UFSC
Araranguá

Verificação e Validação (V&V) de Software

Teste de Software

Engenharia de Software II

Profa. Andréa Sabedra Bordin

Roteiro

- O que é?
- Conceitos fundamentais
- Por que testar ?
- Tipos de Teste de Software
  - Teste de Unidade
  - Teste de Integração
  - Teste de Aceitação
  - Projeto de Casos de Teste

O que é?

- É o processo de executar um programa com o objetivo de encontrar **defeitos** (Myers, 2004).
  - É, portanto, uma atividade de V&V dinâmica.

Conceitos fundamentais

- **Erro**: trata-se de uma ação humana (ex.: não entendimento de como um cálculo é realizado)
- **Defeito**: Causado por um erro de entendimento (ex.: código com fórmula de cálculo mal escrita)
- **Falha**: Tentativa de execução de um defeito. (ex.: execução de um cálculo gerando resultados indevidos)

- **Falha** é um evento; **defeito** é um estado do software, causado por um **erro**.

A ideia básica dos testes é que os defeitos podem se manifestar por meio de falhas observadas durante a execução do software. Essas falhas podem ser resultado de uma especificação errada ou falta de requisito, dentre outros. Assim, uma falha é o resultado de um ou mais defeitos.

Por que testar?

- Fornecer maior segurança aos clientes;
- Oferecer maior continuidade do serviço ao negócio do cliente;
- Reduzir gastos em correção de falhas;
- **Melhorar a qualidade do software.**

Testes como processo

- Testes não devem ser tratados apenas como uma atividade no ciclo de vida de software, mas sim como um **processo**.

- O processo de teste deve ocorrer em paralelo com outras atividades do processo de desenvolvimento de software (análise de requisitos, projeto de software e implementação) e envolve também atividades de planejamento.

Custos do teste de software

A **regra 10 de Myers** apresenta que o custo da correção de um defeito tende a aumentar quanto mais tarde ele for encontrado.

Myers, G.J., *The Art of Software Testing*, 2nd edition, John Wiley & Sons, 2004.

Perspectiva de teste

- Bons testadores necessitam de um conjunto especial de habilidades. Um testador deve abordar um software com a atitude de questionar tudo sobre ele (McGregor e Sykes, 2001).
- A perspectiva de teste é um modo de olhar qualquer produto de desenvolvimento e questionar a sua validade.
- A perspectiva de teste requer que um fragmento de software demonstre não apenas que ele executa de acordo com o especificado, mas que executa **apenas** o especificado (McGregor e Sykes, 2001).
  - O software faz o que deveria fazer e somente isso?

Técnicas de Testes

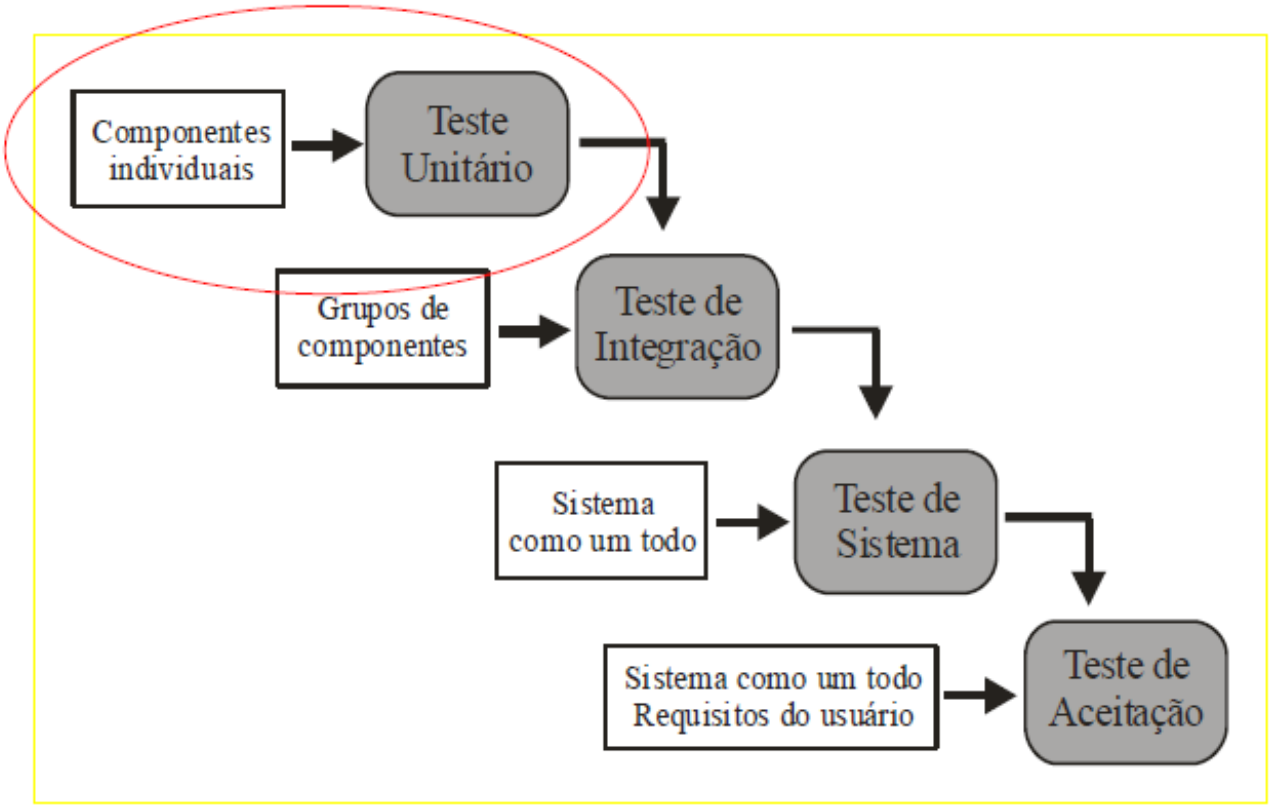
- Caixa-preta (*black-box* ou funcional)
  - Apenas a estrutura externa do sistema é conhecida.

