



FET

Fundamentos da Engenharia de Testes

ctr.rsinet.com.br
FET – Fundamentos da Engenharia de Testes – Versão: 2.0

2007 © Copyright RSI Informática Ltda 

Programação do Curso

- Apresentação, Introdução e Objetivos
- Fundamentos do Teste
- Teste Durante o Ciclo de Vida do Software
- Técnicas Estáticas
- Técnicas de Modelagem de Teste
- Gerenciamento de Teste
- Ferramentas de Suporte a Teste
- Referências



Instalações e Material

- Material de Treinamento
 - Apostila
 - Apostila do *Syllabus Foundation Level*
 - Caderno de exercícios
 - Simulado do exame de certificação



Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0

3

ctr.rsinet.com.br



Recomendações



Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0

4

ctr.rsinet.com.br



Expectativa



Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0

5

ctr.rsinet.com.br



Apresentação, Introdução e Objetivos

Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0

6

ctr.rsinet.com.br



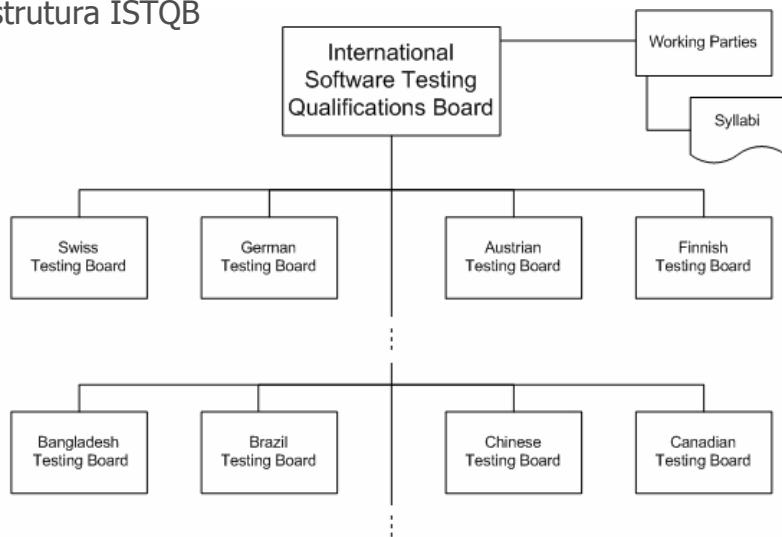
Apresentação, Introdução e Objetivos

■ ISTQB e BSTQB

- O ISTQB (*International Software Testing Qualifications Board*) é um conselho internacional responsável pela qualificação internacional em testes para a qual este curso prepara.
- O ISTQB é responsável pela definição de:
 - Syllabus: o corpo de conhecimento requerido para alcançar certificação ISTQB.
 - Glossário de termos de teste de software.
 - Código de conduta e ética que devem ser seguidos por todos os conselhos regionais que o compõem.
- O ISTQB é composto por diversos conselhos em mais 40 países.
- O BSTQB (*Brazilian Software Testing Qualifications Board*) é o conselho nacional do Brasil.
 - Único órgão que tem permissão de aplicar a certificação do ISTQB no país.
 - Tem a missão de promover o profissionalismo e reconhecimento da disciplina de testes e qualidade de software no Brasil.

Apresentação, Introdução e Objetivos

■ Estrutura ISTQB



Apresentação, Introdução e Objetivos

- A certificação ISTQB em números:
 - Número de certificados (CTFL e CTAL) no mundo:
 - Maio/06: 33.218
 - Abril/07: 56.032
 - Março/08: 81.855
 - Março/09: 111.446
 - O Brasil
 - Em três anos de formação e reconhecimento do BSTQB, o número acumulado de certificados:
 - 2006: 20 (+8 pelo ASTQB)
 - 2007: 104
 - 2008: 274
 - 2009 (abril/09): 391

Apresentação, Introdução e Objetivos

- *Foundation Level Syllabus* do ISTQB de 2007.
 - Apresenta todo o corpo de conhecimento necessário para a certificação.
 - Desenvolvido por oito autores de sete países: Thomas Müller (presidente), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson e Erik van Veendendaal.
 - Posteriormente atualizado por quatro revisores: Thomas Müller (presidente), Dorothy Graham, Debra Friedenberg e Erik van Veendendaal.
 - Revisado por diversos pares e aprovado por conselhos regionais.
- Os termos vêm do Glossário ISTQB em português.
 - Na prática há uma grande variação na terminologia, inclusive nas traduções aceitas e usadas dos termos em inglês.
 - Nem todos os termos no syllabus são definidos no glossário.

Apresentação, Introdução e Objetivos

- O exame para a certificação CTFL, *Certified Tester Foundation Level*, do ISTQB/BSTQB.
 - Composto de 40 questões de múltipla escolha. É necessário mais de 60% de acerto para passar (mínimo de 25 questões).
 - Período de 1 hora para responder as questões e preencher o gabarito (~1,5 min por questão!).
 - Aplicado simultaneamente em várias capitais brasileiras.

	Tempo sugerido no syllabus	Porcentagem do tempo total	Número de questões (aproximado)
capítulo 1	115	15	6
capítulo 2	115	15	6
capítulo 3	60	10	4
capítulo 4	255	30	12
capítulo 5	180	20	8
capítulo 6	80	10	4

Apresentação, Introdução e Objetivos

- O curso
 - Este é um curso de treinamento preparatório para obtenção da certificação CTFL.
 - O material deste curso é credenciado pelo BSTQB. 
Brazilian Software Testing Qualifications Board
 - O material foi preparado para ajudá-lo a tornar-se um profissional de teste mais efetivo e eficiente.
- Exercícios
 - O curso apresenta diversos exercícios, mantenha um caderno para fazer anotações.
- O instrutor
- Horários e *coffee-break*
- **Você pode se certificar!!**

Apresentação, Introdução e Objetivos

- Os principais tópicos cobertos pelo curso coincidem com os capítulos do *Foundation Level Syllabus ISTQB*.
 1. Fundamentos do Teste
 2. Teste Durante o Ciclo de Vida do Software
 3. Técnicas Estáticas
 4. Técnicas de Modelagem de Teste
 5. Gerenciamento de Teste
 6. Ferramentas de Suporte a Teste

Apresentação, Introdução e Objetivos

- Seus objetivos com o curso.
 - Tome alguns minutos escrevendo o que você gostaria de obter deste curso.
 - Não deixe o curso terminar com algum objetivo não atendido!
-
-
-
-
-
-
-

Apresentação, Introdução e Objetivos

- Este curso é **SEU!**
- Participe:
 - ? Faça perguntas;
 - ! Faça comentários;
 - 👉 Compartilhe experiências;
 - 👎 Outras opiniões e desacordos são bem vindos!



Boas seções têm muito disso...

- Por que você está aqui?
- O que você quer aprender, discutir, e ensinar?



... assim como disso.

Capítulo 1: Fundamentos do Teste

Capítulo 1: Fundamentos do Teste

Seções:

1. Por Que É Necessário Testar?
2. O Que É Teste?
3. Princípios Gerais do Teste
4. Fundamentos do Processo de Teste
5. A Psicologia do Teste

Capítulo 1: Fundamentos do Teste

Seção 1.1 – Por Que É Necessário Testar?

- Conceitos chave:
 - Como os *bugs* podem causar dano?
 - Os *bugs* e seus efeitos.
 - A necessidade do teste.
 - O papel do teste na garantia da qualidade.

As Ameaças dos *Bugs*

- Companhia/Empresa
 - Reputação relativa à qualidade danificada.
 - Custos de manutenção altos e não previsíveis.
 - Atrasos inesperados nos ciclos de lançamento do software.
 - Falta de confiança no sistema.
 - Processos.
- Ambiente
 - Poluição.
 - Desperdício.
- Pessoas, sociedade, governo
 - Perda de postos de emprego.
 - Perda de vidas.
 - Perda de direitos.
 - Perda de missões.
 - Perda de guerras.

De Onde os *Bugs* Vêm e o Que os *Bugs* Fazem

- As pessoas cometem erros que adicionam *bugs* (defeitos) no sistema.
 - Requisitos e especificação de projeto.
 - Código (lógica de negócio e interface de usuário).
 - Documentação (eletrônica e cópia física).
- Quando um código defeituoso é executado, falhas ocorrem.
- Se essas falhas são visíveis aos clientes, usuários, ou outros interessados, pode resultar em uma insatisfação com a qualidade do sistema.

De Onde Vêm os *Bugs* e as Falhas?

- *Bugs* ocorrem devido a...
 - Falibilidade de programador, analista, e outros contribuidores individuais (incluindo testadores).
 - Pressão de tempo.
 - Complexidade de código, infraestrutura, ou problema a ser resolvido.
 - Mudança e junção de tecnologia.
 - Muitas interações do sistema.
- Falhas ocorrem devido a *bugs* e...
 - Condições ambientais.
 - Mau uso (deliberado ou acidental).



Teste para Gerenciar Riscos de Qualidade

- Riscos e restrições para um projeto de software:
 - Funcionalidades: Conjunto correto.
 - Prazo: Rápido o suficiente.
 - Custo: Aceitavelmente barato.
 - Qualidade: Útil para os clientes/lançamento/próximo passo.
- O teste provê a informação para guiar o projeto, reduzir e gerenciar os riscos, e reparar os problemas importantes.
- O teste também pode corresponder às necessidades de conformidade, contratuais, e regulatórias.

O Que "Qualidade" Significa para Você?

- "Apto ao uso" vs. "Conformidade com os requisitos".
- Teste e qualidade.
 - O teste fornece confiança quando encontra poucos *bugs*.
 - Testes OK: reduz o nível de risco de qualidade.
 - Teste não OK: dá a chance de aumentar a qualidade.
 - O conjunto de teste permite uma avaliação da qualidade.
- Quais são as características importantes de qualidade para seu sistema? Você as está testando (suficientemente)?
- Teste, garantia da qualidade, e melhoria da qualidade.
 - Idealmente, o teste é parte de uma estratégia maior de garantia da qualidade para um projeto.
 - Para projetos futuros, analise as causas raízes dos defeitos encontrados nos projetos atuais, e caminhe para reduzir a incidência.

Capítulo 1: Fundamentos do Teste Seção 1.1 – Por Que É Necessário Testar?

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 1: Fundamentos do Teste

Seção 1.2 – O Que É Teste?

- Conceitos chave:
 - Objetivos comuns do teste.
 - O propósito do teste.....
no desenvolvimento do software, manutenção, e operações...
... para encontrar defeitos, prover confiança e informação, e prevenir defeitos.

Objetivos do Teste

- Objetivos gerais típicos do teste:
 - Encontrar **falhas** e prover os programadores com a informação que eles necessitam para corrigir os **bugs** importantes.
 - Adquirir **confiança** sobre o nível de qualidade do sistema.
 - **Prevenir** defeitos (através de envolvimento no início de desenvolvimento, em revisões e projeto de testes avançados).
 - Fornecer **informação** sobre os aspectos mais importantes da qualidade do sistema sob teste.
 - Auxiliar a gerência a entender a **qualidade** do sistema.
- Você pode pensar em outros objetivos para seus projetos?
- Você pretende alinhar seus planos e ações com os objetivos de teste estabelecidos pela gerência?

Fases do Teste e Objetivos

- Teste de Unidade/Componente: Encontrar *bugs* nas partes individuais do sistema sob teste antes que as partes sejam totalmente integradas no sistema.
- Teste de Integração Encontrar *bugs* nos relacionamentos e interfaces entre pares e grupos de componentes do sistema sob teste a medida que as partes se juntam.
- Teste de Sistema: Encontrar *bugs* no comportamento geral ou particular, funções, e respostas do sistema sob teste como um todo.
- Teste de Aceite/Piloto: Demonstrar que o produto está pronto para entrega/liberação, ou avaliar a qualidade e fornecer informação sobre o risco da entrega/liberação.
- Teste de Manutenção: Checar erros introduzidos durante a realização de mudanças no software.
- Teste Operacional: Avaliar as características não funcionais do sistema, tais como confiabilidade e disponibilidade, geralmente no ambiente operacional.

Efetividade e Eficiência

Efetivo:

- Produzir um resultado decidido, decisivo, ou desejado; impressionante.
- Para sermos testadores **efetivos** nós devemos selecionar o objetivo apropriado e os resultados desejados.

Eficiente:

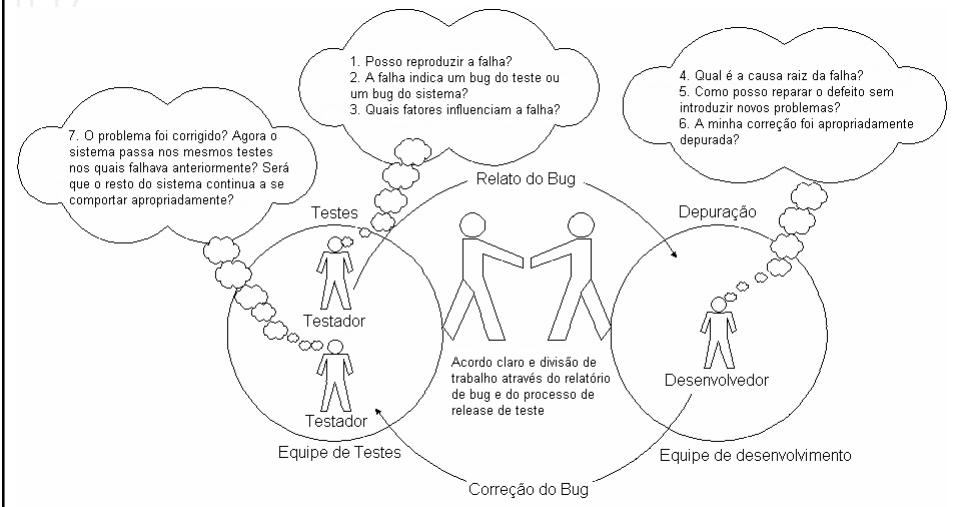
- Produtivo para o efeito desejado; especialmente produtivo sem desperdício.
- Para sermos testadores **eficientes** nós devemos alocar recursos (tempo e dinheiro) de forma apropriada.

Esses termos possuem significado maior no contexto de todo o desenvolvimento ou processo de manutenção, não apenas no processo de teste.

Teste vs Depuração

- Teste encontra falhas que são causadas por *bugs*.
- A depuração...
 - ...identifica a causa raiz de uma falha...
 - ...corrige o código...
 - ...e checa que o defeito tenha sido corrigido corretamente.
- Teste de confirmação garante que a correção resolveu a falha observada.
- As diferentes responsabilidades que devem ser preservadas:
 - **Testadores testam.**
 - **Programadores depuram.**

Encontrar – Depurar – Confirmar



Além da Execução do Teste

- Testar não é apenas executar testes em um sistema em execução.
- Outras atividades de teste, antes e depois da execução, incluem...
 - Planejar e controlar.
 - Escolher as condições de teste.
 - Projetar casos de teste.
 - Checar resultados de teste.
 - Avaliar critério de saída.
 - Relatar resultado de teste.
 - Tarefas de encerramento dos testes.
- Nós vamos voltar nesse ponto mais à frente.

Capítulo 1: Fundamentos do Teste Seção 1.2 – O Que É Teste?

- *Exercício – Os Testes do Triângulo (Myers)*
- Um programa aceita três inteiros representando os comprimentos dos lados de um triângulo.
- Ele tem como saída: “escaleno” (nenhum lado igual), “isósceles” (dois lados iguais), ou “equilátero” (três lados iguais).
- Escreva um conjunto de casos de teste efetivo (encontra bugs comuns) e eficiente (o menor número de testes possível). Normalmente, um caso de teste consiste de uma ação do testador, dados, e resultados esperados, mas aqui a ação é genericamente “dados de entrada”.
- Discuta.

Capítulo 1: Fundamentos do Teste
Seção 1.2 – O Que É Teste?

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 1: Fundamentos do Teste

Seção 1.3 – Princípios Gerais do Teste

- Conceitos chave:
 - O teste revela a presença de *bugs*.
 - Impossibilidade de teste exaustivo.
 - Benefícios de testar cedo.
 - Aglomerados de *bugs*: o agrupamento de defeitos.
 - O paradoxo do pesticida.
 - O teste deve se adaptar a necessidades específicas.
 - Falácia da ausência de erros.

