Introdução ao Teste de Software

Microsoft | Innovation Center
PUCRS

Prof. Marco Mangan
Prof. Ricardo M. Czekster
Faculdade de Informática
04/09/2017

Súmula

- Fase I: Teste de Software S2B
- Conteúdos abordados
  - Introdução ao Teste de Software
  - Plano de Teste, gerenciamento de testes
  - Automação de Testes de Software
  - Testes unitários

Súmula

- Fase II: Desenvolvimento de trabalho prático
- Atividades
  - Estudo de uma ferramenta quanto às suas funcionalidades, especificação, etc
  - Criação de Plano de Teste completo
  - Modelagem de testes

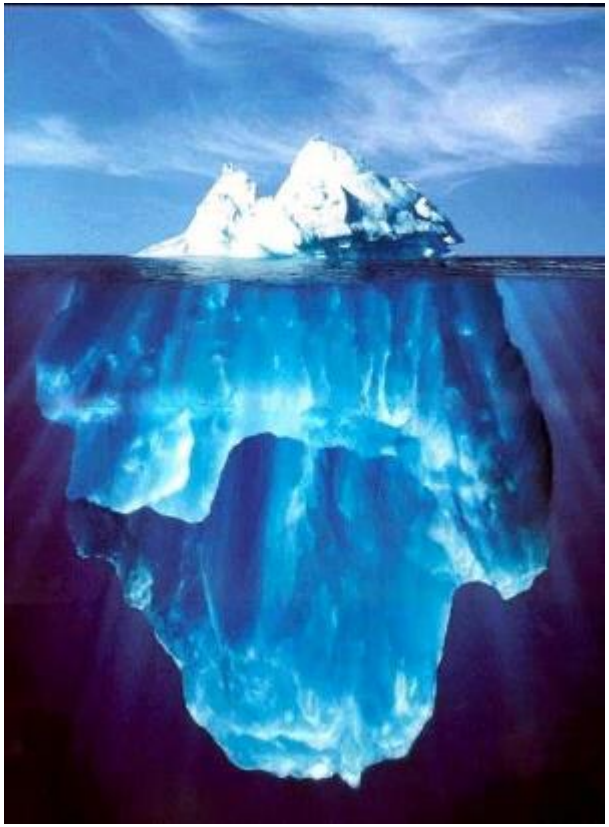
RECOMENDAÇÕES

Recomendações

- Curso é +- teórico
- Dúvidas não perguntadas são dúvidas não respondidas
- Ir até o final
- Cuidar com as faltas
- Apresentação de um relatório de desempenho ao final do curso para todos os alunos (possivelmente empresas)

Detalhe

- Exercícios pedidos em sala de aula são apenas o **início** do assunto!



OUTRAS INFORMAÇÕES

Bibliografia básica de teste

- Introdução ao Teste de Software
  - Delamaro, Maldonado, Jino
- Teste e Análise de Software
  - Pezzè, Young
- *Practical Software Testing*
  - Ilene Burnstein
- Syllabus (básico) do ISTQB - *International Software Testing Qualifications Board*
- Existem MUITAS referências para teste...
  - Muitas destas muito teóricas (impraticáveis)

Students to Business – Trilha de Teste de Software

FASE I

TESTE DE SOFTWARE: INTRODUÇÃO E CONCEITOS BÁSICOS

Conceitos

- Introdução ao teste de software
- Verificação & Validação
- Tipos de teste
- Ciclo de desenvolvimento de software
- Teste funcional e não-funcional de software
- Carreira de testador de software
- Documentação de testes
- Criação de planos de testes

Hoje

- Assunto é *Teste de Software*
- Explorar:
  - ***mentalidade*** para teste de software
  - **teoria** de teste de software
  - necessidade e motivação
- **Introdução** aos conceitos mais importantes para as próximas aulas
- **Preparação** para um planejamento completo de teste de um produto/software

Processos de Engenharia e Verificação

- Disciplinas de engenharia:
  - Civil, mecânica, elétrica, eletrônica, química etc
  - Atividades de projeto e construção
  - Atividades de verificação ➡ identificação de defeitos
- Engenharia de software
  - Voltada para o desenvolvimento de Sist. de Software
  - Atividades de projeto e construção
  - Atividades de verificação

Processos de Engenharia e Verificação

- Atividades de verificação
  - Adequadas para a construção repetitivas de itens
  - Adequadas para itens customizados
  - Adequadas para itens altamente críticos

Processos de Engenharia e Verificação

- Verificação de produtos produzidos em série
  - Repetição de conjuntos pré-definidos de testes
  - Indicam quando o produto atingiu padrões de qualidade pré-definidos
  - Se o grau de automação é grande ➡executados por amostragem
- Verificação de produtos únicos
  - Conjuntos especializados de teste
  - A complexidade cresce com a complexidade e variedade dos componentes

Processos de Engenharia e Verificação

- Software
  - Um dos mais variados e complexos produtos produzidos de forma regular
  - Requisitos de qualidade dependentes
    - Do domínio de aplicação
    - Ambiente de execução
    - Público alvo
    - Outros
  - Exige técnicas sofisticadas e específicas para cada produto desenvolvido

Dificuldades de teste de software

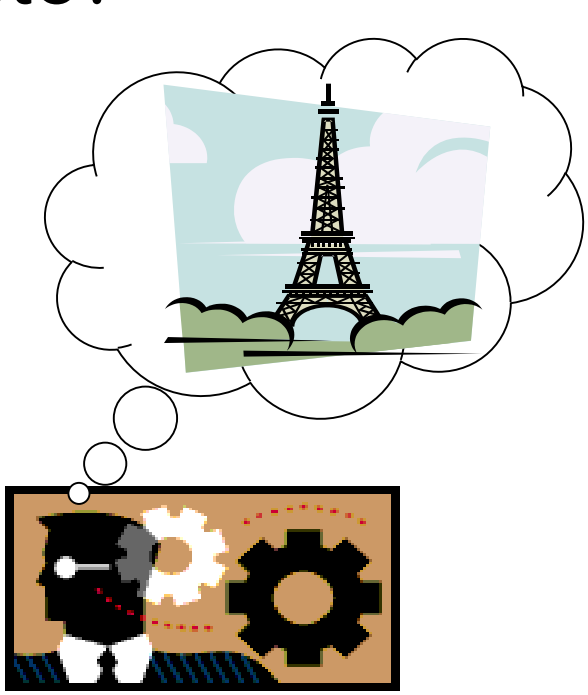
- Por que é tão difícil testar software?
  - Porque é algo dinâmico
  - Porque envolve pessoas
  - ?

Dificuldades de teste de software

- Por que é tão difícil testar software?
  - Porque é algo dinâmico
  - Porque envolve pessoas
  - Porque as regras de negócios podem ser complexas...
  - Porque as tecnologias mudam muito rapidamente
  - Porque as equipes mudam a toda hora
  - Porque novas necessidades surgem a cada momento
  - ...

Diferença entre Verificação e Validação

- Validação: "Estamos construindo o produto correto?"



Cliente

Desenvolvedor

Não atende a expectativa do Cliente

Diferença entre Verificação e Validação

- Verificação: "Estamos construindo o produto corretamente?"

Cliente

Desenvolvedor

Não atende
os requisitos funcionais
ou não-funcionais

Técnicas de V&V

- Existem dois tipos de técnicas de V&V
  - Estáticas
    - Não requer que o sistema seja executado
    - Exemplo: **revisões** (*inspections* e *walkthroughs*)
  - Dinâmicas
    - Requer trabalhar com uma representação executável do sistema
    - Exemplo: **teste**

Técnicas de V&V

- Revisões:
  - Identificam a correspondência entre um artefato (programa, projeto, diagrama) e sua especificação
  - Não demonstram se o artefato é operacionalmente útil ou se suas características não-funcionais atendem os requisitos desejados
  - Técnica estática
  - Variações:
    - Revisões formais, *peer review*, inspeção, w*alkthrough etc*
    - *O* que varia é o grau de formalismo do procedimento

Quais são as atividades de teste?

- Apenas executar um software?
  - Parte de teste, mas não tudo
- Atividades de teste existem antes e depois da execução
  - Planejamento e controle
  - Escolha de condições de teste
  - Definição e design de casos de teste
  - Avaliação e análise de sistemas sob teste (SUT)

Objetivos de teste

- Encontrar e prevenir **defeitos**
- Conhecer o grau de **qualidade** de um produto
- Prover informações para tomada de decisões
- Se usado desde o início, previne que defeitos sejam introduzidos no código
- **Verificar** e **validar** produtos

Vale a pena?

- Percepção de retorno não imediata
  - Qual o **valor** de se testar um produto?
- Não garante qualidade **total**
- Documentação
  - Excesso de documentação?
  - Clientes concordaram/aprovaram?
  - Revisões na documentação custam caro?
  - Qual o impacto na velocidade do projeto?

Vale a pena?

- **Não** testar um software/produto seria pior
  - Como a qualidade seria atestada?
- O que é melhor fazer?
  - Revisar o processo de teste na organização
  - Está custando muito caro?
  - Como pode-se melhorar?
  - Questionar-se sempre
- Posicionamento quanto aos testes
  - **Encontrar** ou **prevenir** defeitos?

Qualidade

- *ISO/IEC 9126 Software engineering: Product quality*
  - http://en.wikipedia.org/wiki/ISO/IEC_9126

O Teste no Ciclo de Desenvolvimento de Software

- **Categorias de teste:**
  - Funcional x Estrutural
    - Especificação x Código
  - Estático x Dinâmico
    - Não executa x executa código
  - Estatístico
    - Enfoque na confiabilidade
  - Aspectos não-funcionais
    - Desempenho, stress, segurança etc
- Cada categoria inclui diversas técnicas !!

Teste – níveis de teste

- Teste Funcional
  - Teste das principais funcionalidades de sistemas
  - Altera o sistema desenvolvido, diversas iterações
- Teste Não-Funcional
  - Testes de outras características: tempo de resposta, memória utilizada etc
  - Também chamado de *Teste de Desempenho*
  - Inicia **após** estabilidade do sistema desenvolvido
- Interesse principal: ***qualidade do produto***

Tipos de teste

- Funcional
- Não-Funcional

- de Unidade
- de Integração
- de Sistema
- Exploratório
- de Aceitação
- de Regressão
- de Fumaça
- de Stress, Robustez
- Baseado em Riscos
- ...

Tipos de teste – ...

- Mais testes...
  - de *baseline*
  - de segurança
  - de instalação
  - de implantação
  - de compatibilidade
  - de concorrência
  - de conversão
  - de *backup*
  - de recuperação
  - ...

Resumindo...