- Caixa-branca *(White-box ou estrutural)*
  - A estrutura interna (código) do sistema é conhecida e usada pelo testador.

Fases de Teste

- A atividade de teste é dividida em fases com objetivos distintos.
- De uma forma geral, pode-se estabelecer como fases (Delamaro et al., 2007):
  - Teste de Unidade
  - Teste de Integração
  - Teste de Sistema
  - Teste de aceitação

Fases de Teste



Teste de Unidade

- Tem como foco as **menores unidades** de um programa.
- Uma unidade é um componente de software que não pode ser subdividido.
  - Na **programação procedural** uma unidade pode ser um programa individual, função, procedimento, etc.
  - Na **programação orientada a objetos**, a menor unidade é um método que pode pertencer a uma classe.
- Nesta fase espera-se encontrar **defeitos relacionados a algoritmos incorretos ou mal implementados.**

Teste de Unidade

- Faz uso intensivo de técnicas que exercitam **caminhos específicos nas estruturas de controle**.

- **Técnica de Caixa-branca**.
  - Pode ser aplicado à medida que ocorre a implementação das unidades.
  - Pode ser realizado pelo próprio desenvolvedor.

Benefícios do Teste de Unidade

- Aumenta a confiança na mudança/manutenção de código.
  - Se os bons testes unitários são escritos e, se são executados cada vez que qualquer código é alterado, é possível se capturar prontamente quaisquer defeitos introduzidos devido à mudança.
- Os códigos são mais reutilizáveis.
  - A fim de tornar o teste de unidade possível, **o código precisa ser modular**.
  - Isto significa que os códigos são mais fáceis de reutilizar.
- O **custo de corrigir um defeito** detectado durante o teste de unidade **é menor** em comparação com os defeitos detectados em níveis mais elevados.