O Teste Revela a Presença de Bugs: Uma Parábola

- Você tem uma bela horta, mas um dia você vê folhas comidas nos tomateiros.
- "Oh não", você pensa, "*Eu tenho lagartas na minha horta!*"
- Você sabe que existem pragas na sua horta.
- Mas se você não tivesse visto os sintomas, você poderia ter **certeza** de que não havia pragas na horta?
- Algumas pragas são fáceis de descobrir, outras não.

- Os testes podem revelar a presença de *bugs*, mas não podem provar sua ausência.



Missão Impossível: Teste Exaustivo

- :(“Apenas tenha certeza de que o software funciona antes que nós o despachemos...”
- Esta frase é demonstravelmente impossível.
 - Os caminhos de execução em software não trivial são quase infinitos.
 - Grandes fluxos de dados separados no espaço (características) e tempo (dados estáticos).
 - Pequenas mudanças podem causar regressões que não são lineares em relação ao tamanho da mudança.
 - Uma miríade de perfis de uso e configurações em campo: alguns desconhecidos e alguns que não há como saber.
 - Linha de base: teste exaustivo, ou seja, todas as combinações de entradas e pré-condições, não é possível.

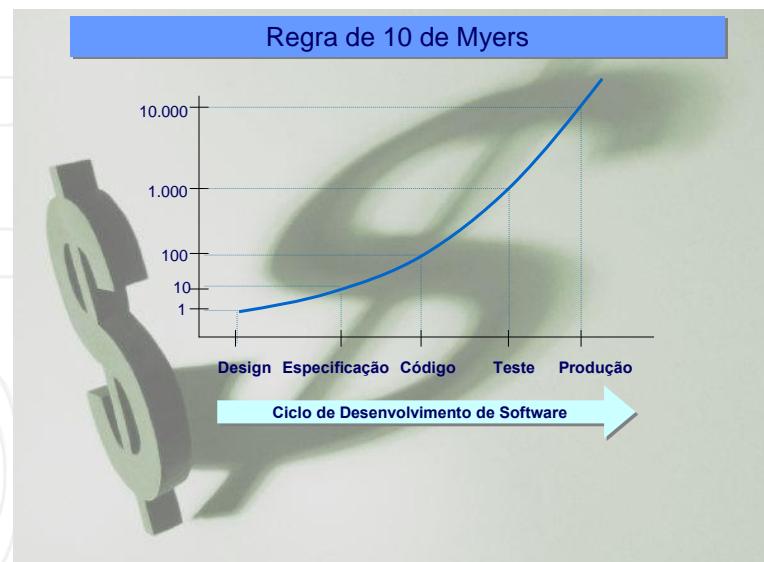
Neutralizando Expectativas de Teste Exaustivo

- Teste exaustivo como forma de provar que o software funciona é uma expectativa (errônea) comum.
- MÁS expectativas criam problemas para os profissionais de teste e equipes de teste.
 - Altas demandas inatingíveis no grupo de teste.
 - Sensação de incompetência quando essas demandas não são atendidas.
- Testadores devem estar prontos para comunicar como o teste pode contribuir para o software (com palavras que os *stakeholders* do projeto possam compreender).

Benefícios da Garantia de Qualidade e Teste Precoce

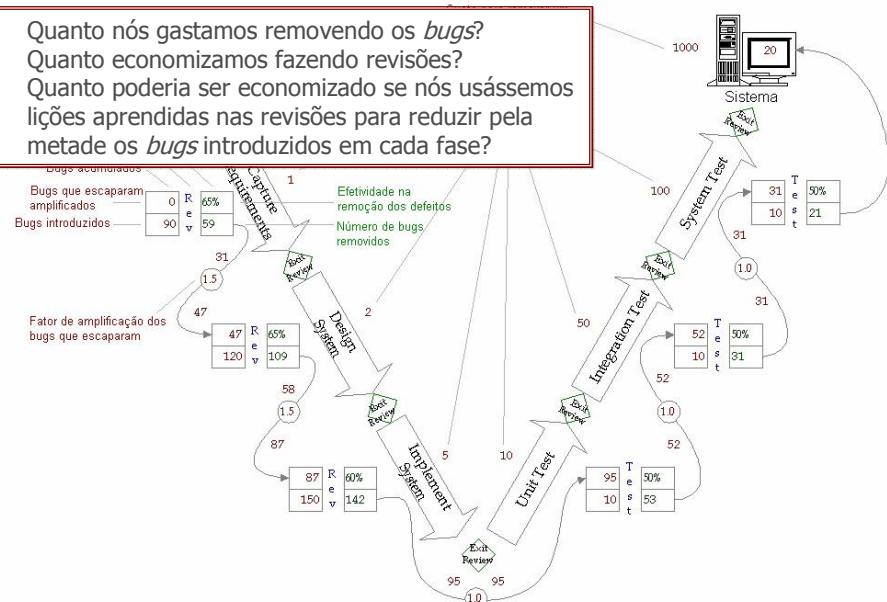
- O custo de um *bug* tende a aumentar enquanto o projeto continua.
- Muitos dos custos associados com *bugs* de pré-versões tendem a ser associados com o esforço requerido para removê-los, portanto custos mais altos significam prazos maiores.
- Quanto mais *bugs* entram na atividade de teste ou de garantia da qualidade, mais *bugs* escaparão daquela atividade.

O Custo do Erro – A Regra de 10 de Myers



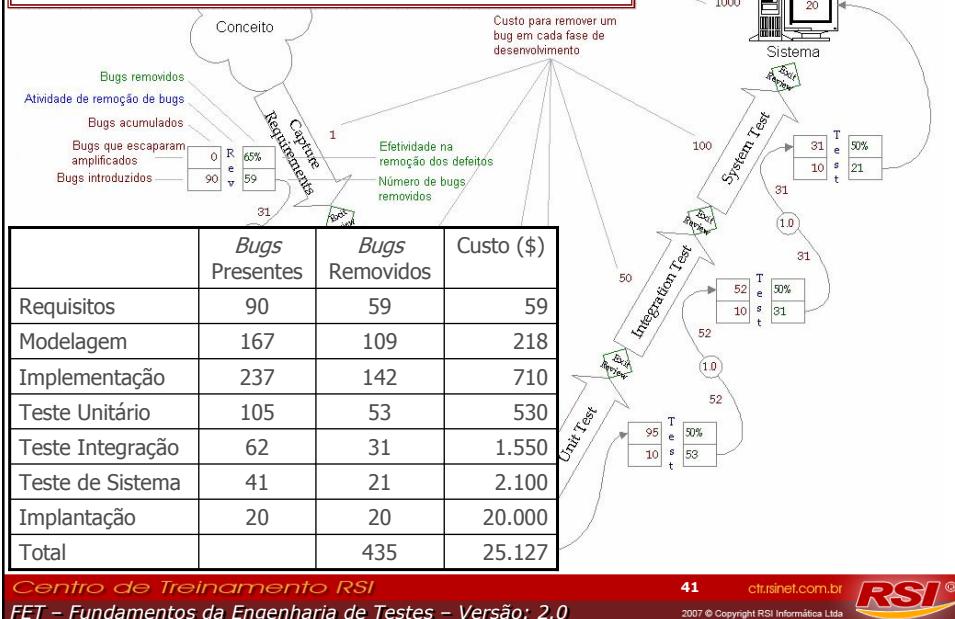
Modelo V

1. Quanto nós gastamos removendo os *bugs*?
2. Quanto economizamos fazendo revisões?
3. Quanto poderia ser economizado se nós usássemos lições aprendidas nas revisões para reduzir pela metade os *bugs* introduzidos em cada fase?



Modelo V

1. Quanto nós gastamos removendo os bugs?



Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0

41

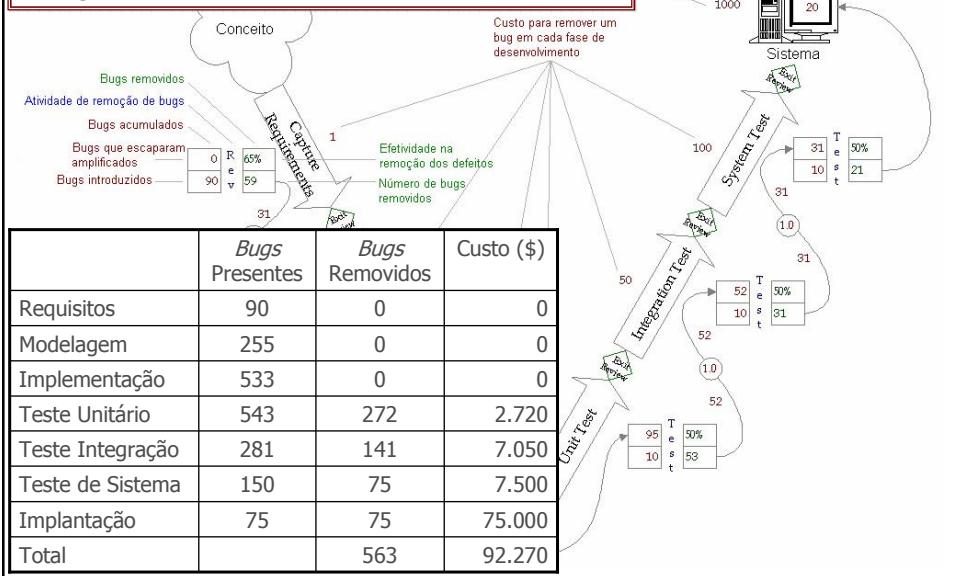
ctr.rsinet.com.br



2007 © Copyright RSI Informática Ltda

Modelo V

2. Quanto economizamos fazendo revisões?



Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0

42

ctr.rsinet.com.br



2007 © Copyright RSI Informática Ltda

Modelo V

3. Quanto poderia ser economizado se nós usássemos lições aprendidas nas revisões para reduzir pela metade os bugs introduzidos em cada fase?

	Bugs Presentes	Bugs Removidos	Custo (\$)
Requisitos	45	29	29
Modelagem	84	55	110
Implementação	119	71	355
Teste Unitário	53	27	270
Teste Integração	31	16	800
Teste de Sistema	20	10	1.000
Implantação	10	10	10.000
Total		218	12.564

Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0

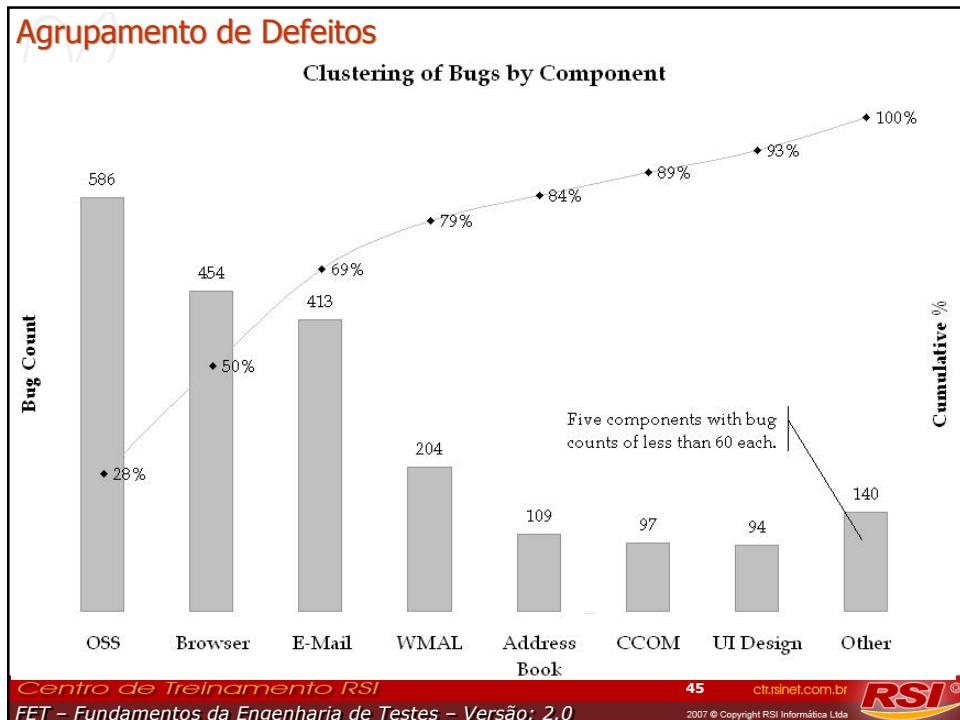
43 ctr.rsinet.com.br RSI®
2007 © Copyright RSI Informática Ltda

Agrupamento de Defeitos

- Estudos têm demonstrado a distribuição desigual dos bugs:
 - MVS: 38% dos bugs em campo em 4% dos módulos.
 - IMS: 57% dos bugs em campo em 7% dos módulos.
- Capers Jones relata que a excessiva presença de módulos suscetíveis a erro causa 50% de redução da produtividade na manutenção do software.

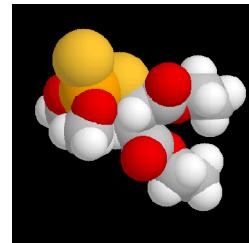
Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0

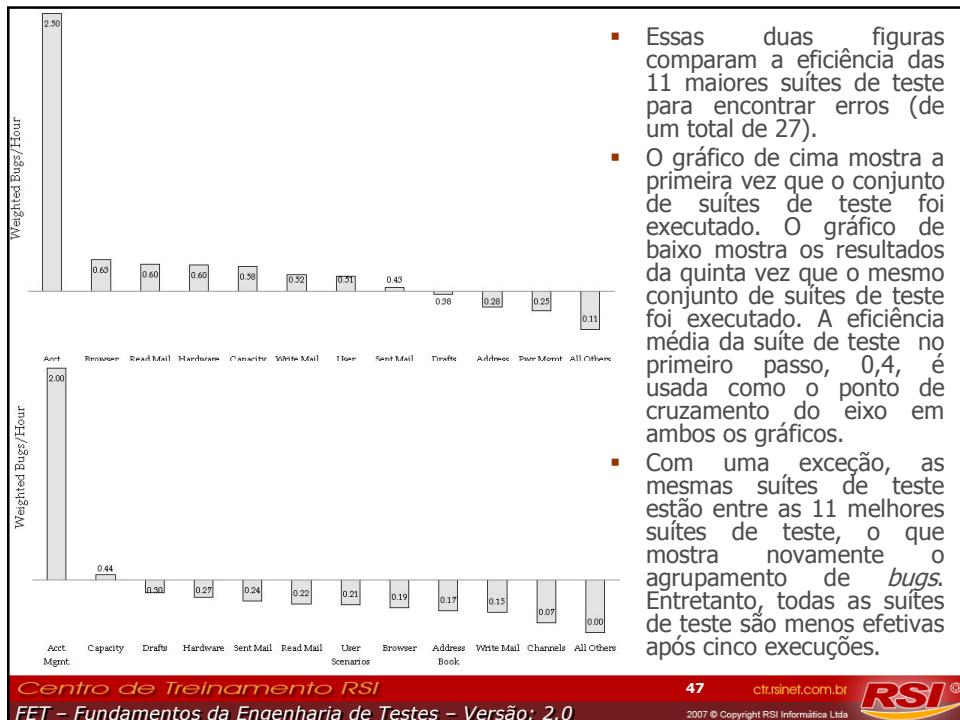
44 ctr.rsinet.com.br RSI®
2007 © Copyright RSI Informática Ltda



O Paradoxo do Pesticida

- Retorne à sua horta.
- Você pulveriza pesticida na sua horta, e as lagartas morrem, mas o pesticida não é efetivo contra todas as pragas.
- Da mesma forma como os pesticidas se tornam menos efetivos, assim ocorre com os testes.
- Testes funcionais não podem encontrar *bugs* de desempenho.
- Experimente novas técnicas de teste – já que o objetivo dos testes é encontrar *bugs*!





Ilusão da Ausência de Erros

- Encontrar e corrigir muitos *bugs* não garante a satisfação do usuário, cliente, e/ou *stakeholder*.
- Muitos produtos com número baixo de defeitos falharam no mercado.
- Projetos de sucesso balanceiam suas forças competitivas em termos de características, prazos, orçamento, e qualidade.

