pvh( );

INSTITUTO FEDERAL
GOIÁS

Instituto Federal de Educação, Ciência e Tecnologia
Campus Inhumas

TESTES DE SOFTWARE

Visão Geral de Testes de Software
Tópico 2 do plano de ensino

Prof. Me. Victor Hugo Lázaro Lopes

AGENDA

3. Processo de teste de software;
- **3.1 Axiomas & conceitos-chave;**
- **3.2 Culturas de teste;**
- **3.3 Tipos de teste, modelos de teste;**
- **3.4 Desenvolvimento orientado a testes;**
- **3.5 Ciclo de vida de testes, Planejamento de Teste & Plano de testes;**
- **3.6 O curso geral de um teste: planejamento, execução e avaliação;**
- **3.7 Gerenciamento do processo de testes;**
- **3.8 Extremme Programming e Testes de Software;**

Alguns axiomas:

- **É impossível testar um programa completamente**

Um testador iniciante pode pensar que sim e ficar frustado. Quatro razões-chave:

- O número de combinações de entradas é enorme;
- O número de combinações de saídas é enorme;
- O número de caminhos através do software é enorme;
- Especificação de software é subjetiva, em que o “bug” esteja nos olhos de quem vê.

Exemplo simples: para testar uma calculadora básica de um SO, existem infinitas combinações. Para testar uma soma simples, podemos testar 1 + 1 e não dar erro, mas podemos testar a soma 1,99 + 3,3439 e ocorrer um erro.

Alguns axiomas:

- **Teste de software é um exercício baseado em risco**

Se você decide não testar todas as possibilidades do cenário, você está assumindo um risco.

Deve haver um ponto ótimo entre quantidade de “bugs” existentes eliminados e o custo do teste.

O ponto ótimo de teste representa o risco aceitável ao menor custo possível.

Alguns axiomas:

- **Teste não mostar que “bugs” não existem**

Você é um exterminador encarregado de examinar a casa à procura de insetos. Você inspeciona a casa e encontra evidências de insetos: insetos vivos, insetos mortos e ninhos de insetos.

Você visita outra casa e não encontra insetos. Você olha nos locais padrões e não vê sinais de insetos. Você pode achar insegos mortos, ou ninhos de mortos, mas não necessariamente evidências de que insetos existem.

Você pode afirmar, categoricamente, que não existem insetos na casa examinada?

Alguns axiomas:

- **Quanto mais “bugs” você acha, mais “bugs” existirão**

Programadores têm dias ruins, gerando código-fonte com possibilidade de erros.

Programadores, enquanto pessoas, cometem os mesmos erros, pois possuem hábitos.

Alguns bugs indicam a ponta do *iceberg*.

Alguns axiomas:

- **O paradoxo do pesticida**

Certos “bugs” adquirem resistência ao “pesticida” utilizado, em que quanto mais você conserta e testa, piores se tornam.

- **Nem todos “bugs” encontrados serão consertados**

Bug encontrado e não corrigido torna-se uma parte do risco, em que o desafio é escolher o risco mais conveniente ao projeto.

Conceitos-Chave

- **Precisão e Acurácia**

Quando se testa algo, a resposta que retorna tem precisão ou acurácia? Ambos?

Precisão: proximidade entre os valores obtidos pela repetição do processo de mensuração.

Acurácia: proximidade da medida relativamente ao verdadeiro valor da variável.

Conceitos-Chave

- **Precisão e Acurácia**

Vejamos um simples jogo de dardos, cujo o objetivo primário é acertar o centro do alvo na parede. O teste executa 3 lançamentos, com os possíveis resultados:

- Os 3 dardos atingem o alvo, mas não em seu centro, ao seu redor, em locais diferentes: não há precisão nem acurácia.
- Os 3 dardos não atingem o alvo, porém atingem num mesmo lugar exato fora do alvo: há precisão, mas não acurácia.
- Os 3 dardos atingem o alvo, mas não em seu centro. Eles atingem lugares muito próximos ao centro: há acurácia, mas sem precisão.
- Os 3 dardos atingem o alvo em seu centro e no mesmo lugar exatamente: há precisão e acurácia.

Qualquer software precisa de precisão e acurácia: acertar, independente da situação, os mesmos valores (precisão) e os valores desejados (acurácia).

Conceitos-Chave

- **Qualidade e Confiabilidade**

Qualidade: grau de excelência. Se um referido produto possuir alta qualidade, isto quer dizer que vai de encontro com as necessidades do usuário.

Confiabilidade é a medida de quão confiável é o produto, isto é, o quanto ele é bom em não “danificar” dados ou processos em seu uso.

Culturas de Teste

Percebe-se que as culturas de cada empresa se classificam dentro de uma faixa:

- **Cultura do OBA-OBA**

É a visão do programador que pensa e diz para si mesmo: “para que melhorar, se daqui a alguns anos vou me aposentar?”, ou “para que melhorar logo, se ainda tenho muitos anos pela frente?”.

Nestes ambientes imaturos, implantar teste é um desafio ainda maior, no qual os envolvidos representarão resistência: anticorpos organizacionais.

Culturas de Teste

Percebe-se que as culturas de cada empresa se classificam dentro de uma faixa:

- **Cultura do Pensamos Que Sabemos**

Bastante comum. Tem alguma coisa de teste implementado, mas em vez de ser a favor passa a ser contra, porque os processos passam a ser mais uma "pedra no meio do caminho", e não uma ponte para a qualidade do produto.

Pior: pensam que estão no caminho certo.

Culturas de Teste

Percebe-se que as culturas de cada empresa se classificam dentro de uma faixa:

- **Cultura do Sabemos Que Não Sabemos**

Tem consciência de suas limitações e onde quer chegar, e está no caminho certo. Quando começa a implementar testes há um salto significativo e quantitativo na qualidade das aplicações.

Culturas de Teste

Percebe-se que as culturas de cada empresa se classificam dentro de uma faixa:

- **Cultura do Sabemos Que Sabemos Pouco**

Possui certa maturidade, e que aprende com seus erros. Já possui processos de testes automatizados.

- **Cultura do Sabemos Que Sabemos Bem**

  Já tem um processo maduro de desenvolvimento e de testes de software mas se depara com a questão: Continuar se renovando ou continuar no caminho que tem dado certo? Se tirarmos uma foto vemos que o processo é bom, mas se tirarmos várias fotos no decorrer do tempo, vemos que pouco mudou.

Culturas de Teste

- **O paradoxo do pesticida**

Certos “bugs” adquirem resistência ao “pesticida” utilizado, em que quanto mais você conserta e testa, piores se tornam.

- **Nem todos “bugs” encontrados serão consertados**

Bug encontrado e não corrigido torna-se uma parte do risco, em que o desafio é escolher o risco mais conveniente ao projeto.

Tipos de teste

- **Teste de unidade:** Teste em nível de componente ou classe;
- **Teste de Integração:** Garante que as unidades funcionem quando combinados;
- **Teste de Sistema:** Testa a aplicação como um todo, funcionando conforme o esperado**;**
- **Teste Operacional:** testa a capacidade operacional do sistema;
- **Teste Negativo-Positivo:** Garante que a aplicação vai funcionar no “caminho feliz” de sua execução e vai funcionar no seu fluxo de exceção;
- **Teste de Regreção:** testa feito após inserir nova funcionalidade, em que retorna-se a versão anterior pra evitar os defeitos já conhecidos e corrigidos;
- **Teste Caixa Preta:** Teste do sistema sem se preocupar com o código e instruções internas;

Tipos de teste

- **Teste de Caixa Branca:** Objetiva testar o código e instruções internas;
- **Teste Beta:** Testar a aplicação em produção;
- **Teste de Verificação de Versão:** Verificações realizadas antes da liberação de uma nova versão;
- **Teste de Interface:** Verifica a navegabilidade e os objetos de tela;
- **Teste de Aceitação:** Teste exploratório voltado a validação dos desejos do usuário, dando aceite ou não;
- **Teste de Estresse:** Testar a aplicação em condições extremas e inesperadas;
- **Teste de Volume:** Testar a quantidade de dados envolvidos;
- **Teste de Configuração**: Testar se a aplicação funciona corretamente em diferentes ambientes de hardware e software;

Tipos de teste

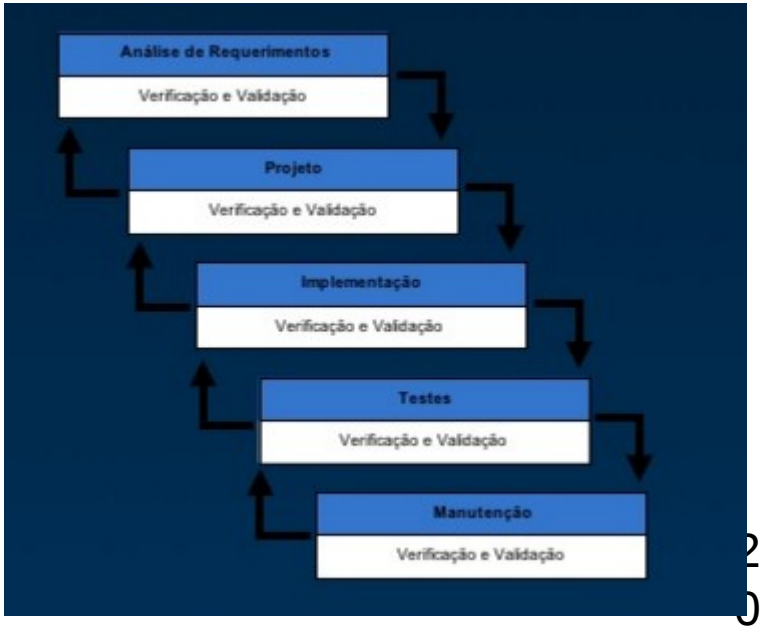
- **Teste de Documentação:** Avalia a documentação;
- **Teste de Integridade:** Testar a integridade dos dados armazenados;
- **Teste de Segurança:** Testar a segurança da aplicação nas mais diversas formas;
- **Monkey test:** o teste do macaco é de forma aleatória e inesperada, normalmente sem planejamento;

Modelos de Testes VS Metodologias

- **Modelo:** conjunto de técnicas que representa, ou que tenta representar algo da realidade ou algum conceito;
- **Metodologia:** conjunto de técnicas, métodos e estratégias voltadas para a criação e/ou execução de algo, que em geral pode ser um ou mais produtos ou serviços.
- Em testes, existem vários modelos de testes que visam otimizar, criar ou representar um processo de teste de software.