Teste de Integração

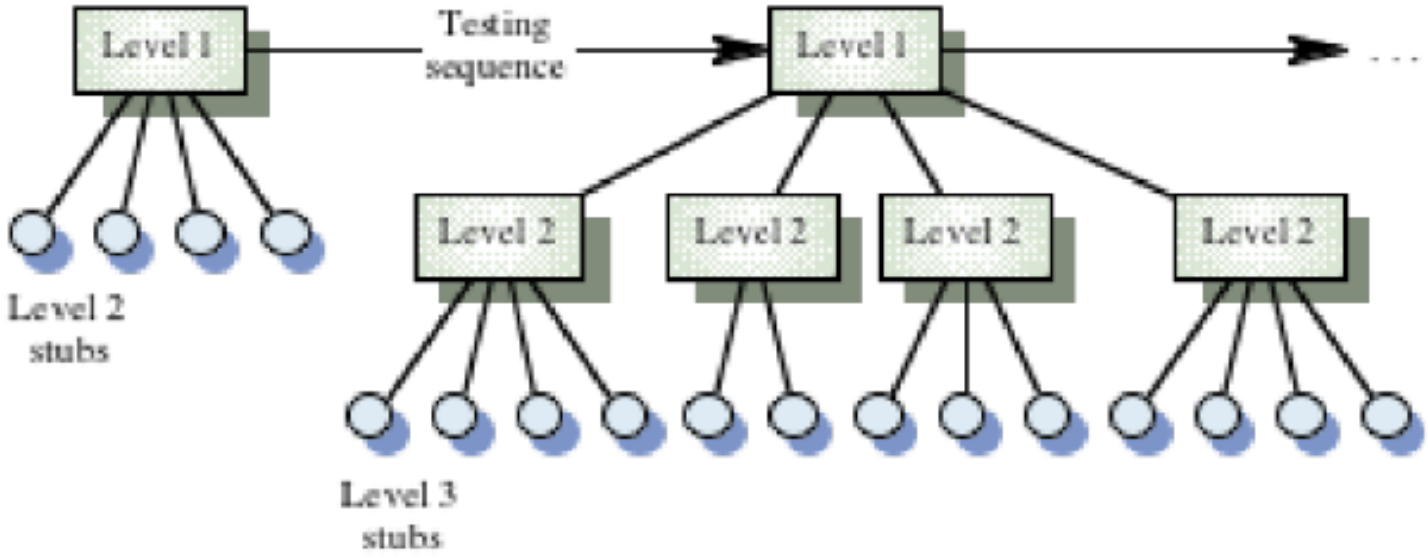
• Deve ser realizado após serem testadas as unidades individualmente.
• Mesmo que todos os módulos estejam funcionando individualmente, não se pode garantir que eles funcionarão em conjunto.
• Deve-se verificar se as partes ou componentes funcionam em conjunto, se são chamados corretamente.
• Requer grande conhecimento das estruturas internas do sistema e, por isso, geralmente é executado pela própria equipe de desenvolvimento (Delamaro et al., 2007).
• Todas as técnicas de teste se aplicam, com destaque para o de **caixa-preta ou teste funcional**.

Teste de Integração
Abordagem de Integração *top-down*

- “Esqueleto” do sistema é desenvolvido primeiro.
- Componentes são adicionados a ele.
- O programador trabalha supondo que o código de baixo nível já esteja pronto.
- Exemplo:
  - Pode-se codificar chamadas à **verificação de CPF**, mesmo sabendo que ela ainda não existe.
  - Em seu lugar, pode haver uma rotina “fantasma” – ***stub -*** estrutura de programa criada para substituir um módulo subordinado ao módulo sob teste.

Teste de Integração
Abordagem de Integração *top-down*

- Começa pelos componentes de alto nível e vai descendo na hieraquia de componentes, de acordo com a arquitetura do software.
- Os sub-componentes são representados por *stubs*.
- Stubs tem a mesma interface, mas não precisa ter funcionalidade. Basta retornar valores esperados.



Teste de Integração
Abordagem de Integração *top-down*

- Vantagens
  - Verificar inicialmente os comportamentos mais importantes do sistema.
- Desvantagens
  - Muitos *stubs* são necessários.
  - O teste para ser efetivo necessita de muitos *stubs*, caso contrário, na integração dos níveis mais baixos poderão acontecer problemas.

Teste de Integração
Abordagem de Integração *bottom-up*

- A integração é feita a partir do nível mais baixo da hierarquia.
  - Integração de componentes que fornecem serviços comuns a outros módulos/componentes.
  - Exemplo: rotina que valida CPF pode ser chamada em vários pontos de um programa.
- Deve ser escrito código que invoque as rotinas de baixo nível, testando-as com diversas combinações de parâmetros.
  - As rotinas escritas para tais testes são conhecidas como ***drivers***, pois sua função é acionar o código que deve ser testado.

Teste de Integração
Abordagem de Integração *bottom-up*

- Para cada combinação é criado um **driver** que coordena a entrada e a saída dos casos de teste.
- O módulo é testado.
- O *driver* é substituído pela combinação de módulos correspondentes, que passam a interagir com os módulos do nível superior.