O Teste Deve se Adaptar às Necessidades

- Diferentes projetos, organizações, e produtos têm diferentes necessidades de teste.
- Existem as melhores práticas de teste (e são discutidas neste curso) mas você precisa adaptá-las ao seu projeto.
- Falhar em adequar a equipe de teste e seus métodos a essas necessidades, resulta comumente na dissolução das equipes de teste.

Capítulo 1: Fundamentos do Teste Seção 1.3 – Princípios Gerais do Teste

- *Exercício – Princípios de Teste Observados (e Não)*
- Pense em um projeto recente.
- Anote quais dos princípios gerais do teste que você pode lembrar foram observados.
- E sobre os princípios que você pode lembrar que não foram observados (isto é, violados)?
- Discuta.

Capítulo 1: Fundamentos do Teste
Seção 1.3 – Princípios Gerais do Teste

Exercícios de Fixação

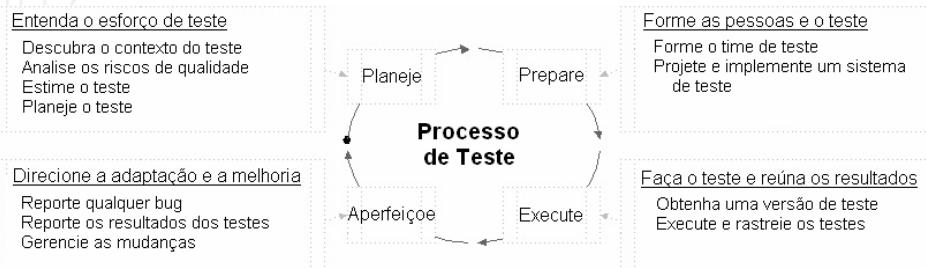
- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 1: Fundamentos do Teste

Seção 1.4 – Fundamentos do Processo de Teste

- Conceitos chave:
 - Planejamento e controle.
 - Análise e modelagem.
 - Implementação e execução.
 - Avaliação de critérios de saída e apresentação de relatório.
 - Atividades de encerramento de teste.

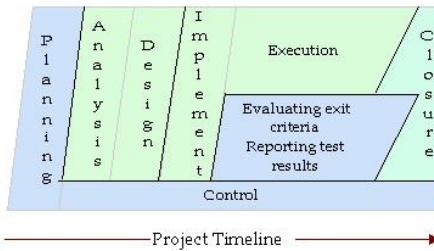
Processos de Teste: Processos Típicos de Teste



O papel do engenheiro de teste é focado em algumas dessas atividades, porém não em todas, dependendo de como os papéis estão definidos. Entretanto, o engenheiro de testes efetivo e eficiente deve entender como o processo de teste funciona e como ele se ajusta ao projeto completo dentro de uma perspectiva macro.

Processo Fundamental de Teste ISTQB

- Você pode pensar no processo de teste em termos das seguintes etapas:
 - Planejamento e controle.
 - Análise e modelagem.
 - Implementação e execução.
 - Avaliação de critérios de saída e apresentação de relatório.
 - Atividades de encerramento de teste.
- Essas etapas podem se sobrepor, serem concorrentes, e/ou interagir.



Planejamento e Controle

Planejamento:

- Determinar o escopo do teste, riscos, objetivos e estratégias.
- Determinar os recursos requeridos de teste.
- Implementar as estratégias de teste.
- Programar a análise e projeto dos testes.
- Programar a implementação, execução e avaliação dos testes.
- Determinar os critérios de saída de teste.

Controle:

- Medir e analisar os resultados.
- Monitorar e documentar o progresso, cobertura e critério de saída de teste.
- Iniciar ações corretivas.
- Tomar decisões.

Muitas atividades de planejamento e controle envolvem a obtenção de acordo, apoio e consenso entre a equipe e a gerência de projeto.

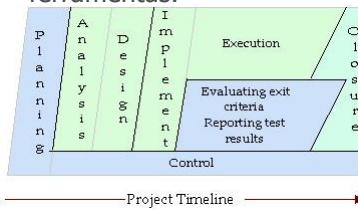
Análise e Modelagem

Análise:

- Revisar a base de teste (por exemplo, as especificações de requisitos e projeto, arquitetura da rede/sistema, riscos de qualidade).
- Identificar e priorizar as condições de teste, requisitos de teste ou objetivos de testes, e requisitar dados dos testes baseados em análise de itens de teste (por exemplo, seus comportamentos, especificação, e estrutura).
- Avaliar a possibilidade de teste dos requisitos e do sistema.

Modelagem:

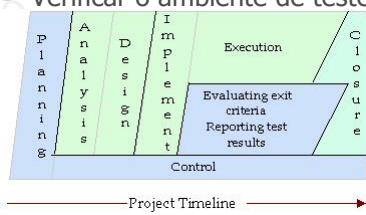
- Modelar e priorizar combinações de dados de teste, ações, e resultados esperados.
- Identificar os dados de teste necessários para condições e casos de teste.
- Modelar o ambiente de teste.
- Reconhecer a infraestrutura, ferramentas.



Implementação e Execução

Implementação:

- Desenvolver, implementar, e priorizar os casos de teste, criar dados, escrever procedimentos
- Criar o ambiente preparado de teste, *scripts*.
- Organizar suítes de testes e sequências de procedimentos de teste.
- Verificar o ambiente de teste.



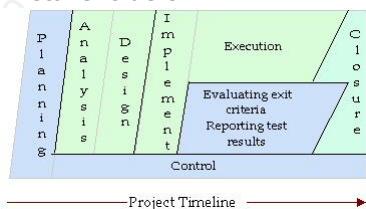
Execução:

- Executar os casos de teste (manuais ou automatizados).
- Armazenar os resultados de teste, e as versões do software sob teste, ferramentas de testes e *testware*.
- Comparar os resultados reais com os esperados.
- Relatar e analisar os incidentes.
- Repetir testes corrigidos e/ou atualizados.
- Executar testes de confirmação e/ou regressão.

Critérios de Saída, Relato, e Encerramento

Critérios de saída e relatório:

- Comparar os registros de testes mediante os critérios de saída no plano de teste.
- Determinar se são necessários mais testes ou se o critério de saída especificado deveria ser alterado.
- Escrever um relatório resumido de teste para os *stakeholders*.



Encerramento:

- Confirmar os entregáveis de teste, resoluções finais ou relatórios de *bugs* postergados, e o aceite do sistema.
- Finalizar e arquivar o *testware*, o ambiente de teste e a infraestrutura de teste.
- Entregar o *testware* para a organização de manutenção.
- Fazer uma retrospectiva para captar melhorias para versões futuras, projetos e processos de teste.

Capítulo 1: Fundamentos do Teste
Seção 1.4 – Fundamentos do Processo de Teste

- *Exercício – Etapas de Teste e Tarefas Executadas*
- Lembre-se de um projeto recente.
- Anote quais das etapas e tarefas do processo de teste ISTQB foram realizadas.
- Anote quais das etapas e tarefas do processo de teste ISTQB não foram realizados.
- Anote se algumas etapas se sobrepujaram ou se foram completamente paralelas.
- Discuta.

Capítulo 1: Fundamentos do Teste
Seção 1.4 – Fundamentos do Processo de Teste

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 1: Fundamentos do Teste

Seção 1.5 – A Psicologia do Teste

- Conceitos chave:
 - Fatores psicológicos para sucesso do teste.
 - Objetivos claros.
 - Auto teste e teste independente.
 - Comunicação respeitosa.
 - O programador e o testador visam o sucesso.

Certeza Versus Progresso

- Uma pesquisa completa pode conduzir a relatórios indiscutíveis de *bug*, casos de testes inquestionavelmente corretos, etc.
- Tal certeza causa satisfação intelectual e reduz relatórios de *bugs* rejeitados e irreprodutíveis.
- Mas a certeza também pode consumir muito tempo e esforço para o retorno e atrasar o progresso.
- Projetos de teste se adaptam às pressões de prazos ou falham.
- Testar computadores é engenharia de testes.
 - Não é uma busca pela verdade – isto é ciência.
 - Engenharia é produzir objetos úteis para os clientes.

Má Interpretação do Resultado do Teste

- De um lado você pode
 - Relatar um comportamento correto como um *bug*.
 - Atribuir severidade ou prioridade excessivamente altas.
 - Superestimar o significado dos *bugs*.
- De outro lado você pode
 - Falhar em detectar ou relatar um comportamento incorreto.
 - Atribuir severidade ou prioridade excessivamente baixas.
 - Subestimar o significado do *bug*.

Algumas dicas para evitar esses erros

1. Faça os testadores terem intervalos para assim não perderem eventos importantes
2. Automatize aonde for prático
3. Defina os resultados esperados de forma mais clara possível
4. Atribua os testadores corretos para cada tarefa de execução de teste
5. Use revisões por pares para execução do teste e relatórios de *bugs*

Entretanto... reconheça que execução perfeita do teste demora muito.
Defina "suficientemente bom" para o teste e viva com essa decisão

Pessimismo Profissional

- Explore possibilidades deprimentes da falha:
 - Antecipe as piores possibilidades de forma a atingir a melhor qualidade que pode ser alcançada do produto.
 - Não como adversário, mas apenas com um ponto de vista diferente do que dos programadores.
 - Lembre: assumir que nada irá falhar durante o teste nega toda a história da computação.
- ⚠ Aviso: **não** é uma permissão para ofender!
- Não mire nos programadores como alvos com os relatórios, ou ridicularize a falha.
- Desafio: ser positivo, agradável, e portador de más notícias, tudo ao mesmo tempo.



Pierre tem pessimismo, mas talvez não profissionalismo.

Más Notícias == Má Pessoa??

- Testadores são recebidos algumas vezes com má vontade porque trazem notícias de problemas no projeto.
- A chave é o **pessimismo profissional**.
 - Não assuma o papel de "Campeão Solitário da Qualidade".
 - Qualidade do produto é uma decisão de negócio.
 - Riscos de qualidade são medidos mediante outros riscos.
 - Não faça críticas sobre os esforços de desenvolvedores em corrigir o *bug*.
 - Nunca se vanglorie, mesmo que você estivesse certo e todos os outros errados.
- Juan de Mariana: "*A maior das tolices é esforçar-se em vão, e esgotar-se sem conquistar qualquer coisa, a não ser ódio.*"



A morte ou
um guia
amigável?



Curiosidade Balanceada

- Balanceie a necessidade por perfeição em qualquer área com necessidade de cobrir muitas áreas em um espaço curto de tempo.
- Engenheiros de teste efetivos e eficientes:
 - Têm um talento para consumir tempo onde estão os *bugs*.
 - Podem fazer uma completa isolação do *bug* rapidamente.
- Engenheiros de teste **não** efetivos e ineficientes:
 - Escrevem testes para procurar por *bugs* improváveis, de baixo impacto.
 - Gastam horas pesquisando *bugs* triviais.

Foco

- Dois problemas de foco:
 - Perseguir questões de forma limitada, perdendo a visão de prioridades mais importantes.
 - Distrair-se das tarefas chave.
- Balanceie e reavalie as prioridades constantemente.
- Permaneça focado nos objetivos do projeto de teste.
- Testadores bem equilibrados podem atingir os objetivos de teste com uma sinalização clara do seu gerente de teste.



Definindo Habilidades do Testador

- Leitura.
 - Especificações, e-mails, casos de teste, etc.
- Escrita.
 - Casos de teste, relatórios de *bugs*, documentação de teste, etc.
- Não dependente da sua “língua nativa”.
- Estatística e outras operações matemáticas.
- Habilidades pertinentes à tecnologia, projeto e teste.
 - Tecnologia: linguagens de programação e mais, como sistemas operacionais, redes, *HTML/Web*, etc.
 - Domínio da aplicação: bancária, fatores humanos, aplicações de escritório, etc.
 - Teste: produção de *scripts*, exploração e ataque ao sistema, automatização, modelagem do desempenho, etc.

Capítulo 1: Fundamentos do Teste
Seção 1.5 – A Psicologia do Teste

- *Exercício – Psicologia em Ação*

- Reflita sobre as atitudes e comportamentos dos testadores de maior sucesso que você conhece.
- Até qual nível eles mostram os aspectos psicológicos discutidos nesta seção?
- Quais outros elementos da personalidade deles e habilidades você acredita que os leve ao sucesso?
- Discuta.

Capítulo 1: Fundamentos do Teste
Seção 1.5 – A Psicologia do Teste

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 1: Fundamentos do Teste

Exercícios de Fixação

- Faça os exercícios no formato do exame para o capítulo 1.
- Discuta.

Capítulo 2: Teste Durante o Ciclo de Vida do Software

Capítulo 2: Teste Durante o Ciclo de Vida do Software

Seções:

1. Modelos de Desenvolvimento de Software
2. Níveis de Teste
3. Tipos de Teste: o Alvo do Teste
4. Teste de Manutenção

Capítulo 2: Teste Durante o Ciclo de Vida do Software

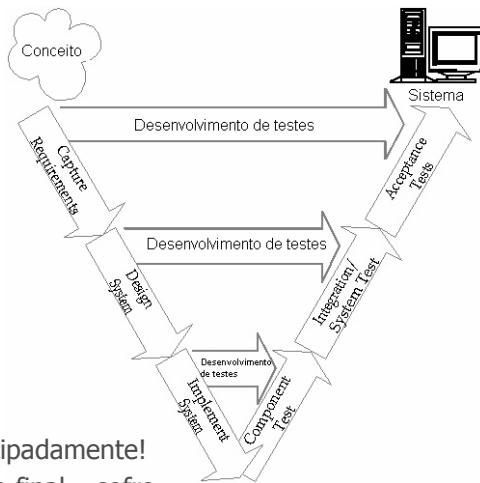
Seção 2.1 – Modelos de Desenvolvimento de Software

- Conceitos chave:
 - O relacionamento entre desenvolvimento e atividades de teste.
 - Adaptação dos modelos de desenvolvimento de software para o contexto do projeto e do produto.
 - Razões para diferentes níveis de teste.

Modelo "V" ou Modelo Sequencial

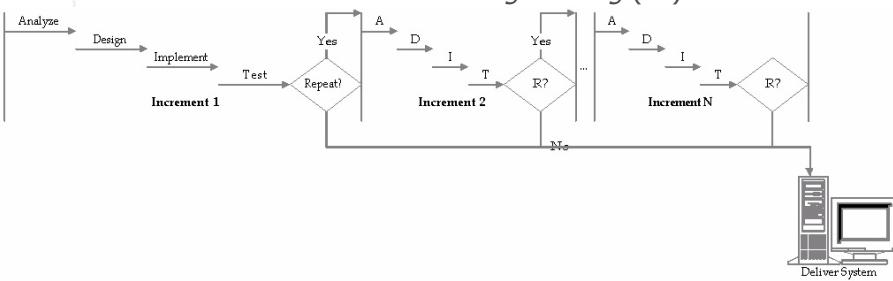
- Geralmente direcionados por cronograma, orçamento e risco.
- Faça os níveis mais profundos do projeto (detalhes), depois faça a construção e, finalmente, faça o teste.
- Modelo intuitivo e familiar.
- Vence o caos!

△ É difícil planejar tão a fundo antecipadamente!
 △ Quando o plano falha, o teste – no final – sofre.



Modelo Iterativo, Evolucionário ou Incremental

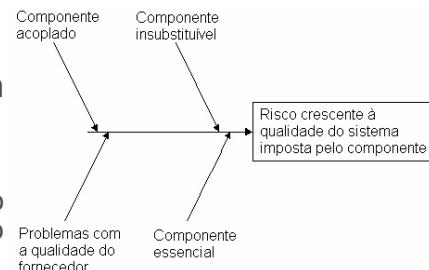
- Dirigido por risco e cronograma para atingir uma janela de mercado ou a data de entrega estipulada.
- O conjunto de funcionalidades cresce ao redor da funcionalidade central.
- É possível entregar (alguma coisa) a qualquer tempo uma vez que a funcionalidade está pronta.
- Está tornando-se uma abordagem popular.
- Varia em formalidade desde *Extreme Programming* (XP) ao RAD e RUP.



△ Ainda é uma tentação entregar um sistema com funcionalidades com *bugs*
 △ No mundo “ágil”, o papel do teste ainda está em desenvolvimento.

