UFSC
Araranguá

Inspeção de Software

Engenharia de Software II

Profa. Andréa Sabedra Bordin

Inspeção de Software

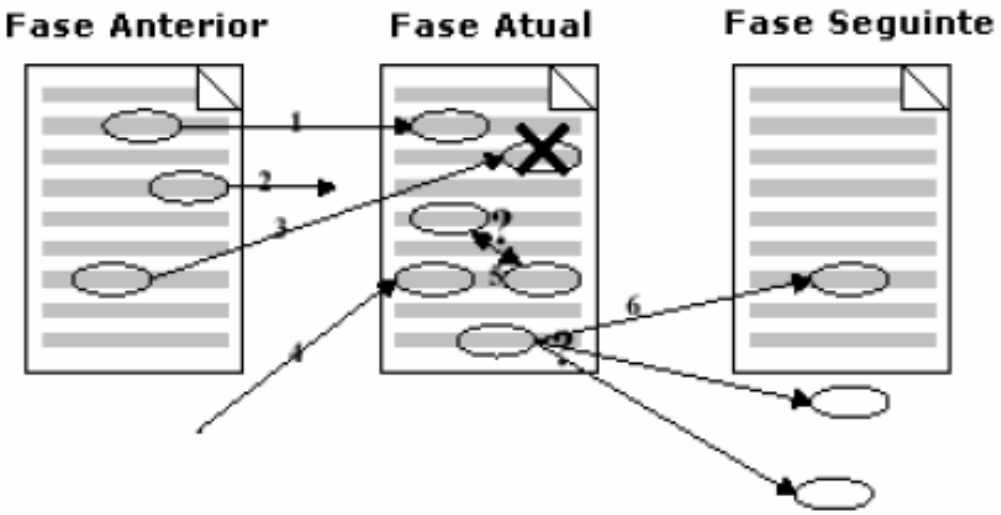
Inspeção de Software

- Todas as etapas do processo de desenvolvimento de software são suscetíveis à incorporação de **defeitos**, que podem ser detectados pela **inspeção** e posteriormente removidos.

- O custo da correção de defeitos aumenta com o decorrer do processo de desenvolvimento (Boehem e Basili, 2001).

- **Revisões de artefatos de software** tem se mostrado um **abordagem eficiente e de baixo custo** para encontrar defeitos logo após terem sido introduzidos (Ciolkowiski *et al*, 2002).

Inspeção de Software



- 1. Informação transformada corretamente.
- 2. Informação perdida durante a transformação.
- 3. Informação transformada incorretamente.
- 4. Informação estranha introduzida.
- 5. Ocorrência de múltiplas transformações inconsistentes a partir de uma mesma fonte de informação.
- 6. Possibilidade de múltiplas transformações inconsistentes a partir de uma mesma fonte de informação.

Inspeção de Software

- Pode ser aplicada a qualquer representação legível* do software.
  - Por exemplo: Revisão cuidadosa, linha por linha, do **código fonte** do programa.
- O objetivo **não é corrigir problemas** e sim encontrá-los para que o desenvolvedor corrija depois.
- Identificam entre 60 e 90% dos erros de programa.

- O **objetivo da inspeção** é aumentar a **qualidade** dos artefatos produzidos ao longo do processo de desenvolvimento.