Teste de Integração
Stubs e Drivers

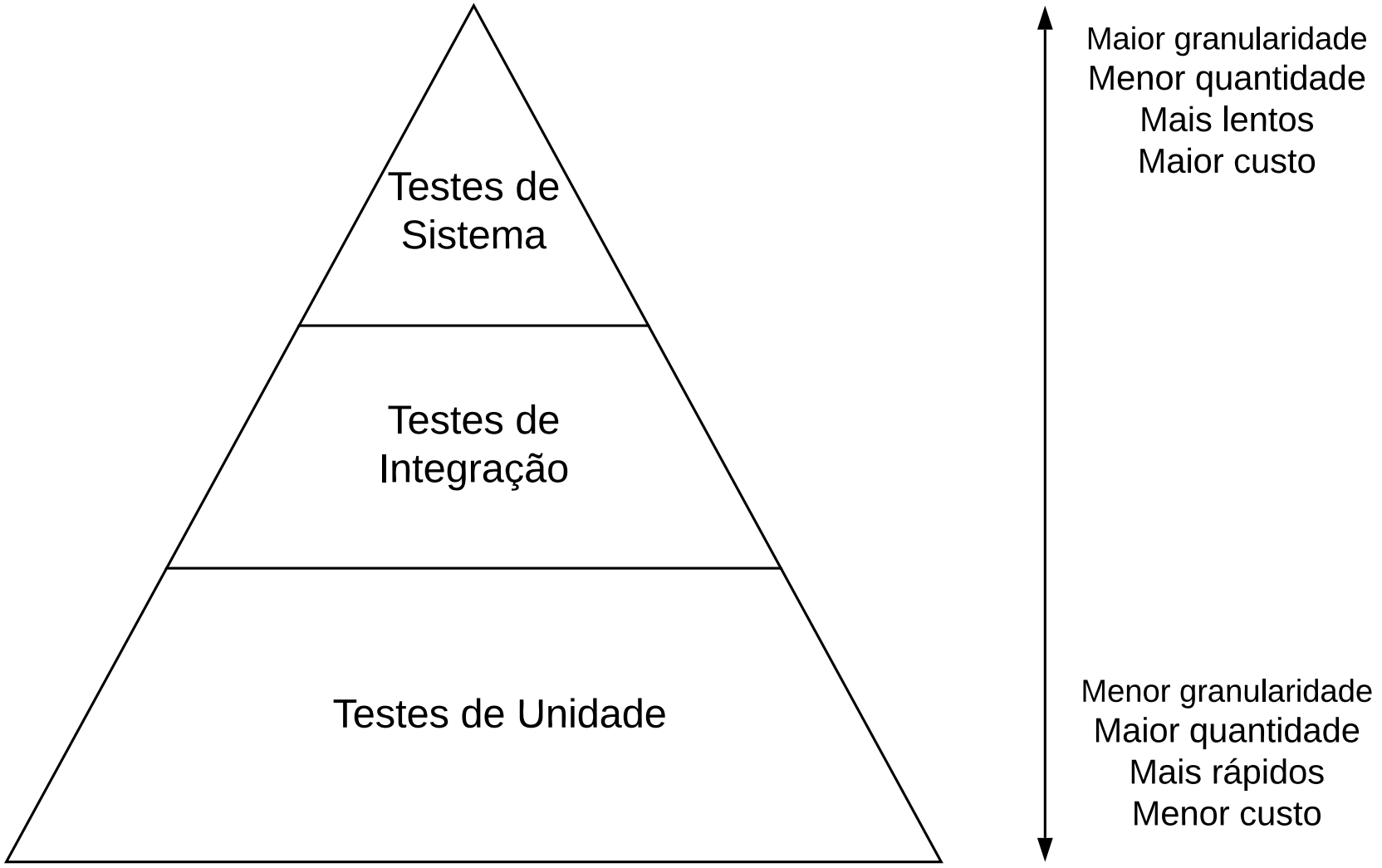
- *Stub*
  - Quando um componente A que vai ser testado chama operações de outro componente B, que ainda não foi implementado, pode-se criar uma **implementação simplificada** de B, que deve retornar valores.
- *Driver*
  - Quando B é o modulo que está implementado, mas o módulo A (que chama as funções de B) não está, deve ser implementada uma simplificação de B.

Teste de Sistema

- Uma vez integradas todas as partes, inicia-se o teste de sistema.
- Quando realizado por uma equipe de testes, o objetivo é verificar se as funcionalidades contidas na especificação de requisitos foram corretamente implementadas.
- Quando realizado por usuários, o objetivo é validar o sistema (Teste de Aceitação).
- É uma boa prática que essa fase seja realizada por testadores independentes.
- Geralmente emprega teste funcional
  - Pode-se usar o diagrama de casos de uso como fonte de funcionalidades.

