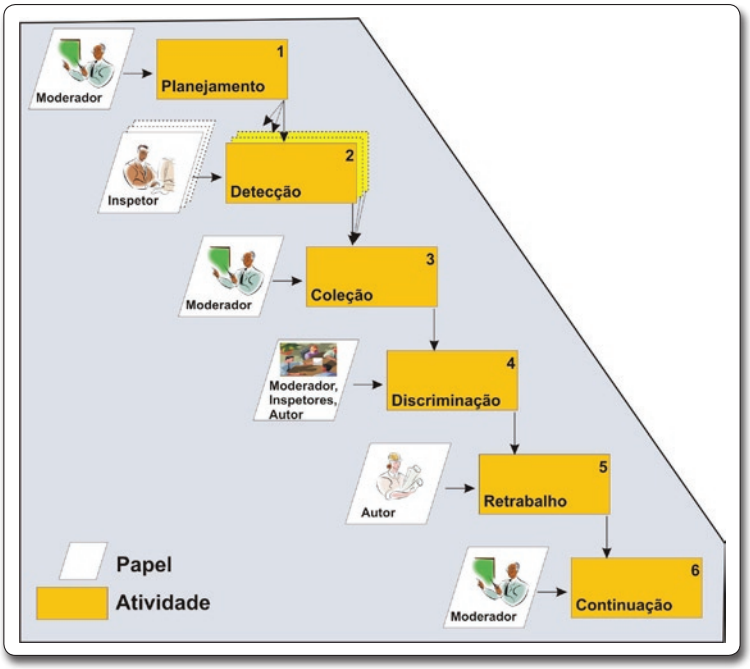

Figura 6. Reorganização do processo de inspeção, adaptada de (SAUER et al., 2000).

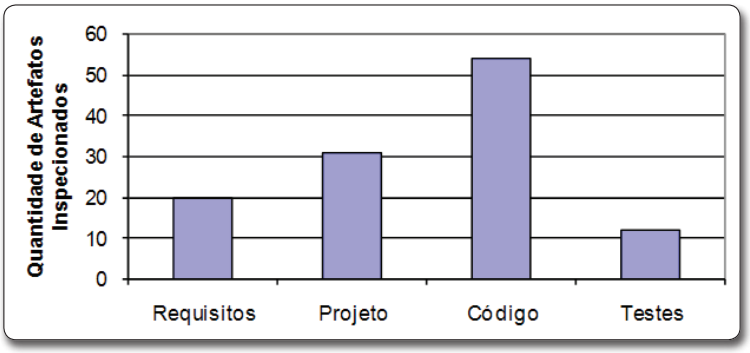

Figura 7. Distribuição do Uso de Inspeção nos Diferentes Artefatos. Adaptado de (LAITENBERGER e DEBAUD, 2000).

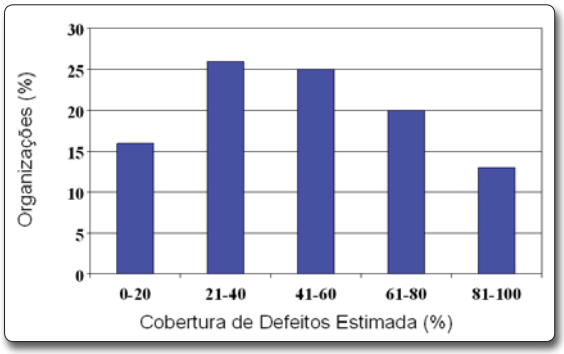

Figura 8. Cobertura de defeitos estimada para as organizações participantes do survey. Adaptado de (CIOLKOWSKI et al., 2003).

mente. Cerca de 60% das organizações responderam não conduzir uma atividade de detecção de defeitos regularmente.

• (3) Organizações raramente utilizam revisões de software no contexto de um programa sistemático de avaliação e melhoria do processo. Mais da metade das empresas participantes do survey respondeu não coletar dados (40%) ou coletar dados e não os utilizar para realizar uma análise (18%).

A utilização pouco sistematizada de revisões na prática tem sido um fator de confusão entre os resultados esperados e os obtidos através de sua aplicação. A **Figura 8** ilustra a cobertura de defeitos estimada das revisões realizadas pelas organizações participantes do survey.