*código ou documentação

Inspeção de Software

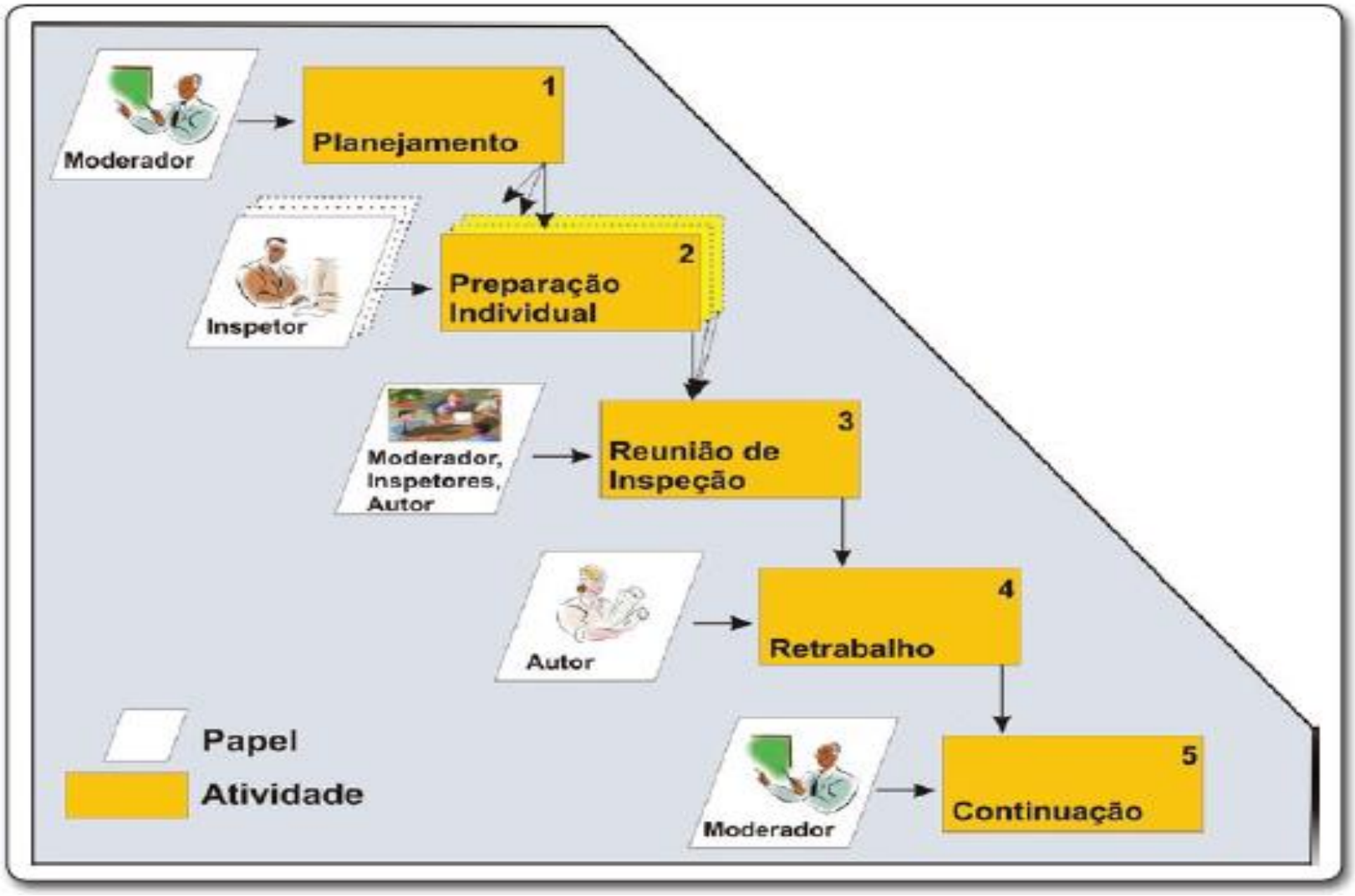
- **Inspeção de programa** é um tipo de inspeção de software
  - Detecção de defeitos de programa.
  - Erros de lógica, condições errôneas, não conformidade com padrões de projeto.
- Outros tipos de revisão:
  - Artefatos de modelagem, cronograma, etc.

- **Processo formal** realizado em equipe.

Inspeção de Software
Papéis do processo de inspeção

- Em uma inspeção, cada um dos participantes recebe um ou mais papéis e responsabilidades.
- Os papéis de uma inspeção tipicamente são distribuídos em:
  - Autor:
    - Responsável pelo artefato assim como por sua correção.
    - Entrega o produto de trabalho e documentos associados aos participantes.
    - Esclarece as dúvidas relativas ao produto a ser inspecionado.
  - Moderador, Leitor, Escritor (relator)
  - Inspetor
    - Encontra erros, omissões e inconsistências.
  - Coordenador das Inspeções

Processo de Inspeção de Software



Processo de inspeção de software, conforme definido por Fagan.

Processo de Inspeção de Software

1. O **moderador** planeja a inspeção: alocação de pessoal e de recursos necessários.
2.O **autor** apresenta o programa explicando o que o mesmo se propõe a fazer.
3.Cada membro da equipe estuda o programa procurando por erros.
4.Os erros são apresentados pelo **leitor** e registrados pelo **relator.**
5.O **autor** corrige os problemas identificados.
6.O **moderador** chefe decide sobre a necessidade de conduzir outra inspeção.

- ***Checklist*** pode auxiliar no processo de inspeção de software.
- Inspeções são aprimoradas com a experiência da equipe.

Para alguns autores a inspeção **aumenta os custos** no começo do processo de software.