Teste de Aceitação

- Testes funcionais, realizados pelo usuário, objetivando demonstrar a conformidade com os requisitos do software.
- É útil para aproximar o cliente final do resultado esperado pelo sistema.
- Os testes podem ser de duas categorias:
  - **Alfa**: Feitos por usuários, geralmente nas instalações do desenvolvedor. Observam e registram/problemas.
  - **Beta**: Feitos por usuários, geralmente em suas próprias instalações, sem supervisão do desenvolvedor. Problemas detectados são então relatados ao desenvolvedor.



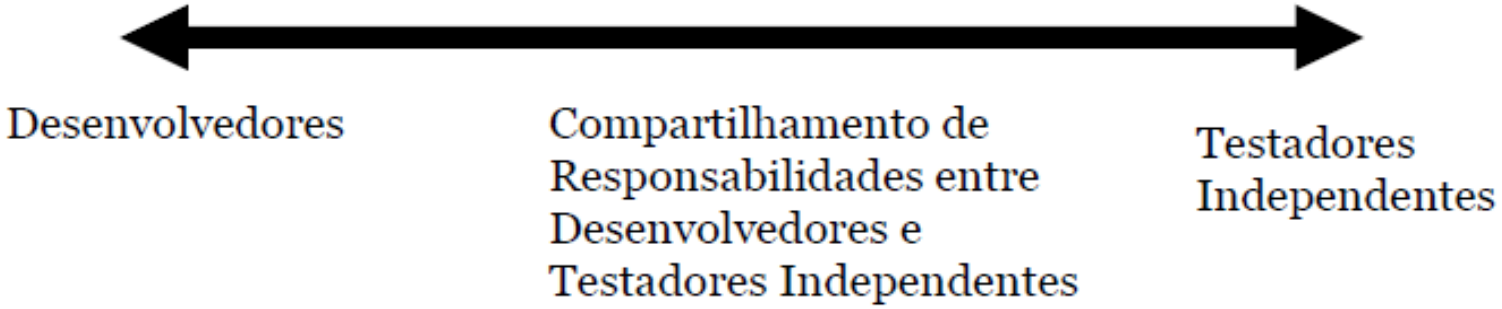
https://dl.acm.org/doi/book/10.5555/1667109

Processo de Teste

- Independentemente da fase de teste, o processo de teste inclui as seguintes atividades:
  1. Planejamento de Teste
  2. Projeto de Casos de Teste
  3. Implementação de Casos de Teste
  4. Execução
  5. Análise

Processo de Teste
1. Planejamento do teste

- Quem deve realizar os testes?



Processo de Teste
1. Planejamento do teste

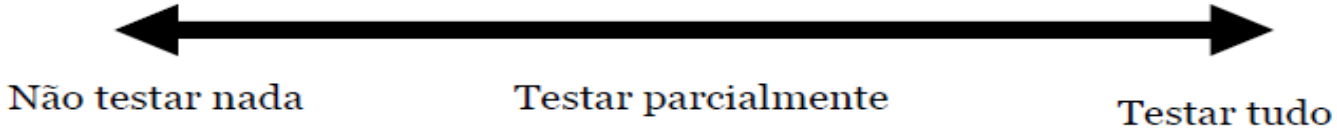
- **Desenvolvedor**:
  - Código é resultado de seu trabalho. Logo, procurar defeitos é, de certo modo, a destruição do mesmo, e o próprio desenvolvedor não tem motivação psicológica para projetar casos de teste que demonstrem que seu produto tem defeitos (dissonância cognitiva).
  - Dificilmente o desenvolvedor consegue criar um caso de teste que rompa com a lógica de funcionamento de seu próprio código (Koscianski e Soares, 2006).
- **Testador Independente**:
  - Assume a “perspectiva de teste”.
  - Pode representar papéis distintos (testador de unidade, de integração e de sistema).
  - Ser responsável por todos os testes, em todas as fases, pode tornar o processo lento e pouco produtivo.

Processo de Teste
1. Planejamento do teste

- Compartilhamento de Responsabilidades:
  - Divisão por Fases:
    - Desenvolvedores testam unidades, muitas vezes em paralelo com a implementação das mesmas.
    - Testadores independentes testam integração e sistema.
  - Divisão por Atividades do Processo de Teste:
    - Testadores independentes são responsáveis pelo Planejamento, Análise e Projeto de Casos de Teste
    - Desenvolvedores são responsáveis pela construção e execução dos casos de teste.

Processo de Teste
1. Planejamento do teste

- O que testar? Que partes devem ser mais cuidadosamente testadas?