Integração do Sistema

- Muitos projetos envolvem a integração de componentes.
- Opções de minimização de risco
 - Integre, rastreie, e gerencie o teste do fornecedor em um esforço de teste distribuído.
 - Confie no teste de componente do fornecedor.
 - Estabeleça a qualidade/capacidade de teste do fornecedor.
 - Despreze e substitua o teste dele (ahh!).
 - Veja a política a ser adotada nas últimas duas opções.
- Planeje você mesmo os testes de integração e de sistema.



Acoplado: Forte interação ou consequência de falha entre componente e sistema.

Insubstituível: poucos componentes similares disponíveis.

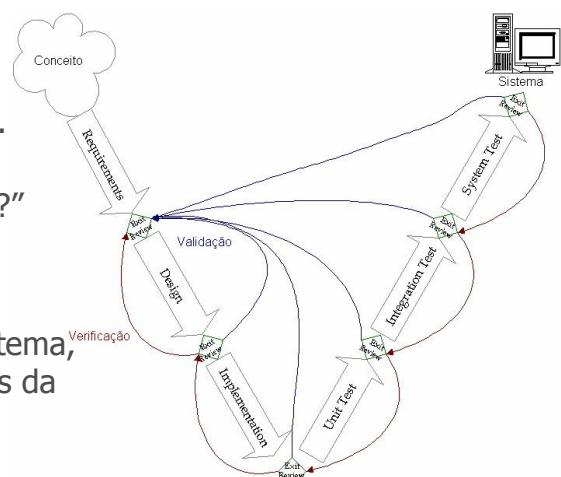
Essencial: funcionalidades chave no sistema indisponíveis se o componente não trabalhar corretamente.

Problemas de qualidade do fornecedor: Probabilidade aumentada de um componente ruim

Verificação & Validação

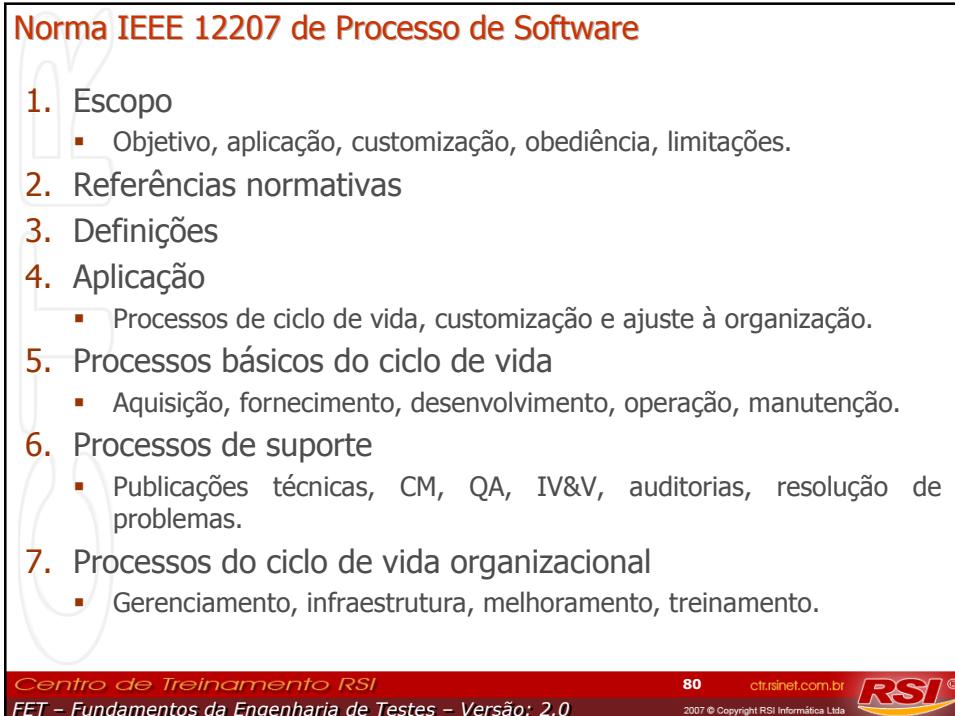
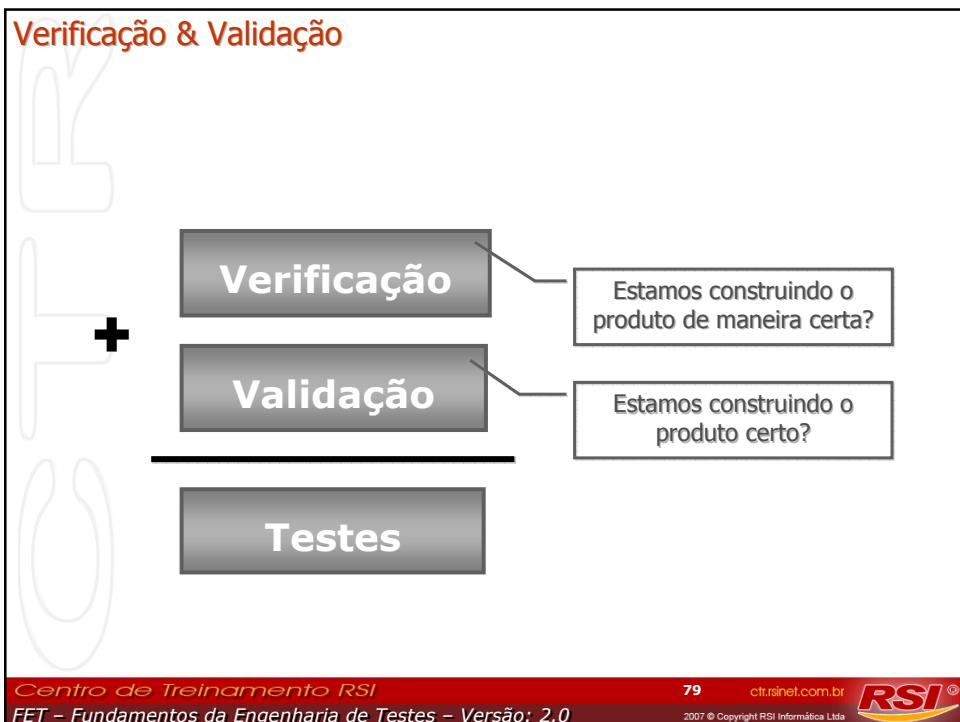
Verificação:

- Procura por *bugs* nos entregáveis da fase.
- “Estamos construindo o sistema corretamente?”



Validação:

- Procura por *bugs* no sistema, baseado nos entregáveis da fase.
- “Estamos construindo o sistema correto?”



Processo de Maturidade CMMI

- *Capability Maturity Model Integration*: modelo de cinco níveis criado pelo *Software Engineering Institute* (SEI).
 - **Inicial**: imprevisível, pobremente controlado, reativo.
 - **Gerenciado**: processos estabelecidos em nível de projeto, frequentemente reativos.
 - **Definido**: processos estabelecidos permeando a organização, geralmente pró-ativos.
 - **Gerenciado quantitativamente**: processos medidos e controlados na organização.
 - **Otimizado**: foco na melhoria contínua, geralmente direcionado por dados.
- O teste foi pouco enfatizado no CMM, levando a modelos específicos de teste como Processos de Teste Críticos, Melhoria do Processo de Teste, Modelo de Maturidade de Teste.

Independentemente dos Modelos....

- Características gerais do bom teste:
 - Atividade de teste para cada atividade de desenvolvimento (por exemplo, teste unitário e atividade de implementação).
 - Níveis de teste focados em objetivos, com coordenação para evitar lacunas, e/ou sobreposição.
 - Análise e projeto do teste começam cedo – prevenção de *bugs*.
 - Testadores envolvidos em quaisquer revisões que eles estiverem aptos a comparecer, levando seu ponto de vista exclusivo.
- Você pode combinar ou reorganizar os níveis de teste desde que você mantenha essas características em mente.

Capítulo 2: Teste Durante o Ciclo de Vida do Software
Seção 2.1 – Modelos de Desenvolvimento de Software

▪ *Exercício – Modelos e Realidade*

- O famoso especialista em qualidade W. E. Deming disse, “Todos os modelos estão errados; alguns são úteis.”
- Indique qual modelo de ciclo de vida nesta seção aplica-se melhor ao seu último projeto (ou, se não havia modelo organizacional, indique “codifica e corrige”).
- Quanto você acredita que o modelo foi útil?
- Até que nível, se houve algum, ele foi causador de danos?
- Discuta.

Capítulo 2: Teste Durante o Ciclo de Vida do Software
Seção 2.1 – Modelos de Desenvolvimento de Software

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 2: Teste Durante o Ciclo de Vida do Software

Seção 2.2 – Níveis de Teste

- Conceitos chave:
 - Principais objetivos do teste para cada nível.
 - Objetos típicos de teste para cada nível.
 - Alvos típicos de teste para cada nível.
 - Produtos do trabalho de teste para cada nível.
 - Participantes do teste para cada nível.
 - Tipos de defeitos e falhas para cada nível.

Teste de Componente (Unidade)

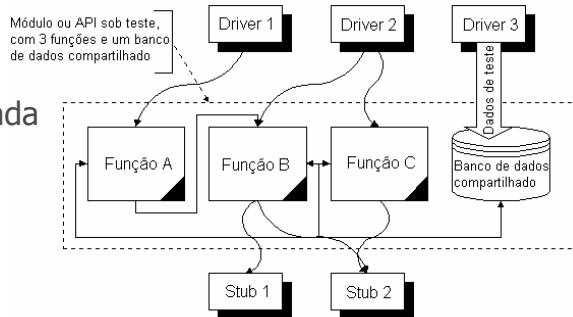
- Objetivos: Encontrar *bugs*, adquirir confiança, e reduzir os riscos nas partes individuais do sistema sob teste antes da integração do sistema.
- Base: Código, banco de dados, requisitos/modelagem, riscos de qualidade.
- Tipos de teste: Funcionalidade, uso de recursos, desempenho, estrutural.
- Item Sob Teste: Varia. O menor item testável independentemente (função ou classe) ou um componente separado fornecendo serviços para os outros.
- Ambiente preparado e ferramentas: nível API (*drivers* e *stubs*), *freeware* e comercial.
- Responsável: Geralmente programadores, mas o nível de proficiência e grau de execução varia.

Processo de Teste de Componente/Unidade

- Teste de componente/unidade tipicamente...
 - ...envolve acesso ao código.
 - ...é executado no ambiente de desenvolvimento.
 - ...requer *drivers*, *stubs*, e/ou ambiente preparado.
 - ...é feito pelo programador que escreveu o código.
- Frequentemente, os *bugs* são corrigidos quando encontrados sem qualquer relato, o que reduz a transparência do processo de desenvolvimento com respeito à qualidade.
- Desenvolvimento dirigido a teste / teste primeiro (*test driven development*)
 - Desenvolva um conjunto de testes de unidade;
 - Construa e integre o código;
 - Execute os testes e depure até passar nos testes.

Drivers e Stubs

- Durante o teste de unidade, componente, e integração – e para teste das APIs – é frequentemente necessário simular partes do fluxo de chamadas alcançáveis a partir do módulo(s) sob teste.
- Configuração de dados também necessária algumas vezes.
- **Driver:** função(s) que chama o módulo(s) sob teste.
- **Stub:** função(s) chamada – diretamente ou indiretamente – pelo módulo(s) sob teste.



Teste de Integração

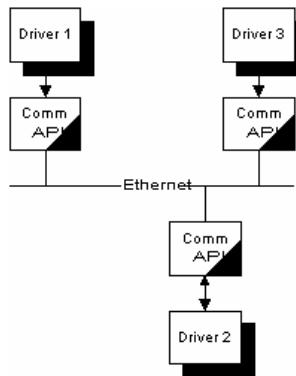
- Objetivos: Encontrar *bugs*, adquirir confiança, e reduzir o risco nos relacionamentos e interfaces entre pares e grupos de componentes no sistema sob teste assim que as partes forem unidas.
- Base: Projeto, arquitetura, esquemas, fluxo de dados, riscos de qualidade.
- Tipos de Teste: Funcionalidade, uso de recursos, desempenho.
- Item Sob Teste: *Builds* ou *backbones*.
- Ambiente preparado e ferramentas: nível API e CLI, *freeware* e comercial.
- Responsável: Idealmente ambos testadores e programadores, mas frequentemente nenhum.

Técnicas de Integração

- Big bang
 - Pegue todos os módulos testados; junte-os; teste.
 - Rápido, mas onde está o *bug*?
 - Por que esperar até que todo o código esteja escrito para iniciar a integração?
- Bottom up
 - Inicie com os módulos da camada mais baixa; use *drivers* apropriados; teste.
 - Repita o processo, trocando os *drivers* por módulos, até terminar.
 - Bom isolamento do *bug*, mas e se os problemas estiverem na camada mais alta?
- Top down
 - Como a *bottom up*, mas iniciando a partir da camada mais alta e usando *stubs*.
 - Bom isolamento do *bug*, mas e se os problemas estiverem na camada mais baixa?
- Backbone
 - Inicie com módulos críticos; construa o *backbone* inicial; use *drivers* e *stubs*; teste.
 - Repita o processo, substituindo *stubs* e *drivers* pelos módulos na ordem de risco.
 - Bom isolamento do *bug* e encontra *bugs* de integração na ordem de risco.

Técnica de Integração *Backbone (backbone 0)*

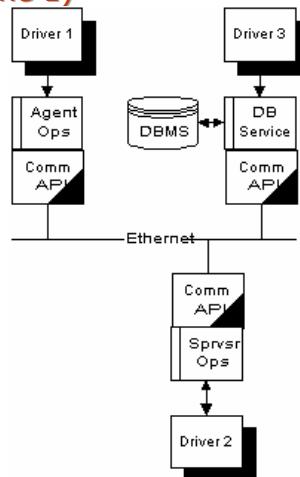
- Considere o seguinte exemplo (baseado em um projeto real).
- Nós começamos com um *backbone* básico.
 - APIs de comunicação.
 - Arquitetura básica de rede.
- Teste a funcionalidade básica, tratamento e recuperação de erros, confiabilidade, e desempenho.
- Risco de qualidade: A arquitetura básica do sistema é impraticável?



Backbone 0: Testa o software básico de comunicação e a arquitetura de rede.

Técnica de Integração *Backbone (backbone 1)*

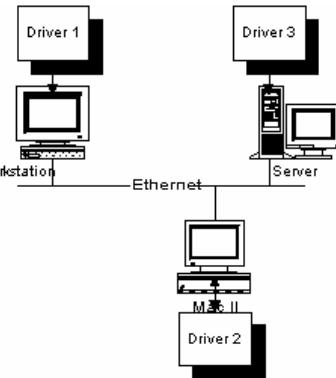
- Adicione os módulos que implementam as mais críticas e principais operações e serviços.
- Novamente, teste a funcionalidade básica, tratamento e recuperação de erros, confiabilidade, e desempenho.
- Risco de qualidade: As operações e funções principais integram-se com a camada de transporte?
- Continue o processo com o próximo nível de risco de qualidade...



Backbone 1: Testa algumas operações e serviços principais através da API de comunicação e da rede

Técnica de Integração *Backbone (backbone n)*

- Como no passo anterior, nós estamos testando (neste caso, usando *drivers* de teste automatizados) *end-to-end* através das GUIs nos sistemas hospedeiros.
- Risco de qualidade: O sistema totalmente integrado funciona?
- O *backbone* final para o teste de integração também é o primeiro *build* totalmente integrado para teste de sistema.



Backbone n: Testar *end-to-end* o sistema integrado completamente. Assim que funcionar, estaremos prontos para o teste de sistema.

Níveis do Teste de Integração

- Pode haver mais de um nível de teste de integração em um projeto.
 - Teste de integração de componente: testa a interação entre unidades e componentes seguindo o teste de unidade/componente.
 - Teste de integração de sistema: testa a interação entre todo o sistema seguindo o teste de sistema.
- Teste de integração de sistema é complexo.
 - Múltiplas organizações controlando as interfaces do sistema, fazem com que as mudanças sejam perigosas.
 - Processos de negócio podem caracterizar sistemas.
 - Questões relacionadas a compatibilidade de hardware/software podem aparecer.

