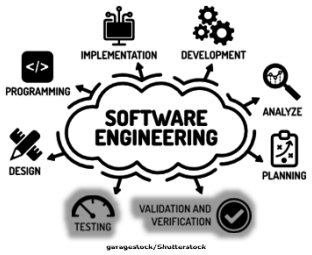
**Aula 1**

## Teste de *Software*

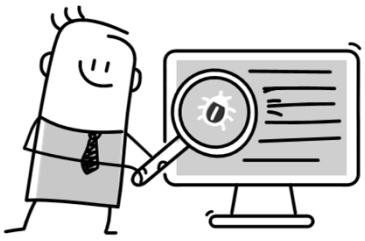
**Profª Maristela Weinfurter**

## Conversa Inicial

## Teste de *software*




garagestock/Shutterstock



Semanche/Shutterstock

## Fundamentos de processos de testes

## Fundamentos de processos de testes

- **Teste de *software* é uma atividade:**
  - **Abrangente**
  - **Complexa**
  - **Pode ser baseada em princípios:**
    - **Da qualidade de *software***
    - **Da engenharia de *software***

AnnaKoles/Shutterstock

## Fundamentos de processos de testes

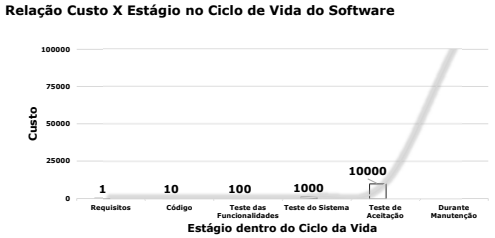
- **Segundo Hambling (2015), o recomendado é realizarmos pequenos testes em pedaços de *software* de forma exaustiva**

## Fundamentos de processos de testes

- **Tabela 1: custo comparativo para correção de erros. Elaborado com base em Hambling, 2015.**

| Estágio no qual o erro foi encontrado | Comparativo de Custos |
|---|---|
| 1. Requisitos | $1 |
| 2. Código | $10 |
| 3. Teste das Funcionalidades | $100 |
| 4. Teste do Sistema | $1.000 |
| 5. Teste de Aceitação | $10.000 |
| 6. Durante Manutenção | $100.000 |

## Fundamentos de processos de testes

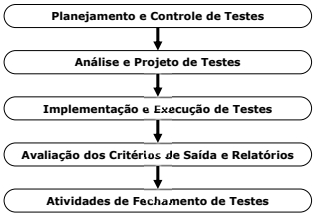



## Fundamentos de processos de testes

- **Teste é um processo. Deve ser detalhado como um processo de teste fundamental aplicável em todas as etapas**
- **A parte mais visível do teste é o ato da sua própria execução. Precisamos preparar a execução de testes, analisar os que foram executados e verificar se foram concluídos**

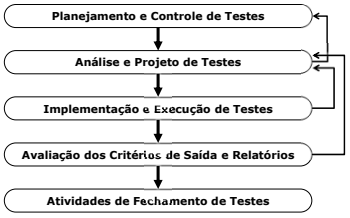
## Fundamentos de processos de testes

- **Figura 2: Processos Fundamentais de Teste (Hambling, 2015)**




## Fundamentos de processos de testes

- **Figura 3: Iteração de Atividades de Teste (Hambling, 2015)**



## Fundamentos de processos de testes

- Desenvolver e priorizar casos de teste, criar dados de teste, escrever procedimentos de teste e, opcionalmente, preparar equipamentos de teste e escrever *scripts* de teste automatizados
- Coletar casos de teste em suítes de teste, em que os testes podem ser executados um após o outro, para maior eficiência

## Fundamentos de processos de testes

- Verificar se a configuração do ambiente de teste está correta
- Executar casos de teste na ordem determinada. Isso pode ser feito manualmente ou usando ferramentas de execução de teste

## Fundamentos de processos de testes

- Manter um registro das atividades de teste, incluindo o resultado (aprovado/reprovado) e as versões de *software*, dados, ferramentas e *testware* (*scripts* etc.)
- Comparar os resultados reais com os resultados esperados

## Fundamentos de processos de testes

- Relatar discrepâncias como incidentes com o máximo de informações possível, incluindo, se possível, análise causal (defeito de código, especificação de teste incorreta, erro de dados de teste ou erro de execução de teste)

## Fundamentos de processos de testes

- Sempre que necessário, repetir as atividades de teste quando as alterações forem feitas após os incidentes levantados. Isso inclui a reexecução de um teste que falhou anteriormente para confirmar uma correção (reteste), a execução de um teste corrigido e a execução de testes aprovados anteriormente para verificar se os defeitos não foram introduzidos (teste de regressão)

## Fundamentos de processos de testes

- Avaliando os critérios de saída:
  - Verificar se os critérios de saída previamente determinados foram atendidos
  - Determinar se são necessários mais testes ou se os critérios de saída especificados precisam ser alterados
  - Redigir o resultado das atividades de teste para os patrocinadores do negócio e outras partes interessadas