Processo de Teste
1. Planejamento do teste

- **Princípio de Pareto**
  - Nos custos de desenvolvimento de software:
    - "80% do esforço de desenvolvimento (em tempo e recursos) produz 20% do código, enquanto o restante (80%) é produzido com apenas 20% do esforço" .
  - Nos testes de software:
    - "80% das falhas de software é gerado por 20% do código do software, enquanto os outros 80% gera apenas 20% das falhas".

Processo de Teste
1. Planejamento do teste

- Esse princípio sugere que os esforços devem ser **concentrados nas partes mais importantes e/ou frágeis**.
- Um bom teste é ao mesmo tempo econômico e encontra o máximo de defeitos (Koscianski e Soares, 2006).
- Análise de Requisitos de Teste:
  - Quais as partes mais importantes?
  - Quais as mais frágeis?
  - Enfim, que partes devem ser mais cuidadosamente testadas?

Processo de Teste
1. Planejamento do teste

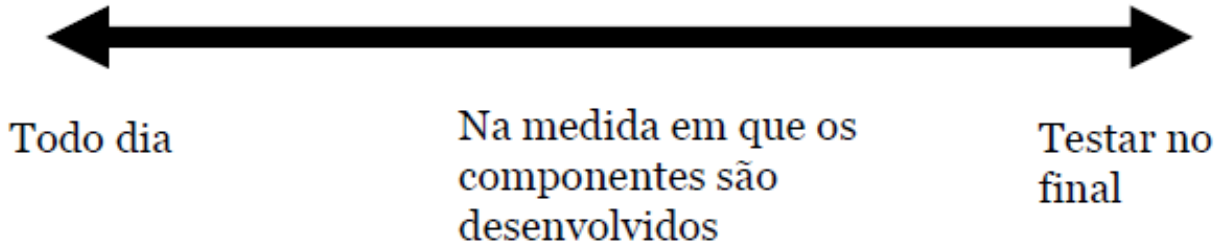
- Quanto teste é adequado?

Processo de Teste
1. Planejamento do teste

- É impossível testar tudo.
- Testes podem somente mostrar a presença de erros, mas não a sua ausência.
- Cobertura de Teste pode ser medida pelo menos de duas maneiras (McGregor e Sykes, 2001):
  - Em relação a requisitos: quanto dos requisitos especificados deve ser testado? Considerar requisitos funcionais e não funcionais.
  - Em relação à execução de linhas de código (teste de caminhos possíveis).

Processo de Teste
1. Planejamento do teste

- Quando testar?



Processo de Teste
1. Planejamento do teste

- Teste como uma atividade do processo linear de software (testar no final) x Teste como um processo paralelo ao processo de desenvolvimento (teste todo dia)

- Planejar, analisar e projetar testes à medida que o processo de desenvolvimento progride.
- Utilizar informações do ciclo de vida (incrementos) e dos requisitos (casos de uso) para definir o cronograma de testes.

Processo de Teste
1. Planejamento do teste

Cronograma de teste:

- **Teste de Unidade**: depende da produção das unidades pelo processo de desenvolvimento.
- **Teste de Integração**: depende do ciclo de vida (incrementos / iterações) e da estrutura (subsistemas / casos de uso). Pode ser programado em intervalos específicos.
- **Teste de Sistema**: sua execução deve ocorrer nas entregas e, portanto, depende do ciclo de vida.

Processo de Teste
1. Planejamento do teste

- Como qualquer atividade de planejamento, planejar testes envolve a **gerência de riscos** e a realização de **estimativas de esforço**, recursos (pessoas, equipamentos, software), tempo e custo.
- Fatores que afetam as estimativas / riscos:
  - Software sendo testado: tipo de software, plataforma de implementação etc.
  - Equipamentos e software requeridos para o teste
  - Modelo organizacional (testadores x desenvolvedores)
  - Nível de Cobertura.
- Usar dados históricos.

Processo de Teste
1. Planejamento do teste

- Como qualquer atividade de planejamento, um **plano de testes** deve ser elaborado, descrevendo o escopo, o processo de teste definido, os recursos alocados, estimativas, cronograma e riscos.

- Exemplo de Modelo de Plano de Teste: Padrão IEEE 829-1998.