No entanto, para outros, ela tem se mostrado **eficiente e de baixo custo** para encontrar defeitos, reduzindo o retrabalho e melhorando a qualidade dos produtos.

Exemplo: inspeção de um documento de especificação de requisitos

Defeito 16

Localização	RS - Página 11
Classificação	Informação Inconsistente
Severidade	Baixa
Descrição	Na descrição dos casos de uso, para um mesmo "tipo" de condição, ora há pós-condição ora não há, por exemplo, cadastramento com sucesso de um item.
Comentário de Correção	CORRIGIDO Do UC07 ao UC17 não tinha pós-condição.

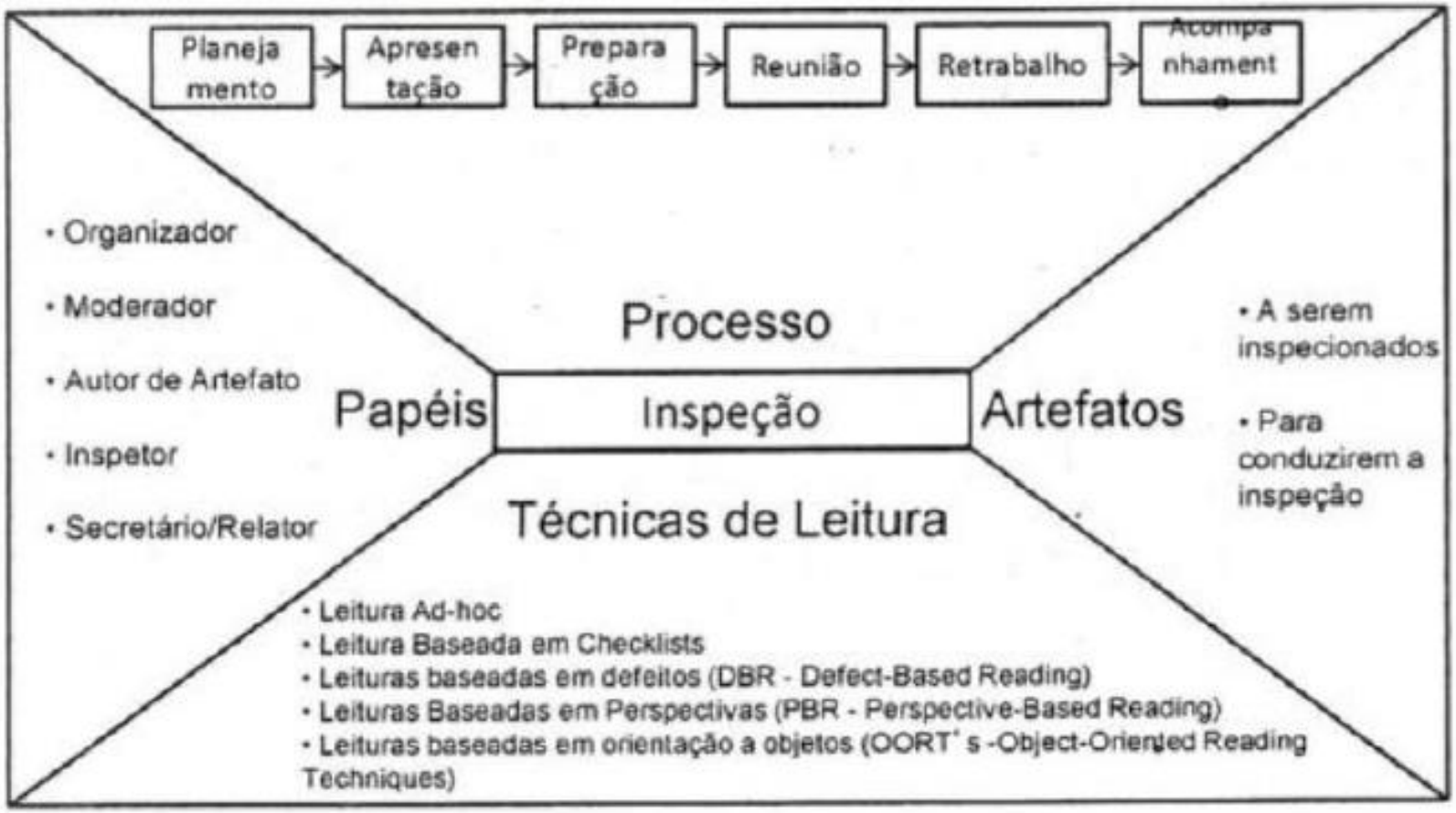
Comentários (3)

1.	Moderador	Não entendi direito qual o problema, poderia especificar mais?
2.	Inspetor 2	Sim, o que eu quis dizer é que uma pós condição do tipo "o sistema armazena as modificações com sucesso" em alguns casos aparece e outras, não. Se 'cadastrar informção com sucesso" não é uma pós-condição 'relevante" acredito que nenhum caso de uso deveria ter tal 'tipo" de pós-condição.
3.	Moderador	OK, sugiro que tentemos padronizar isto, vou classificar como defeito bom baixa severidade.