Visando apoiar a realização de revisões na prática de forma mais sistemática foram elaboradas propostas de apoio ferramental. Algumas destas propostas se encontram descritas na sessão seguinte.

Ferramentas de Apoio ao Processo de Inspeção

Entre as ferramentas que apóiam a coordenação e as diversas atividades do processo de inspeção que tem sido efetivamente avaliadas e utilizadas recentemente na indústria de software destacamos as seguintes: GRIP (GRoupware supported Inspection Process), IBIS (Internet Based Inspection System) e ISPIS (Infra-estrutura de Suporte ao Processo de Inspeção de Software). Esta última sendo tecnologia nacional desenvolvida na COPPE/UFRJ, em processo de registro junto ao INPI. Uma descrição destas ferramentas se encontra a seguir:

• GRIP (Grünbacher et al., 2003). Nasceu da iniciativa de se adaptar um sistema COTS (Comercial Off The Shelf) de apoio a trabalho colaborativo (GSS) (Groupware Support System) para realizar inspeções em requisitos de software. Assim, GRIP fornece um framework e ferramentas colaborativas para a realização de inspeções assíncronas com equipes geograficamente distribuídas. GRIP fornece suporte a um processo próprio de inspeção que envolve quatro atividades: planejamento, inspeção individual, reunião assíncrona, e avaliação da inspeção. A ferramenta não fornece apoio a técnicas específicas de detecção de defeitos, apenas ad-hoc. Uma taxonomia é fornecida junto com o artefato a ser inspecionado. Os elementos desta taxonomia representam, de forma hierárquica, os tópicos abordados no artefato a ser inspecionado. As discrepâncias dos inspetores são então agrupadas de acordo com os elementos da taxonomia. Assim, uma discrepância na descrição do caso de uso Mantendo Usuário poderia ser adicionada utilizando a seguinte navegação pela taxonomia: Requisitos Funcionais > Casos de Uso > Mantendo Usuário > Descrição. Os artefatos em si são lidos na ferramenta mais conveniente ao inspetor. Desta forma, GRIP é capaz de lidar com diferentes tipos de artefato, bastando que uma taxonomia para o artefato seja criada descrevendo sua estrutura. Na reunião GRIP fornece apoio para a classificação das discrepâncias que de acordo com um estudo experimental, mostrou reduzir o esforço da reunião e apoiar a classificação correta de discrepâncias;

• IBIS (LANUBILE et al., 2003). Utiliza a web em conjunto com notificações por e-mail para apoiar inspeções assíncronas com equipes geograficamente distribuídas. IBIS visa explicitamente apoiar a reorganização do processo de inspeção proposta em SAUER et al. (2000) e permitir a sua realização de forma sistematizada. IBIS não limita o tipo de artefato a ser inspecionado e na detecção de defeitos atualmente permite apenas a utilização das técnicas ad-hoc e de checklists. Na reunião de inspeção desta proposta as discussões ocorrem através de um fórum de discussão, onde mensagens podem ser acrescentadas e o moderador pode realizar a classificação das discrepâncias como defeito ou falso positivo. IBIS não fornece apoio aos pontos de tomada de decisão do processo de inspeção, sendo as atividades de planejamento e de continuação do processo tratadas como simples cadastros, sem apoio para a realização de suas tarefas. IBIS tem sido utilizada em estudos experimentais recentes para obter conhecimento na área de inspeções de software e para avaliar aspectos da reorganização do processo de inspeção. A **Figura 9** ilustra o apoio de IBIS ao registro de discrepâncias na atividade de detecção de defeitos utilizando um checklist.

• ISPIS (KALINOWSKI e TRAVASSOS, 2004). Esta ferramenta também visa apoiar a reorganização do processo de inspeção proposta em SAUER et al. (2000). Entretanto, ISPIS foi desenvolvida utilizando uma metodologia experimental, levando em consideração conhecimento a respeito de inspeções de software selecionado criteriosamente, dando preferência a conhecimento efetivamente avaliado. ISPIS é a única ferramenta que possibilita o uso automatizado deste conhecimento sem que a equipe de inspeção precise adquiri-lo através de revisões bibliográficas e aplicá-lo manualmente. Assim, os pontos de tomada de decisão do processo são apoiados por resultados de técnicas de estimativa e dados quantitativos