Teste de Sistema

- Objetivos: Encontrar *bugs*, adquirir confiança, e reduzir riscos nos comportamentos global e particular, funções, e respostas do sistema sob teste como um todo.
- Base: Requisitos, projeto de alto nível, casos de uso, riscos de qualidade, experiência, listas de checagem, ambientes.
- Tipos de Teste: Funcionalidade, segurança, desempenho, confiabilidade, usabilidade, portabilidade, etc.
- Item Sob Teste: Sistema completo, no ambiente de teste o mais realístico possível.
- Ambiente preparado e ferramentas: API, CLI, ou GUI, *freeware* e comercial.
- Responsável: Tipicamente testadores independentes.

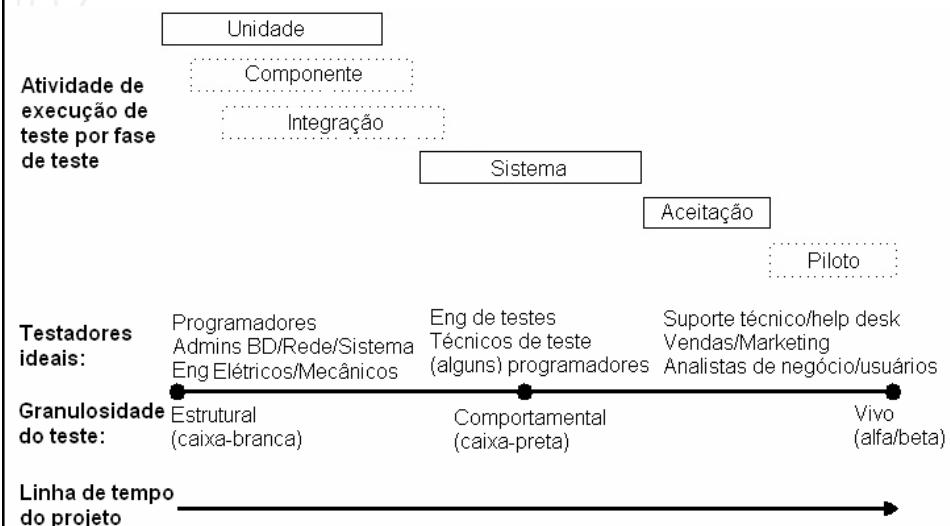
Teste de Aceite

- Objetivo: Demonstrar que o produto está pronto para entrega/lançamento.
- Base: Requisitos, contratos, experiência.
- Tipos de Teste: Funcional, portabilidade, desempenho.
- Item Sob Teste: Sistema completo, algumas vezes no ambiente de produção ou do cliente.
- Ambiente preparado e ferramentas: Usualmente GUI.
- Responsável: Geralmente usuários ou clientes, mas também testadores independentes.

Variações no Teste de Aceite

- Teste de aceite do usuário: Usuários de negócios verificam a aderência aos objetivos funcionais.
- Teste operacional: aceite pelos administradores do sistema (por exemplo, *backup/restore*, recuperação de desastre, gerenciamento de usuário, manutenção, segurança).
- Teste de contrato e regulamento: Verificação da conformidade aos requisitos, regulamentos, ou normas contratualmente acordados ou de exigência legal.
- Teste alfa e beta (ou de campo): Teste e construção de confiança por clientes potenciais ou já existentes, com beta teste e teste de campo sendo executados no ambiente(s) corrente.

Teste Pervasivo: Precoce, Multifuncional

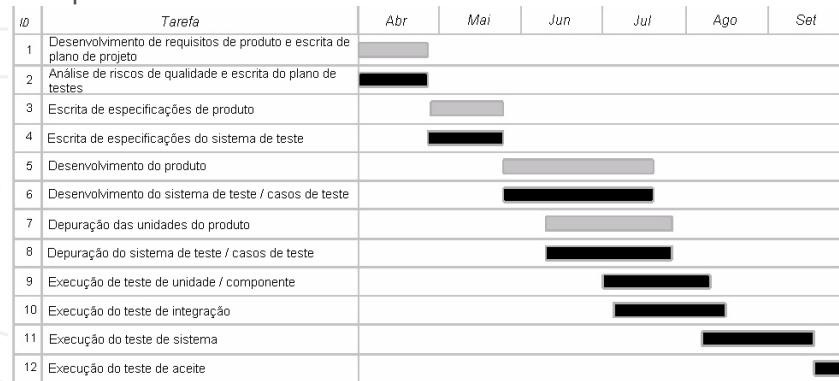


Por que Teste Pervasivo?

- Diferentes participantes podem testar diferentes especificidades.
 - Diferentes habilidades para diferentes estágios.
- Diferentes especificidades enfatizadas em cada nível (ou fase).
 - Teste unitário: primariamente estrutural.
 - Teste de sistema: primariamente comportamental.
 - Teste de aceite: primariamente real (sistema vivo).
- É importante ser flexível, pois:
 - Técnicas de teste de várias granulosidades podem ser úteis em todas as fases de execução do teste.
 - Execução de fase de teste sobrepõe dependência nos critérios de entrada e saída.
 - Nem todas as fases de teste ocorrem em todos os projetos.

Quando o Teste Impregna os Projetos

- Tarefas de teste ocorrem durante todo o esforço de desenvolvimento.
- A execução do teste é planejada com múltiplos ciclos para permitir tempo de correção.
- Funcionalidades serão retiradas ou adiadas para iterações posteriores ao invés de adiar as datas de entrada nas fases de teste ou entrar antes de estar pronto.



Capítulo 2: Teste Durante o Ciclo de Vida do Software
Seção 2.2 – Níveis de Teste

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 2: Teste Durante o Ciclo de Vida do Software

Seção 2.3 – Tipos de Teste: o Alvo do Teste

- Conceitos chave:
 - Principais tipos ou alvos de teste de software.
 - Testes funcionais e não funcionais.
 - Testes estruturais.
 - Testes de confirmação e regressão.
 - Uso dos testes funcionais, não funcionais, e estruturais nos vários níveis.

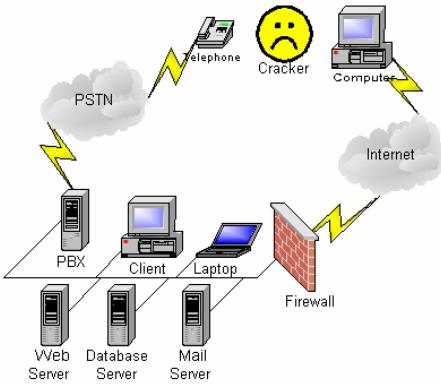


Funcionalidade

- Ação cabível ou requerida que não está presente, está inacessível, ou seriamente danificada.
 - Calculadora sem a função de adição.
 - Função de adição implementada, a tecla “+” não funciona.
 - Apenas soma números inteiros, não números reais.
- Ação correta, resultado errado.
 - Função soma: $2+2=5?$
- Ação correta, resultado correto, efeito colateral errado.
 - Função dividir: $2/2=I$ (formato de numeral romano).
- A funcionalidade do sistema, subsistema ou componente é descrita em documentos como especificação de requisitos, casos de uso, ou especificação funcional. Entretanto, geralmente algumas funções permanecem não documentadas, e os testadores precisam entender o significado de “comportamento razoável”.

Segurança

- Ameaças de segurança incluem:
 - Vírus.
 - Invasão de servidores.
 - Negação de serviço.
- Suposições erradas comuns:
 - Codificação (HTTPS) no servidor Web resolve problemas de segurança.
 - Comprar um *firewall* resolve problemas.
 - Administradores de rede ou de sistemas inábeis podem resolver problemas.
- Conhecimento de campo especializado de teste.



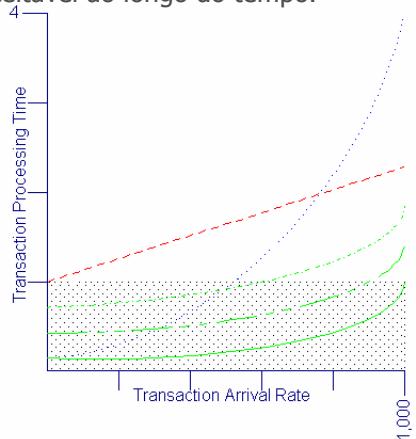
O determinado – ou entediado – invasor pode entrar em um dos seus servidores. Uma exploração pode conduzir o invasor a outras vulnerabilidades – e a dados valiosos.

Interoperabilidade

- Interoperabilidade com programas, sistemas operacionais, bancos de dados,
- Inclui:
 - Interfaces.
 - Troca de dados.
- Pode ser um-a-um ou como uma coleção de componentes.

Desempenho e Confiabilidade

- Desempenho.
 - Muito devagar pela curva de desempenho.
 - “Joelho” inaceitável na curva desempenho.
 - Degradação de desempenho inaceitável ao longo do tempo.
- Confiabilidade.
 - Sistema falha em completar funções normais.
 - Sistema funciona normalmente, mas cai ou trava aleatoriamente



Estresse, Capacidade, e Volume

- Estresse: condições extremas causam falha.
 - Combinações de erro, capacidade, e volume.
- Capacidade: problemas de funcionalidade, desempenho, ou confiabilidade devido a diminuição de recursos.
 - Preenche o disco ou memória em 80% ou mais.
- Volume: problemas de funcionalidade, desempenho, ou confiabilidade devido a taxa de fluxos de dados.
 - Executa mais de 80% de transações classificadas por minuto, número de usuários simultâneos, etc.

Manutenção e Manutenibilidade

- Manutenção
 - Processos de instalação e desinstalação de atualizações e correções não funcionam.
 - Configurações não podem ser alteradas apropriadamente (ex., *plug-and-play*, *hot plugging*, adicionar espaço em disco, etc.).
- Manutenibilidade
 - O próprio software (código fonte) não é passível de manutenção.
 - Bancos de dados não atualizáveis.
 - Bancos de dados crescem monotonicamente (consistentemente, nunca decrescendo).
 - Software não testável eficientemente durante a manutenção; por exemplo, regressão excessiva.

Usabilidade e Interface com Usuário

- Um sistema pode funcionar apropriadamente mas não pode ser utilizável pelo cliente pretendido.
- Interfaces enfadonhas que não seguem *workflows*.
- Funcionalidade inacessível.
- Inapropriadamente difícil para os usuários aprenderem.
- Mensagens de instrução, ajuda, e erro que são enganosas, confusas, ou com ortografia errada.

Configuração e Portabilidade

- Uma única plataforma pode ser configurada de muitas maneiras diferentes pelo software.
- Uma família de plataformas pode suportar várias configurações de hardware.
- As mudanças de configuração são tratadas?
 - Adicionar espaço em disco ou outras formas de armazenamento.
 - Adicionar memória.
 - Atualizar ou adicionar CPU.
- Portabilidade para vários ambientes.

Outros Testes Funcionais/Não-Funcionais

- Localização (interface com usuário)
- Localização (operacional)
- Conformidade com normas e regulamentos
- Tratamento e recuperação de erros
- Recuperação de desastre.
- Trabalhados em rede/Internet ou distribuídos
- Momento (*timing*) e coordenação
- Qualidade dos dados
- Conversão de dados
- Operações
- Instalação
- Desinstalação
- Tratamento de dados e tempo
- Documentação
- E muitos outros...

Testes Baseados na Estrutura

- Testes baseados em como o sistema é construído.
 - Código
 - Dados
 - Projeto
- Cobertura estrutural (caixa branca) pode ser medida após os testes funcionais e não-funcionais baseados na especificação (caixa preta) serem executados para checar omissões.
- Este tópico será coberto em maior profundidade mais tarde...

Regressão e Confirmação

- Teste de regressão averigua os efeitos das mudanças, pois mesmo mudanças pequenas, localizadas, isoladas, nem sempre geram efeitos pequenos, localizados, ou isolados.
- Estratégias de regressão são cobertas na próxima seção.
- Teste de confirmação confirma que...
 - Mudanças feitas no sistema estão presentes.
 - Correções de *bugs* introduzidas no sistema resolveram os sintomas observados.
- Possibilidade de nova execução dos testes ajuda nos testes de regressão e confirmação.
- Automatização, coberta em profundidade, também auxilia.

Padrão de Qualidade ISO 9126

Características / Subcaracterísticas:

- **Funcionalidade:** Conveniência, precisão, interoperabilidade, segurança, conformidade.
 - **Confiabilidade:** maturidade (robustez), tolerância a falhas, capacidade de recuperação, conformidade.
 - **Usabilidade:** facilidade de entendimento, capacidade de aprendizagem, operação, atratividade, conformidade.
 - **Eficiência:** comportamento temporal, utilização de recursos, conformidade.
 - **Manutenibilidade:** facilidade de análise, possibilidade de efetuar mudanças, estabilidade, testabilidade, conformidade.
 - **Portabilidade:** adaptabilidade, instalabilidade, coexistência, substituição, conformidade.
- } Alvo de teste funcional
- } Alvo de teste não-funcional

Capítulo 2: Teste Durante o Ciclo de Vida do Software Seção 2.3 – Tipos de Teste: o Alvo do Teste

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 2: Teste Durante o Ciclo de Vida do Software

Seção 2.4 – Teste de Manutenção

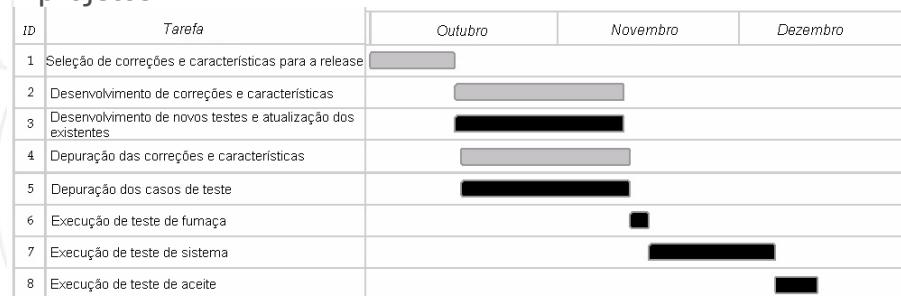
- Conceitos chave:
 - Motivos para o teste de manutenção.
 - Teste de manutenção versus teste de nova aplicação.
 - Papel do teste de regressão e análise de impacto.

Motivos para Manutenção

- Três gatilhos típicos para manutenção e teste de manutenção:
 - **Modificação:** melhorias, correções de *bugs*, mudanças no ambiente operacional, correções.
 - **Migração:** um novo ambiente suportado.
 - **Retirada:** fim da vida de um subsistema ou troca de todos os gatilhos do sistema.
- Teste de manutenção objetiva a própria mudança – e o que foi mudado e não deveria ser mudado.

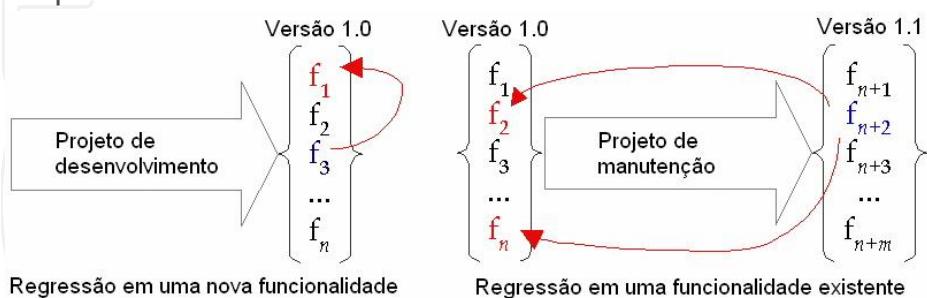
Versões de Teste de Manutenção

- A regressão é o grande risco para versões de manutenção.
- Algumas organizações tentam colocar uma grande versão com valiosas funcionalidades dentro de uma pequena versão de manutenção.
- Tempo escasso para desenvolver novos testes.
- Falham os princípios básicos para estimar o teste de grandes projetos.



Regressão

- Regressão (comportamento errado de uma função, atributo, ou funcionalidade previamente correta devido a mudança):
 - Local (correção cria novo *bug*).
 - Exposta (correção revela *bug* existente).
 - Remota (correção em uma área quebra alguma coisa em outra área).
- Regressão pode afetar tanto funcionalidades existentes quanto novas.