- MUITOS tipos testes
- Seria legal definir **cada** teste em **uma** frase, não?
  - *Taglines*: a seguir

Mais importantes e *taglines*

- Teste funcional
  - *"teste uma função por vez, garanta execução"*
- Teste baseado em riscos
  - *"teste os maiores defeitos antes"*
- Teste de stress
  - *"execute até a exaustão dos recursos"*
- Teste de regressão
  - *"repita os testes a cada mudança"*

Mais importantes e *taglines*

- Teste exploratório
  - *"aprendizagem, planejamento e teste **simultaneamente**"*
- Teste aleatório
  - *"novos casos de teste a cada execução"*
- Teste de desempenho
  - *"quanto de recursos são consumidos por cada operação do sistema?"*

Terminologia

- Padrão IEEE 610:1990
  - Defeito (*error/bug/defect/mistake*): interação humana que produz um resultado incorreto
  - Erro (*fault*): manifestação de um defeito em uma especificação de software
  - Falha (*failure*): comportamento diferente do esperado
  - http://dis.unal.edu.co/~icasta/ggs/Documentos/Normas/610-12-1990.pdf
- IEEE 1044:2009
  - http://www.ctestlabs.org/neoacm/1044_2009.pdf

Terminologia – Microsoft

- Defeito, Erro, Falha, *Bug*
  - Um problema em geral
- Requisito (*user story, scenario, work item*)
  - tudo que define um requisito do sistema, algo que deve ser realizado

Princípios de teste de software

1. **Testes mostram a presença de defeitos**
2. **Teste exaustivo é impossível**
   - testar todas as combinações de entradas é impraticável
3. **Teste em todas as etapas**
   - teste deve ser introduzido desde o início do projeto
4. **Agrupar defeitos por módulo/componente/classe**
   - observar a densidade de defeitos nos módulos
5. **Paradoxo do *pesticida***
   - se os mesmos testes são sempre repetidos, eventualmente os casos de teste não acharão novos defeitos. Revisar os testes e criar novos.
6. **Intensidade de testes é dependente de contexto**
   - software para hospitais é testado em intensidade maior do que um site de compras
7. **Falácia da ausência de erros**
   - se não preenche os requisitos dos usuários, não adianta consertar os defeitos

Premissas básicas de teste

- Com testes é possível detectar a **existência**, mas não é possível garantir a **ausência** de erros
- Impossível testar **todas** as possibilidades de execução
- Testes (funcionais e não-funcionais) possuem natureza **destrutiva**

Premissas básicas de teste

- Objetivo de muitas equipes
  - *zero defect release*

Premissas básicas de teste

- Descobrir e consertar os defeitos '**certos**'
  - Os que possuem mais visibilidade nos clientes
  - Observar os caminhos da aplicação mais usados
  - Baseados nas principais funcionalidades, UCs
  - Dependente de projeto
- Sempre existirão defeitos...
  - Aprenda a viver com isso!

Premissas básicas de teste

- Atividade de teste
  - processo de executar um programa com a intenção de descobrir um defeito
- Bom caso de teste
  - aquele que tem uma elevada probabilidade de revelar um defeito ainda não descoberto
- Teste bem-sucedido
  - aquele que revela um defeito ainda não descoberto

Como as organizações criam as diferenças entre as equipes

- Testadores
- Desenvolvedores
- Área de negócios

Trabalho colaborativo

- Testadores, desenvolvedores, negócios
- Entregar produto com alto **valor** agregado

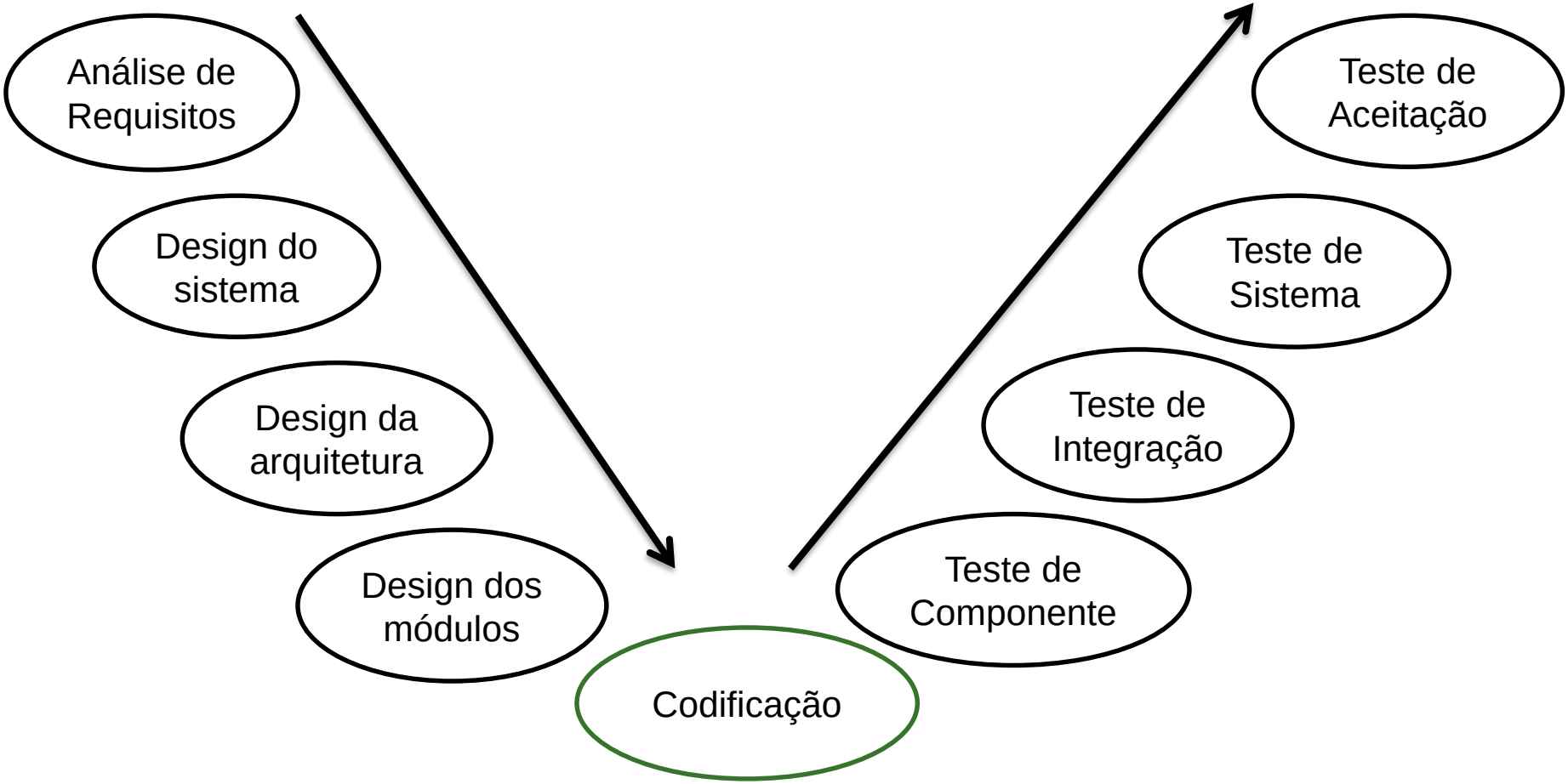
Introdução

- Teste → qualidade de software
- Todos envolvidos no desenvolvimento de software trabalham juntos

- *Testers x Developers x Architects x Management*
  - *'finger-pointing'*
  - Muito tempo capturando e corrigindo defeitos
  - Clientes descobrindo defeitos ao executar produto

MODELOS DE DESENVOLVIMENTO DE SW

Modelos de Desenvolvimento de Software – Modelo V (*V-Model*)



Modelos de Desenvolvimento de Software

- Modelo V (*V-Model*)
  - Teste de Componente (unidade, ou *unit*)
  - Teste de Integração
  - Teste de Sistema
  - Teste de Aceitação

Teste de Componente (unitários)

- Também conhecidos como: *unit, program, module testing*
- Procuram por defeitos e verificam o funcionamento de
  - Módulos, programas, objetos, classes etc
- Feitos de forma isolada do resto do sistema
- Podem ser preparados e automatizados antes da codificação (*Test Driven Development*)

Testes de Integração

- Teste entre componentes, interações entre partes de outros sistemas (S.O., Sistema de Arquivos etc)
- Realizado após teste de componente
- Escopo da integração determina a dificuldade de se isolar um sub-sistema
  - Pode aumentar o risco e adicionar tempo para resolver o problema
- Testes de comunicação entre módulos

Testes de Sistema

- Testa o comportamento geral do sistema/produto
- Ambiente de teste deve ser parecido com ambiente de produção para melhores resultados
- Pode incluir testes baseados em especificações de requisitos, UCs, ou outros
- Investiga requisitos funcionais e não-funcionais do sistema e aspectos de qualidade

Testes de Aceitação

- Testes que envolvem os clientes / *stakeholders* / usuários do sistema
- Estabelecer confiança sobre o sistema
- Não é objetivo procurar defeitos
- Determina características não-funcionais do sistema

Teste de regressão

- Realizada a cada nova implementação do sistema
- São re-executados **todos** os testes existentes para verificar se a nova mudança quebra o sistema em outras partes
- **Novos** testes devem ser escritos para a nova funcionalidade
- Boa prática: escrever um teste de regressão a cada defeito consertado

Teste baseado em modelos

- *Model Based Testing:* muito material
- Baseado em negócios
  - UML: Diagrama de Atividades, Casos de Uso, Máquina de estado
- Mais formais
  - Cadeias de Markov, Redes de Petri
- Mais informações sobre alguns destes testes na aula que vem

Design de testes

Técnicas de teste

Microsoft | Innovation Center
PUCRS

Técnicas de *design* de testes

- Teste caixa-preta
- Teste caixa-branca
- Teste caixa-cinza

- Existem outras técnicas...

Técnicas de *design* de testes

- Teste caixa-branca (*white-box*)
  - Análise de cobertura
  - Teste de unidade
- Teste caixa-preta (*black-box*)
  - Não se conhece o código-fonte do programa
  - Verifica entrada e saída (e observa os requisitos)
- Teste caixa-cinza (*gray-box*)
  - Apenas alguns módulos ou trechos são conhecidos

Teste caixa-preta

- Também chamado de *specification-based* ou *black-box*
- Teste de requisitos funcionais
- Funções incorretas ou ausentes
- Erros de interface
- Erros de acesso (interno e externo)
  - Banco de dados, *Web Services*
  - Arquivos inexistentes, etc
- Erros de desempenho
- Erros de inicialização e término

Teste caixa-preta – técnicas

- Partição de Equivalência
  - *Equivalence partitioning*
- Análise do Valor Limite
  - *Boundary Value Analysis*
- Tabela de Decisão
  - *Decision Table Testing*
- Teste de Transição de Estados
  - *State Transition Testing*
- Teste de Caso de Uso
  - *Use Case Testing*

Teste caixa-preta

- Código fonte não disponível
- Usado para testar sistemas diversos
  - Por exemplo: de competidores
- Uso de **especificação** do produto
- Pode ser usado como técnica para geração de testes
  - Identificar casos de teste mais importantes
  - Identificar categorias e opções mais importantes

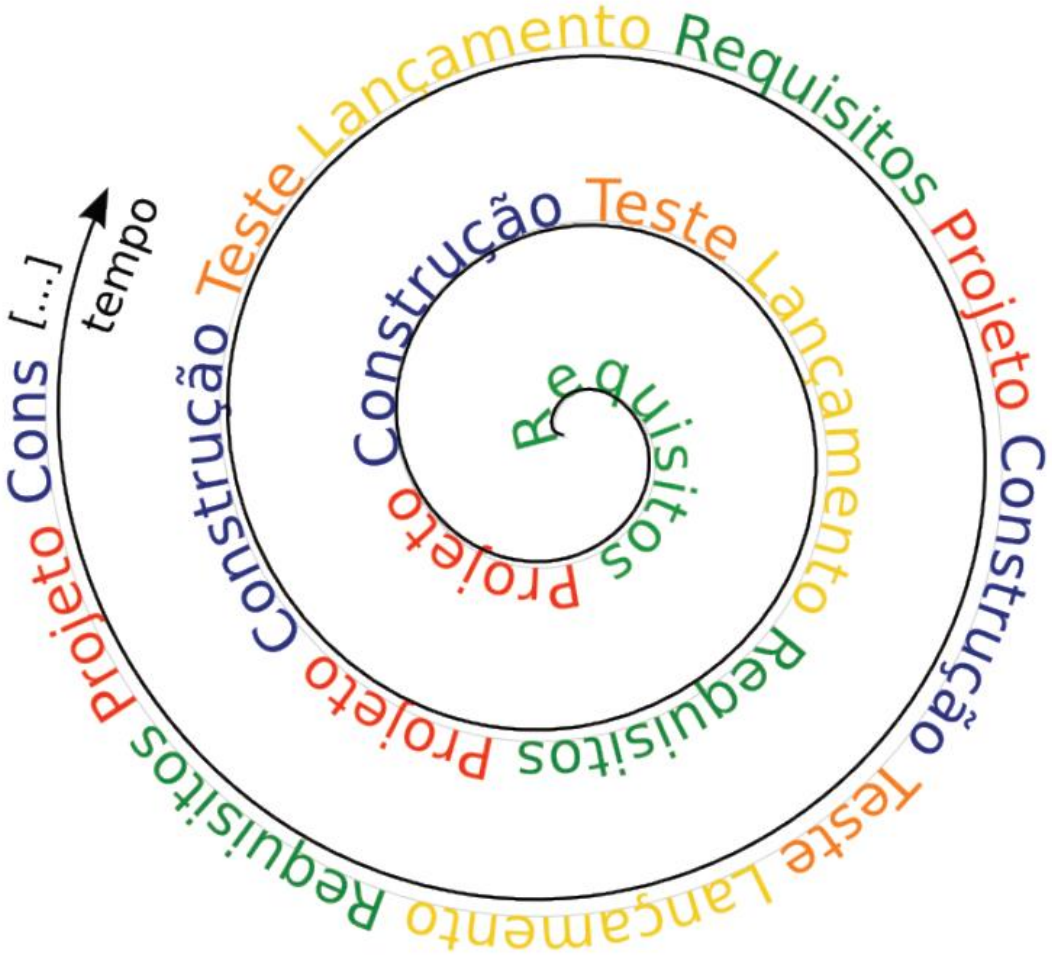
Teste caixa-preta

- Usado juntamente com teste caixa-branca
- Teste caixa-branca consegue saber todos os módulos implementados
  - Testa apenas o que existe
- Teste caixa-preta olha a especificação
  - Testa o que pode não existir, ou seja, não ter sido implementado
- Vantagens e desvantagens de cada um

