Disciplina: **206580 - Verificação e Validação**

Técnicas de Ver & Val

Prof. Ricardo Ajax
ricardoajax@unb.br

Agenda

- Análise Estática x Análise Dinâmica
- Técnicas de Ver&Val Estáticas
- Definição de Exercício

Análise Dinâmica x Análise Estática

- Dinâmica
  - Visa exercitar o software com o uso de dados reais;
  - Verificar se as saídas obtidas estão de acordo com as saídas esperadas;
  - Só é possível quando um protótipo ou versão executável está disponível;

Análise Dinâmica x Análise Estática

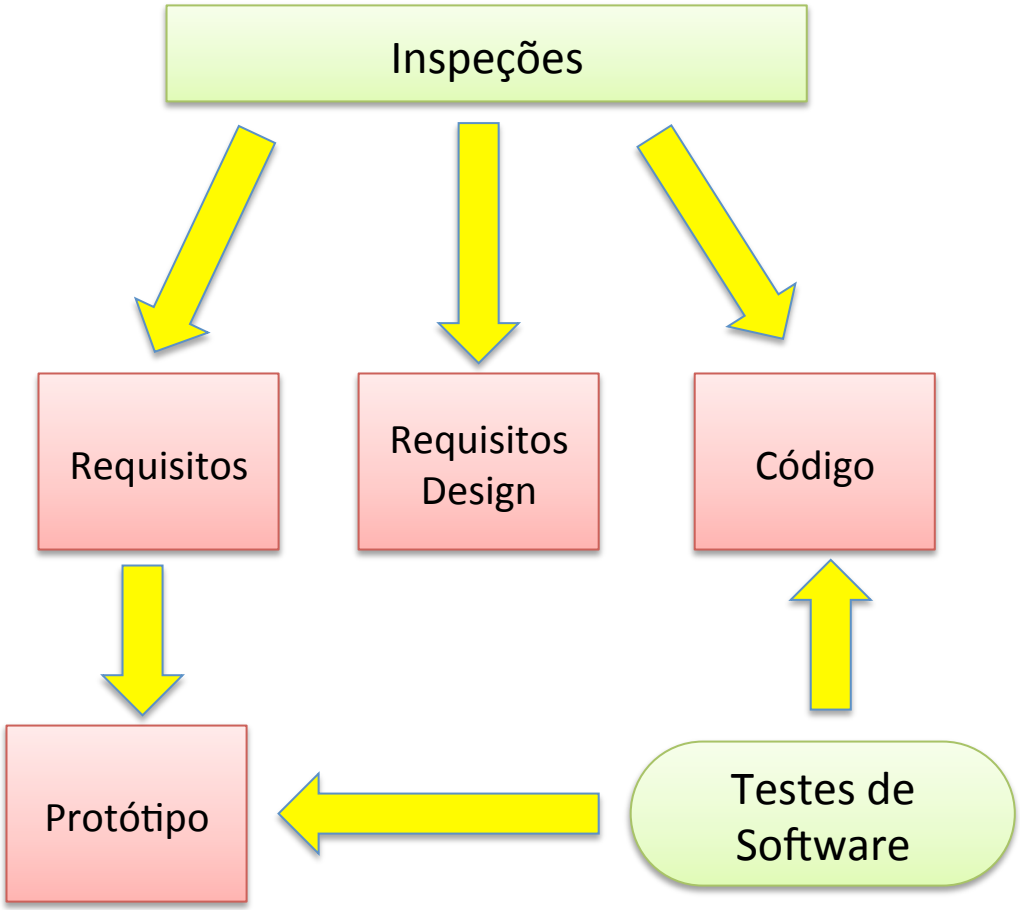
- Estática
  - Não necessita de uma versão executável
  - Pode ser usada em todas as fases do desenvolvimento
  - Pode ser automática
  - Permite verificar correspondência entre um programa e sua especificação
  - **Não** permite demonstrar que o software é útil operacionalmente
  - **Não** permite verificar propriedades emergentes como desempenho e confiabilidade