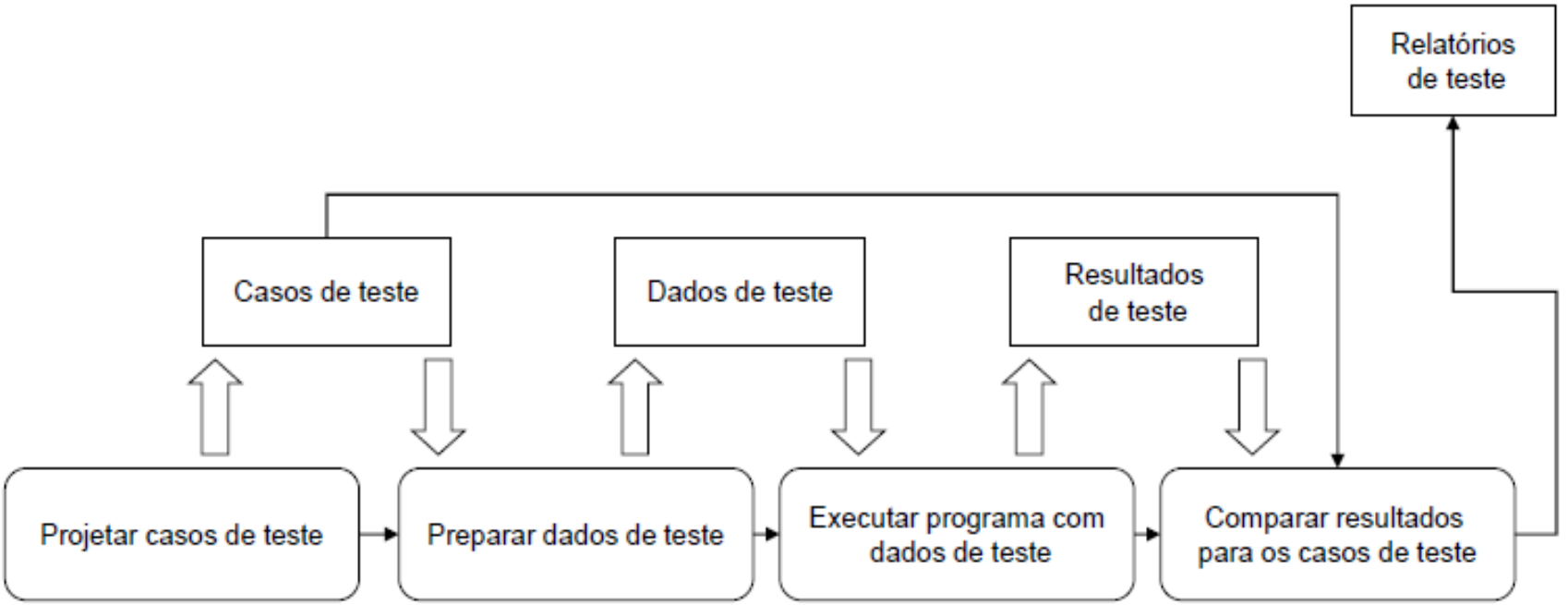
Processo de Teste
1. Planejamento do teste

- Identificador único para o plano de testes.
- Introdução
- Itens: identifica os itens a serem testados
- Funcionalidades: identifica as funcionalidades que serão testadas ou não, bem como os motivos
- Processo: especifica as principais atividades, técnicas e ferramentas
- Critérios de aceite: especifica os critérios de aceite para cada item.
- Suspensão: especifica os critérios para suspender parte ou todo o processo de teste

Processo de Teste
1. Planejamento do teste

- Produtos: identifica os artefatos produzidos pelo processo de teste (planos, casos de teste, relatórios, etc.).
- Tarefas: detalhamento e dependência entre as atividades
- Ambiente: identifica as necessidades do ambiente de teste (hardware e software).
- Responsabilidades: identifica os responsáveis pelas atividades, pelo ambiente de teste e pelos itens de teste.
- Treinamento: especifica as necessidades de treinamento e identifica opções.
- Cronograma.
- Riscos: plano de riscos.
- Aprovações: especifica os nomes e cargos dos responsáveis por aprovar o plano.

Processo de Teste
1. Planejamento do teste



Caso de Uso: Alterar Senha de Usuário

Fluxo Principal:

P 1 – Usuário seleciona o menu "Alterar Senha"
P 2 – O sistema gera um código aleatório
P 3 – O sistema carrega a tela "Alteração de Senha"
P 4 – Usuário entra com sua identificação, a senha atual, a nova senha, a confirmação da nova senha e o valor do código aleatório (A1)
P 5 – O usuário confirma a alteração (A2)
P 6 – O sistema valida a senha. (E1) (E2)
P 7 – O sistema emite a mensagem de indicação de sucesso
P 8 – O caso de uso é finalizado

Fluxo Alternativo 1: O usuário não visualiza código aleatório

A 1.1 – O usuário seleciona opção "Não consegui visualizar o código"
A 1.2 – O sistema retorna ao fluxo principal (P2)

Fluxo Alternativo 2: O usuário cancela a alteração

A 2.1 – O usuário seleciona a opção cancelar
A 2.2 – O sistema cancela a operação
A 2.3 – O caso de uso é finalizado

Fluxo de Exceção 1: Confirmação de senha nova não confere com a mesma

E.1.1 – O sistema identifica que a nova senha fornecida e a confirmação da mesma, não conferem.
E.1.2 – O sistema emite a mensagem "A confirmação da Nova Senha não confere."
E.1.3 – O sistema retorna ao fluxo principal (P4) para entrada da confirmação da senha nova do usuário.