Teste caixa-preta

- Maiores informações e exemplos na próxima aula
  - Técnicas
  - Exemplos
  - etc

Integração contínua e testes



Conceitos Preliminares

- Suíte de teste:
  - Um conjunto de casos de teste
- Teste ou execução de teste:
  - Corresponde a atividade de executar casos de teste e observar seus resultados
- Critério de adequação:
  - É um predicado que pode ser verdadeiro ou não para um par (programa, suíte de teste)
  - Exemplos:
    - Cobertura de estados
    - Cobertura de transições
    - Cobertura de caminho

Conceitos Preliminares

- Como expressar um teste:
  - Descrição textual dos passos e resultados
  - Script de uma ferramenta de capture-playback
  - **Classes ou métodos de teste**

Students to Business – Trilha de Teste de Software

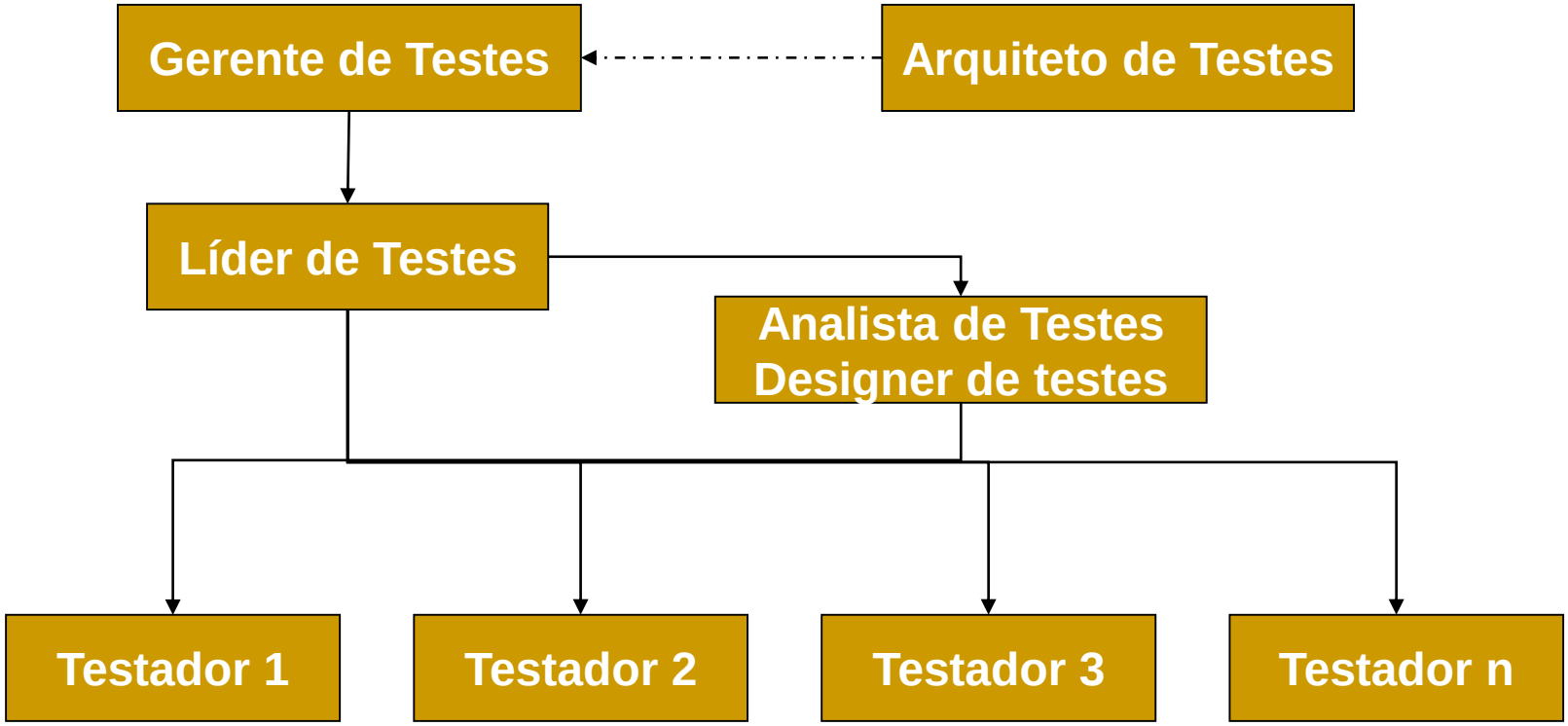
PROCESSO, DEFEITOS E PLANOS DE TESTE

Agenda

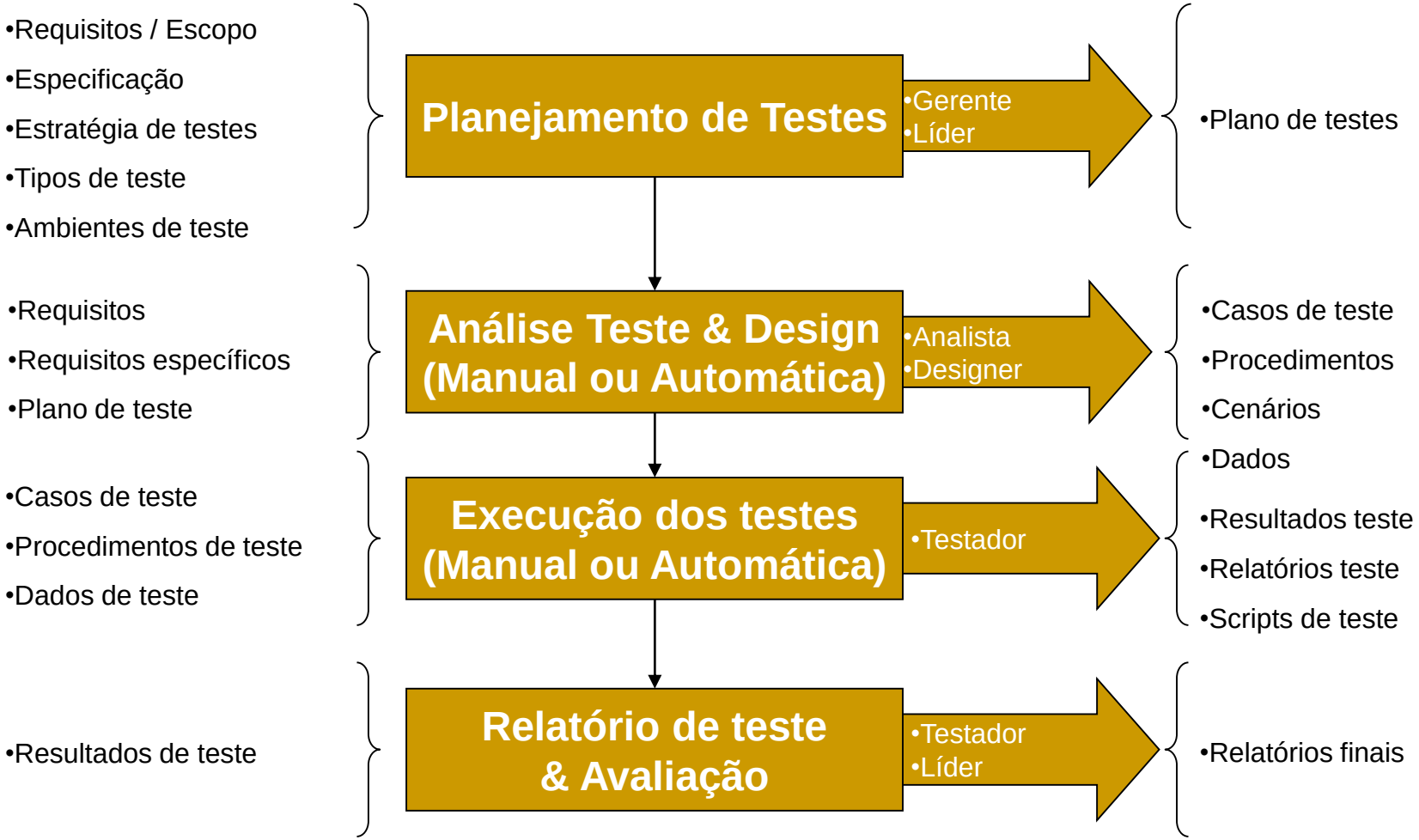
- Decisões para a criação de testes
- Teste de Nível de Sistema
- Criação de Planos de Teste
- Exemplos de Planos de Teste
  - Modelos e boas práticas
- Registro de defeitos
- Teste de Regressão e Aceitação
- Templates, exemplos e exercícios

ORGANIZAÇÃO E PROCESSO DE TESTE DE SOFTWARE

Organização básica de times



Processo geral de teste



1. Planejamento de testes

- Gerente ou líder devem ter um plano inicial para os testes
  - Escopo dos testes definidos
  - Estratégias definidas
  - Riscos do projeto (casos especiais, ou de alto valor)
  - Escolha de teste manual ou automático ou ambos
  - Estimativas para o tempo de testes e cronograma
  - Identificação do ambiente de testes

1. Planejamento de testes

- Plano de testes
  - Revisado pela equipe (desenvolvimentos, analista de negócios, cliente)
  - Aprovado pelo gerente de projeto e pelo cliente
  - Plano de testes deve ser revisado para mudanças no projeto

2. Análise de teste e design

- Analista e designer de testes derivam casos de teste a partir dos requisitos
- Observam requisitos funcionais e não funcionais
- Os Casos de teste
  - cobrem todos os aspectos de cada requisito
  - cobrem todos os testes definidos nas estratégias de teste

2. Análise de teste e design

- Se automação for necessária
  - Designer deve escrever os scripts baseado nos casos de teste
- Casos de teste
  - revisados pelo líder de projetos, desenvolvedores, outros testadores, líder de teste, analista de negócios e cliente
  - aprovação pelo líder de teste e cliente
  - revisados para inclusão de novas funcionalidades ou descoberta de defeitos