Análise Dinâmica x Análise Estática

- Inspeções de software, Revisões em Pares, Walkthroughs-> Estáticas

- Testes de Software -> Dinâmica

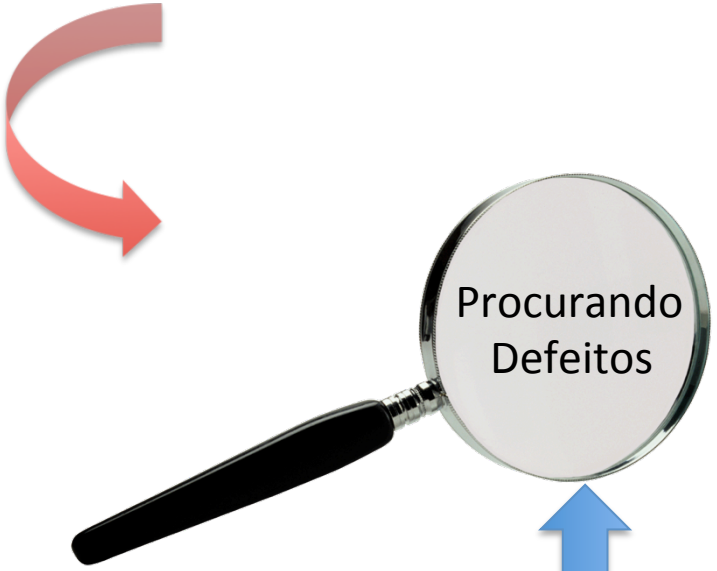
Técnicas e estágios do processo



Técnicas de Verificação Estática

Podem cometer
ERROS

Que podem
Virar defeitos



Se não achados
Podem conduzir a ???

Verificação Estática

- Estudos e experimentos têm demonstrado que as **inspeções** são muito úteis na descoberta prévia de defeitos evitando que eles se constituam em falhar posteriores;
- 60% dos erros podem ser detectados usando-se inspeções informais (Fagan, 1986)
- Alguns estudos realizados (Selby e Basili,1987; Gilb e Graham 1993) constataram que a **revisão estática de código, além de eficiente, tende a ser menos dispendiosa** que o **teste**

Vantagens da Verificação Estática

- Uma única sessão de inspeção pode descobrir vários erros
  - Não há preocupação com a interação entre os erros. Ao contrário do teste (análise dinâmica), em que erros encontrados podem mascarar outros erros.
- Versões incompletas de um sistema podem ser avaliadas sem custos adicionais
  - No teste de versões incompletas pode ser necessário por exemplo criar *stubs* ou massa de dados específicas, afim de simular comportamentos e, portanto, permitir a execução de testes.
- Análise estática pode ir além dos defeitos e verificar a qualidade do código quanto a aspectos como aderência a padrões, facilidade de manutenção, complexidade etc
- Localização do erro

Dificuldades de adoção

- Os custos incorrem desde o início do desenvolvimento
- Engenheiros de software relutam em aceitar que a inspeção pode ser mais eficiente que o teste
- Inspeções levam tempo e parecem diminuir a velocidade do processo de desenvolvimento, gerando o sentimento que são mais caras. Porém, quanto mais cedo se descobere um defeito e se evita que ele vire uma falha, menor o custo para a sua resolução.

=> Ou seja: O tempo e os custos podem ser recuperados com menor esforço, pois menos "coisas" precisam ser reparadas.

Revisões de Código na Google

- "*All code that gets submitted needs to be reviewed by at least one other person, and either the code writer or the reviewer needs to have readability in that language. Most people use Mondrian to do code reviews, and obviously, we spend a good chunk of our time reviewing code.*"

-- Amanda Camp, Software Engineer, Google

http://www.niallkennedy.com/blog/2006/11/google-mondrian.html

Revisões de Código na Facebook

- *"At Facebook, we have an internally-developed web-based tool to aid the code review process. Once an engineer has prepared a change, He or she submits it to this tool, which will notify the person or people he/she has asked to review the change, along with others that may be interested in the change - - such as people who have worked on a function that got changed.*
- *At this point, the reviewers can make comments, ask questions, request changes, or accept the changes. If changes are requested, the submitter must submit a new version of the change to be reviewed. All versions submitted are retained, so reviewers can compare the change to the original, or just changes from the last version they reviewed. Once a change has been submitted, the engineer can merge her change into the main source tree for deployment to the site during the next weekly push, or earlier if the change warrants quicker release."*