Modelos de Testes VS Metodologias

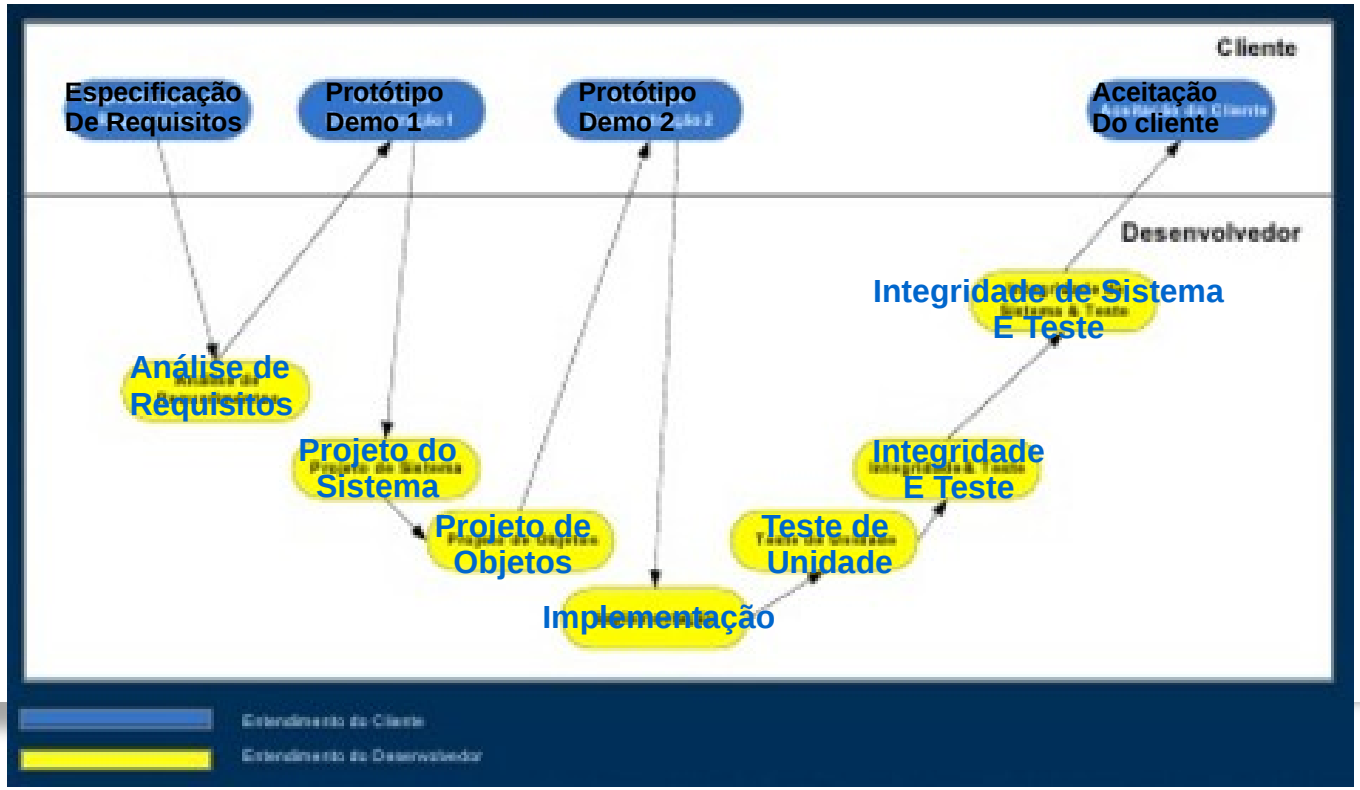
- **Modelo Waterfall (cascata):** baseado no modelo de software em cascata, em que cada tarefa é completamente concluída para poder seguir à próxima, e se for preciso retornar à etapas anteriores, deve-se repetir todas na sequência.
  Nos teste de software, este modelo inclui verificação e validação após cada uma das etapas.



Modelos de Testes VS Metodologias

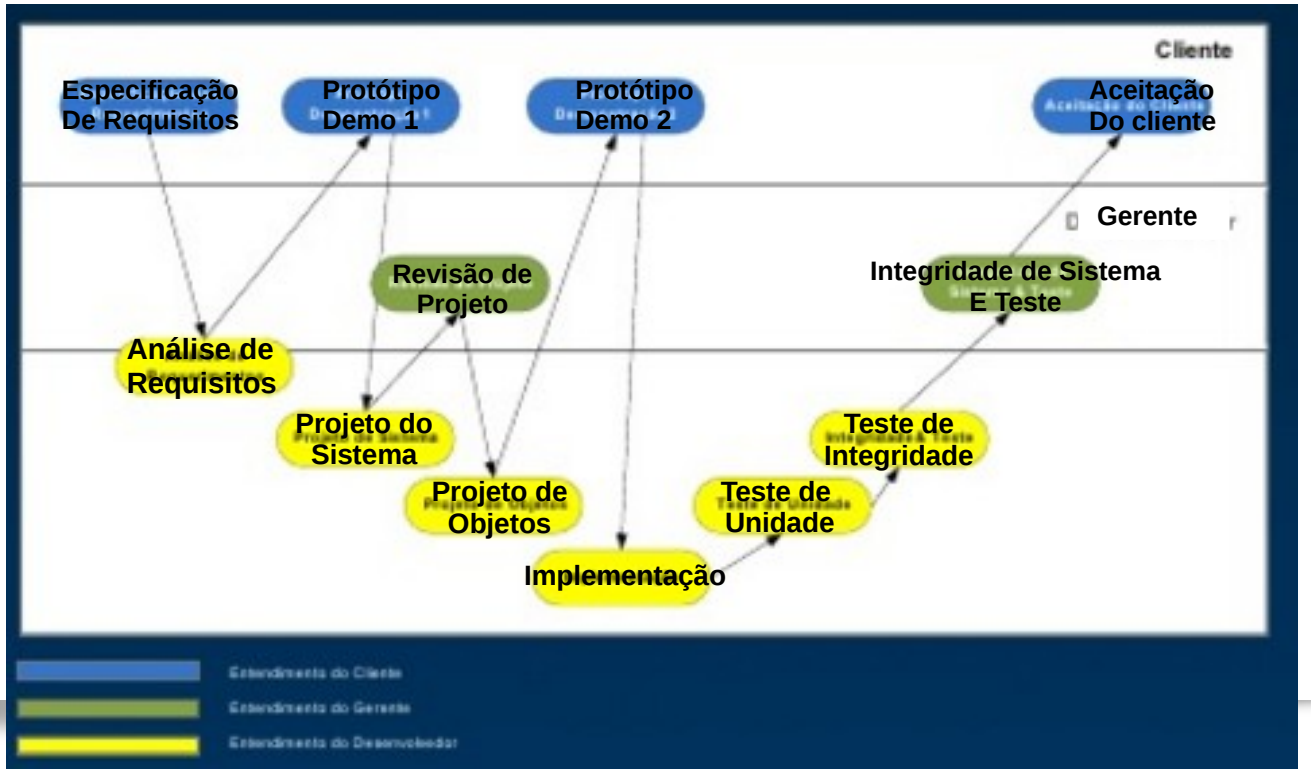
Modelo SawTooth (serra dentada): Mostra a interação entre o usuário e o desenvolvedor. O enfoque do desenvolvedor é simples, atuando semelhante a um pedreiro que faz uma obra numa casa, executando sob encomenda.

Ponto fraco: o programador sobrecarregado deixa a qualidade de lado.



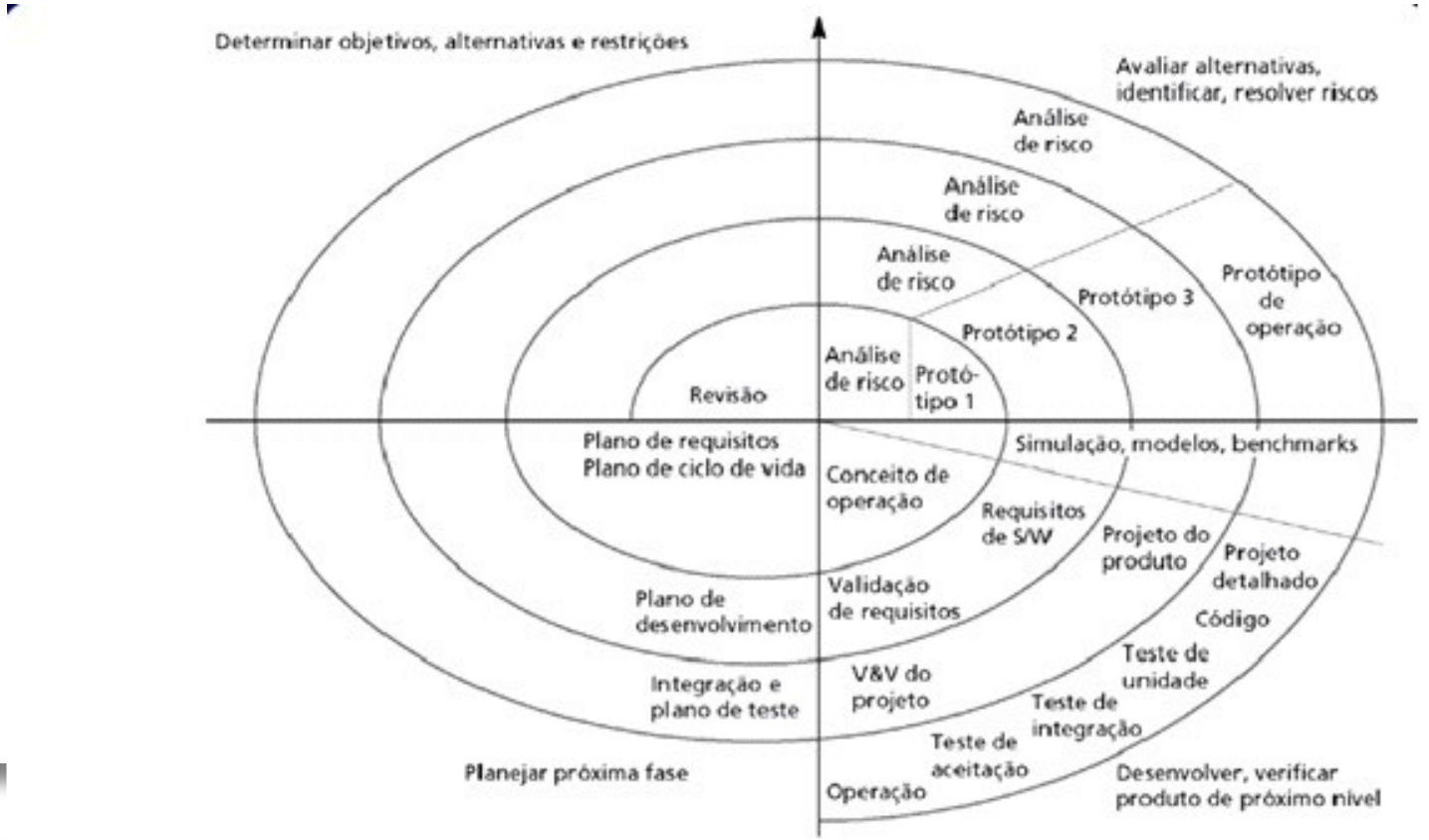
Modelos de Testes VS Metodologias

Modelo SharkTooth: Semelhante ao SawTooth, mas com a participação de um gerente revisando o processo, minimizando o gap de padrões de desenvolvimento e qualidade.



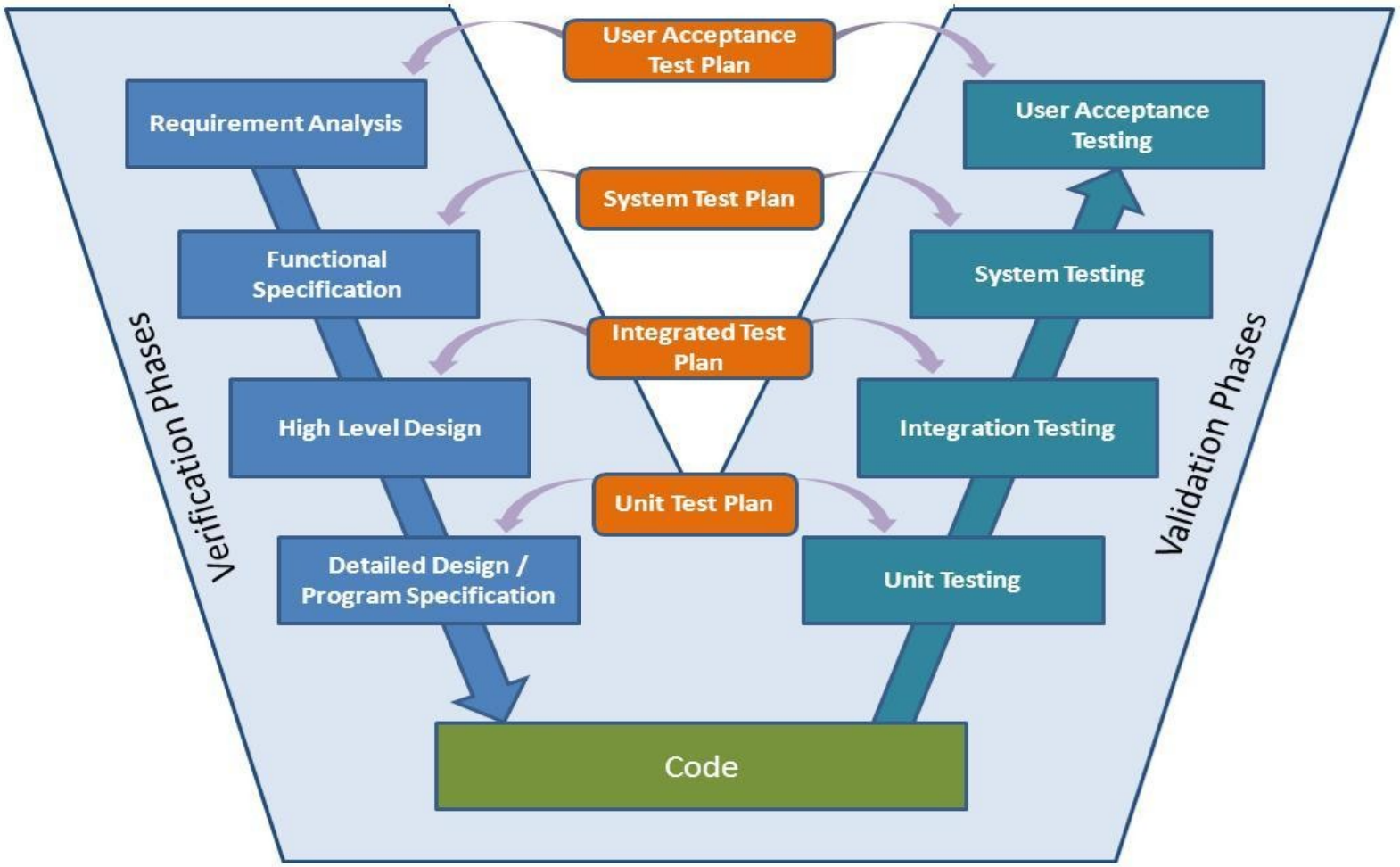
Modelos de Testes VS Metodologias

Modelo Espiral: é cíclico e fa uso da prototipação, os testes estão devidamente explicitados. Plano de testes é posterior ao projeto do sistema.