Fluxo de Exceção 2: Campo Requerido Não Fornecido ou Inválido

E.2.1 – Usuário não entra com campo requerido
E.2.2 – O sistema emite a mensagem "Campo requerido ausente ou inválido."
E.2.3 – O caso de uso é finalizado

Cenário	Caso de Teste	Entradas	Resultado Esperado
Cenário 1	CT1 – Abertura da tela de alteração de senha	Seleção do menu "Alterar Senha"	Tela com a imagem de um código aleatório e campos: identificação, senha atual, nova senha, a confirmação da nova senha e valor do código aleatório
	CT2 – Alteração de senha com sucesso	Identificação = CPF cadastrado; Senha atual = válida; Nova senha e Confirmação = a1b2c3; Código = válido	Alteração de senha efetivada no banco e mensagem de indicação de sucesso
Cenário 2	CT3 – Regerar o código aleatório	Seleção da opção "Não consegui visualizar o código"	Novo código aleatório gerado.
Cenário 3	CT4 – Cancelar alteração de senha	Seleção da opção Cancelar	Cancelamento da operação de alteração de senha
Cenário 4	CT5 – Confirmação de senha diferente da nova senha	Identificação = CPF cadastrado; Senha atual = válida; Nova senha = a1b2c3; Confirmação: a1b2c4; Código = válido	Mensagem "A confirmação da Nova Senha não confere."
Cenário 5	CT6 – Identificação inválida	Identificação = CPF não cadastrado; Senha atual = válida; Nova senha e Confirmação = a1b2c3; Código = válido	Mensagem "Campo requerido ausente ou inválido."
	CT7 – Senha atual inválida	Identificação = CPF cadastrado; Senha atual = inválida; Nova senha e Confirmação = a1b2c3; Código = válido	Mensagem "Campo requerido ausente ou inválido."
	CT8 – Código aleatório diferente do exibido	Identificação = CPF cadastrado; Senha atual = válida; Nova senha e Confirmação = a1b2c3; Código = inválido	Mensagem "Campo requerido ausente ou inválido."
	CT9 – Identificação ausente	Identificação = Não preencher; Senha atual = válida; Nova senha e Confirmação = a1b2c3; Código = válido	Mensagem "Campo requerido ausente ou inválido."
	CT10 – Senha atual ausente	Identificação = CPF cadastrado; Senha atual = Não preencher; Nova senha e Confirmação = a1b2c3; Código = válido	Mensagem "Campo requerido ausente ou inválido."
	CT11 – Nova senha ausente	Identificação = CPF cadastrado; Senha atual = válida; Nova senha = Não preencher; Confirmação = a1b2c3; Código = válido	Mensagem "Campo requerido ausente ou inválido."
	CT12 – Confirmação de senha ausente	Identificação = CPF cadastrado; Senha atual = válida; Nova senha = a1b2c3; Confirmação = Não preencher; Código = válido	Mensagem "Campo requerido ausente ou inválido."
	CT13 – Código aleatório ausente	Identificação = CPF cadastrado; Senha atual = válida; Nova senha = a1b2c3; Confirmação = a1b2c3; Código = Não preencher	Mensagem "Campo requerido ausente ou inválido."

Automatização do Processo de Teste

- O processo de teste tende a ser extremamente **dispendioso** e é muito importante **utilizar ferramentas de apoio ao teste** para buscar aumentar a produtividade.
- A automação de testes tem o objetivo de reduzir o envolvimento humano em atividades manuais repetitivas.
- O grande **benefício** da automação de testes não é a execução dos testes mais rápida e a qualquer hora do dia ou da noite, mas o aumento da amplitude e profundidade da cobertura dos testes.
- Há diversos tipos de ferramentas de teste.

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.
- http://istqbexamcertification.com/what-is-validation-in-software-testing-or-what-is-software-validation/
- 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.
- •Brazilian Software Test Qualification Board. http://www.bstqb.org.br/