- Ryan McElroy, Software Engineer, Facebook

Análise Estática Técnicas

- Informais
- Auditoria
- Walkthroughs
- Revisão em pares
- Inspeções
- Análise estática automática

Verificação Estática

- Verificar a **consistência** e **completude** dos documentos ou código
- Certificar-se que os **padrões** ou **normas** aplicáveis foram seguidos
- Descobrir **problemas** no software ou na documentação do projeto

Auditoria

- Revisores **independentes**
- Utilização de checklists e questionários
- Embasamento em normas e padrões da área
- Difícil encontrar auditores experientes

Auditoria

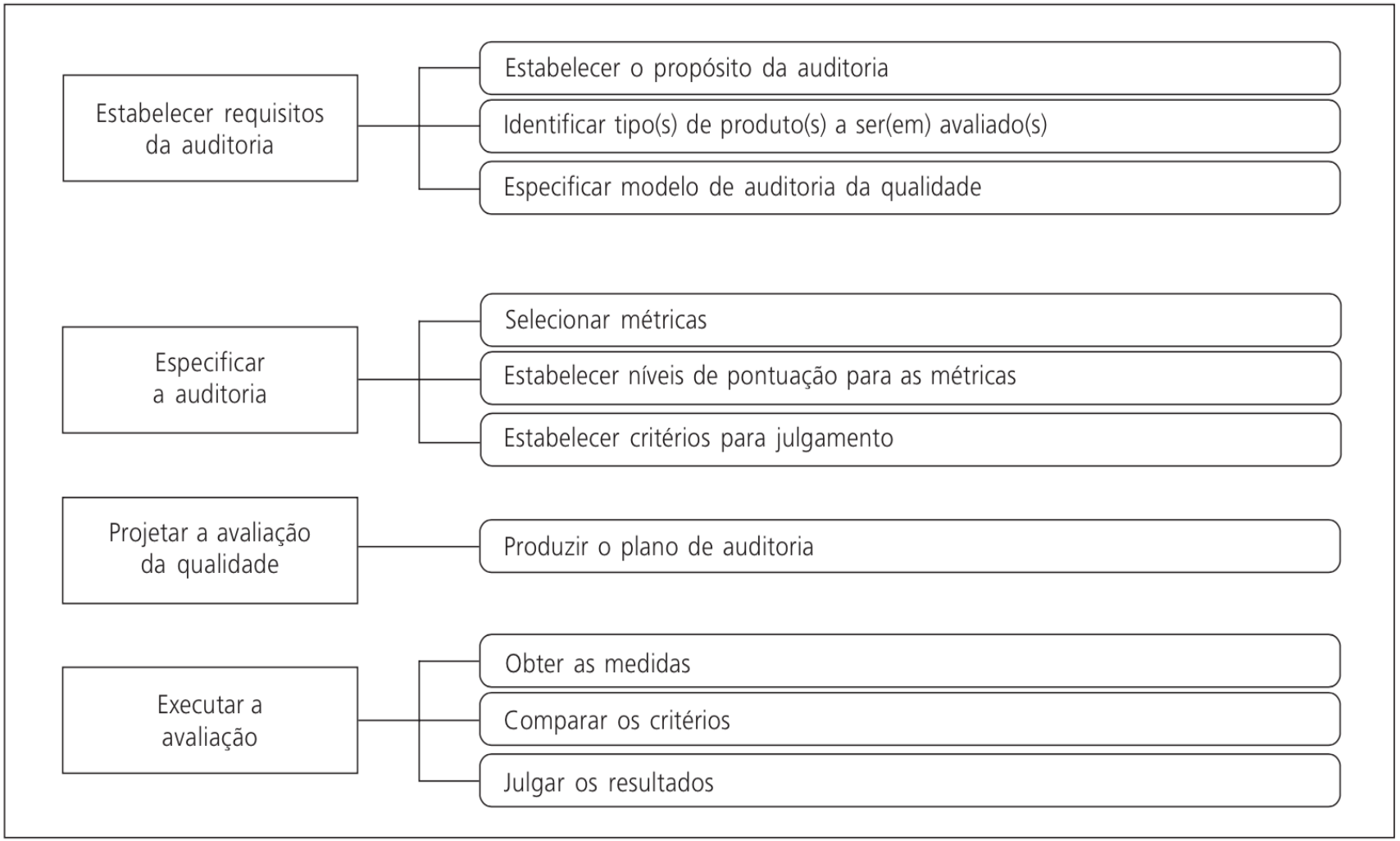


Figura 1: Etapas e fases do roteiro de auditoria da qualidade de *softwares*

Fonte: adaptada da NBR ISO/IEC 14598-1 (2001, item 6, figura 3).

[EK Marçal, 2009]

Revisão por Pares

Ou *Peer-Review*

- É um processo de revisão que usa pares para revisar os aspectos do ciclo de vida de desenvolvimento que lhes são mais familiares
- Autor e revisor executam a mesma função no ciclo de vida
- Deve ser estruturada (procedimentos, relatórios finais)
- Realizada com base em *check-lists*
- Desvantagem: problemas em fazer ou aceitar críticas
- Obs.: é também o método de revisão de trabalhos científicos

Walkthrough

- Consiste em uma checagem do produto
- Pode ser multidisciplinar: equipe faz perguntas e levanta questionamentos
- Menos estruturada, apenas procedimentos básicos
- Pode ser solicitada pela própria equipe de desenvolvimento para avaliar se determinadas questões estão sendo resolvidas da melhor forma possível
- Desvantagem: depende da habilidade e interesse da equipe participante

Inspeções

- Revisões estruturadas
- Produto é avaliado contra critérios de entrada ou contra a especificação do produto.
- Revisão minunciosa por pessoas experientes.
- Responsabilidades divididas: moderador, autor, revisor
- Geralmente dirigidas por *checklists* de verificação de erros