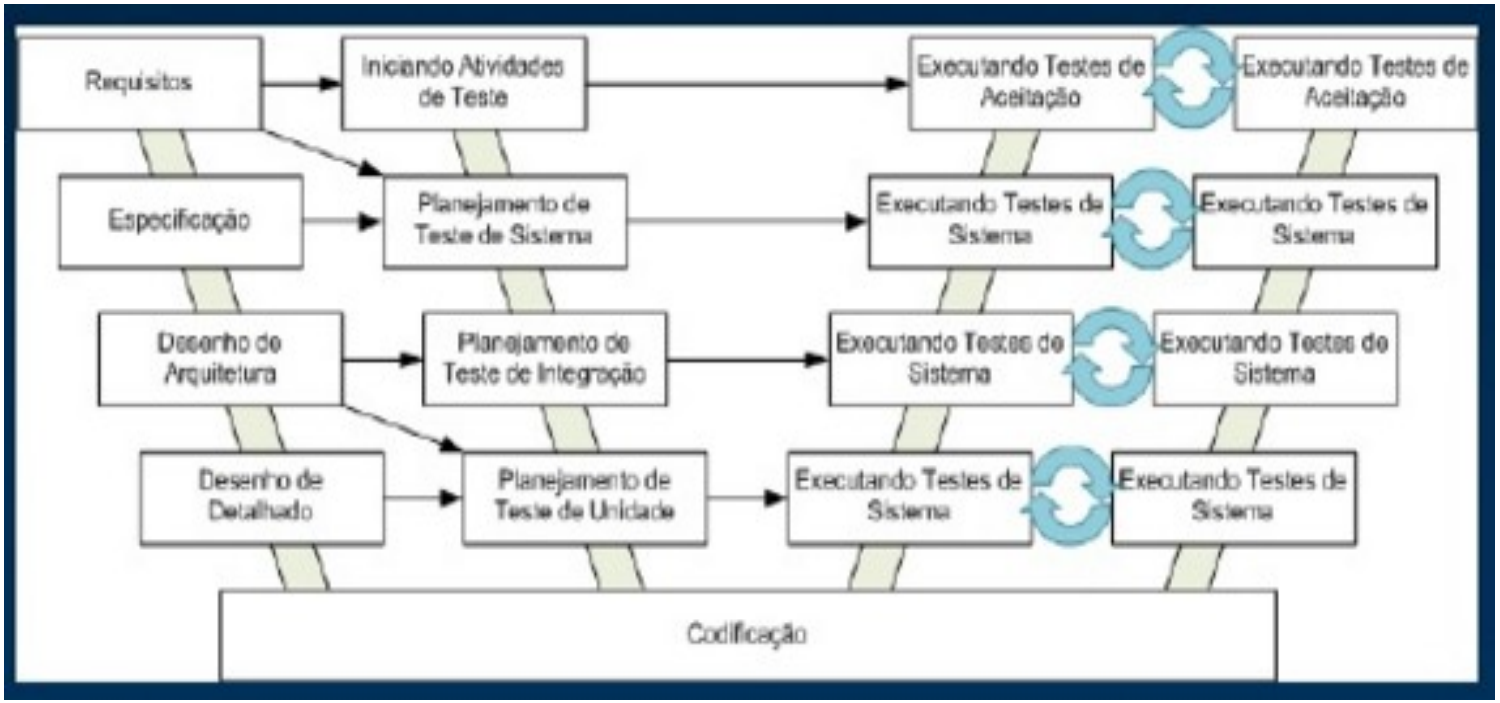
Modelos de Testes VS Metodologias

Modelo V: descreve graficamente as fases individualmente em forma de “V”, que denota Verificação (atende aos requisitos F e NF?) e Validação (atende às necessidades do cliente?).



Modelos de Testes VS Metodologias

Modelo W: baseado no modelo V. Permite uma melhor eliminação dos bugs desde o início, pois para cada etapa do V teríamos uma etapa de testes. Foca na maximização dos testes.



Teste Manual x Teste Automatizado

- Há um consenso entre vários consultores especialistas de que o melhor caminho é a automação dos testes.
  - Já pensou alocar 1000 pessoas para fazer apenas um simples teste de carga na aplicação?

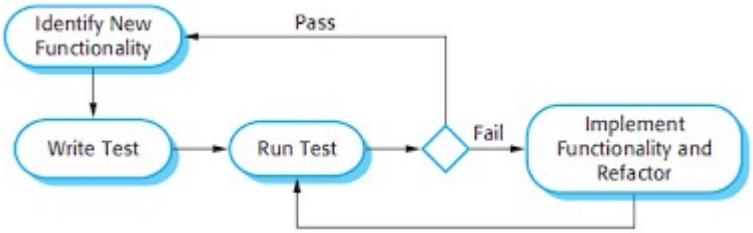
Projetos de automação de testes devem ser implementados com muito cuidado. A seleção da ferramenta de automação de testes, bem como o que deverá ser automatizado ou não, deve ser analisada de forma clara.

Teste Manual x Teste Automatizado

- Quando a Automação faz sentido?
  - Testes que usam técnicas de regressão ou de confirmação, gerando grande repetição de tarefas;
  - Quando faz uso de testes aleatórios que utilizam caminhos aleatórios, com grande quantidade de dados de testes;
  - Testes de carga, volume, capacidade;
  - Performance e confiabilidade;
  - Quando qualquer tipo de teste exiga a repetição contínua.

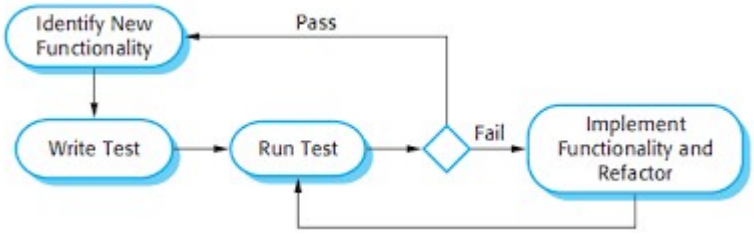
Desenvolvimento Orientado(Digido) a Testes - TDD

- Test-Driven Development é uma abordagem para o desenvolvimento de programas em que se intercalam testes e desenvolvimento (SOMMERVILLE, 2011).
- Você não caminha para o próximo incremento até que o código desenvolvido passe no teste.



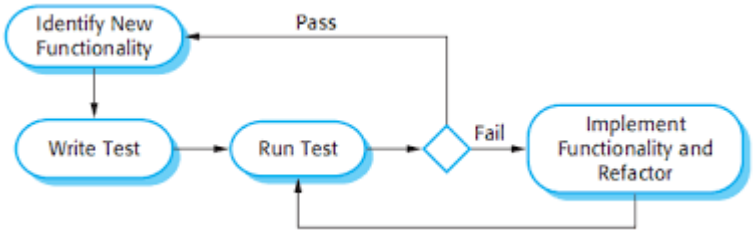
Desenvolvimento Orientado(Digido) a Testes - TDD

- Começa identificando o incremento de funcionalidade necessário. Deve ser pequeno e implementável em poucas linhas de código (idealmente);
- Escreve um teste para essa funcionalidade e o implementa como um teste automatizado. Isto significa que o teste pode ser executado e relatará se passou ou falhou;
- Executa-se o teste, junto com todos os outros testes implementados. Como a implementação do incremento não ocorreu, haverá falha proposital, mostrando que o teste acrescentará algo ao conjunto de testes;
- Implementa-se o incremento e executa novamente o teste. Isso pode envolver refatoração do código existente;
- Depois que todos os testes forem executados com sucesso, você caminha para implementar a próxima parte da funcionalidade.



Desenvolvimento Orientado(Digido) a Testes - TDD

- Benefícios apontados na literatura:
  - Ajuda os programadores a clarear suas ideias sobre o que um segmento de código supostamente deve fazer, pois força-o a pensar no resultado final da implementação antes dela efetivamente ocorrer;
  - Cobertura de código: todo segmento de código que você escreve dever ter pelo menos um teste associado;
  - Depuração simplificada: quando um teste falha, a localização do problema deve ser óbvia, não sendo necessário uso de recursos para localizar o problema;
  - Documentação de sistema: os testes em si agem como uma forma de documentação que descreve o que o código deve estar fazendo. Ler os teste torna mais fácil a compreensão do código;
  - Revelou-se uma abordagem de sucesso para projetos de pequenas e médias empresas, com relatos de programadores satisfeitos. Nenhum relato de TDD levar à redução de qualidade.



Ciclo de vida de teste de software

- O ciclo de vidas de teste de software é composto das etapas:
  - **Planejamento**: se estabelece o que vai ser testado, em quanto tempo e em que momento os testes serão interrompidos;
  - **Preparação**: o objetivo é preparar toda a estrutura do ambiente de testes como: equipamentos, configuração de hardware e softwares usados (sistemas operacionais, browsers, etc.), criação da massa de dados de teste, pessoal, ferramentas de automação, entre outros;
  - **Especificação:** a atividade principal é elaborar e revisar os cenários e roteiros de testes;
  - **Execução:** executa-se os testes planejados e registrar os resultados obtidos;
  - **Entrega:** onde arquiva-se toda a documentação e descreve-se todas as ocorrências do projeto relevantes para a melhoria do processo.

Planejamento de testes

- Planejamento de testes é um processo, que visa gerar como artefato um plano de testes (MOLINARI, 2011).
- É algo crítico no processo de software de uma empresa.
- Situação nacional, segundo MCT: 25 % planeja seus testes de forma eficaz e formal.
- O planejamento facilita ou melhora quatro itens:
  - Organização, objetividade, previsibilidade e gerenciamento.
- Tem como entraves:
  - Perfil do gerente e elaborador de testes, prazos e a fragilidade dos requisitos.

Planejamento de testes

- Problemas em nível macro:
  - Indefinição de requisitos de testes;
  - Recursos para ferramentas de teste desprezados;
  - Prazo apertado do projeto impactando o projeto de testes;
  - Pouco conhecimento para o planejamento;
  - Planejamento pouco abrangente;
  - Planejamento de alto risco.