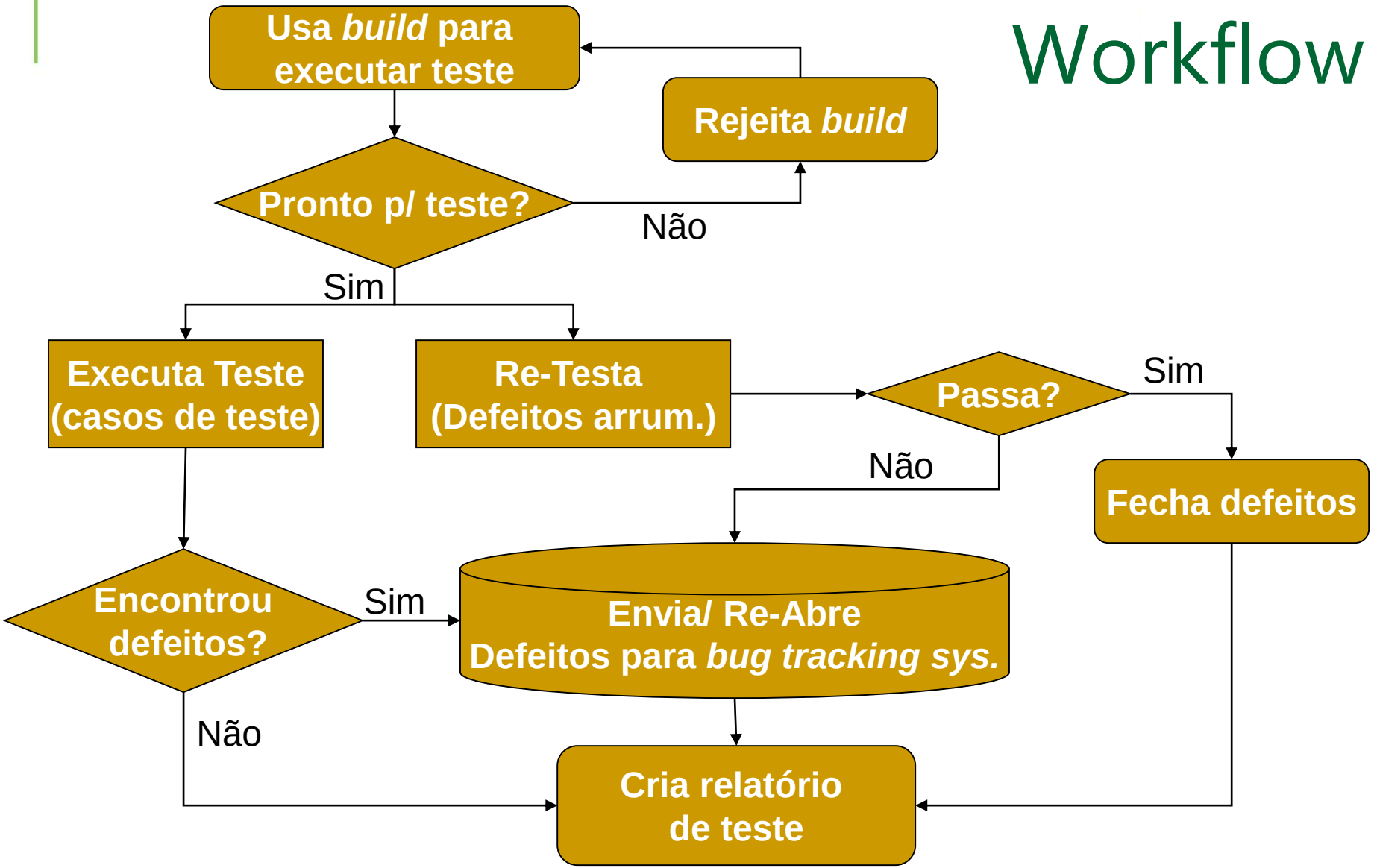
3. Execução dos testes

- Testadores seguem as atribuições definidas pelo líder de teste
- Executam testes seguindo os casos de teste
- Re-testam defeitos que foram descobertos e consertados
- Geram relatórios de defeitos
- Acompanham os defeitos até que sejam resolvidos

4. Relatórios de teste

- Gerente e líder de testes analisam defeitos usando sistema de *bug tracking*
- Geram o relatório de defeitos e avaliação de testes
- Calculam e revisam as métricas de testes
- Determinam se os critérios de teste passaram e se a fase de teste deve ser finalizada

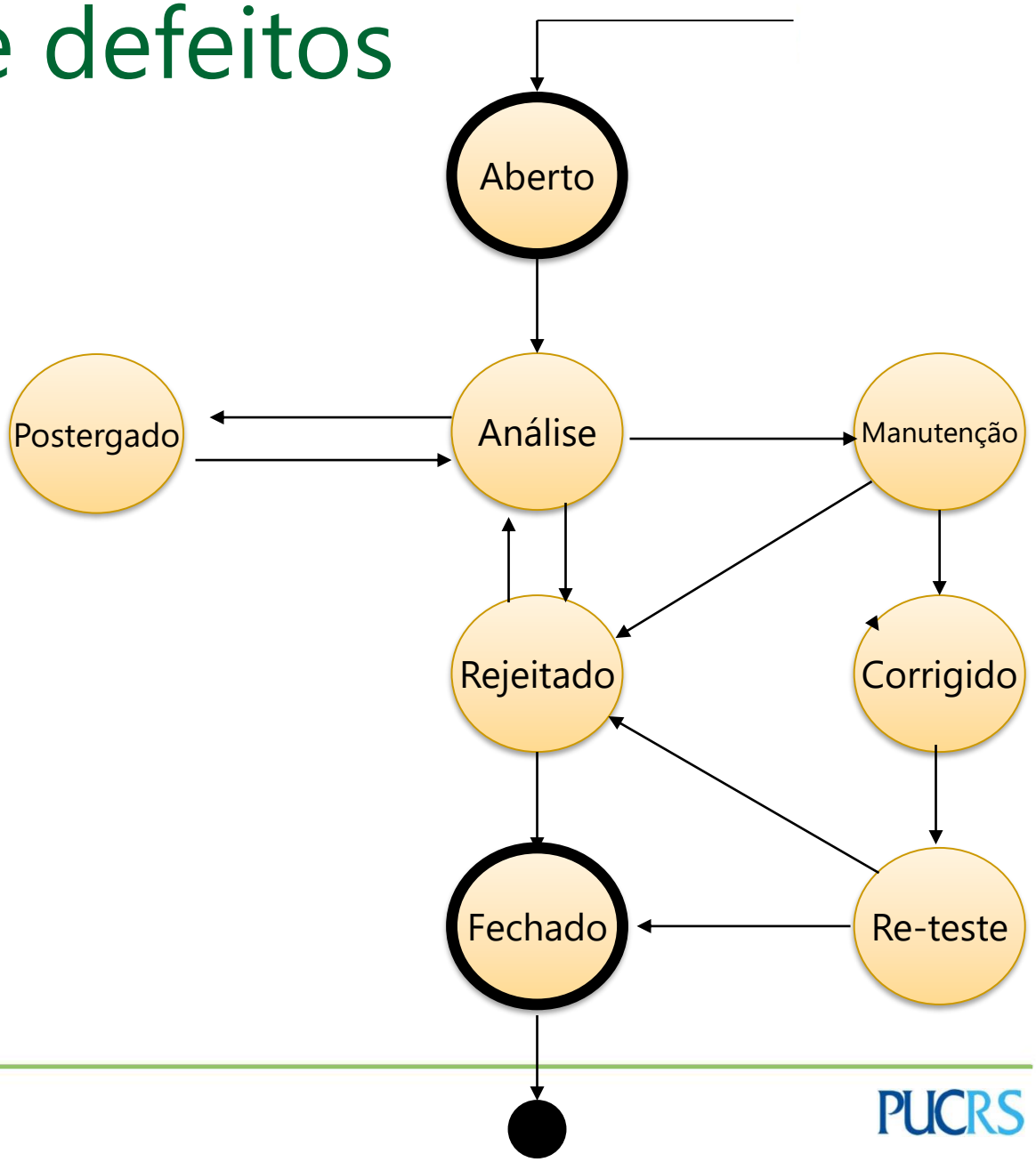
Workflow


Usa *build* para executar teste
Rejeita *build*
Pronto p/ teste?
Não
Sim
Executa Teste (casos de teste)
Re-Testa (Defeitos arrum.)
Passa?
Sim
Não
Fecha defeitos
Encontrou defeitos?
Sim
Envia/ Re-Abre Defeitos para *bug tracking sys.*
Não
Cria relatório de teste

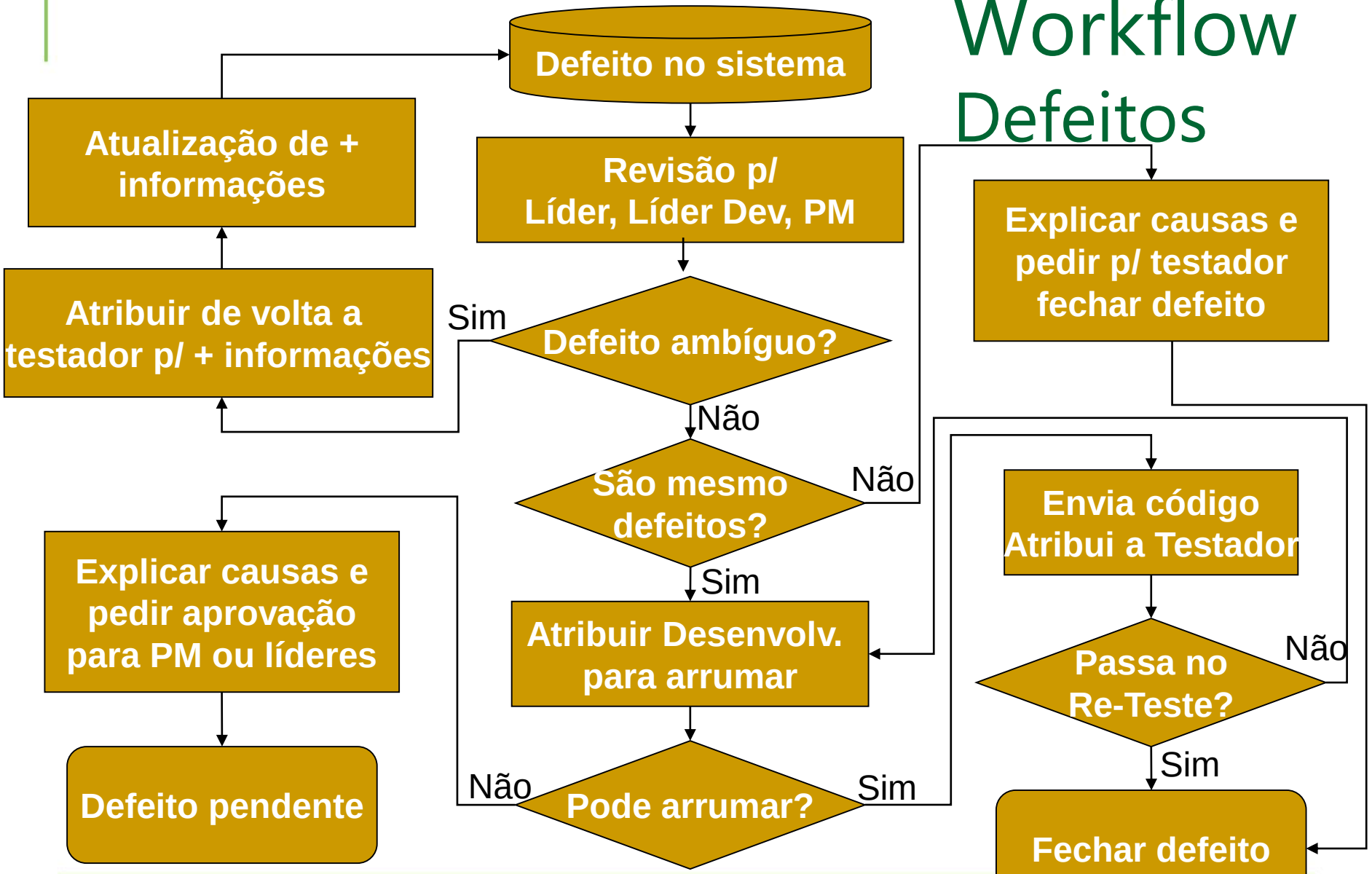
REGISTRO DE DEFEITOS

Registro de defeitos

- Um bom sistema de BugTracking deve permitir o cadastramento tanto das etapas do ciclo de desenvolvimento como do ciclo de vida dos Bugs.