Exemplo de *checklist* de defeitos de código-fonte

Classe de defeitos	Checagem de inspeção
Defeitos nos dados	Todas as variáveis do programa são iniciadas antes de seu uso? Todas as constantes foram denominadas O limite superior de vetores deve ser igual ao tamanho do vetor ou ao tamanho –1? Se as strings de caracteres são utilizadas, um delimitador é explicitamente designado? Existe alguma possibilidade de overflow de buffer
Defeitos de controle	Para cada declaração condicional, a condição está correta? Cada laço está certo para terminar? As declarações compostas estão corretamente entre parênteses? Em declarações 'case', todos os casos são levados em conta? Se um identificador break é requerido após cada caso em declarações 'case', esse identificador foi incluído?
Defeitos de entrada/saída	Todas as variáveis de entrada são utilizadas? Todas as variáveis de saída tem um valor designado antes de saírem? Entradas inesperadas podem fazer com que os dados sejam corrompidos?
Defeitos de interface	Todas as chamadas de funções ou métodos tem o número correto de parâmetros? Os tipos formais e reais de parâmetros combinam? Os parâmetros estão na ordem certa?
Defeitos de Gerenciamento de armazenamento	Se uma estrutura encadeada é modificada, todos os ponteiros foram corretamente redesignados? Se o armazenamento dinâmico é utilizado, foi alocado espaço correto? O espaço é explicitamente liberado, depois que não é mais necessário?
Defeitos de gerenciamento de exceções	Todas as possíveis condições de erros foram levadas em conta?

Processo de Inspeção de Software
Outra visão



Laitenberger, Oliver e DeBaud, Jean-Marc. An Encompassing Life Cycle Centric Survey of Software Inspection. The Journal of Systems and Software, vol 50, 2000

Técnicas de Leitura de Artefatos

Existem algumas técnicas indicadas para apoiar a inspeção de artefatos de software, conhecidas como **“técnicas de leitura”**, que fornecem diretrizes para a detecção de defeitos durante a inspeção.

- **Leitura Baseada em *Checklist* (LBCh**), composta de uma lista de perguntas que auxilia os inspetores a encontrar defeitos no produto, durante a inspeção.
- **Leitura Baseada em Cenário (LBCe**), que descreve como procurar defeitos com base em um conjunto de instruções e diretrizes denominadas de cenários.
- **Leitura Baseada em Perspectiva (LBPe**), se baseia na ideia de cenários, porém aplica a inspeção a partir dos diferentes pontos de vista assumidos pelos *stakeholders* do sistema inspecionado.

Classificação de defeitos

Omissão: quando informações importantes são deixadas de lado.

Comissão: quando uma informação errada é introduzida no documento.

Classe	Tipo	Descrição
Omissão	Funcionalidade Omitida (FO)	Alguma informação, relativa à descrição do comportamento esperado do sistema, não aparece no documento.
	Performance Omitida (PO)	Alguma informação, relativa à descrição da performance desejada, não aparece no documento, ou aparece de forma inaceitável.
	Ambiente Omitido (AO)	Alguma informação, relativa à descrição do hardware, do software, do banco de dados e do pessoal envolvido, não aparece no documento.
	Interface Omitida (IO)	Alguma informação, relativa à forma como o sistema interagirá ou se comunicará com componentes que estão fora do escopo do sistema, não aparece no documento.
Comissão	Informação Ambígua (IA)	Um termo importante, uma frase ou uma sentença, essenciais para o entendimento do sistema não foi definido no documento, ou foi definido de forma que possa causar confusão.
	Informação Inconsistente (II)	Duas sentenças contradizem-se mutuamente ou expressam ações de que não estão corretas ou não podem ser executadas.
	Funcionalidade Incorreta (FI)	Alguma sentença expressa um fato que não pode ser verdade de acordo com as condições especificadas.
	Seção Incorreta (SI)	Alguma informação está em um local errado dentro do documento.
Outros (O)		Defeitos que não se enquadram nos tipos acima.