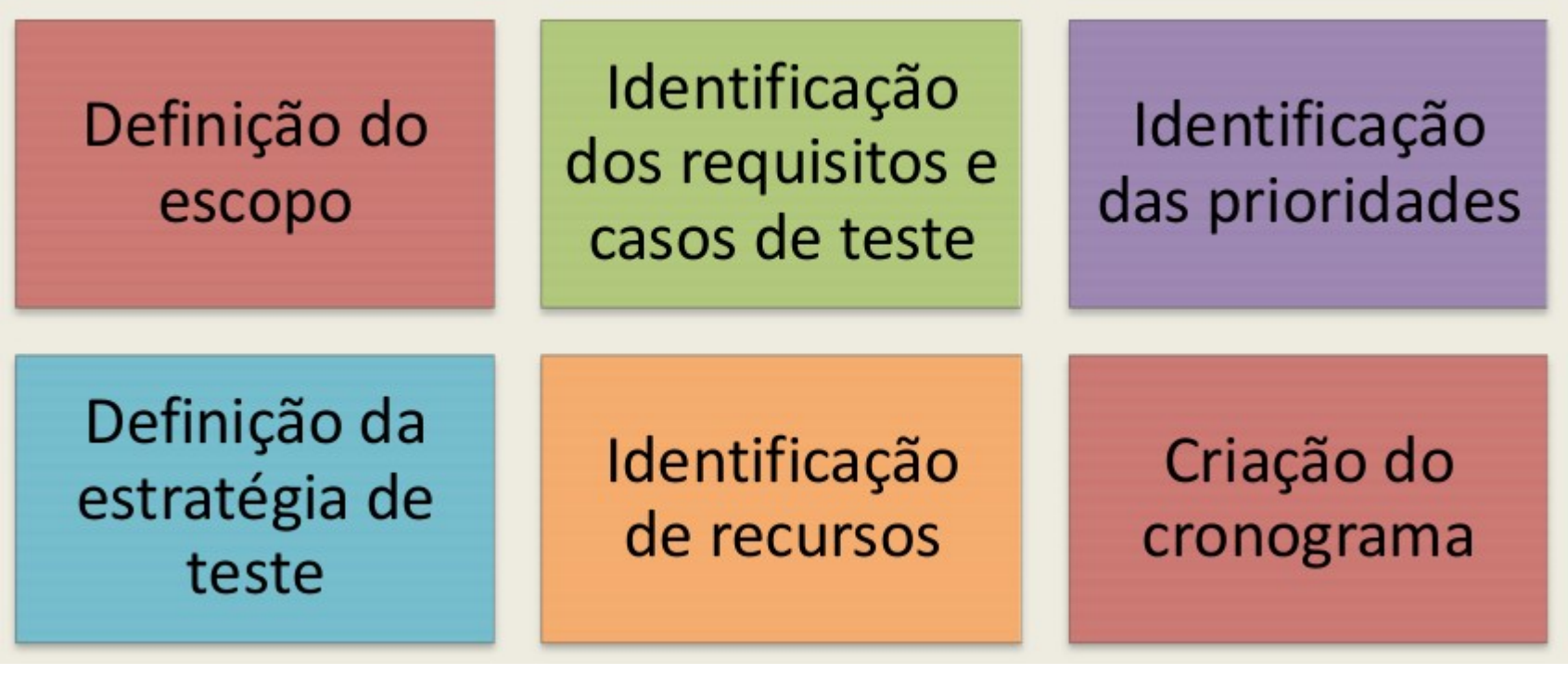
Planejamento de testes

- Conceitos importantes:
  - **Planejamento de testes**: é o processo em si de planejamento;
  - **Plano de testes**: artefato principal e fruto do planejamento;
  - **Requisitos de Teste**: é a meta ou aquilo que se quer testar na aplicação. Ex.: teste de inclusão de fotos em produtos de um e-commerce. Um importante requisito de teste seria testar o correto upload, manipulação pelo script e salvamento em pastas e BD.
  - **Casos de Teste**: são as situações de teste, ou o teste em si. Ex.: upload de foto em formato JPEG de grande formato; upload de arquivo em formato GIF de grande peso; upload de arquivos TXT. Um caso de teste pertence (deve) a um único requisito.

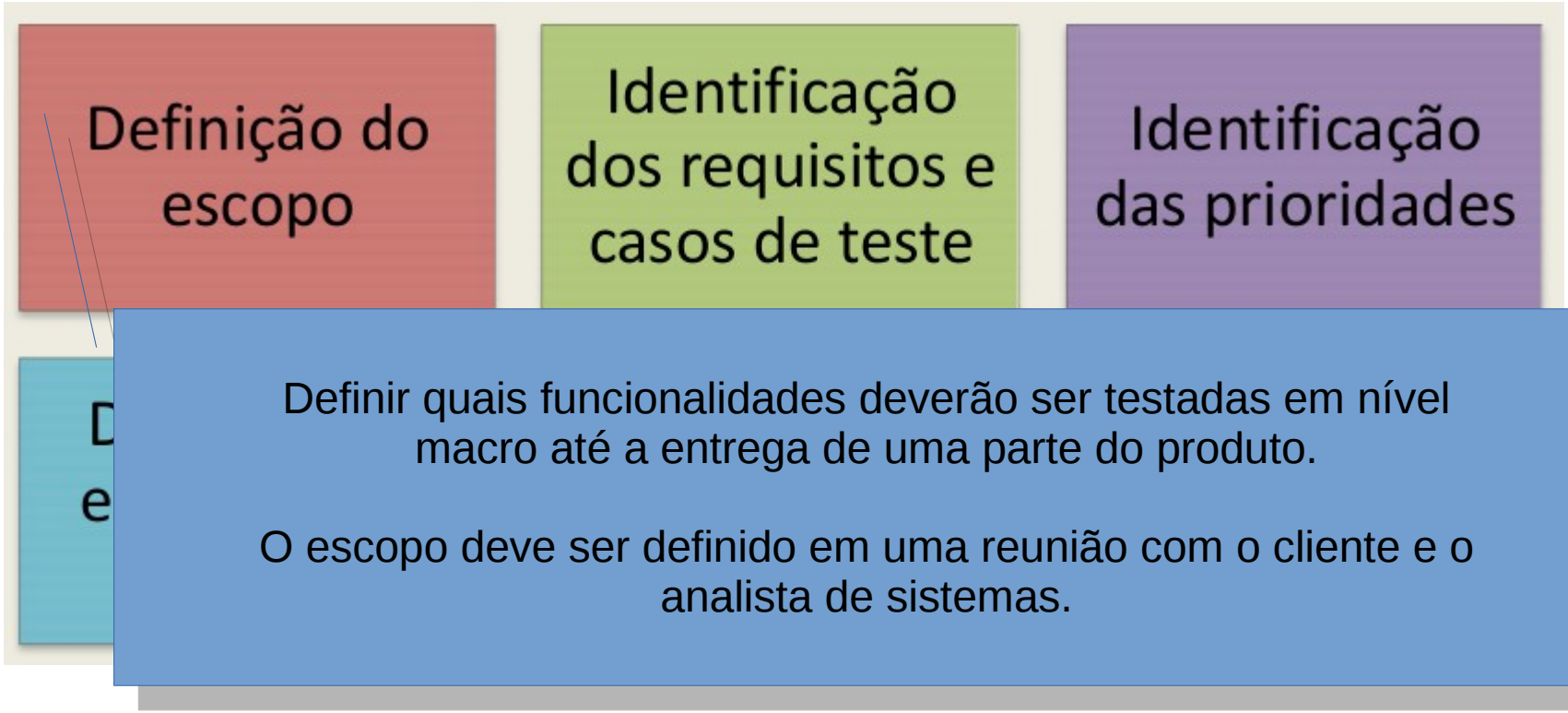
Planejamento de testes

- Conceitos importantes:
  - **Procedimentos de Testes**: são os passos necessários para realizar um teste. Podem refletir em um ou mais casos de testes ou requisitos de testes. Ex.: 1) Acessar o CMS; 2) selecionar a função de listagem de produtos; 3) clicar no icone “foto” do produto que fará o upload da foto; 4) selecionar a foto no computador local e clicar em enviar; 5) aguardar o upload completo e finalização do procedimento; 6) sair da aplicação;
  - **Script de Teste:** é a implementação do procedimento de teste através de uma ferramenta de teste automatizado;
  - **Ponto de Verificação:** checagem realizada no script para verificar se o teste foi bem sucedido. Ex.: verifique se a mensagem do upload foi “Foto enviada com sucesso”.