Procedimento de inspeção de Software

- Autor
  - Criar ou manter o produto a ser revisado
  - Efetuar as correções
  - Apresentar o produto
- Revisor
  - Revisar o produto
  - Participar da reunião de inspeção

Procedimento de inspeção de Software

- Moderador
  - Coordenar revisão
  - Garantir que o procedimento está sendo seguido
  - Monitorar os prazos de execução das fases
  - Distribuir os produtos aos revisores
  - Acompanhar a correção dos defeitos
  - Moderar conflitos
  - Coletar medições

Procedimento de inspeção de Software

- Passo 1: Apresentar produto (opcional, mas recomendável)
- Passo 2: Revisar Produto (orientado ou não a checklists)
- Passo 3: Reunir equipe (comunicar defeitos)
- Passo 4: Corrigir defeitos (retrabalhar o artefato)
- Passo 5: Verificar correção de defeitos (nova rodada de inspeção?)

Exs: Classes de Defeitos - Requisitos

- Funcionalidade omitida
- Desempenho omitido
- Ambiente omitido
- Interface omitida
- Informação ambígua
- Informação inconsistente
- Funcionalidade incorreta
- Outros

Exs: Classes de Defeitos - Código

Depende, mas geralmente:

- .Defeitos de sintaxe:
  - Uso inadequado de comandos e funções
- Defeitos de Dados
  - Inicialização de variáveis, definição de constantes, limites de vetores, etc
- Defeitos de Controle
  - Terminação correta de loops, condições de declarações condicionais, todos os casos possíveis em declarações “case”, etc
- Defeitos de Entrada/Saída
  - Todas as variáveis de entrada são usadas, variáveis de saída tem valores atribuídos, entradas inesperadas, etc

Exs: Classes de Defeitos - Código

- Defeitos de Interface
  - Chamadas de funções e métodos tem o número correto de parâmetros, parâmetros na ordem correta, etc
- Defeitos de Gerenciamento de Armazenamento
  - Alocação de espaço no armazenamento dinâmico liberação de memória, etc
- Defeitos de Gerenciamento de Exceções
  - Todas as condições possíveis de erro foram consideradas?