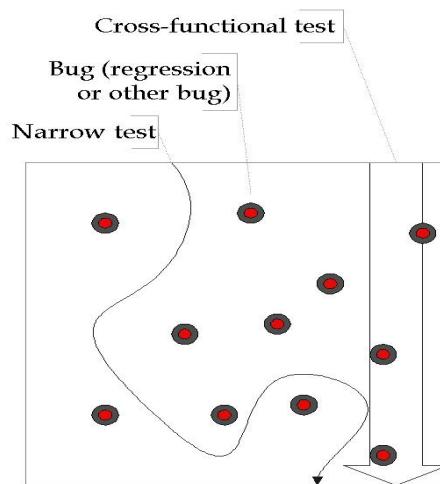


Estratégia de Regressão 1: Repita Todos os Testes

- Se nossos testes correspondem a riscos importantes de qualidade, então a repetição de todos os testes deveria encontrar as regressões mais importantes.
- Automatização é o único meio prático para grandes e complexos sistemas.
- Nós cobriremos automatização mais a fundo em uma seção posterior.

Estratégia de Regressão 2: Repita Alguns Testes

- Frequentemente, automatização total é impossível.
- Selecione alguns testes para repetir
 - Rastreabilidade
 - Análise de mudança/impacto
 - Análise de risco
- Use testes multifuncionais para “teste de regressão acidental”.
- Pode usar cobertura de código para avaliar o nível de risco.



Outras Três Estratégias de Regressão

- Libere versões mais lentamente.
 - Libere a cada seis meses ao invés de todo mês.
 - Repetição parcial ou mesmo completa pode aumentar a cobertura em versões maiores.
- Combine correções de emergência com o processo de versões mais lentos para permitir flexibilidade enquanto mantém baixo o risco de regressão.
- Use teste pelo cliente ou usuário:
 - Beta para software de mercado de massa.
 - Versão piloto, estágio ou fase, paralelo para TI.

Capítulo 2: Teste Durante o Ciclo de Vida do Software Seção 2.4 – Teste de Manutenção

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 2: Teste Durante o Ciclo de Vida do Software

Exercícios de Fixação

- Faça os exercícios no formato do exame para o capítulo 2.
- Discuta.

Capítulo 3: Técnicas Estáticas

Capítulo 3: Técnicas Estáticas

Seções:

1. Revisão e o Processo de Teste
2. Processo de Revisão
3. Análise Estática por Ferramentas

Capítulo 3: Técnicas Estáticas

Seção 3.1 – Revisão e o Processo de Teste

Conceitos chave:

- Produtos do trabalho de software e técnicas estáticas.
- A importância e o valor das técnicas estáticas.
- A diferença entre técnicas estáticas e dinâmicas.
- Os objetivos da análise estática e das revisões, e comparação de objetivo com teste dinâmico.

Teste Estático

- Revisões e ferramentas:
 - Revisões variam desde informais até a muito formais.
 - Ferramentas podem executar alguns tipos de testes estáticos.
 - Técnicas estáticas podem ser usadas para requisitos e projetos, além de código, esquemas de banco de dados, documentação, testes,...
- Modelos e protótipos:
 - Um diagrama de um sistema complexo geralmente pode revelar problemas de projeto que podem ficar escondidos nas palavras.
 - Um diagrama feio significa muitos *bugs*.
- Casos e dados de teste:
 - A análise de teste e projeto baseado em requisitos e especificações de projeto é uma forma de revisão estruturada.
 - Análises de teste e projeto geralmente revelam problemas.

Ferramentas Estáticas

- Análise estática:
 - Redação problemática: verificadores de gramática/ ortografia.
 - Programação perigosa: J-Test, Safer C, lint...
 - Medição: Análise de complexidade.
- Simulações de sistema:
 - Simulador de sistemas de propósito geral.
 - Modelagem de desempenho / ferramentas de pesquisa de operações.
 - Planilhas.

Revisões: Custos e benefícios

- Custos:
 - O tempo necessário para executar as revisões.
 - O esforço necessário para obter e analisar métricas.
 - A melhoria do processo.
- Benefícios:
 - Prazos menores (devido a remoção eficiente de *bugs*).
 - Períodos de teste menores e custos menores de teste.
 - Produtividade do desenvolvedor.
 - Qualidade melhorada do produto (que reduz os custos no futuro).
- Revisões de todos os gêneros são comprovadamente técnicas de alto retorno para melhoria da qualidade.

Relacionando Teste Estático e Dinâmico

Similaridades:

- Busca identificar defeitos.
- Trabalha melhor quando um amplo conjunto transversal de *stakeholders* é envolvido.
- Poupa tempo e dinheiro da empresa.

Diferenças:

- Cada técnica pode encontrar diferentes tipos de defeitos de modo mais efetivo e eficiente.
- Técnicas estáticas encontram defeitos ao invés de falhas.

Capítulo 3: Técnicas Estáticas
Seção 3.1 – Revisão e o Processo de Teste

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 3: Técnicas Estáticas

Seção 3.2 – Processo de Revisão

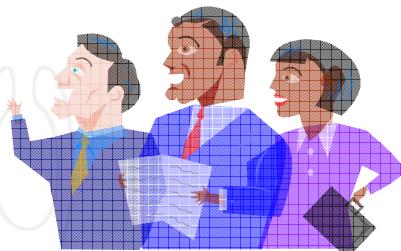
- Conceitos chave:
 - Fases, papéis e responsabilidades de uma revisão formal típica.
 - As diferenças entre diferentes tipos de revisão.
 - Os fatores para revisões de sucesso.

Tipos de Revisão

- **Informal:** nenhum processo formal é usado (gráficos no corredor, testes camaradas, programação por par), mesmo assim útil, barata, popular.
- **Técnica:** processo documentado e definido de remoção de defeitos, envolvendo pares e especialistas técnicos mas não gerentes.
- **Acompanhamento** (*walkthroughs*): o autor “conduz” seus pares “através” do documento ou código.
- **Inspeções:** um moderador treinado (que não seja o autor) lidera a equipe de inspeção (com papéis definidos) através de um processo formal de inspeção (regras, listas de checagem, critérios de entrada e saída), que inclui obtenção de métricas de remoção de defeitos.
- Quando acompanhamento, revisões técnicas ou inspeções são realizadas por um grupo de pares, a revisão pode ser chamada de revisão por pares.

Consenso e Entendimento

- Estar ambíguo ou incompleto pode esconder o significado real das especificações.
- É necessário acordo e entendimento uniforme das especificações.



Muito antes de qualquer código existir, a especificação precisa ser entregue a um grupo de testes externo para ser examinada em busca de integridade e clareza. Como [V. A.] Vyssotsky [do projeto meios de segurança do laboratório Bell] diz, os próprios desenvolvedores não podem fazer isso. **“Eles não lhe dirão que não entenderam; eles alegremente inventarão seu caminho através das lacunas e obscuridades.”**

- Fred Brooks
The Mythical Man-Month, 1975

Um Processo de Revisão Genérico

1. Planejamento
2. Kick-off
3. Preparação
4. Encontro de revisão
5. Retrabalho/reparo
6. Acompanhamento

Inclui estimativa e planejamento, treinamento de participantes, etc.

Esse passo do processo são repetidos a cada item revisado. A preparação toma comumente uma a duas horas. O encontro toma de uma a duas horas em conjunto. Retrabalho/reparo trata da correção dos *bugs* encontrados.

Acompanhamento inclui os itens individualmente assim como todo o processo de análise de melhoria, avaliação de defeito (*bug*) removido na fase de revisão de término (encontro de término), etc.

Os detalhes do processo de revisão dependem do tipo específico de revisão usado no projeto.

Papéis e Responsabilidades

- Moderador: Lidera/conduz as reuniões de revisão.
- Escrevente ou secretário: Coleta informação sobre as descobertas.
- Autor: Descreve, explica, responde perguntas sobre o item.
- Revisor/inspetor: Descobre defeitos (*bugs*) no item.
- Gerente: planeja, consegue recursos e treinamento, apóia, analisa métricas do processo.
- Em alguns casos, uma pessoa pode desempenhar múltiplos papéis:
 - Autores algumas vezes atuam como moderadores.
 - Um dos revisores pode atuar como secretário.
 - As especificidades são determinadas pelo tipo da revisão.

Sugestões para Revisões de Sucesso

- Forneça treinamento.
- Revise o produto, não o autor.
- Estabeleça e siga uma programação e objetivos.
- Limite o debate.
- Mantenha o foco em encontrar, não corrigir, os problemas.
- Limite e selecione cuidadosamente os participantes.
- Faça anotações.
- Insista na preparação (por exemplo, tendo anotações enviadas pelas pessoas).
- Desenvolva uma lista de checagem para cada tipo de item que é revisado.
- Use as técnicas corretas.
- Garanta apoio do suporte.
- **Aprenda e torne-se melhor!**

Bugs Comuns em Requisitos e Projeto

- Ambiguidades: O que exatamente isto significa?
 - Ex.: O sistema deve permitir aos usuários terem ISP e-mail.
 - Que ISPs? Qual tamanho de e-mails? Anexos?
- Estar incompleto: Ok, e então?
 - Ex.: Após três senhas inválidas, o sistema deve bloquear a conta do usuário.
 - Por quanto tempo? Como desbloqueia? Quem pode desbloquear?
- Não testável: Como posso checar este item?
 - Ex.: Sistema deve prover 100% disponibilidade.
 - Nenhuma técnica de teste conhecida para demonstrar disponibilidade perfeita.
- Dependências excessivas, acoplamentos e complexidade.
 - Procure por diagramas de projeto feios e requisitos confusos.

Norma IEEE 1028 para Revisões de Software

1. Visão geral
 - Objetivo, escopo, adaptação, organização, aplicação.
2. Referencias
3. Definições
4. Revisões gerenciais
 - Responsabilidades, entradas/saídas, critérios de entrada/saída, procedimentos.
5. Revisões técnicas
 - Responsabilidades, entradas/saídas, critérios de entrada/saída, procedimentos.
6. Inspeções
 - Responsabilidades, entradas/saídas, critérios de entrada/saída, procedimentos, coleta de dados, melhoria de processo.
7. Acompanhamento
 - Responsabilidades, entradas/saídas, critérios de entrada/saída, procedimentos, coleta de dados, melhoria de processo.
8. Auditorias
 - Responsabilidades, entradas/saídas, critérios de entrada/saída, procedimentos.

Capítulo 3: Técnicas Estáticas Seção 3.2 – Processo de Revisão

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 3: Técnicas Estáticas

Seção 3.3 – Análise Estática por Ferramentas

- Conceitos chave:

- Defeitos e erros típicos identificados pela análise estática quando comparadas com revisões e técnicas dinâmicas.
- Benefícios típicos da análise estática.
- Lista de defeitos típicos de código e projeto identificados pelas ferramentas de análise estática.

Análise Estática e Teste Dinâmico

- Assim como as revisões, a análise estática procura por defeitos no código fonte do software e nos modelos de software.
- Ao contrário do teste dinâmico, a análise estática é feita sem executar realmente o sistema.
- A análise estática envolve a análise do sistema ou seus componentes por uma ferramenta, enquanto que o teste dinâmico nem sempre envolve ferramentas.
- Análise estática pode encontrar defeitos que são difíceis de encontrar ou isolar no teste dinâmico.
 - Exemplos incluem questões de manutenibilidade, uso inseguro de apontadores.
 - Isolamento é mais fácil porque você encontra o *bug*, não o sintoma.

O Que Podemos Analisar?

- Código do programa (ex. fluxo de controle e fluxo de dados).
- Modelos do programa (ex., simulações).
- Saída gerada como HTML e XML.
- Documentos de requisitos e projeto.

Benefícios da Análise Estática

- Detecção de *bugs* mais cedo e barata (antes de iniciar a execução dos testes).
- Avisos sobre onde agrupamentos de *bugs* podem existir, devido a programação perigosa, alta complexidade, etc.
- Localização de *bugs* que o teste dinâmico pode não encontrar.
- Detecção de dependências e inconsistências nos modelos de software (por exemplo, assim como problemas de link em páginas Web).
- Manutenibilidade melhorada do código e do projeto.
- Prevenção de defeitos baseada nas métricas obtidas e lições aprendidas da análise.

Bugs Típicos Encontrados na Análise Estática

- Referenciar uma variável com um valor indefinido.
- Interface inconsistente entre módulos e componentes.
- Variáveis que nunca são usadas.
- Código inatingível (morte).
- Violações das normas de programação.
- Vulnerabilidades de segurança.
- Violações de sintaxe do código e modelos do software.

Usando Ferramentas de Análise Estática

- Usuários típicos são....
 - Programadores, geralmente durante o teste de componente e de integração.
 - Projetistas e arquitetos de sistema durante o projeto.
- Durante a introdução inicial mediante a um sistema existente, as ferramentas de análise estática podem produzir um grande número de avisos.
- Compiladores fazem alguma análise estática, mas estão disponíveis muitas outras ferramentas sofisticadas.

Capítulo 3: Técnicas Estáticas
Seção 3.3 – Análise Estática por Ferramentas

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 3: Técnicas Estáticas

Exercícios de Fixação

- Faça os exercícios no formato do exame para o capítulo 3.
- Discuta.

Capítulo 4: Técnicas de Modelagem de Teste

Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0

151 ctr.rsinet.com.br
2007 © Copyright RSI Informática Ltda 

Capítulo 4: Técnicas de Modelagem de Teste

Seções:

1. Identificando as Condições de Testes e Projetando os Casos de Testes
2. Categorias das Técnicas de Modelagem de Teste
3. Técnicas Baseadas em Especificação ou Caixa Preta
4. Técnicas Baseadas em Estrutura ou Caixa Branca
5. Técnicas Baseadas na Experiência
6. Escolhendo as Técnicas de Teste

Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0

152 ctr.rsinet.com.br
2007 © Copyright RSI Informática Ltda 

Capítulo 4: Técnicas de Modelagem de Teste

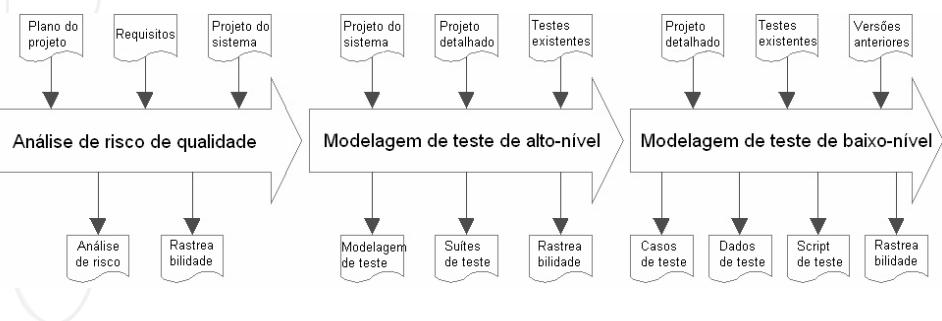
Seção 4.1 – Identificando as Condições de Testes e Projetando os Casos de Testes