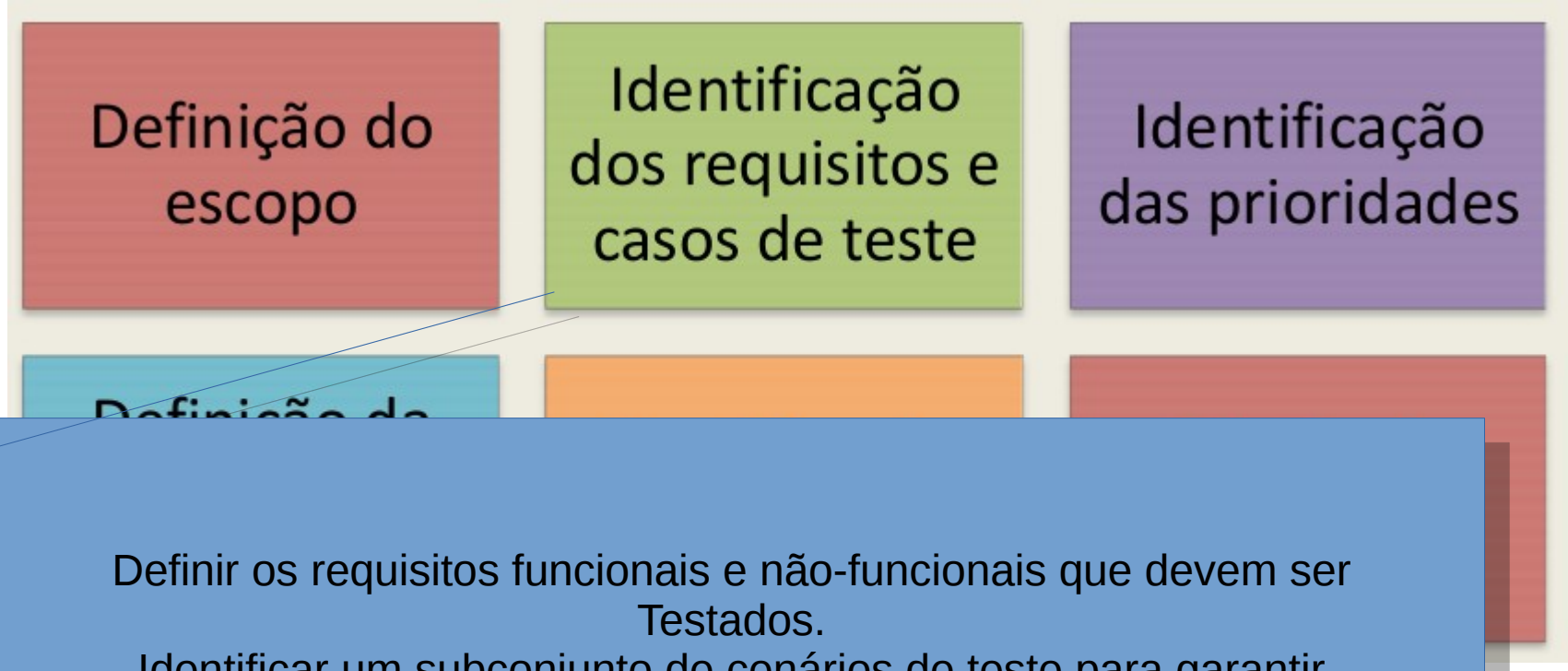
Planejamento de testes



Planejamento de testes

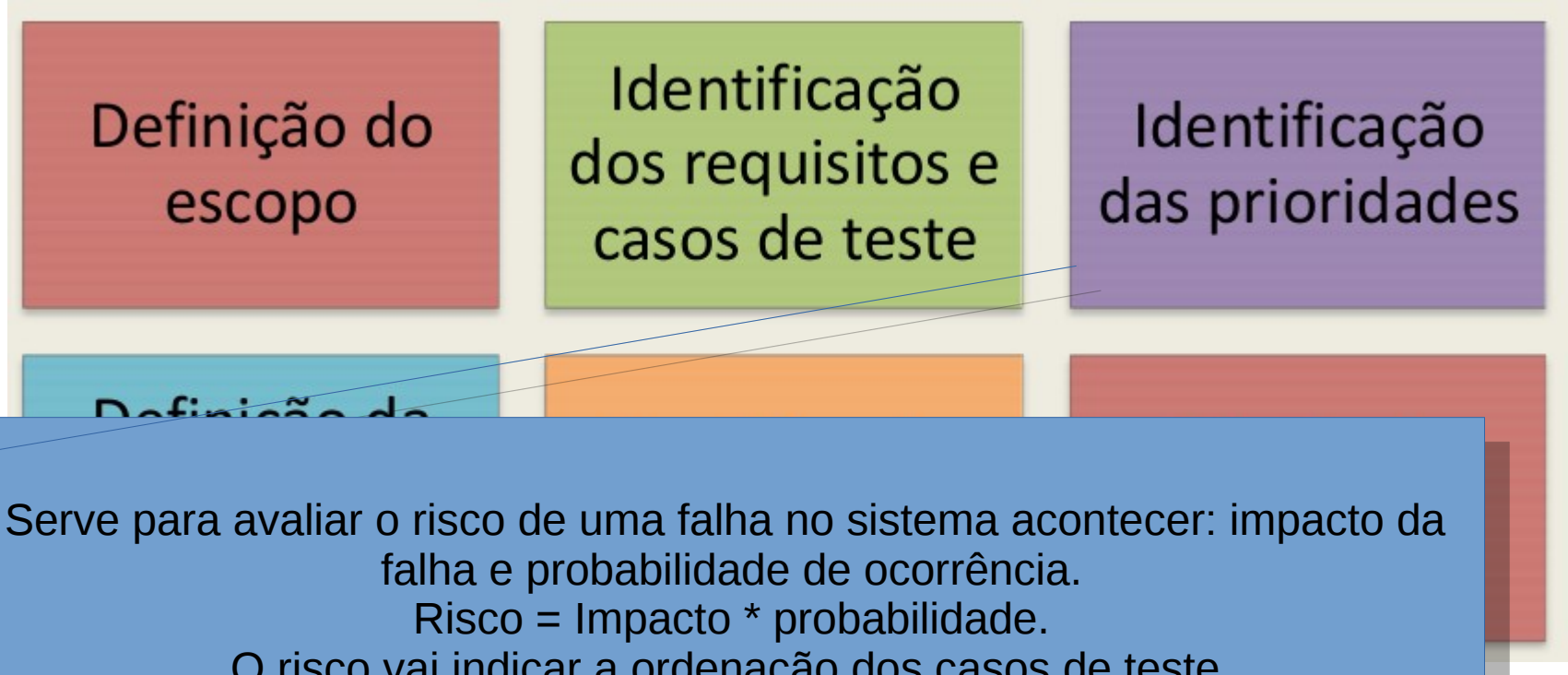


Planejamento de testes



Definir os requisitos funcionais e não-funcionais que devem ser Testados.
Identificar um subconjunto de cenários de teste para garantir que o sistema funciona em todos os casos.
Definir casos de testes para cada cenário a ser testado.

Planejamento de testes



Serve para avaliar o risco de uma falha no sistema acontecer: impacto da falha e probabilidade de ocorrência.
Risco = Impacto * probabilidade.
O risco vai indicar a ordenação dos casos de teste.

Planejamento de testes

-

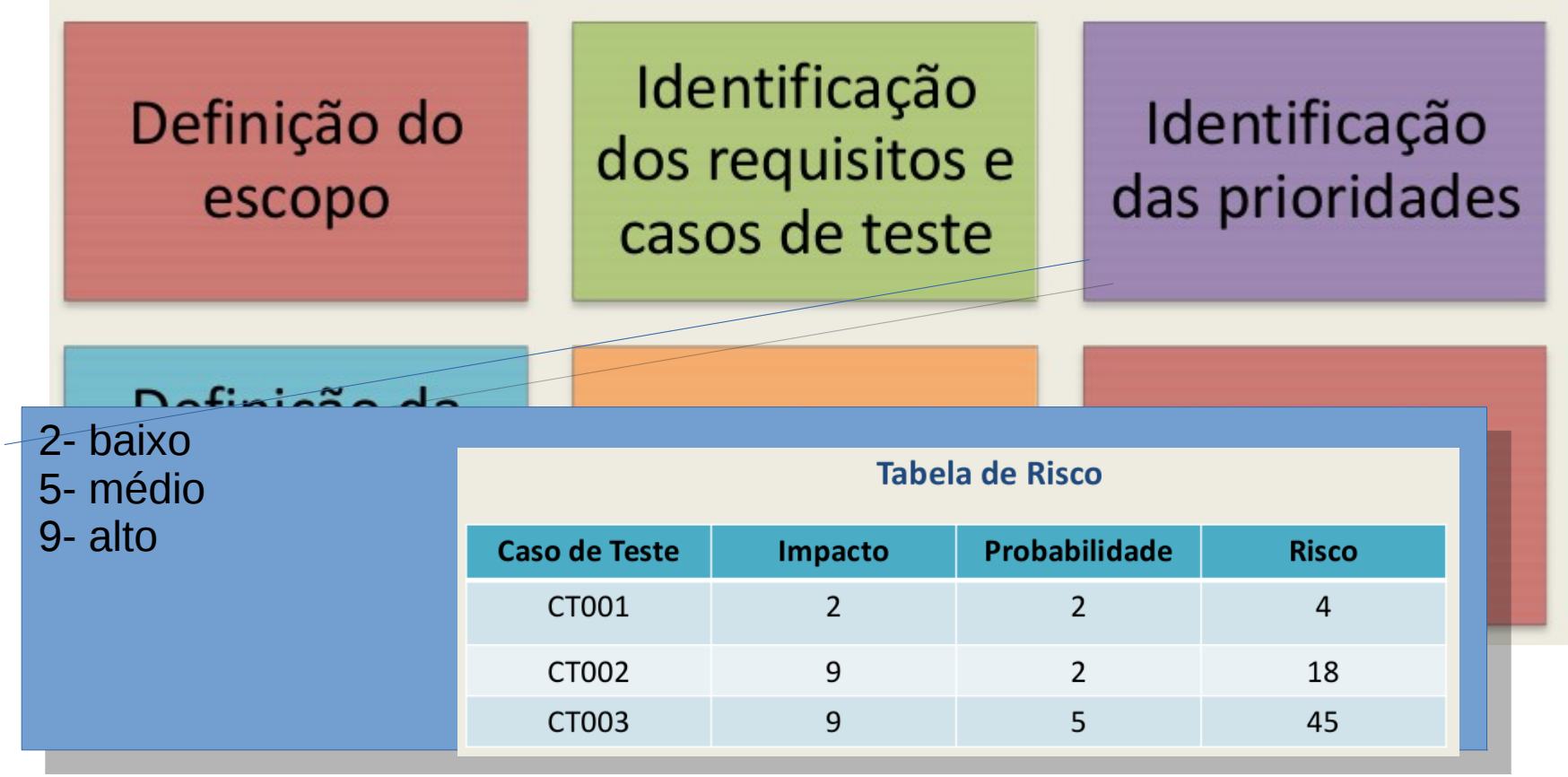


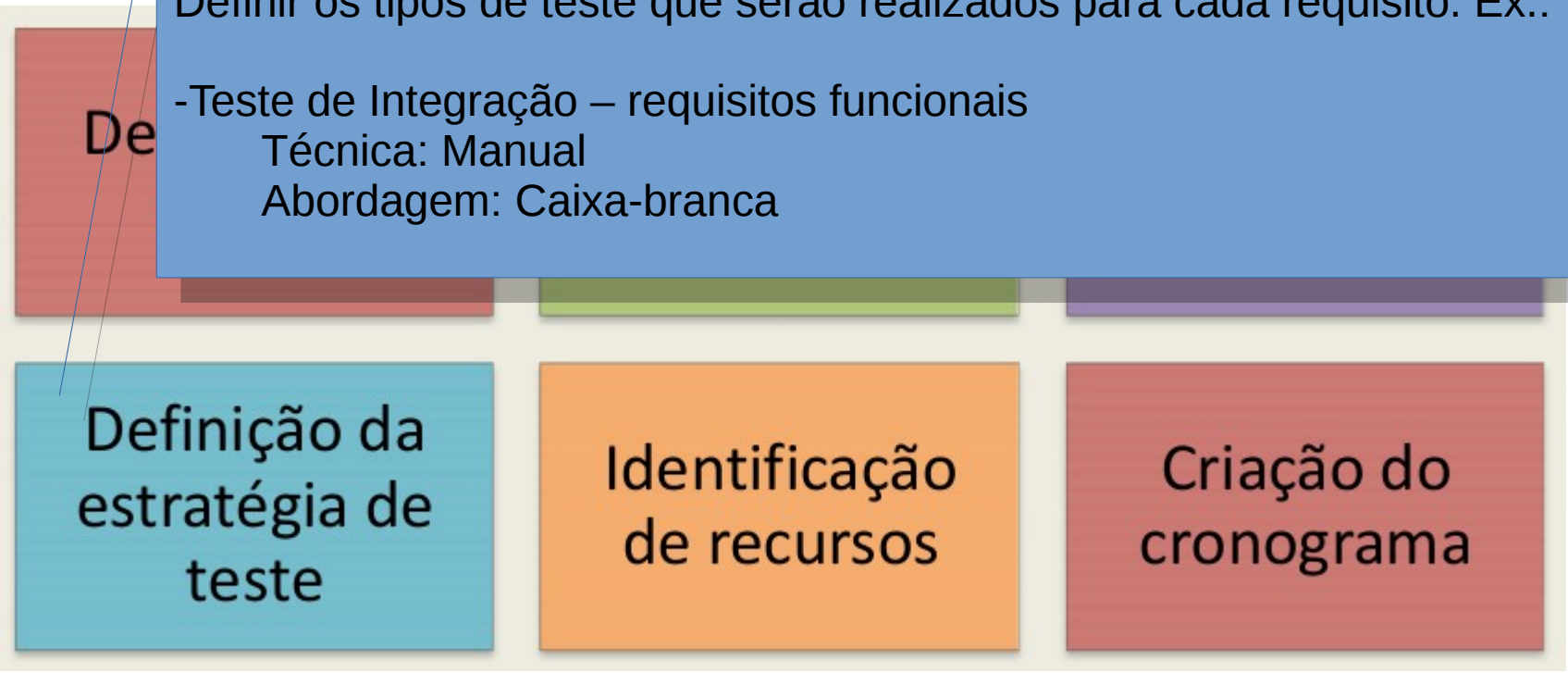
Tabela de Risco

Caso de Teste	Impacto	Probabilidade	Risco
CT001	2	2	4
CT002	9	2	18
CT003	9	5	45

Planejamento de testes

- Definir os tipos de teste que serão realizados para cada requisito. Ex.:

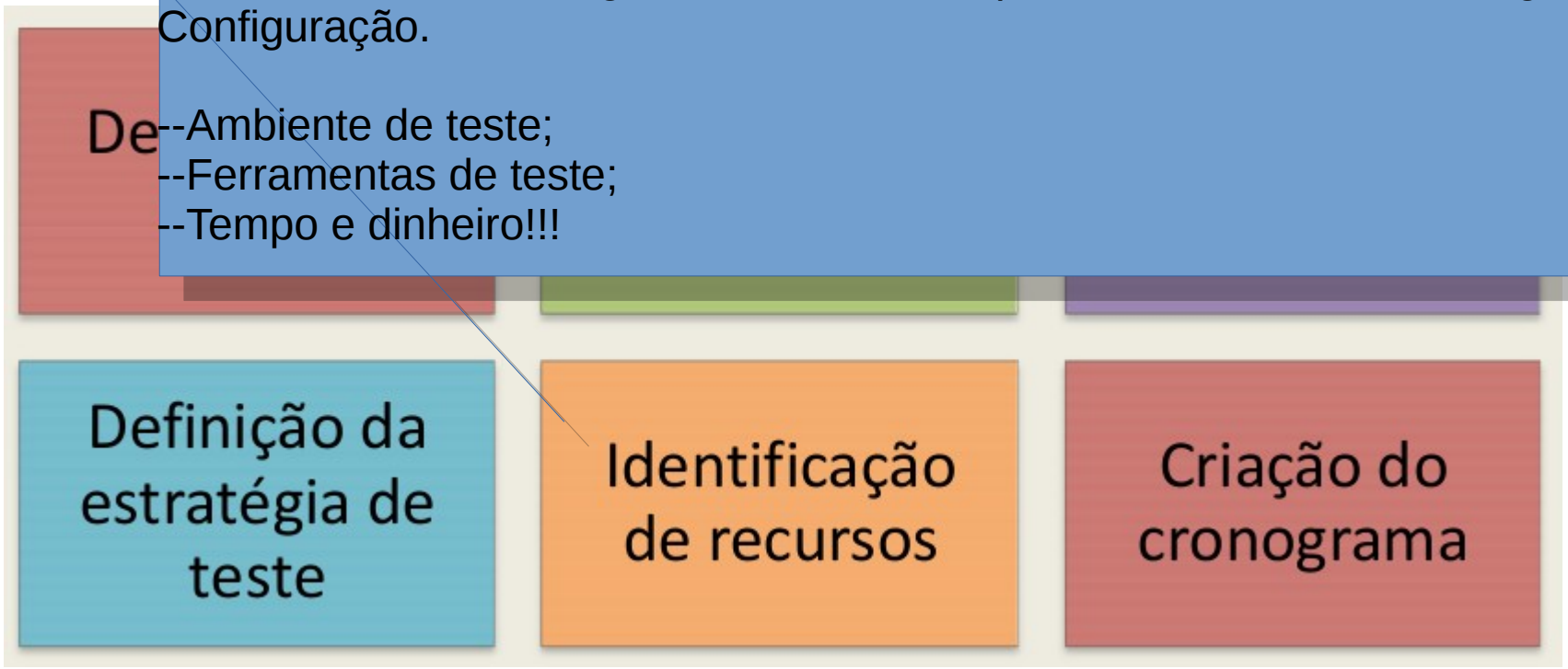
  -Teste de Integração – requisitos funcionais
  Técnica: Manual
  Abordagem: Caixa-branca