Workflow Defeitos



Registro de defeitos

- *"Eu estava usando o sistema e então ele parou de funcionar"*
- **Prefira**
  - **Abra a aplicação**
  - **Clique em Procurar, escolha Palavra**
  - **Digite "defeito" no campo *Palavra***
  - **Clique no botão OK**

Registro de defeitos

- **Precisão** ao relatar defeito
- **Claridade**
  - Almeje a reprodução do erro por outros
  - *"no more no repro"*
  - Utilize as ferramentas ao seu favor
- Coloque **um** defeito por relatório
- Nenhum defeito é simples o suficiente para não ser registrado
- Separe ocorrência de **fatos** de **especulações**

Registro de Defeitos

- A medida que se aplicam os testes são detectados defeitos
- Os defeitos precisam ser registrados em um sistema de acompanhamento de defeitos ou *BugTracking*.
- Um sistema de *BugTracking* permite registrar o ciclo de vida de um defeito até seu fechamento.
- Permite também a coleta de métricas sobre a efetividade do processo de testes

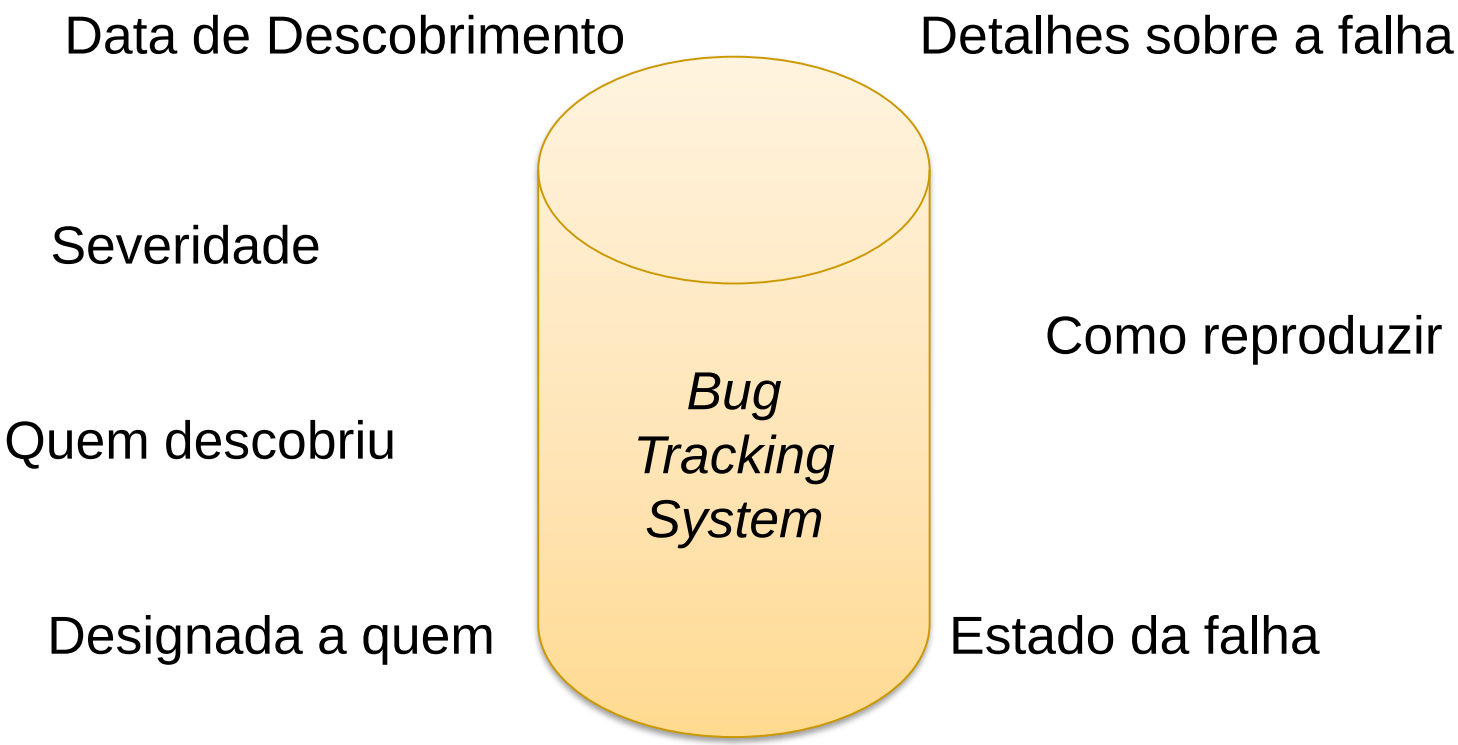
Sistemas de *bug tracking*

- MS MTM
- Bugzilla
  - http://www.bugzilla.org/
- Mantis
  - http://www.mantisbt.org/
- Trac (*issue tracking system*)
  - http://trac.edgewall.org/
- Redmine (*flexible project management*)
  - http://www.redmine.org/

Registro de defeitos

- Um software de larga escala pode ter uma série de defeitos, encontrados por diversas pessoas
- A pessoa que conserta o defeito (desenvolvedor) é diferente da pessoa que encontra o defeito
- Este processo não pode ser feito de forma informal
  - Deve ser possível a qualquer momento do projeto saber quantos defeitos existem e quem está consert.
- Defeitos são registrados e acompanhados até seu conserto
- Registro de defeitos é uma das melhores práticas da indústria de testes

Registro de defeitos



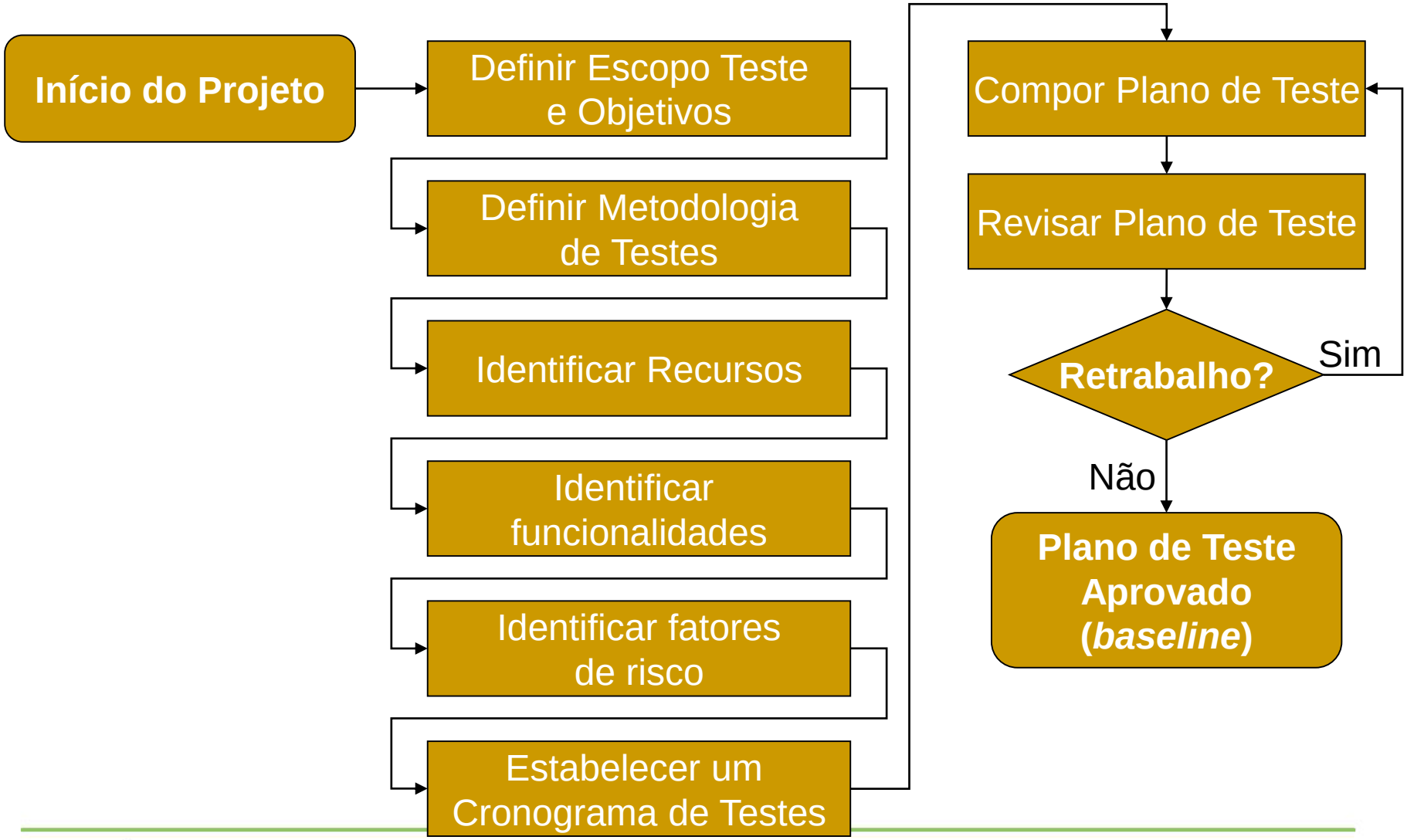
Registro de defeitos

- Saber registrar a **severidade** de um defeito é importante
- Severidade determina a prioridade do conserto dos defeitos
  - Determina os esforços de programação (recursos)
- Por exemplo, supondo 4 níveis
  - Crítico: Todos devem parar para consertar!
  - Severo: Tem um grande impacto no sistema
  - Menor: Defeito isolado, não afeta outros módulos
  - Cosmético: sem impacto na funcionalidade

Registro de defeitos

- Idealmente, todos os defeitos devem ser fechados
- As vezes, empresas liberam software com defeitos conhecidos (severidade baixa?)
- Organizações possuem padrões para liberação de produtos no mercado
- Registro de defeitos são usados para determinar a tendência de chegada e saída de defeitos
  - Produtividade de equipe de teste e desenvolvimento

Workflow geral



Atividade prática

- Procurar duas ferramentas de *bug tracking*
- Listar seus principais recursos
- Comparar as ferramentas em termos de funcionalidades

- Dúvidas
  - Qual funcionalidades procurar?
  - Outras...

PROCESSO DE TESTE

Processo de Teste – Fases

- **Geração** dos testes
  - Envolve análise da especificação do software
  - Escolha das funcionalidades que serão testadas
  - Determinação de como será executado o teste
  - Especificação de *scripts* de teste
- **Execução** dos testes
  - Desenvolvimento de um ambiente de teste em que o *script* pode ser executado (MS VS ou similar)
  - Execução do script de teste
  - Análise dos resultados

Processo de Teste – Outras Fases

- **Gerenciamento** e manutenção
  - Aplicação de testes em outros momentos
  - Gerenciar scripts, integração ao sistema de controle de versões
  - Observar os requisitos e mapear para novos testes
  - Organizar os testes, excluir passos conforme evolução do sistema
  - Utilizar (e documentar) outras ferramentas de teste

Processo de Teste – Automação

- **Automação** de testes
  - Clicar em um botão e mostrar todos os *bugs* a cada adição de funcionalidade ou *build*
  - Análise custo x benefício
  - Conhecimento do código-fonte e dos módulos construídos
  - Ferramentas para automação de teste
    - Devem permitir a criação de scripts e o seu gerenciamento
    - Fácil manipulação para novas funcionalidades

DECISÕES PARA CRIAÇÃO DE TESTES

Técnicas de geração de casos de teste

- Espaço de entrada de um programa ou conjunto de funções é grande
  - pode ser considerado infinito
- Por essa razão o teste aleatório não costuma ser o mais indicado
- Objetivo das técnicas de geração de casos de teste é
  - identificar, dentre este conjunto grande de casos de teste possíveis, os casos potencialmente mais reveladores de erros

Decisões para criação de Testes

- Questões importantes:
  - Quem deve realizar os testes?
  - O que testar?
    - Que partes devem ser mais cuidadosamente testadas?
  - Quanto teste é adequado?
  - Quando testar?
  - Como o teste deve ser realizado?