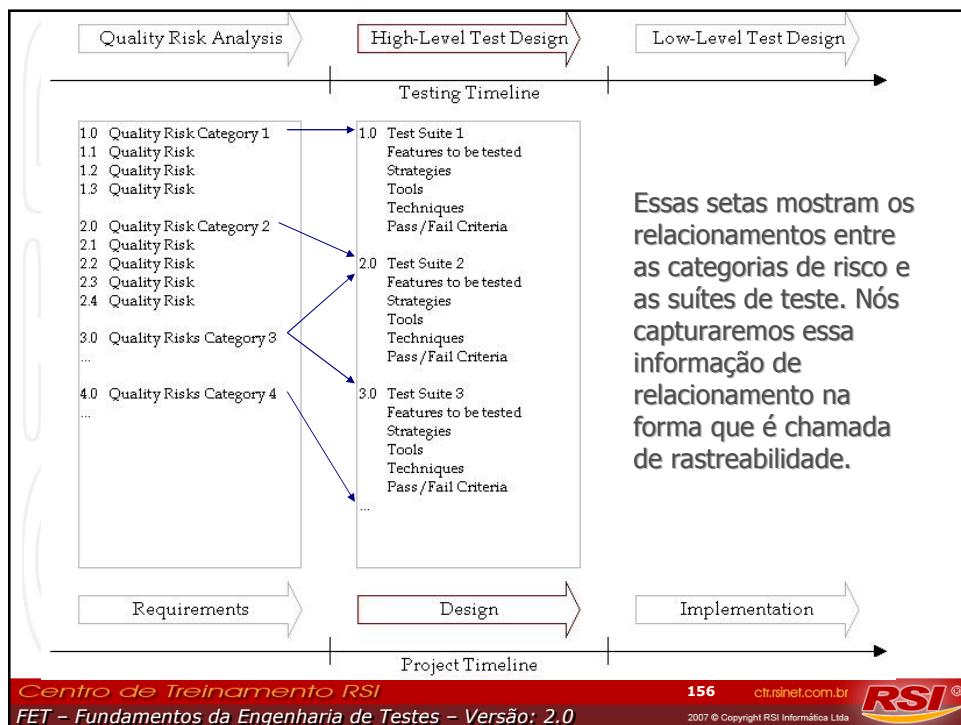
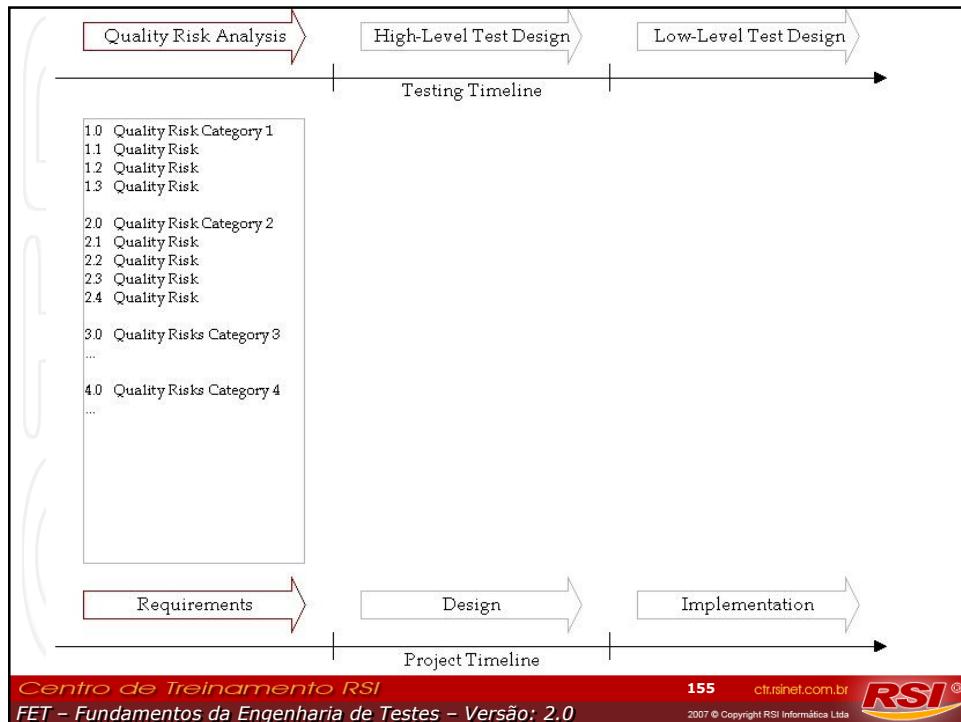
■ Conceitos chave:

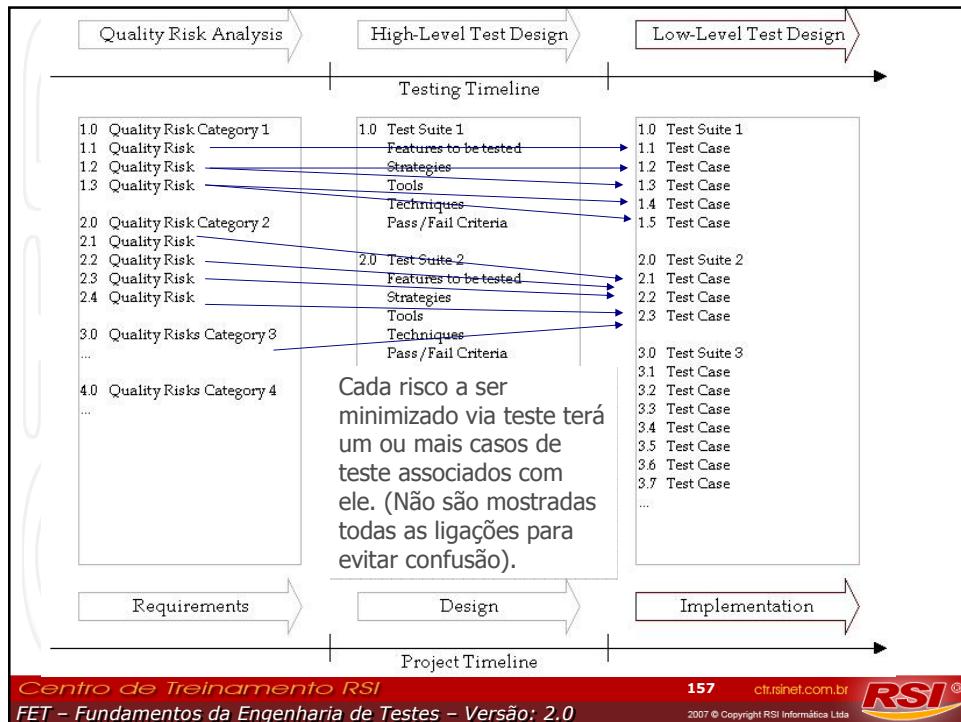
- Basear o teste na análise de risco.
- Determinar o nível de risco com probabilidade e impacto.
- Especificar projetos, casos , procedimentos de teste.
- Traduzir os casos de teste em procedimentos de teste.
- Relacionar casos de teste e procedimentos de teste.
- Desenvolver a programação da execução dos testes.

Fases do Desenvolvimento do Teste

- Desenvolvimento do teste geralmente progride em fases:
 - Análise (riscos de qualidade).
 - Projeto de teste em alto nível.
 - Projeto de teste em baixo nível (implementação).
- Entradas externas usadas para criar entregáveis internos (*testware*).







O Que é Risco de Qualidade ou Produto?

- **Risco**
 - A possibilidade de um resultado negativo ou indesejado.
 - A probabilidade de um risco tornar-se um resultado é...
 $>0, <1$ no futuro...
0 ou 1 no passado.
- **Risco de qualidade ou produto**
 - A possibilidade que o sistema falhará em satisfazer os clientes, usuários, ou outros *stakeholders*.
 - Uma família de *bugs* possíveis está atrás dos riscos de qualidade.

Como Podemos Analisar os Riscos de Qualidade?

Informal	ISO 9126	Modo de falha e análise de efeito
Inicia com as categorias clássicas de riscos de qualidade	Inicia com seis características principais de qualidade	Inicia com categorias, características e subsistemas
Funcionalidade, estados e transações, capacidade e volume, qualidade dos dados, tratamento e recuperação de erro, desempenho, padrões e localização, usabilidade, etc.	Funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade, portabilidade (FRUEMP), então decompõe em subcaracterísticas chave para seu sistema	<i>Stakeholders</i> chave listam possíveis modos de falhas, prevêem seus efeitos no sistema, usuário, sociedade, etc., atribui severidade, prioridade, e probabilidade, então calcula um número de prioridade de risco (RPN)
Determina prioridade para testar cada risco de qualidade com os <i>stakeholders</i> chave	Determina a prioridade para testar cada subcaracterística com os <i>stakeholders</i> chave	<i>Stakeholders</i> usam RPN para dirigir a profundidade e para o teste

Indiferentemente à técnica, o fator chave é a funcionalidade transversal da participação do *stakeholder*, consenso, e a melhor visão geral possível.

Riscos de qualidade são problemas em potencial do sistema, o quais podem comprometer a satisfação do usuário.

Número de prioridade do risco: Traz métrica ao risco.

Risco de negócio (operacional): impacto do problema.

Risco técnico: Probabilidade do problema.

Rastreamento das informações aos requisitos, projeto, ou outras bases de risco.

Risco de qualidade	Risco Técnico	Risco de negócio	# Prior. risco	Extensão do teste	Rastreamento
Categoria 1					
Risco 1					
Risco 2					
Risco n					

Uma hierarquia de categoria de riscos pode ajudar a organizar a lista e ajudar sua memória.

1 = Muito alto
2 = Alto
3 = Médio
4 = Baixo
5 = Muito baixo

O produto dos riscos técnicos e de negócio, de 1 a 25.

1-5 = Extensivo
6-10 = Abrangente
11-15 = Ocasional
16-20 = Oportuno
21-25 = Relata bugs

Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0

160 ctr.rsinet.com.br RSI®
2007 © Copyright RSI Informática Ltda

Dicas para Análise de Risco

- Use uma equipe de *brainstorming* multifuncional.
- Identifique os itens de risco, então defina o nível de risco.
- Somente separe itens de risco quando necessário para distinguir entre diferentes níveis de risco.
- Considere riscos técnicos e de negócio.
 - Risco técnico: probabilidade do problema, impacto do problema no sistema.
 - Risco de negócio: probabilidade do uso, impacto do problema nos usuários.
- Acompanhe e re-alinhe a análise de risco, teste, e o projeto em marcos chave do projeto.

IEEE 829 Especificação de Modelagem de Teste

- A especificação de modelagem de teste descreve uma coleção de casos de teste e condições de teste em alto nível, e inclui as seguintes seções:
 - Identificador da especificação de projeto de teste.
 - Recursos a serem testados (nesta suíte de teste).
 - Refinamentos da abordagem (técnicas específicas, ferramentas, etc.).
 - Identificação do teste (rastrear aos casos de teste na suíte).
 - Funcionalidade critérios de passa/falha (por exemplo, oráculo de teste, base de teste, sistemas legados, etc.).
- Essa coleção de casos de teste geralmente chamada de uma suíte de teste.
- Sequenciamento (suítes de teste e casos dentro das suítes) geralmente dirigidos pela prioridade do risco e do negócio e afetada pelas restrições, recursos, e progresso do projeto.

IEEE 829 Especificação de Caso de Teste

- A especificação de caso de teste descreve os detalhes de um caso de teste, e inclui as seguintes seções:
 - Identificador da especificação do caso de teste.
 - Itens de teste (o que deve ser entregue e testado).
 - Especificações de entrada (entradas do usuário, arquivos, etc.).
 - Especificações de saída (resultados esperados, incluindo telas, arquivos, temporização, etc.).
 - Necessidades de ambiente (hardware, software, pessoas, suporte,...).
 - Requisitos especiais de procedimento (intervenção do operador, permissões, etc.).
 - Dependências entre casos (se necessário configurar pré-condições).
- Na prática, casos de teste variam显著mente em esforço, duração, e número de condições de teste cobertas.

IEEE 829 Especificação de Procedimento de Teste

- Uma especificação de procedimento de teste descreve como executar um ou mais casos de teste, e inclui as seguintes seções:
 - Identificador da especificação do procedimento de teste.
 - Propósito (por exemplo, quais testes serão executados).
 - Requisitos especiais (habilidades, permissões, ambiente, etc.).
 - Passos do procedimento (entrada, configuração, início, prosseguir [os próprios passos], medição dos resultados, encerramento/suspensão, reinicio [se necessário], parada, terminar/diminuir, contingências).
- Procedimentos de teste são geralmente embutidos nos casos de teste.
- Um procedimento de teste é algumas vezes referenciado como um *script* de teste, e pode ser manual ou automatizado.

Cobertura do Caso de Teste (Rastreabilidade)

- Medir ou correlacionar testes mediante áreas de preocupação.
- Usada como meio de medir e melhorar os testes.
- Tipos práticos:
 - Especificações dos requisitos.
 - Especificações de projeto.
 - Áreas funcionais.
 - Riscos de qualidade.
 - Configurações.
- Esta é uma técnica comumente usada para garantir cobertura completa do teste.

Espec	Caso de teste												Total
	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	2.1	2.2	2.3	2.4	
1.1	1	1		1	1		2						6
1.2	2		2	1				1	2				8
1.3													0
1.4		2										2	4
1.5	1			1			1	1					4
1.6					2								2
1.7			2	1		2		2					7
1.8					2						1		3
1.9	1			1						2			4
1.10		1					1	1		1			4
Total	5	4	2	5	4	3	3	4	2	2	7	1	0

- Uma técnica:
 - Use planilha
 - Liste casos de teste e área de cobertura para medir
 - 0: nada; 1: indireto; 2: direto

Capítulo 4: Técnica de Modelagem de Teste Seção 4.1 – Identificando as Condições de Testes e Projetando os Casos de Testes

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 4: Técnicas de Modelagem de Teste

Seção 4.2 – Categorias das Técnicas de Modelagem de Teste

■ Conceitos chave:

- Motivos para teste baseado na especificação (caixa preta), baseado na estrutura (caixa branca), e baseado na experiência.
- Técnicas comuns de caixa preta e caixa branca.
- Características e diferenças entre teste baseado na especificação, teste baseado na estrutura, e teste baseado na experiência.

Três Tipos de Técnicas de Projeto de Teste

- **Baseado na especificação**
 - Crie testes primariamente pela análise da base de teste.
 - Procure por *bugs* no modo do sistema se comportar.
- **Baseado na estrutura (caixa branca)**
 - Crie testes primariamente pela análise da estrutura do componente ou sistema.
 - Procure por *bugs* no modo que o sistema é construído.
- **Baseado na experiência (ataques, listas de checagem, teste exploratório)**
 - Crie testes primariamente baseados no entendimento do sistema, experiência anterior, e suposições sobre *bugs*.
 - Procure por *bugs* nos lugares onde os outros sistemas apresentam *bugs*.
- Testes de caixa preta são baseados em especificação ou experiência, funcional ou não-funcional.

Teste Baseado na Especificação ou Caixa Preta

- Elementos comuns incluem:
 - Modelos formais ou informais usados para especificar o problema a ser resolvido, o software ou seus componentes.
 - Casos de teste sistematicamente derivados desses modelos.
- Exemplos incluem:
 - Partição de equivalência e análise de valor de limite.
 - Diagramas de transição de estado.
 - Tabelas de decisão.

Teste Baseado na Estrutura ou Caixa Branca

- Elementos comuns incluem:
 - Estrutura do sistema (por exemplo, código, projeto, etc.) usada para derivar os casos de teste, por exemplo código e projeto.
 - A extensão da cobertura estrutural pode ser medida por outros casos de teste existentes.
 - Novos casos de teste podem ser sistematicamente derivados para aumentar a cobertura.
- Exemplos incluem:
 - Cobertura de comandos.
 - Cobertura de desvios.

Teste Baseado na Experiência

- Elementos comuns incluem:
 - O conhecimento e experiência usados para derivar casos de teste.
 - Pode considerar conhecimento e experiência do software, seu uso e seu ambiente ou...
 - ...conhecimento sobre defeitos históricos e parecidos, e sua distribuição.
- Exemplos incluem:
 - Ataques.
 - Listas de checagem.
 - Teste exploratório.

Capítulo 4: Técnicas de Modelagem de Teste Seção 4.2 – Categorias das Técnicas de Modelagem de Teste

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 4: Técnicas de Modelagem de Teste

Seção 4.3 – Técnicas Baseadas em Especificação ou Caixa Preta

■ Conceitos chave:

- Escrever casos de teste a partir de modelos de software e de dados usando partição de equivalência, análise valor limite, tabelas de decisão, e diagramas de transição de estado.
- O propósito principal de cada técnica e como a cobertura pode ser medida.
- Teste de caso de uso.

Partição de Equivalência

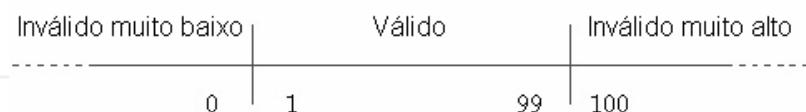
- Divide as entradas, saídas, comportamentos, e ambientes em classes que você acredita que serão tratadas de forma equivalente.
- Define pelo menos um caso de teste para cada partição ou usa valores limites nas partições em que há uma variação.
- Pode usar informações de *marketing* para favorecer classes.
- Por exemplo: testando as impressoras suportadas:
 - Interface física: paralela, serial, USB 1.1, USB 2.0, infravermelho, *firewire*, SCSI, outras?
 - Interface lógica: *postscript*, HPPL, ASCII, outras...
 - Aplicação de imagem: *laser jet*, jato de tinta, *bubble jets*, impressoras matriciais, plotadoras, outras...