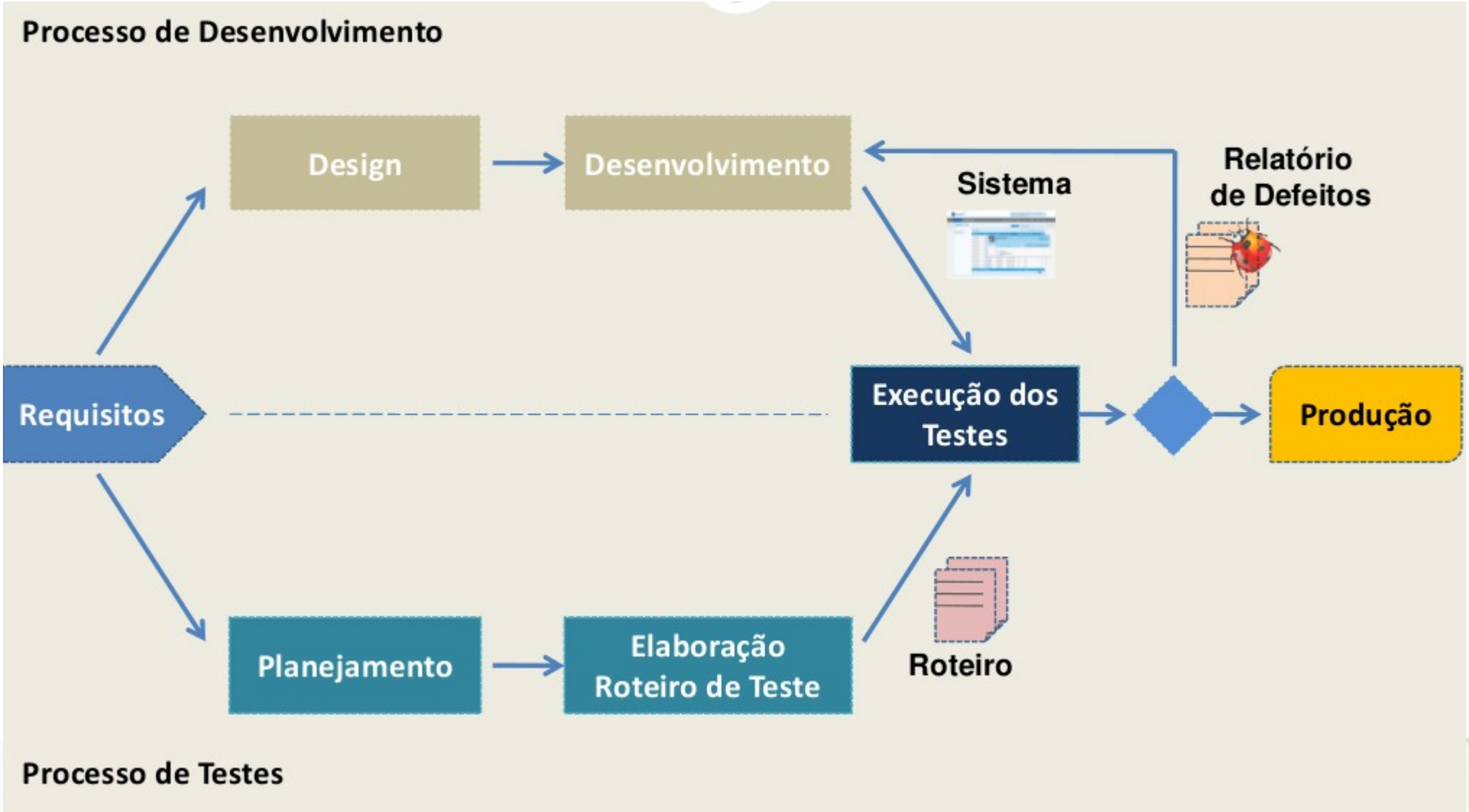
Planejamento de testes

- --Recursos Humanos: gerente de teste, arquiteto de teste, testador e gerente Configuração.

  --Ambiente de teste;
  --Ferramentas de teste;
  --Tempo e dinheiro!!!



Plano de testes



Plano de testes – Caso de Testes

- Pode ser composto por:
  - Nome do caso de teste: “CT001”, por exemplo. Recomenda-se padronização;
  - Prioridade;
  - Pré-condições;
  - Procedimentos;
  - Resultados Esperados;
  - Dados de Entrada;
  - Ambiente;
  - Técnica: Manual ou Automatizada;
  - Iteração.

Plano de testes – Caso de Testes

Caso de Teste:	CT 001 – Senha Inválida
Pré-condições	Estar na tela de login do Administrador
Procedimento	1) O ator informa uma senha inválida e preenche um login válido 2) O ator seleciona a opção OK 3) O sistema verifica se os campos obrigatórios foram preenchidos 4) O sistema verifica se o login do usuário está cadastrado no sistema e se a senha é correta; 5) O sistema exibe a mensagem "Login/Senha inválidos"
Resultado esperado	Mensagem de erro do sistema
Dados de entrada	Um login inválido
Prioridade	Alta
Ambiente	Windows XP; Internet Explorer 6.0 Service Pack 2; Servidor de aplicação Apache 2.0
Técnica	Manual
Iteração	1º Iteração

Plano de testes – Estrutura

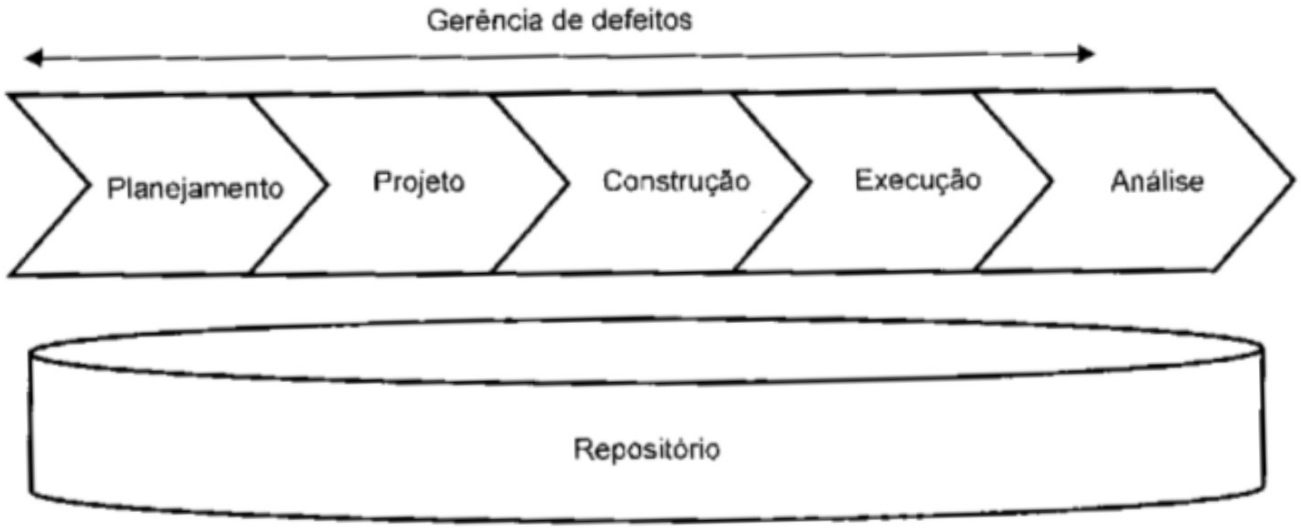
- Testes Estruturais
  - Introdução ou Descrição;
  - Objetivos;
  - Métodos;
  - Objetos a serem testados;
  - Eventos a serem testados;
  - Ferramentas de teste;
  - Execução do teste;
  - Verificação do teste.

- Testes Funcionais
  - Introdução ou Descrição;
  - Objetivos;
  - Métodos;
  - Funcionalidades a serem testadas;
  - Projeto de dados para testes;
  - Construção dos dados de teste;
  - Ferramentas de teste;
  - Execução do teste;
  - Verificação do teste.

Plano de testes – Estrutura segundo IEEE 829-1998

- **Propósito** - descreve o escopo, os recursos, a abordagem e o tempo alocado para as atividades de teste. Identifica os itens e funcionalidades a serem testados, os responsáveis e os riscos.
- **Identificador** – associa um identificador único ao plano de testes específico.
- **Introdução** – resume os itens e funcionalidades a serem testados. Pode haver referências a outros documentos do processo de desenvolvimento.
- **Itens** – Identifica os itens a serem testados, incluindo versão.
- **Funcionalidades** – Identifica as funcionalidades que serão testadas ou não, assim como os motivos.
- **Abordagem** – especifica as principais atividades, técnicas e ferramentas usadas para o teste das funcionalidades. O detalhamento deve ser suficiente para permitir identificação das principais tarefas de teste e a estimativa de tempo para cada uma. As restrições significativas como recursos e prazos, devem ser identificadas.
- **Critérios de aceite** – especifica os critérios de aceite para cada item.
- **Suspensão** – especifica os critérios para suspender parte ou toda a atividade de teste. Especifica as atividades que devem ser repetidas quando o teste for retomado.
- **Produtos – identifica os documentos produzidos, como planos, procedimentos, logs e relatórios.**
- **Tarefas de teste** – identifica as atividades necessárias para preparar e executar os testes, bem como todas as dependências entre as tarefas.
- **Ambiente** – identifica as atividades necessárias para preparar e executar os testes, bem como todas as dependências entre as tarefas.
- **Responsabilidades** – identifica os grupos responsáveis por gerenciar, projetar, executar, verificar e resolver os testes. Identifica os grupos responsáveis pelo ambiente e pelos itens de teste.
- **Treinamento** – especifica as necessidades de treinamento e identifica as opções.
- **Cronograma** – identifica as atividades e os prazos de conclusão. Para cada recurso, como pessoas e ferramentas, especifica os períodos de alocação.
- **Riscos** – identifica os maiores riscos e os planos de contingência.
- **Aprovações** – especifica os nomes e cargos dos responsáveis por aprovar o plano. Devem ser inclusos espaços para assinaturas e data.