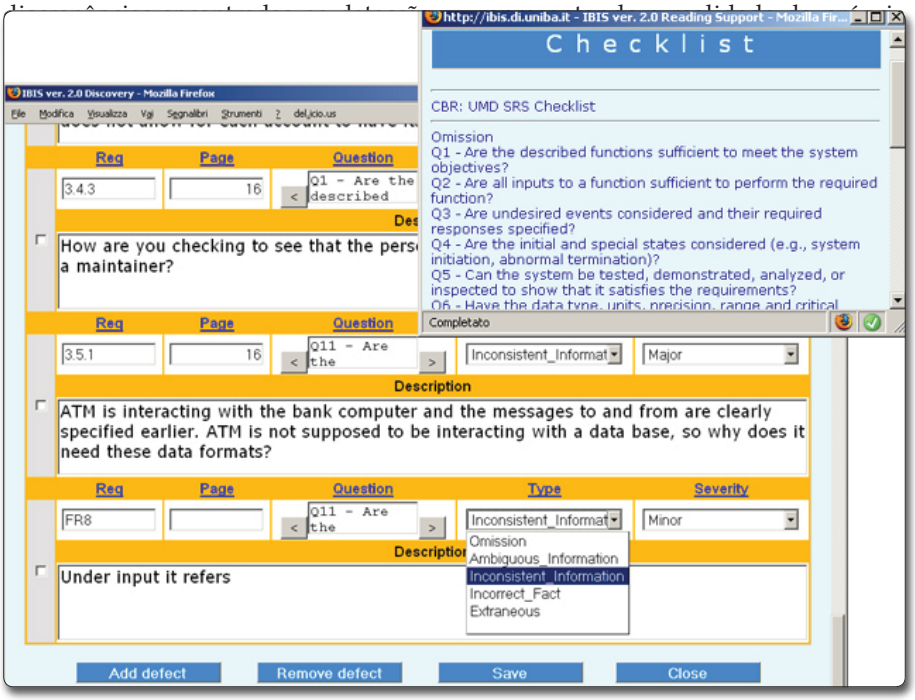

Figura 9. Apoio de IBIS ao registro de discrepâncias na atividade de detecção de defeitos.

para coordenar e apoiar a inspeção de qualquer tipo de artefato de software (documentos de requisitos, de projeto, código, etc) e, opcionalmente, faz uso de um mecanismo de integração para se comunicar com ferramentas de detecção de defeitos existentes para artefatos específicos. Outra característica da ferramenta é permitir reuniões de inspeção tanto síncronas quanto assíncronas. As inspeções com reuniões assíncronas se adequam ao contexto de desenvolvimento global de software. Assim, ISPIS pôde, por exemplo, ser utilizada com sucesso em inspeções com participantes distribuídos entre Rio de Janeiro, Nova Iorque e Londres. A ferramenta tem ainda sido utilizada para a realização de projetos da Fundação COPPETEC. Por representar o estado da arte a respeito de inspeções de software, ISPIS resultou em artigos científicos em conferências renomadas. Sua aplicabilidade na indústria permitiu ainda publicações sob forma de relatos de experiência. A **Figura 10** ilustra algumas telas do apoio de ISPIS à atividade de planejamento.

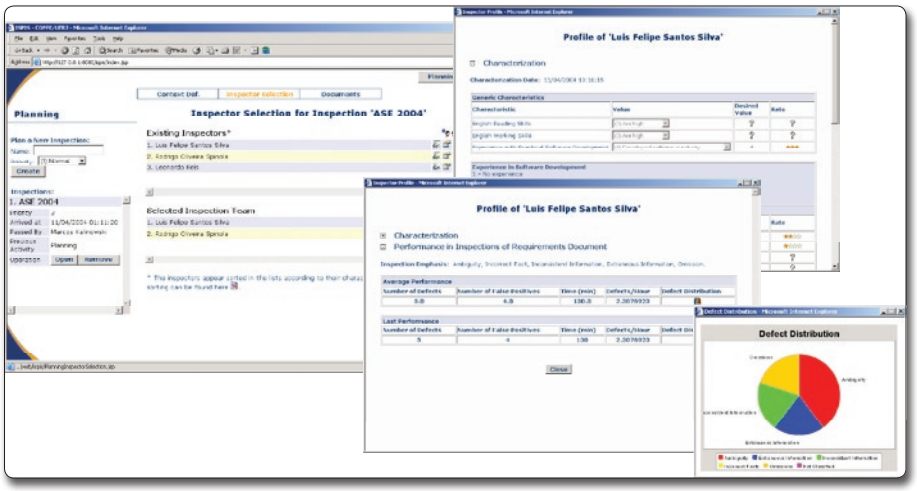

Figura 10. Telas do apoio de ISPIS ao planejamento de uma inspeção de software.