Quem deve realizar os testes?

Desenvolvedores

Compartilhamento de Responsabilidades entre Desenvolvedores e Testadores

Testadores

- Vícios de teste por parte dos programadores
- Testadores podem não ter o que testar

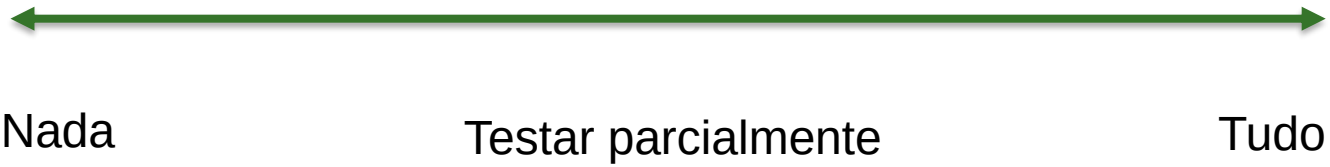
Quem deve realizar os testes?

- Desenvolvedor
  - Código é o resultado do seu esforço e trabalho
  - Procurar problemas é *´traumático´*
  - Não tem motivação para encontrar problemas
    - Pode resultar em mais trabalho...
- Testador
  - Se responsável por muitos testes, torna o processo lento e improdutivo

Quem deve realizar os testes?

- Compartilhamento de responsabilidades
- Mundo ideal: desenvolvedor é um testador
- Desenvolvedores criam código e testes para o código, podendo executar conjuntos de testes
- Testadores revisam testes criados, executam os testes, criam e revisam planos de testes
- **Comunicação** na equipe para entrega

O que testar?



- Princípio de Pareto adaptado ao teste de software
  - 20% dos componentes de software concentram 80% dos defeitos.
- Esforços devem ser concentrados nas partes mais importantes e/ou frágeis.

Quanto teste é adequado?

Nenhum teste

Teste Exaustivo

- O esforço de teste deve olhar o custo x benefício
  - Balanceamento entre tempo/custo do teste e a quantidade de defeitos encontrados
- Cobertura de teste
  - quantidade dos requisitos ou quantidade de linhas de código

Quando testar?

Todo dia

Na medida em que o software é desenvolvido

Testar no final

- Revisões e testes são atividades complementares
- 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 um cronograma de testes

Como testar?

Apenas com o conhecimento da implementação **(como)**

Apenas com o conhecimento da especificação **(o quê)**

Usando o conhecimento da especificação e da implementação **(o quê e como)**

- Definição de **Plano de Testes**
  - Previamente elaborado, descrevendo o escopo, o processo de teste definido, os recursos alocados, estimativas, cronograma e riscos

COMPETÊNCIAS IMPORTANTES

Competências de testadores

- **Competências 'soft'** (atributos não técnicos)
  - Disciplina e perseverança
    - Teste é repetitivo e exige muitos esforços manuais
    - Possuir a habilidade de trabalhar sob pressão devido a prazos
    - Dizer 'não' a gerentes quando qualidade é insuficiente
  - Leitura: estudo de manuais, documentações, especificações
  - Comunicação
    - Tanto verbal quanto escrita é muito importante
    - Perícias 'diplomáticas'
    - Comunicação com equipes técnicas e não técnicas, engenheiros, gerentes, clientes
  - Pensamento negativo: antecipar problemas, avaliar riscos
  - Atitude: 'testar para quebrar'
  - Gerência do tempo e atribuir prioridades aos esforços

Competências de testadores

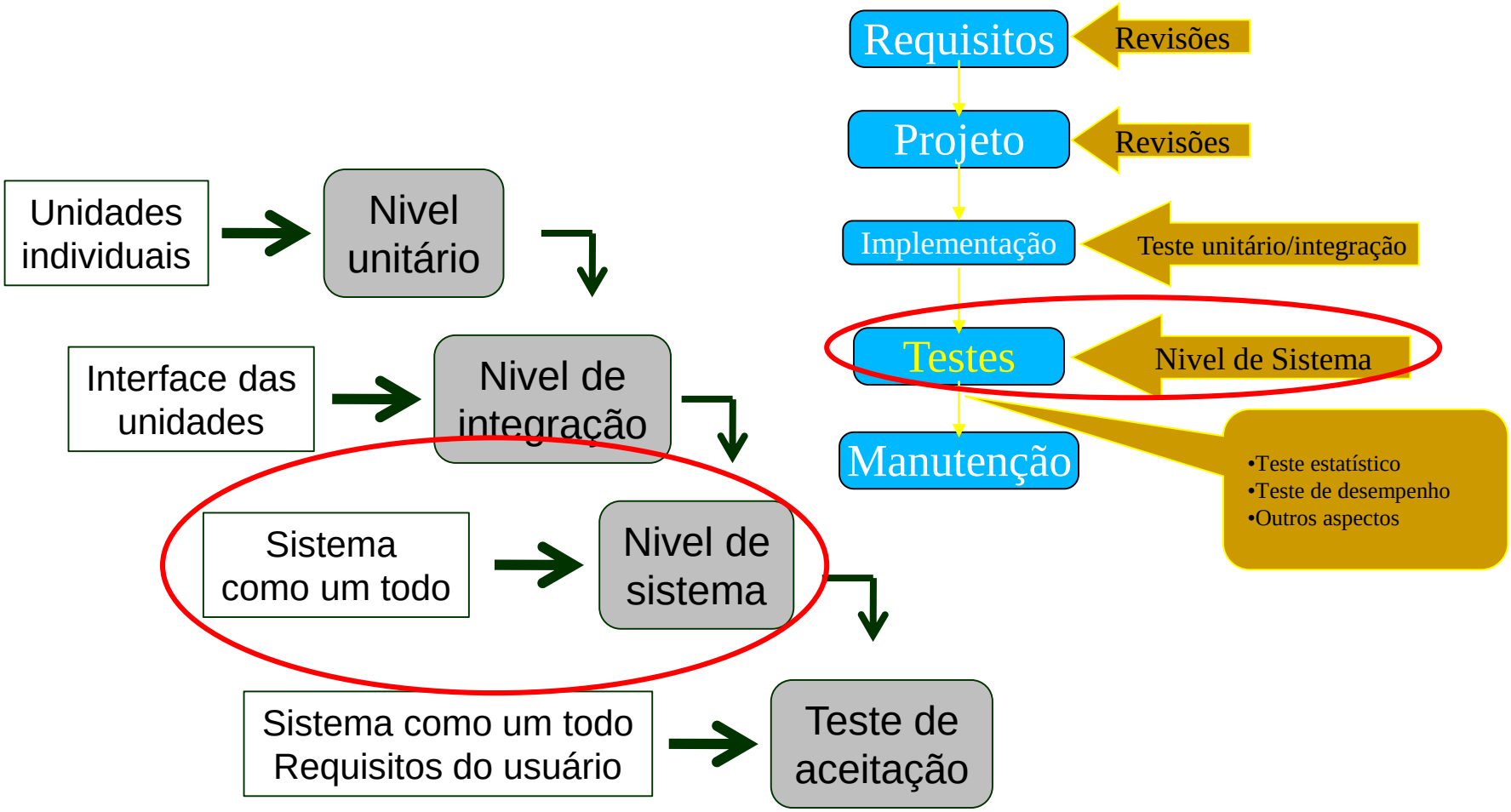
- **Competências técnicas**
  - Familiaridade com desenvolvimento de software arquiteturas, processos
  - Familiaridade com metodologias de teste
  - Entendimento de metodologias de desenvolvimento de software
  - Tradução de requisitos de software em casos de teste
  - Saber como e onde procurar por defeitos
  - Saber como escrever relatórios de defeitos e como reproduzir os defeitos encontrados

TESTE DE NÍVEL DE SISTEMA

Nível de Sistema

- É o teste do sistema como um todo
  - Uma *build* completa
- O teste é "pensado" do ponto de vista do usuário.
- Os testes são executados através da interface do sistema com seu ambiente
  - Interface com o usuário
  - Interface com hardware
  - Interface com outras aplicações

Localizando o Nível de Sistema

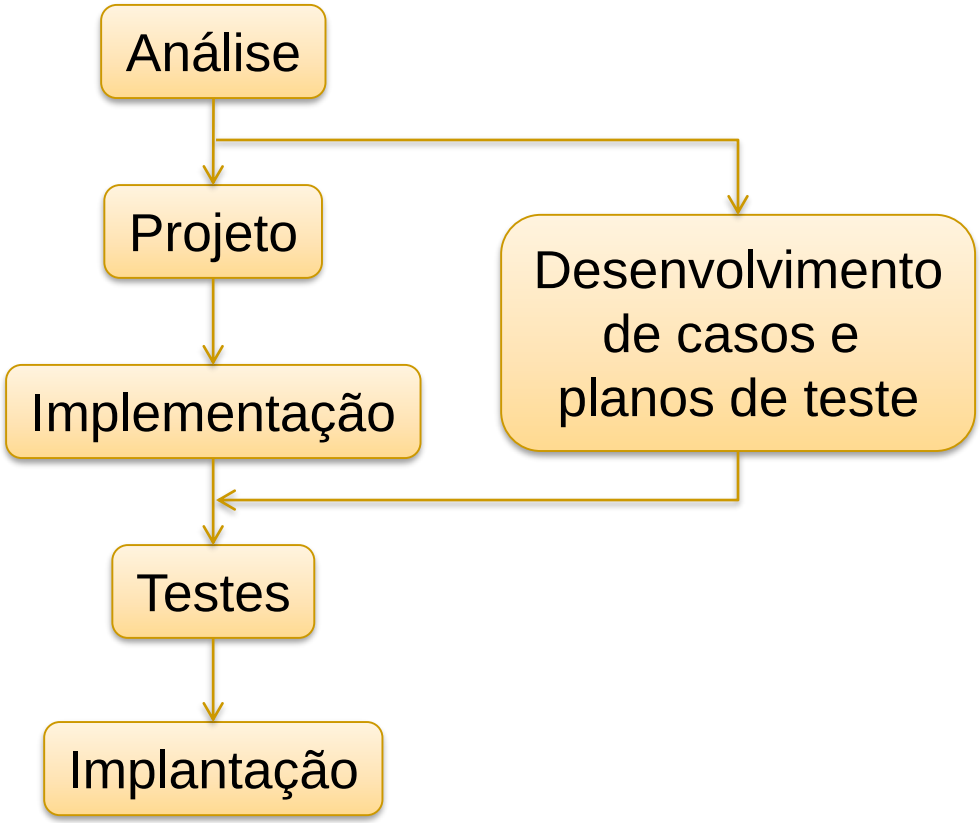


Os testes do nível de sistema são projetados antecipadamente

- O código do programa não é necessário
  - Apenas a descrição do comportamento esperado é necessária
  - Mesmo especificações incompletas e informais podem ser usadas
    - Ainda que não sejam imprescindíveis, especificações completas levam a melhores conjuntos de testes
- O teste antecipado tem efeitos colaterais:
  - Freqüentemente revela ambiguidades e inconsistências na especificação
  - Útil para garantir a testabilidade
  - Explicação da especificação
    - Ou em casos extremos (como em XP), os casos de teste são a especificação