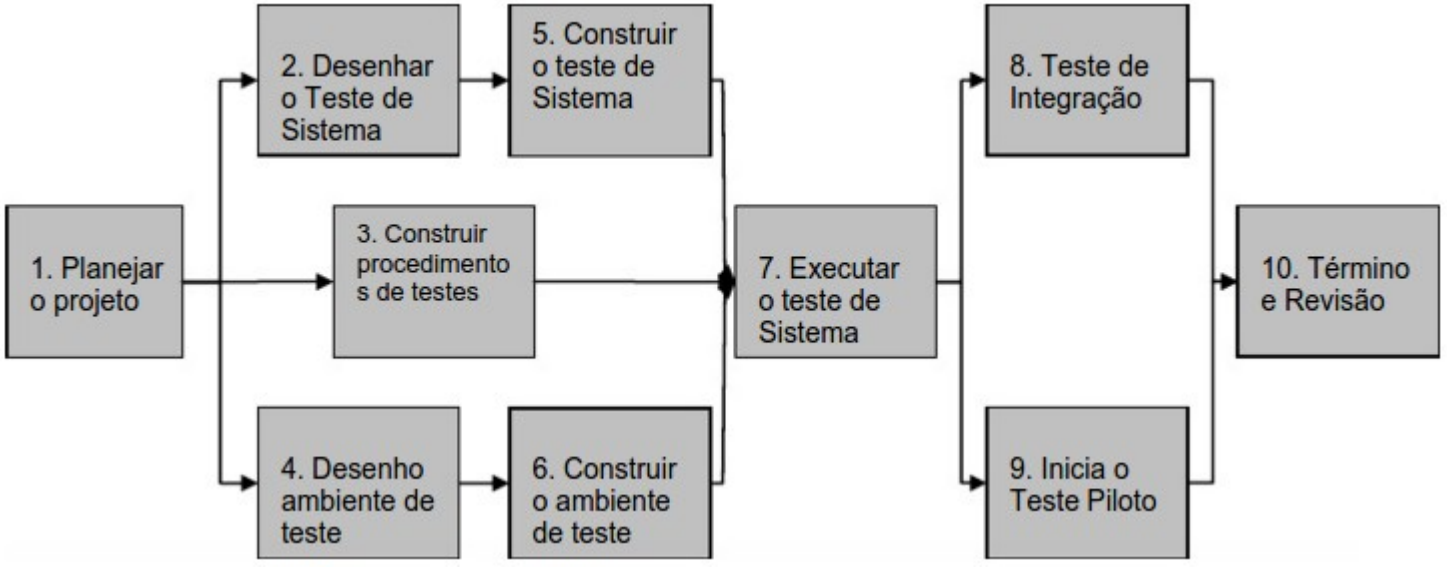
Plano de testes – Arquitetura de Automação de testes



Plano de testes – Artefatos do Processo de Planejamento

- Test People: Quem faz o que.
  - Test Manager: gerente;
  - Test Designer: faz o projeto de teste, elaborando os requisitos e casos de testes;
  - Test Implementor: implementa os scripts;
  - Test Support: dá suporte à ferramenta de automação utilizada;

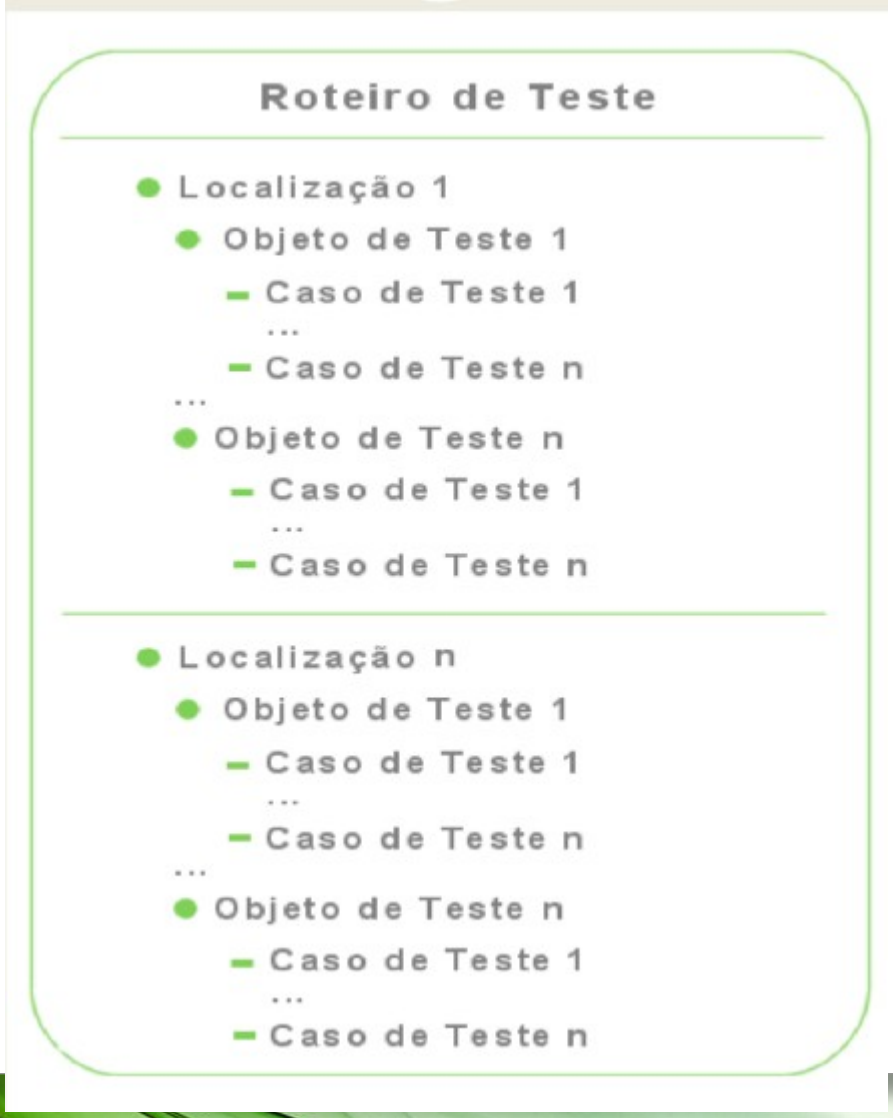
Exemplo de caminho do teste:



Caminho do teste funcional:

1) Revisão da Especificação de Caso de Uso
2) Elaboração de Roteiros de Teste
3) Revisão dos Roteiros de Teste
4) Execução dos Testes
5) Relato dos Defeitos
6) Re-execução dos Testes

Caminho do teste funcional:



Caso de teste:

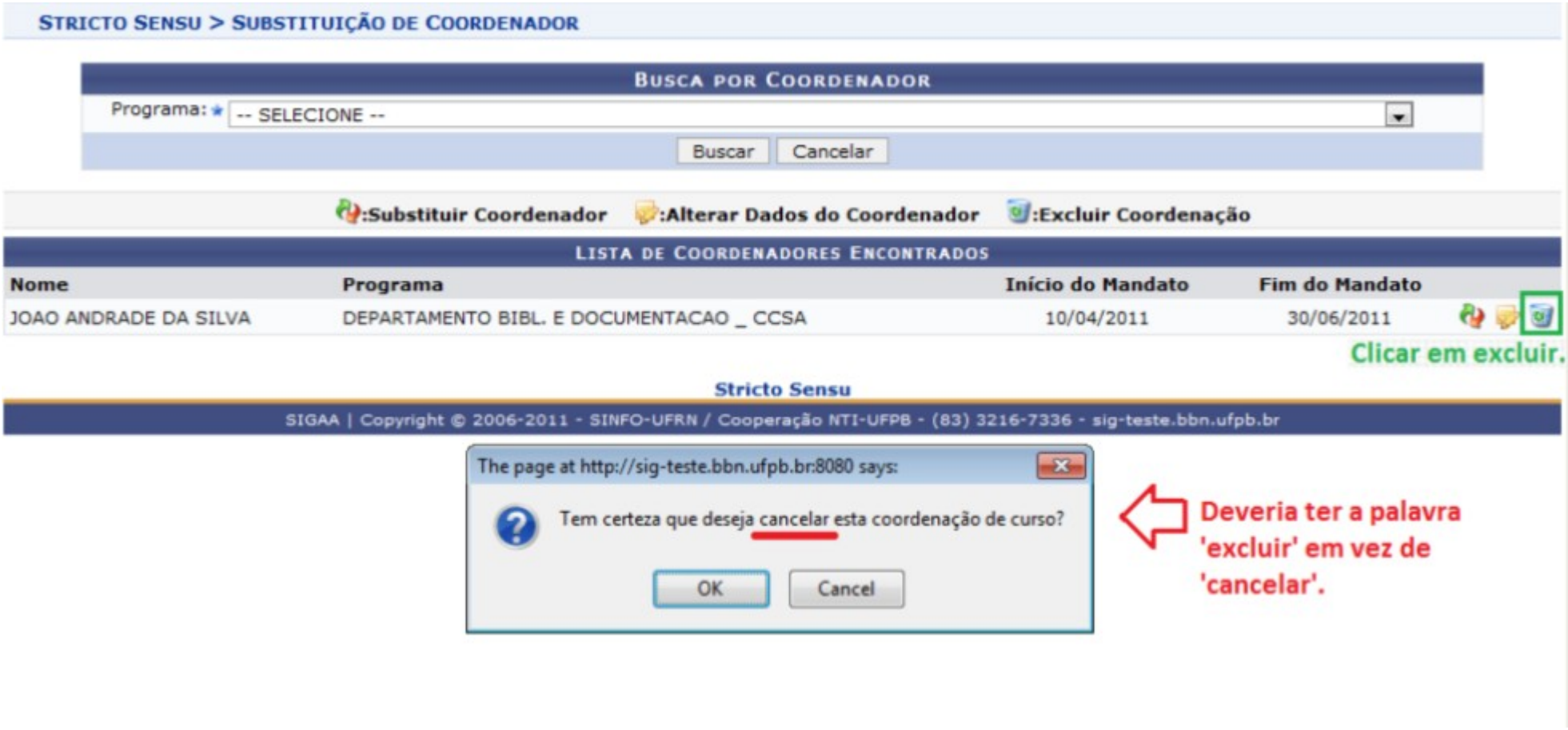
Contador:	**001**
Prioridade:	Média
Localização:	Tela Principal > Login
Objeto de Teste:	Ortografia
Caso de Teste:	Verificar a ortografia das mensagens
Pré - Condição:	**1.** Computador com acesso a internet. **2.** Sistema está disponível.
Procedimento:	**1.** O usuário preenche os campos login e senha. [senha inválida] **2.** O usuário clica no botão 'Entrar' **3.** O sistema exibe uma mensagem de erro.
Resultado Esperado:	A mensagem de erro não possui erros de ortografia.
Evidências:	

Execução do teste:

Procedimento:	**1.** O usuário preenche os campos login e senha. [senha inválida] **2.** O usuário clica no botão 'Entrar' **3.** O sistema exibe uma mensagem de erro.
Resultado Esperado:	A mensagem de erro não possui erros de ortografia.

Evidências: Passou!

Relatório de erros - imagem:



Relatório de erros – descrição textual:

- Localização:
  - SIGAA → Stricto Sensu → Substituir Coordenador
- Passos:
  1. Selecionar um Programa que haja alguma coordenador identificado.
  2. Clicar no botão 'Excluir Coordenação'.
- Evidências:
  1. A mensagem do alerta de confirmação da exclusão está com problema, pois deveria referenciar um exclusão e não um cancelamento. <ver imagem>

Gerenciamento de testes: gerenciando erros

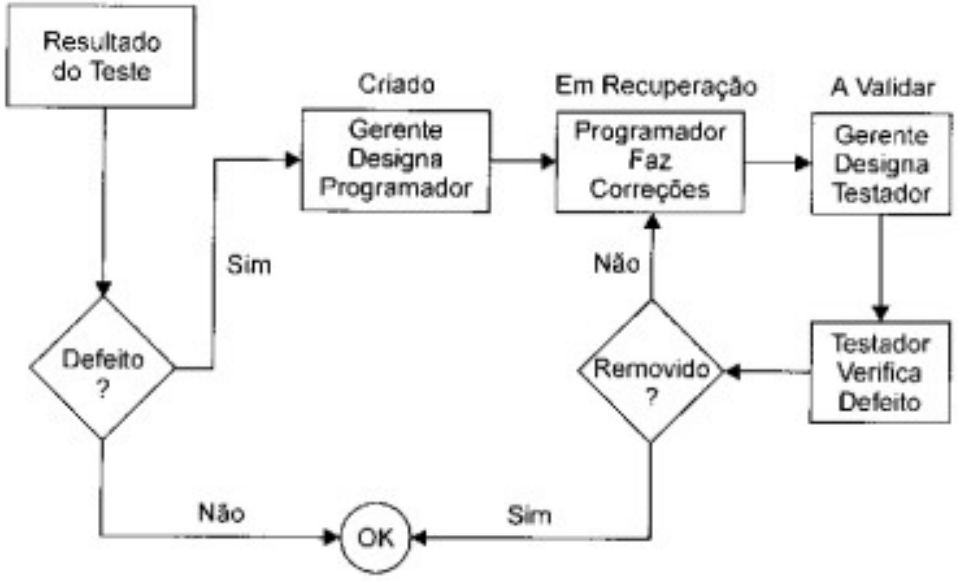
O principal objetivo do processo de gestão de defeitos é evitá-los.