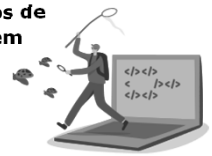
- Para o desenvolvimento que utiliza metodologias ágeis, os processos fundamentais de testes são muito parecidos
- O que pode variar são algumas formas de como utilizamos determinadas atividades de teste

Jerome.Romme/Shutterstock

## Teste e depuração

## Teste e depuração

- Teste e depuração são tipos de atividades diferentes, porém ambas são importantes e complementares


eamesBot/Shutterstock

## Teste e depuração

- Depuração é o processo pelo qual os desenvolvedores passam para identificar a causa de *bugs* ou defeitos no código e realizar correções


eamesBot/Shutterstock

## Teste e depuração

- O teste, por outro lado, é uma exploração sistemática de um componente ou sistema com o objetivo principal de encontrar e relatar defeitos


eamesBot/Shutterstock

## Teste e depuração

- O teste, por outro lado, é uma exploração sistemática de um componente ou sistema com o objetivo principal de encontrar e relatar defeitos. O teste não inclui a correção de defeitos – estes são repassados ao desenvolvedor para correção. O teste, no entanto, garante que as alterações e correções sejam verificadas quanto ao seu efeito em outras partes do componente ou sistema (Brown, 2014)

## Teste e depuração

- Há quem diga que *depuração* é um termo infeliz, em virtude de sua associação com defeitos. O fato é que o que chamamos de *depuração* é uma atividade que nós faremos o tempo todo, seja implementando um novo recurso, seja aprendendo como algo funciona, seja ainda corrigindo um *bug*

## Teste e depuração

- A depuração é uma habilidade frequentemente negligenciada: parece que a maioria dos programadores não encontram base em obras, livros e periódicos

## Teste e depuração

- O primeiro e mais importante princípio de depuração é o processo de eliminação

## Teste e depuração

- A eliminação pode assumir muitas formas, tais como alguns exemplos comuns:
  - Comentar ou desabilitar sistematicamente blocos de código
  - Escrever o código que pode ser coberto por testes de unidade; os próprios testes de unidade fornecem uma estrutura para eliminação
  - Analisar o tráfego de rede para determinar se o problema está no lado do cliente ou do servidor

## Teste e depuração

- Testar uma parte diferente do sistema que tenha semelhanças com a primeira
- Usar a entrada que funcionou antes e alterar essa entrada com uma parte de cada vez até que o problema seja exibido
- Usar o controle de versão para voltar no tempo, um passo de cada vez, até que o problema desapareça
- Funcionalidade de "simulação" para eliminar subsistemas complexos

## Teste e depuração

- Alguns procedimentos mais utilizados durante o processo de depuração de código:
  - Criação de *breakpoints* em partes do código para que possamos ir passo a passo pelo código entre esses pontos de atenção
  - Uma vez colocados os *breakpoints*, podemos executar a depuração sobre o código, parando exatamente nesses *breakpoints*

- Acionamos variáveis, estruturas e outros elementos em que possamos identificar o exato conteúdo no momento no qual o código passa pelo ponto que estamos observando
- Podemos executar a depuração linha a linha de código ou de ponto a ponto de parada (*breakpoints*), observando os conteúdos transformados dentro da funcionalidade que estamos depurando


## Garantia da qualidade e testes

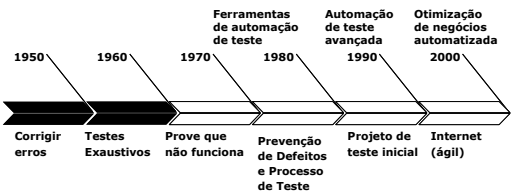
## Garantia da qualidade e testes

- Teste de *software* é a atividade de executar uma série de execuções dinâmicas de programas de *software* após o desenvolvimento do seu código-fonte. É realizado para descobrir e corrigir o maior número possível de erros potenciais antes da entrega ao cliente

## Garantia da qualidade e testes

- Ao longo da história do desenvolvimento de *software*, houve muitas definições e avanços no teste de *software*

## Garantia da qualidade e testes




## Garantia da qualidade e testes

- Os seguintes bons princípios de teste foram propostos (Lewis, 2009):
  - Uma parte necessária de um caso de teste é uma definição da saída ou resultado esperado
  - Os programadores devem evitar tentar testar seus próprios programas

## Garantia da qualidade e testes

- Uma organização de programação não deve testar seus próprios programas
- Inspecione cuidadosamente os resultados de cada teste
- Os casos de teste devem ser escritos para condições de entrada inválidas e inesperadas, bem como válidas e esperadas

## Garantia da qualidade e testes

- Examinar um programa para ver se ele não faz o que deveria fazer é apenas metade da batalha. A outra metade é ver se o programa faz o que não deveria fazer
- Evite casos de teste descartáveis, a menos que o programa seja realmente um programa descartável

## Garantia da qualidade e testes

- Não planeje um esforço de teste sob a suposição tácita de que nenhum erro será encontrado
- A probabilidade da existência de mais erros em uma seção de um programa é proporcional ao número de erros já encontrados nessa seção