Os testes do nível de sistema são projetados antecipadamente

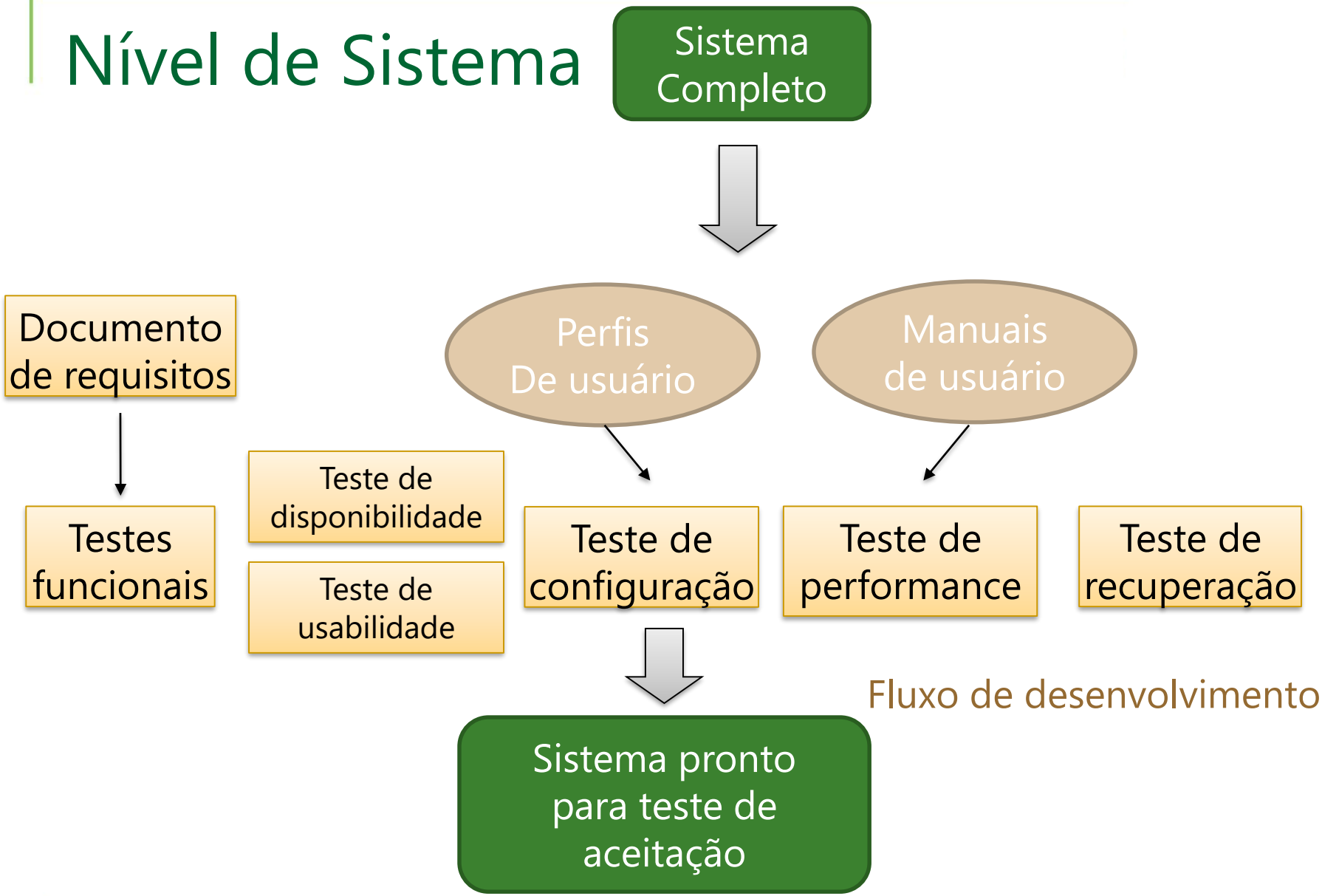
- A especificação de requisitos norteia a elaboração dos casos de teste.
- O desenvolvimento do plano de testes é feito em paralelo e de forma integrada com o desenvolvimento
- Implicação: a equipe de teste deve participar desde a etapa de análise !!!



Aspectos testados

- No nível de sistema testam-se aspectos funcionais e não funcionais:
  - Desempenho
  - Teste de configuração
  - Teste de segurança
  - Teste de recuperação
  - Usabilidade
  - Disponibilidade

Nível de Sistema



Recursos para o Nível de Sistema

- Exige grande quantidade de recursos
- Laboratório
  - Ambiente de testes
  - **Jamais** executar Nivel de Sistema em ambiente de desenvolvimento ou produção
- Tempo:
  - Definição das estratégias
  - Desenvolvimento dos casos de teste
  - Desenvolvimento do plano de teste
  - Aplicação do plano de teste
- Pessoal (equipe de testes)
  - Líderes e Analistas/designers de teste (“engenheiro de teste”)
  - Testadores

Equipe de teste

- Tamanho da equipe varia conforme o projeto e a disponibilidade de recursos
- Objetivo da equipe de teste
  - qualificar o sistema
- **Não** é objetivo da equipe de teste
  - avaliar a equipe de desenvolvimento
- Qual o custo envolvido? Ainda vale a pena?

Teste

- Teste: implementação de um caso de teste
  - Seqüência de
    - Passos
    - Resultados esperados
  - Características
    - Clareza
    - Repetibilidade
  - Exemplo (padrão ANSI IEEE 829/1991)
    - Número do teste
    - Dependências
    - Objetivo
    - Ambiente
    - Passos
    - Resultados esperados

Plano de Teste - Nível de Sistema

- Documento que agrega os testes relacionados a um produto.
- Contém:
  - Dados do produto (identificação, versão etc.)
  - Necessidades de ambiente
  - Conhecimentos e habilidades necessárias
  - Testes
- O versionamento é **fundamental**
  - Builds, Casos de Teste, etc
- A elaboração de um plano de testes pode consumir até 1/3 do tempo dedicado aos testes.

Plano de Teste - Nível de Sistema

- Sugestão
  - cada teste deve verificar apenas uma funcionalidade por vez
- Quando da ocorrência de uma falha, o problema já está isolado.
  - Simplifica-se o relato da falha.
  - Evita-se mascaramento de falhas.

Ferramentas de apoio

- Gerenciadores/editores de planos de teste
  - *Exs: Microsoft Test Manager, Quality Center (Mercury)*
- Execução automática de testes
  - *Capture-playback, Scripts*
  - *Exs: Microsoft VSTS 2010, Qtp (Mercury)*
- Gerenciamento de defeitos (bug-tracking)
  - *Exs: Microsoft VSTS 2010,* Mantis
- Geradores de carga
  - Exs: *Microsoft VSTS 2010, LoadRunner (Mercury)*

PLANO DE TESTES

Exercitar o sistema

- Verificar se os Planos de Teste estão exercitando o sistema como um todo
  - Ou focam apenas em um pedaço
- Difícil de se concretizar sem o código-fonte
- Possibilidade: usar a ideia da complexidade ciclomática para este fim

Redigindo casos de teste

- Como expressar um teste
  - Tabela: <dados de entrada, resultados esperados>
  - Descrição textual dos passos e resultados
  - Script de uma ferramenta de automação
    - Exemplo: capture-playback
  - Classes ou métodos de teste

Bons casos de teste...

- **Encontram defeitos**
  - Objetivo primordial de se testar qualquer software
- **Maximizam contagem de defeitos**
  - Este índice é mais importante que a cobertura
- **Bloqueiam entrega do produto ao cliente**
  - Conseguem parar um produto defeituoso
- **Minimizam custos relacionados ao suporte técnico do produto**
  - menos defeitos, menos ligações ao *helpdesk*

Bons casos de teste...

- **Determinam consonância com especificação**
  - Todas as funcionalidades estão presentes
- **Conformidade a regulação**
  - Em sistemas críticos, verifica a segurança, por ex.
- **Determina qualidade**
  - Qualidade é multidimensional, determinar critérios de qualidade antes, para poder medir no software

TÉCNICAS DE GERAÇÃO DE CASOS DE TESTE

Passos essenciais

- Quebrar a especificação em partes **menores**
- Identificar causas e efeitos
  - Construir uma lista de condições que devem ser satisfeitas conforme as entradas
  - Esta lista pode possuir um identificador **único**
- Construir uma **sequência** de passos para cada condição a ser avaliada
- Testar **múltiplas** entradas de dados para cada condição

Passos essenciais – revisões

- Descobrir os casos de teste mais importantes
- Revisar casos de teste criados
- Quebrar casos complexos em casos mais fáceis
- Unir testes simples (validação de um formulário)
  - Diversos passos para validar todos os campos
  - Falha em um dos passos; se passar todos, sucesso
- Excluir casos repetidos

PLANOS DE TESTE, MODELOS E BOAS PRÁTICAS

Plano de Teste

- Modelo de Plano de Teste
  - Padrão IEEE 829: *Standard for Software and System Test Documentation*
  - http://en.wikipedia.org/wiki/IEEE_829

IEEE Standard 829 for Software Test Documentation

- Plano de Testes
- Especificação de Arquitetura de Teste
- Especificação de Casos de Teste
  - Identificadores para Especificação de Casos de Teste
  - Itens a serem testados
  - Especificação de Entrada e Saída
  - Necessidades de Ambiente (HW,SW)
  - Requisitos especiais de processos
  - Dependências internas para casos de Uso
- Especificação do Procedimento de Testes
- Relatório de Comunicação de Item testado
- Log de Teste

Construção de Planos de Testes

- Objetivos
  - Redução de riscos para o projeto
  - Medir qualidade do produto
- Um Plano de Teste
  - Contém diversos Casos de Teste
- Cada Caso de Teste
  - Descreve os testes para as maiores prioridades do projeto
  - Possui um único objetivo por vez

Qualidades de bons Planos de Teste

- Fáceis de serem lidos
- Não pode ser muito textual
  - Baseado em tópicos
- Rápido e intuitivo para mostrar os resultados
  - Interpretação 'instantânea' para tomada de decisão
- Auto-explicativos
- Curtos e diretos ao ponto

Boas práticas

- Revisar sempre os testes
  - evitar paradoxo do pesticida
- Evidenciar testes repetidos
  - Excluir um dos testes
- Identificar testes que passam sempre e arquivá-los, ou alterar status/criticidade
- Observar os efeitos colaterais dos consertos de defeitos
  - Novos defeitos podem ter sido introduzidos

Práticas ruins – como falhar

- Deixar os testes para o final do projeto
- Buscar 100% de automação de testes
- Excesso de documentação gerada na descoberta de defeito
  - Foco na descoberta e sinalização do defeito
- Deixar usuários encontrar defeitos por não ter criados testes compreensivos
- Nunca revisar testes ou mexer nos existentes

Linguagem de descrição de teste

- Formato imperativo
  - “clique no botão OK”, “digite 10”
- Use nomes exatos e consistentes para os campos
  - Não seja genérico
- Cada teste não tem mais de 10 a 15 passos
  - Se possível
- Siga convenções de nomes

Testes comuns

- Identificador
- Descrição
- Prioridade
- Pré-condição
- Pré-requisitos
- Dependências
- Entradas
- Instruções de utilização
- Resultados esperados
- Resultado Atual
- Pós-condições
- Resultado (passou/falhou)
- Número da versão

Exemplo prático

- Aqui mostrar a tabela
- Mostrar como usar pré-condições
- Mostrar como entrar com dados em um formulário em um passo apenas
- Mostrar o objetivo do caso de teste