Reportar defeitos:

Objetivo	• Reportar de forma clara e objetiva
Precisão	• Isto é um defeito ou poderia ser um erro do testador/usuário?
Neutralizar	• Apenas fatos. Sem humor, sem emoção...
Reproduzir	• Reproduza um defeito ao menos duas vezes antes de reportá-lo
Impacto	• Qual o impacto deste defeito para o cliente?
Evidência	• Sempre que possível registre uma imagem que represente o defeito.

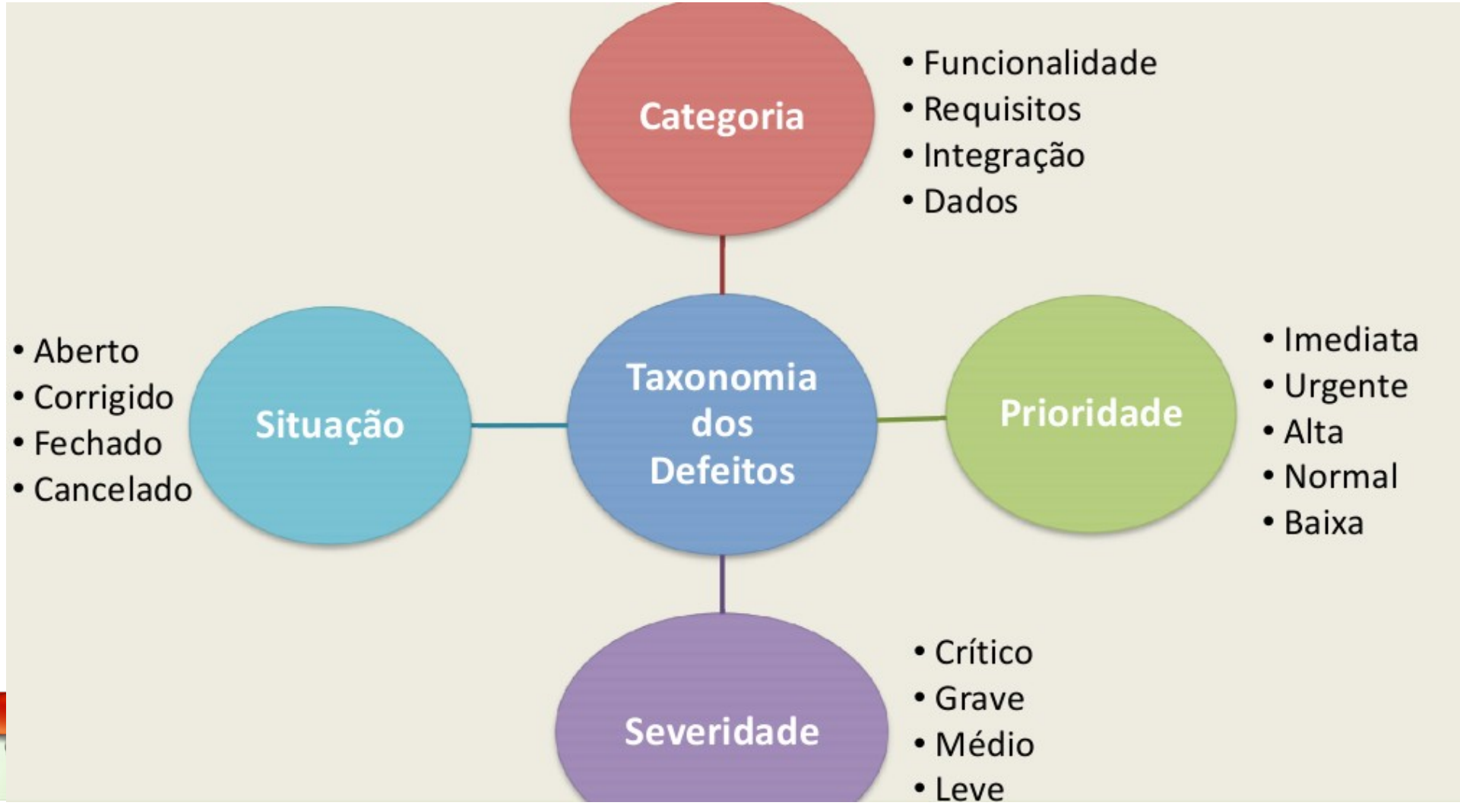
Gerenciamento de testes: gerenciando erros

Defect Tracking: processo de gerência de defeitos, o que incluia atarefa de registrar a descoberta e a solução deles em qualquer ciclo de vida da aplicação.
Típico Workflow:



Gerenciamento de testes: gerenciando erros

Taxonomia de defeitos



Testes em metodos ágeis?

- Os times de XP fazem e dizem que:
  - Os objetos de processamento não são melhorados pela visão não-XP;
  - Focos nos “bugs” são de pouca importância para o projeto;
  - Vários testadores que têm falha e pouco conhecimento técnico ajudam a deuxar lacunas de teste;
  - Testadores tradicionais não se atualizam em termos de técnicas modernas de teste.

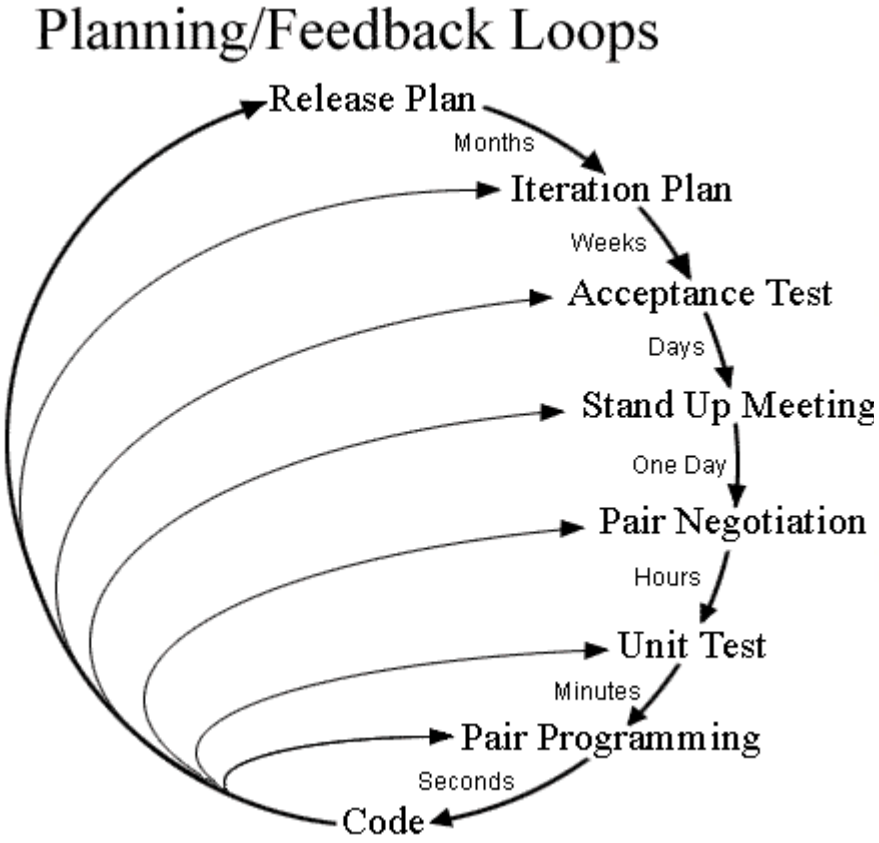
Testes em metodos ágeis?

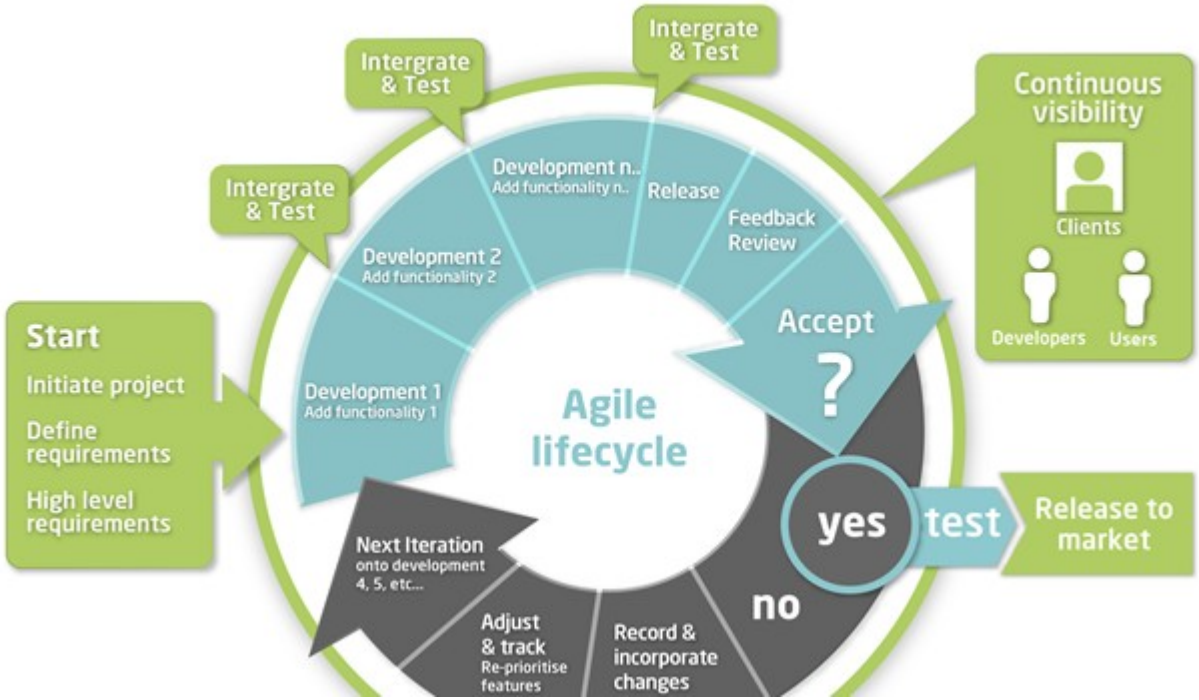
- Os times de XP fazem e dizem que:
  - Os objetos de processamento não são melhorados pela visão não-XP;
  - Focos nos “bugs” são de pouca importância para o projeto;
  - Vários testadores que têm falha e pouco conhecimento técnico ajudam a deuxar lacunas de teste;
  - Testadores tradicionais não se atualizam em termos de técnicas modernas de teste.

Testes em metodos ágeis?

- Em XP a responsabilidade de realizar os testes de software fica a cargo do programador, e os testes de aceitação a cargo do “consumidor” que tenha o poder de validação do negócio envolvido.
- Prega-se que XP salva o custo da necessidade de testadores no projeto.
- Segundo os livros, XP inclui testes de unidade e verificação já imbutidos no processo.

Testes em metodos ágeis?





Referências

Myres , G. F. “The Art of Software Testing”. Ed. John Wiley & Sons, Inc. New Jersey, 2004.

Dijkstra, E. W. "The Humble Programmer". Communications of the ACM 15 (10): 859–866, 1972.

Rios, Emerson. “Teste de software”. Alta Books Editora, 2006.

Gelperin, David, and Bill Hetzel. "The growth of software testing." Communications of the ACM 31.6 (1988): 687-695.

MOLINARI, Leonardo. “Testes de software: produzindo sistemas melhores e mais confiáveis: qualidade de software: soluções, técnicas e métodos”. Érica, 2003.

IEEE . Standard for Soft ware & System. Test Documentation. IEEE 829-2008. New York: IEEE, 2008.

PRESSMAN, Roger S. Engenharia de software: uma abordagem profissional. 7ª Edição. Ed: McGraw Hill, 2011.

SOMMERVILLE, Ian,. Engenharia de Software. São Paulo: Pearson, 9ª edição, 2011.