Análise estática automatizada

– Ferramentas que auxiliam, de forma automatizada, a inspeção de software.
– Complementam os recursos de detecção de erros do compilador.
– Enfatizam situações que poderiam gerar erros quando o programa é executado.
–Exemplo:
  –Variáveis usadas sem serem inicializadas;
  –Variáveis não usadas;
  –Linguagem C : *Lint*
  –Linguagem Java: *CheckList, Find Bugs*

Vantagens da Inspeção

- Nos testes, erros podem ocultar outros erros.
- Versões incompletas podem ser inspecionadas sem custos adicionais.
- Pode considerar atributos de qualidade mais amplos: **padrões**, **facilidade de manutenção**.

Vantagens da Inspeção

- São mais eficazes que os testes
  - Testes de unidade tipicamente encontram de **2 a 4** defeitos por hora.
  - Inspeções de código tipicamente encontram por volta de **10** defeitos por hora.
  - Inspetores experientes são capazes de encontrar **70%** ou mais de defeitos de um produto.
  - Testes de unidade dificilmente superam um rendimento de **50%.**

Vantagens da Inspeção

- Após o teste de unidade, a remoção de defeitos torna-se muito mais cara
  - No teste de integração e de sistemas são necessárias de **10 a 40 horas** do programador para encontrar e corrigir cada defeito.
  - Inspeções tipicamente levam menos de uma hora por defeito.

Vantagens da Inspeção

- No **teste**
  - Você começa com um problema.
  - Em seguida tem que encontrar o *bug.*
  - Depois, deve imaginar a correção.
  - Por fim, implementa e testa a correção.
- Na **inspeção**
  - Você vê o defeito.
  - Então planeja a correção.
  - Finalmente, implementa e revisa a correção.

Vantagens da Inspeção

- No **teste**
  - Se o programa produziu um resultado não usual, você precisa:
    - Detectar que aquilo não foi usual.
    - Descobrir o que o sistema estava fazendo.
    - Encontrar em que ponto estava no programa.
    - Descobrir que defeito poderia causar este comportamento estranho.

Vantagens da Inspeção

- Na **inspeção**
  - Você segue sua própria lógica.
  - Quando encontra um defeito, sabe exatamente onde está.
  - Você sabe o que o programa deveria fazer e não está fazendo.
  - Logo você sabe porque isto é um defeito.
  - Portanto, está em melhor posição para imaginar uma correção completa e eficaz.
  - Quando combinadas com testes, o número de defeitos encontrados pode superar os **90%** de defeitos existentes.

Algumas métricas de inspeção

- Densidade de defeito = Total de Defeitos / Tamanho atual
- Total de Defeitos = Defeitos Principais + Defeitos secundários
- Esforço de Inspeção = Esforço Planejamento + Esforço Visão Geral + Esforço Preparação + Esforço Reunião + Esforço Retrabalho
- Taxa de Preparação = Tamanho Planejado / (Esforço Preparação/ N de participantes)
- Taxa de Inspeção = Tamanho atual / Tempo de Reunião de Inspeção

Outras técnicas de Revisão
Walkthrough

- Reuniões informais para avaliação dos produtos.
- Pouca ou nenhuma preparação requerida.
- O desenvolvedor guia os presentes através do produto.
- O objetivo é comunicar ou receber aprovação.
- São úteis principalmente para **requisitos** e **modelos de software**.

Referências

- Sommerville, Ian. Engenharia de Software - 8ª Edição 2007.
- Pressman, R.S., Engenharia de Software. 6a edição, McGrawHill, 2006.
- Delamaro, M.E., Maldonado, J.C., Jino, M., Introdução ao Teste de Software, Série Campus – SBC, Editora Campus, 2007.
- Koscianski, A., Soares, M.S., Qualidade de Software, Editora Novatec, 2006.
- Myers, G.J., The Art of Software Testing, 2nd edition, John Wiley & Sons, 2004.
- McGregor, J.D., Sykes, D.A., A Practical Guide to Testing Object-Oriented Software, Addison-Wesley, 2001.
- http://istqbexamcertification.com/what-is-validation-in-software-testing-or-what-is-software-validation/
- Brazilian Software Test Qualification Board. http://www.bstqb.org.br/
- **Introdução à Inspeção de Software. http://www-di.inf.puc-rio.br/~kalinowski//publications/KalinowskiS07.pdf**