Recomendações

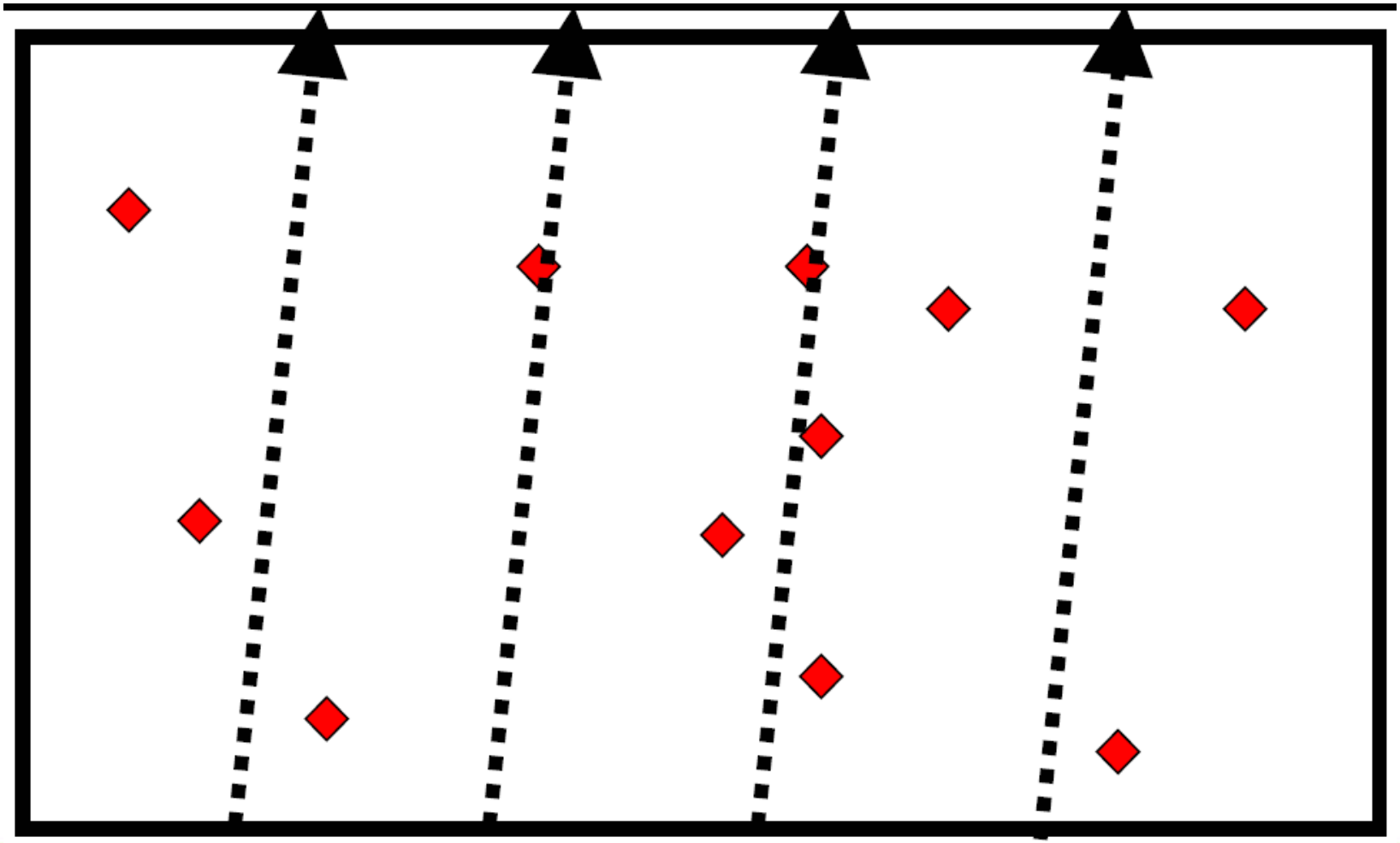
- Escrever testes com os requisitos
- Seguir um padrão, facilitando a comunicação
- Priorizar os testes por valor de negócio
- Agrupar testes por domínio
- Objetivo: focar no **valor** e no **resultado**

TESTES DE REGRESSÃO

Teste de Regressão

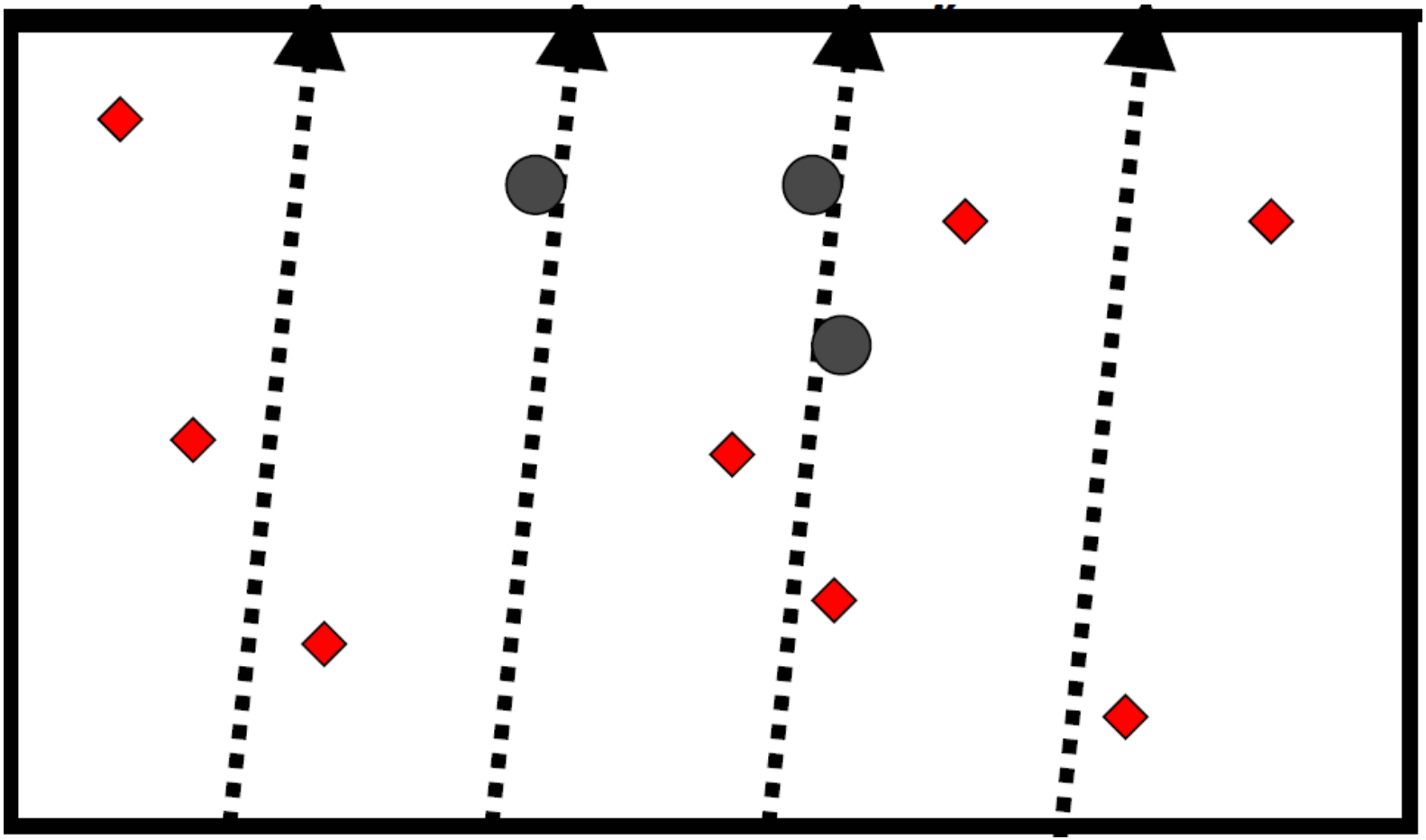
- Realizados a cada mudança no software
- Revisão dos testes efetuados para conter novos testes que exercitem o que foi alterado
- Uso ou não de uma nova equipe de teste
  - Realização dos testes de forma independente

Problema limpar uma mina

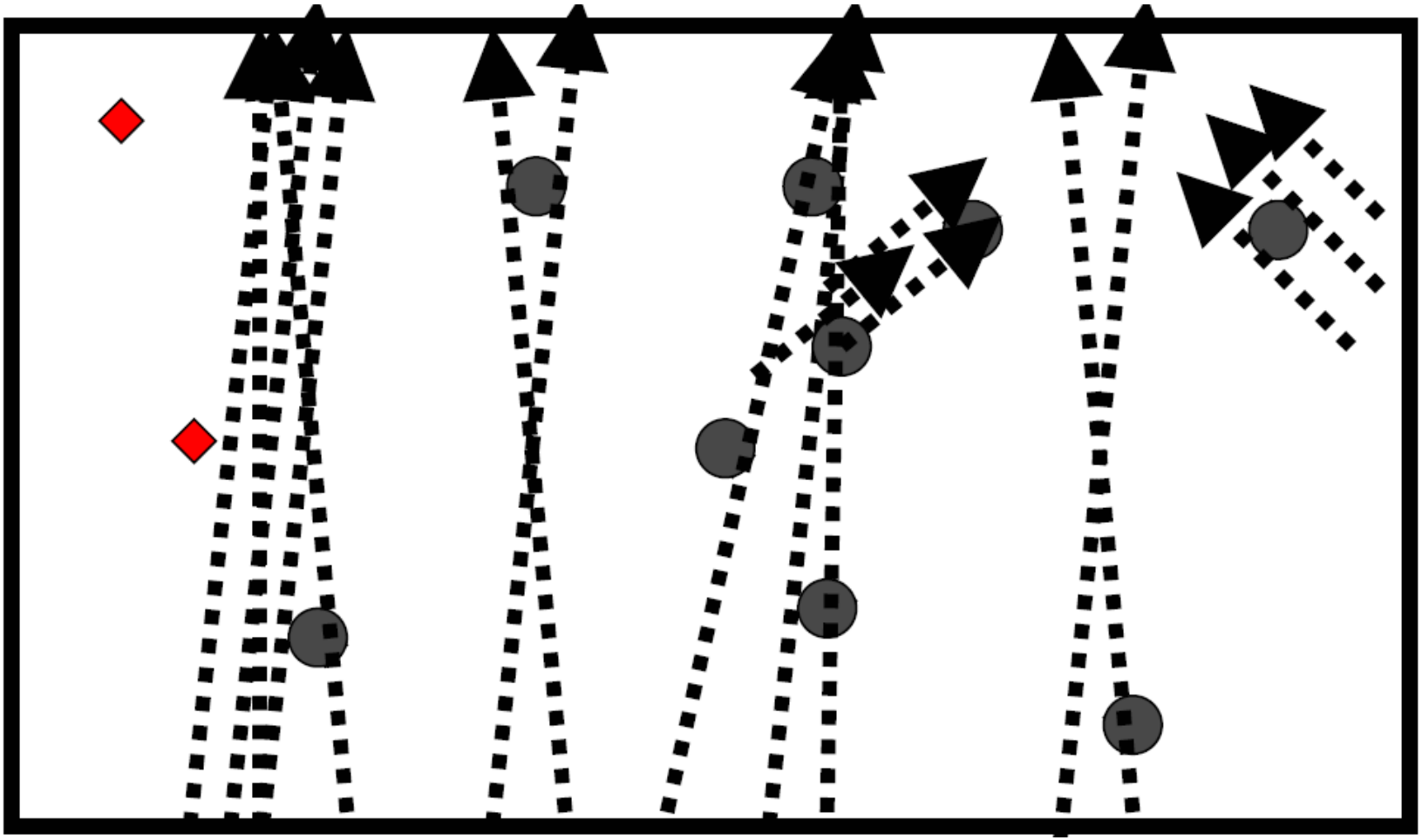


Fonte: Esta analogia foi primeiramente apresentada por Brian Marick.
Estes slides são de James Bach.

Testes repetidos não encontrarão novas minas!



Testes variados são mais efetivos!



TESTES DE ACEITAÇÃO

Teste de Aceitação

- Testam uma funcionalidade, história (*user story*) ou um caso de uso (*use case*)
- Ferramentas
  - MTM, JUnit, Selenium
- Verificam se um requisito foi ou não atendido
- Como fazer este teste?
  - Idealmente antes da implementação (dificuldade)
  - Idealmente serão automatizados
  - Garantem que o código realiza o que se propõe

Teste de Aceitação

- Início do projeto
  - Explorar os requisitos existentes
  - Definir os testes de aceitação para todos os UCs
- A cada iteração (se esta for a metodologia...)
  - Testar se o software atende os requisitos dos UCs

EXERCÍCIOS

Microsoft | Innovation Center
PUCRS

PUCRS

Prof. Marco Mangan
Prof. Ricardo M. Czekster

Faculdade de Informática
04/09/2017