## Garantia da qualidade e testes

- Entre as metodologias ágeis, a programação extrema (XP) é um exemplo de tal tendência. XP é uma abordagem pouco ortodoxa para o desenvolvimento de *software*, e tem sido argumentado que não tem aspectos de *design*

## Garantia da qualidade e testes

- A metodologia de programação extrema propõe um afastamento radical dos processos de desenvolvimento de *software* comumente aceitos. Existem realmente duas regras de XP: (1) faça um pequeno *design;* e (2) sem requisitos, apenas histórias de usuários. Os discípulos da programação extrema insistem que "realmente não existem regras, apenas sugestões"

## Garantia da qualidade e testes

- No modelo XP, o desenvolvedor cria cenários de teste antes de fazer qualquer outra coisa


Bakhtiar Zein/shutterstock

## Garantia da qualidade e testes

- A premissa básica por trás do *design test-first* é que a classe de teste é escrita antes da classe real; assim, o propósito final da classe real não é simplesmente cumprir um requisito, mas simplesmente passar em todos os testes que estão na classe de teste


Bakhtiar Zein/shutterstock

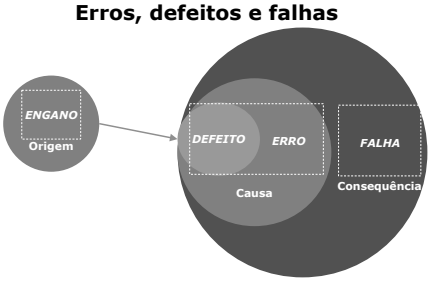
## Erros, defeitos e falhas

## Erros, defeitos e falhas

- O sistema travou durante a produção
- A área de TI cometeu um erro
- Após uma revisão, encontramos um defeito no plano de teste
- Encontrei um *bug* em um aplicativo hoje
- O sistema quebrou
- Uma falha foi relatada no subsistema de monitoramento


ivector/Shutterstock
Leszek Glasner/Shutterstock
Grayscale Studio/Shutterstock

## Erros, defeitos e falhas




## Erros, defeitos e falhas

- Engano: o engano é uma ação humana acidentalmente inclusa dentro de uma classe, uma função ou outro elemento qualquer de um código de programa. Por exemplo, numa fórmula de cálculo de impostos sobre vendas, podemos incluir a fórmula de forma errada ou até mesmo a fórmula errada


MrArtHit/Shutterstock

## Erros, defeitos e falhas

- Defeito:
  o defeito é a consequência de um engano cometido em um trecho de código de programa, o qual resulta em saídas inesperadas ou inconsistentes

Pranch/Shutterstock

## Erros, defeitos e falhas

- **Erro:**
  **o erro, então, se forma devido a um defeito causado, por sua vez, por um engano. Nesse caso, o resultado do *software* é diferente do esperado**

MarcPixel/Shutterstock

Thomas Pajot/Shutterstock

- **Falha:**
  **a falha é uma consequência de um erro. Por exemplo, voltando ao cálculo dos impostos sobre vendas, que teve um defeito em decorrência da escrita de uma fórmula errada, gerou um erro que causou a falha na nota fiscal**

## Causa-raiz e seus efeitos

## Causa-raiz e seus efeitos

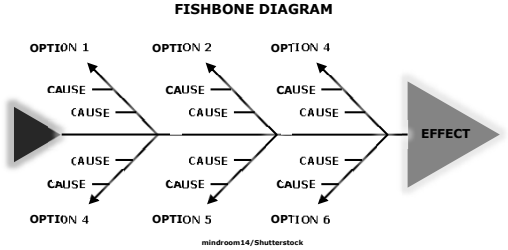
- **A RCA (*Root Cause Analysis*), segundo Latino (2011), é uma das ferramentas mais valiosas para qualquer organização. Isso é especialmente verdadeiro para grandes empresas com uso intensivo de ativos**


sulit.photos/Shutterstock

## Causa-raiz e seus efeitos

- **O desafio de uma RCA eficaz é quando aplicamos os recursos para identificar as causas-raiz de um problema? Há simplesmente muitos problemas que surgem para resolver todos de forma eficaz**

## Causa-raiz e seus efeitos



mindroom14/Shutterstock

**Causa-raiz e seus efeitos**

- **Uma vez coletados todos os erros do *software* em produção, analisamos a causa-raiz do problema, ao agruparmos as causas em grupos. Os erros, uma vez listados e relacionados, atuam na causa e na resolução do problema em definitivo**

- **Algumas questões podem ser levadas em consideração para criação do diagrama de causa e efeito:**
  - **Listar os defeitos encontrados no ambiente**
  - **Listar os problemas dos testes**
  - **Planejar os próximos passos para realização de testes funcionais ou de regressão**

- **Listar *stories* que não estavam como "ready"**
- **Listar ambiguidade em relação às regras de negócio**
- **Listar problemas com cronograma**
- **Listar pontos de questões sobre problemas na concepção ou nos requisitos**