Conclusão

O objetivo de inspeções de software é melhorar a qualidade de artefatos de software através de sua análise, detectando e removendo defeitos antes que o artefato seja passado para a próxima fase do processo de desenvolvimento de software. Conforme visto neste artigo, a aplicação de inspeções entre as atividades do ciclo de vida de software pode trazer diversos benefícios para organizações de software.

O processo de inspeção foi inicialmente elaborado por FAGAN (1976). Argumentos da literatura e estudos experimentais, visando avaliar este processo, vêm indicando reuniões assíncronas para inspeções de software. Tendo em vista as inspeções com reuniões assíncronas SAUER et al., (2000), baseados em estudos experimentais, apresentam uma reorganização com mudanças para reduzir o custo e o tempo total da realização deste tipo particular de inspeção.

Atualmente muitas organizações realizam revisões, mas a forma como as revisões são realizadas ainda é pouco sistematizada e pouco conhecimento da área de inspeções de software é utilizado. Assim o verdadeiro potencial das revisões raramente é explorado, introduzindo um fator de confusão entre os resultados esperados pelas revisões e os obtidos na prática. Visando endereçar este problema e permitir a realização de inspeções mais sistemáticas algumas propostas de apoio ferramental surgiram. Entre estas propostas destacamos a infra-estrutura ISPIS, desenvolvida na COPPE/UFRJ. ●

Referências

- BOEHM, B.W., BASILI, V.R., 2001, "Software Defect Reduction Top 10 List.", IEEE Computer 34 (1): 135-137.
- BOEHM, B.W., ABTS, C., BROWN, A.W., CHULANI, S., CLARK, B.K., HOROWITZ, E., MADACHY, R., REIFER, D., STEECE, B., 2000, Software Cost Estimation with COCOMO II, Prentice Hall.
- BOEHM, B.W., 1981, Software Engineering Economics, Prentice Hall.
- CIOLKOWSKI, M., LAITENBERGER, O., BIFFL, S., 2003, "Software Reviews: The State of the Practice", IEEE Software 20 (6): 46-51.
- FAGAN, M.E., 1976, "Design and Code Inspection to Reduce Errors in Program Development", IBM Systems Journal, vol. 15, no. 3, pp. 182-211.
- GILB, T., GRAHAM, D., 1993, "Software Inspection", Addison-Wesley.
- Grünbacher, P., Halling, M., Biffl, S., "An Empirical Study on Groupware Support for Software Inspection Meetings", Proceedings of the 18th IEEE International Conference on Automated Software Engineering, 2003.
- JONES, C., 1991, "Applied Software Measurement", McGraw Hill.
- Kalinowski, M., Travassos, G. H., "A Computational Framework for Supporting Software Inspections", In: 19th IEEE International Conference on Automated Software Engineering - ASE'04, pp. 46-55, Linz, Austria, 2004.
- LAITENBERGER, O., ATKINSON, C., 1999, "Generalized Perspective Based Inspection to handle Object Oriented Development Artifacts", Proceedings of ICSE 99, p.494-503, Los Angeles, CA, USA.
- Lanubile, F., Mallardo, T., CALEFATO, F., 2003, "Tool support for Geographically Dispersed Inspection Teams", Software Process Improvement and Practice, 2003, 8: 217-231 (DOI: 10.1002/spip.184).
- PRESSMAN, R.S., 2001, Software Engineering: A Practitioner´s Approach, Fifth Edition, McGraw Hill.
- SAUER, C., JEFFERY, D.R., LAND, L., YETTON, P., 2000, "The Effectiveness of Software Development Technical Review: A Behaviorally Motivated Program of Research", IEEE Transactions on Software Engineering, 26 (1): 1-14, January.
- WHEELER, D.A., BRYKEZYNSKI, B., MEESON, R.N., 1996, Software Inspections: An Industry Best Practice, IEEE Computer Society.