Erro C7, X25932

Ocorreu um erro

Contate o administrador

Análise Estática Automática

- Algumas classes de defeitos de códigos podem ser verificadas por um processo automático
- Vem sendo extensamente utilizada, por exigir poucos recursos, ser rápida e apresentar os resultados de forma imediata
- Os analisadores estáticos automatizados complementam os recursos de detecção de erros fornecidos pelos compiladores
- Podem ser usados como parte do processo de Inspeção ou como uma atividade separada do processo de V&V

Análise Estática Automática

- Os tipos de defeitos detectados pelos analisadores estáticos dependem do tipo de linguagem
- Exemplos de classes de defeitos:
  - Defeitos de Dados:
    - Variáveis usadas antes da inicialização
    - Variáveis declaradas, mas não utilizadas
    - Variáveis atribuídas duas vezes, mas não usadas entre as atribuições
    - Possíveis violações de limites de vetor
    - Variáveis não declaradas

Análise Estática Automática

- Exemplos de classes de defeitos:
  - Defeitos de Controle:
    - Código inacessível
    - Ramos incondicionais dentro de *loops*
  - Defeitos de entrada/saída
    - Saída de variáveis duas vezes sem atribuição intermediária

Análise Estática Automática

- Exemplos de classes de defeitos:
  - Defeitos de interface
    - Incompatibilidades de tipo de parâmetro
    - Incompatibilidades de número de parâmetros
    - Não uso de resultados de funções
    - Funções de procedimentos não chamados
  - Defeitos de gerenciamento de armazenamento
    - Ponteiros não atribuídos
    - Ponteiro aritmético
    - Perdas de memória

Análise Estática Automática

- Níveis de verificação que podem ser implementados:
  - **Verificação de erros característicos**: detecção de erros comuns em determinada linguagem de programação
  - **Verificação de erros definidos pelo usuário**: padrões de erros a serem verificados são definidos pelo usuário do analisador estático (por exemplo, padrões específicos de uma organização)
  - **Verificação de asserções**: o analisador executa simbolicamente o código e destaca as asserções incluídas pelos desenvolvedores que não podem ser mantidas
    - Exemplos de analisadores que utilizam esta abordagem: Splint e SPARK Examiner

Análise Estática Automática

- Como resultado da análise as ferramentas podem gerar:
  - **Falsos negativos**: em programas não triviais, não é possível encontrar todos os defeitos
  - **Falsos positivos**: as ferramentas podem reportar problemas em seções de código sem defeitos
- Em geral há um relacionamento inverso entre falsos positivos e falsos negativos:
  - Ferramentas que reportam mais defeitos reais (verdadeiros positivos) em geral tem alta taxa de falsos positivos
  - Opções de configuração nas ferramentas permitem aos usuários controlar a análise favorecendo um ou outro espectro

Exercício 1

- Turma dividida em grupos
- Qual a disciplina do desenvolvimento será escolhida para ter seus produtos analisados?
- Quais os produtos são produzidos pela disciplina
- Quais deles são críticos? Por que?
- Qual deles será verificado
- Quais os produtos de trabalho devem ser verificados em um projeto? Por que?

Exercício 1

- Qual o tipo de verificação estática será escolhido?
- Por que?
- Qual o processo será seguido? (Desenhar)
- Existirá algum produto de suporte à atividade de verificação ou validação? Qual? (ex: Cheklist)
- Qual será a infra-estrutura necessária para a atividade? (hardware, software, ambiente tecnológico)
- Quem serão os envolvidos? Seus papeis?
- Em que artefato os resultados serão apresentados?
- Quais os resultados esperados?
- Que medições serão feitas?
- Execute a verificação ou validação

Exercício 1

Quais foram os resultados?

- Defeitos encontrados
- Criticidade dos defeitos encontrados
- Necessidade de reparos
- Quais as medições foram coletadas
- Quem serão os responsáveis pela correção
- Quais os prazos de correção
- Descreva as Ações corretivas?
- Descreva as Ações preventivas? (caso existam)
- Apresente os resultados para a turma
- Faça uma análise crítica dos resultados

Exercício 1

Relatório: entrega via Moodle

Apresentação: Em sala de aula

Dúvidas?

- Ricardo Ajax
  - Ricardoajax@unb.br