Análise do Valor Limite

- Um refinamento da partição de equivalência que seleciona as margens ou pontos terminais de cada partição para teste.
 - Partição de equivalência procura por *bugs* no código que trata cada classe equivalente.
 - Valores limite são membros de classes de equivalência que também procuram por *bugs* nas definições das margens.
- Somente pode ser usada quando os elementos da partição de equivalência são ordenados.
- Limites não-funcionais (capacidade, volume, etc.) podem ser usados também para teste não-funcional.

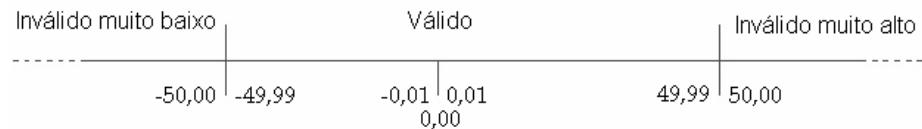
Inteiro

- Quantos itens você gostaria de ordenar?



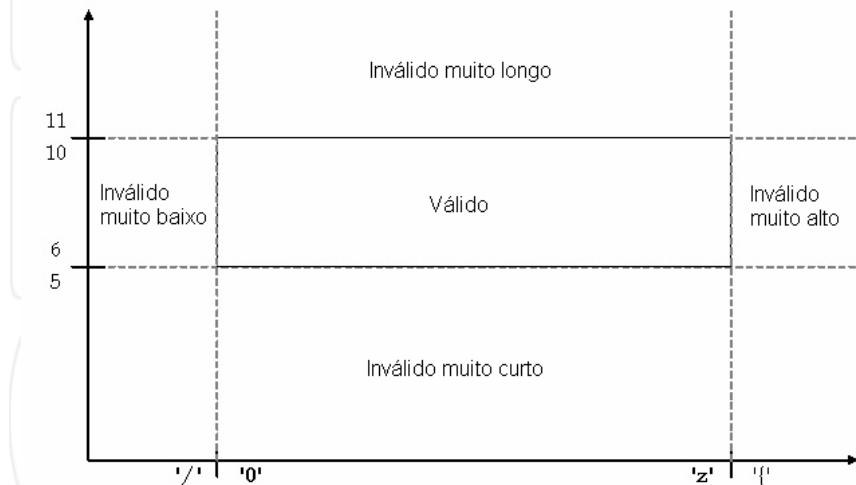
Números Reais

- Qual a média de temperatura?



Caracter e String

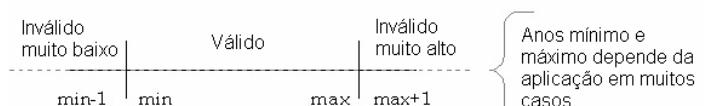
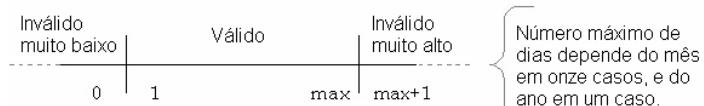
- Senha (6-10 caracteres alfanuméricos)



Data

- Entre a data de partida do seu vôo:

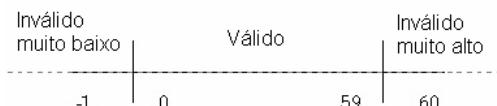
MM/DD/YY



Horário

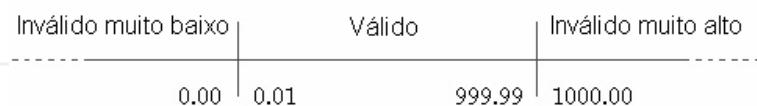
- Entre o horário de partida do seu vôo:

HH:MM:SS



Moeda

- Entre uma oferta de preço (menor que R\$1000,00):



Casos de Uso / Cenário de teste

- Projete vários casos de uso que refletem usos reais e desafiantes do produto.
- Por exemplo, uma aplicação de financiamento para casa:
 - "John e Jenny Stevens têm três filhos..."
 - "...uma casa no valor de R\$200K da qual eles devem R\$130K..."
 - "...dois carros no valor de R\$25K dos quais eles devem R\$17K..."
 - "...rendimentos de R\$45K e R\$75K respectivamente..."
 - "...um pagamento atrasado no cartão Visa e um pagamento atrasado dos carros, dezessete meses e trinta e cinco meses atrás, respectivamente..."
 - "...e eles pedem um financiamento de R\$15K para a casa."
- Algumas metodologias de projeto orientado a objeto incluem casos de uso, portanto isto pode ser uma fonte fácil de testes.

Casos de Uso e Limites

- Revendo nossas condições limite, condições razoáveis de uso afetam as condições limite?
- Se nós temos uma variável contadora de item de quatro bytes sem sinal, nós queremos permitir pedidos de 32.000 itens apenas porque o software suporta tal quantidade?
- Nós permitimos datas de partida após datas de chegada para viagens?
- Aceitar uma entrada “ridícula” é um *bug*?

- O que você acha?

Tabelas de Decisão

- Regras de negócio podem ser especificadas de forma compacta em tabelas de decisão.
 - Decidir como processar um pedido baseado no tamanho, estoque disponível, estado para onde despachar, e assim por diante, está frequentemente em regras de negócio.
 - Elas podem ser mostradas como gráficos de fluxo ou como tabelas.
 - As tabelas de decisão podem produzir casos de teste instantâneos.
- Vamos ver um exemplo.

Tabela de Decisão de ATM

Condição	1	2	3	4	5
Cartão válido	N	S	S	S	S
Senha válida	-	N	N	S	S
Senha inválida=3	-	N	S	N	N
Saldo OK	-	-	-	N	S
Ação					
Rejeita cartão	S	N	N	N	N
Redigita senha	N	S	N	N	N
Prende cartão	N	N	S	N	N
Redigita opção	N	N	N	S	N
Fornece dinheiro	N	N	N	N	S

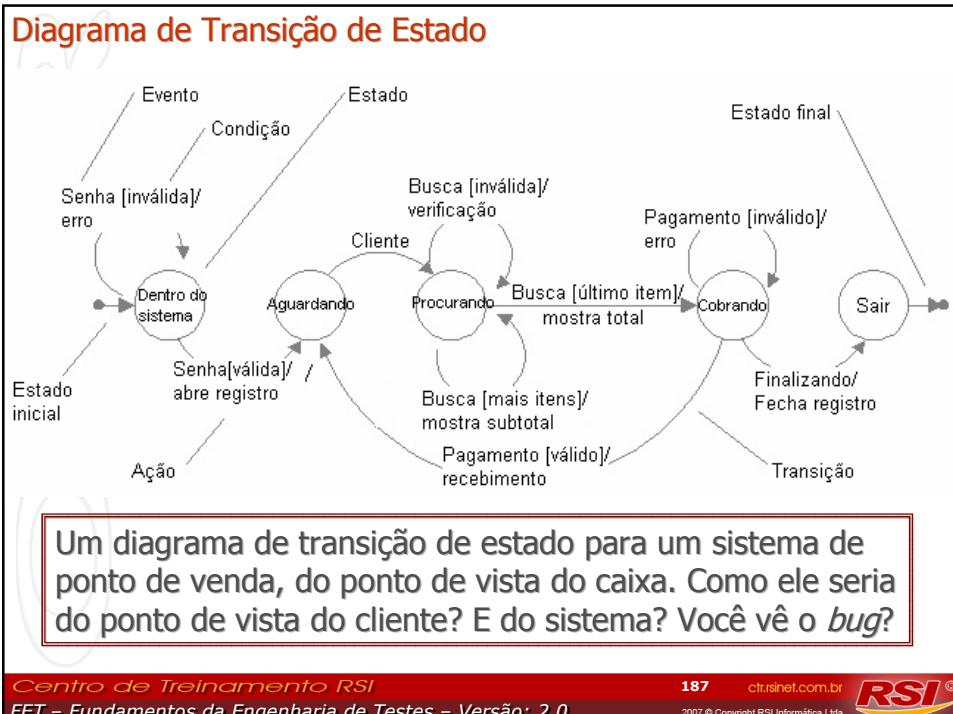
Esta tabela de decisão mostra a lógica de negócio para um ATM. Note que os traços “-” indicam condições que não serão atingidas como parte desta regra.

As regras são mutuamente exclusivas, e somente aquela única regra pode ser aplicada em qualquer momento único de tempo.

Note que a camada de lógica de negócio está usualmente sob a camada de interface com o usuário, portanto neste ponto a checagem básica da sanidade das entradas deveria ter sido feita.

Modelos de Estados Finitos

- Entenda os vários estados que o sistema possui, incluindo qualquer um que seja inicial e final.
- Identifique transições, eventos, condições, e ações em cada estado.
- Use um grafo ou tabela para modelar o sistema e servir como oráculo.
- Para cada evento e condição, verifique a ação e o próximo estado.



Um diagrama de transição de estado para um sistema de ponto de venda, do ponto de vista do caixa. Como ele seria do ponto de vista do cliente? E do sistema? Você vê o *bug*?

Tabelas de Transição de Estado

Estado atual	Evento[Condição]	Ação	Estado novo
Dentro do sistema	Senha [inválida]	Erro	Dentro do sistema
Dentro do sistema	Senha [válida]	Abre registro	Esperando
Dentro do sistema	Cliente	[Indefinido]	[Indefinido]
Dentro do sistema	Busca [qualquer]	[Indefinido]	[Indefinido]
Dentro do sistema	Pagamento [qualquer]	[Indefinido]	[Indefinido]
Dentro do sistema	Finaliza	[Indefinido]	[Indefinido]

Uma tabela de transição de estado pode representar transições de estado complexas que não caberiam num grafo. (Entretanto, a complexidade pode indicar projeto mal feito.) Ela também pode revelar uma situação indefinida, como faz esta porção da tabela para o grafo do slide anterior.

Modelos de Estados Finitos

- Vamos ver outro exemplo.
- Um servidor de impressão (grafo no próximo slide) pode estar:
 - Esperando trabalho.
 - Enfileirando trabalho.
 - Imprimindo trabalho.
 - Esperando intervenção do usuário.
 - Esperando intervenção do operador.
- Um servidor de impressão responde a eventos (entradas do usuário ou impressora) baseado no estado.

Grafo da Máquina de Estados do Servidor de Impressão



Capítulo 4: Técnicas de Modelagem de Teste
Seção 4.3 – Técnicas Baseadas em Especificação ou Caixa Preta

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 4: Técnicas de Modelagem de Teste

Seção 4.4 – Técnicas Baseadas em Estrutura ou Caixa Branca

- Conceitos chave:
 - Cobertura de código.
 - Cobertura de comandos e decisão.
 - Técnicas de projeto de teste de controle de fluxo.

Técnicas Baseadas na Estrutura (Caixa Branca)

- Testes baseados na estrutura são baseados na forma que o sistema trabalha internamente.
 - Determine e atinja um nível de cobertura dos fluxos de controle baseado na análise de código.
 - Determine e atinja um nível de cobertura dos fluxos de dados baseado no código e análise de dados.
 - Determine e atinja um nível de cobertura das interfaces, classes, fluxos de chamadas, e semelhantes baseado na análise das APIs, projeto do sistema, etc.
- Cobertura da estrutura é uma forma de procurar por lacunas dos testes baseados na especificação e na experiência.

Cobertura de Código

- Níveis de cobertura de código:
 - Cobertura de comandos: todo comando executado.
 - Cobertura de desvio (decisão): todo desvio (decisão) tomado em cada direção, verdadeiro ou falso.
 - Cobertura de condição: toda combinação de condições verdadeiras e falsas avaliadas (isto é, a tabela verdade inteira).
 - Cobertura de decisão de multicondição: somente aquelas combinações de condições que podem influenciar a decisão.
 - Cobertura de laço: Todos os caminhos de laço executados zero, uma, e múltiplas (idealmente, máximo) vezes.
- A cobertura de comandos implica na cobertura de desvio (decisão)?

Exemplo de Cobertura de Código: Valor Absoluto

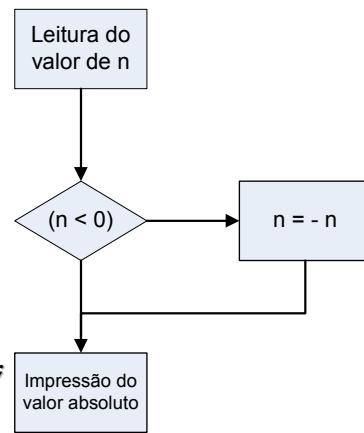
- Considero o seguinte código, no qual é lido um número inteiro e realizada a impressão do seu valor absoluto.

```
#include <stdio.h>

int main () {
    int n;
    printf ("Digite o numero: ");
    scanf ("%d", &n);

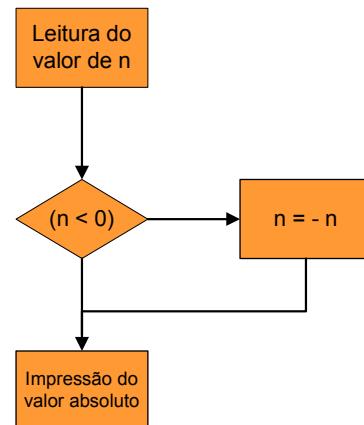
    if (n < 0)
        n = -n;

    printf ("Valor absoluto: %d\n", n);
    return 0;
}
```



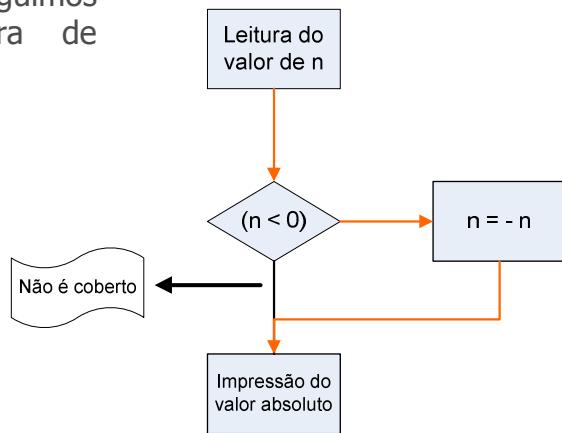
Exemplo de Cobertura de Código: Valor Absoluto

- Para esse código, quais valores de teste para n precisamos para 100% de cobertura de comandos?
 - Basta somente um teste com valor $n < 0$.



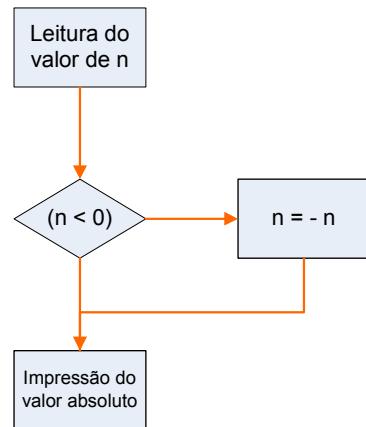
Exemplo de Cobertura de Código: Valor Absoluto

- Com o teste anterior ($n < 0$), não conseguimos 100% de cobertura de desvio.



Exemplo de Cobertura de Código: Valor Absoluto

- Quais valores de teste para n precisamos para 100% de cobertura de desvios?
 - Precisamos de dois testes, $n < 0$ e $n \geq 0$



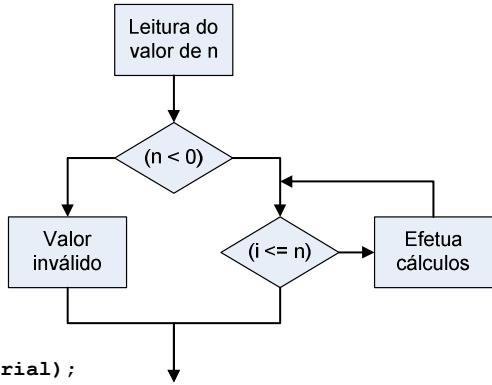
Exemplo de Cobertura de Código: Fatorial

- Considero o seguinte código para calcular o fatorial de um número inteiro dado.

```
#include <stdio.h>

int main () {
    int n, fatorial, i;
    printf ("Digite o numero: ");
    scanf ("%d", &n);

    if (n < 0) {
        printf ("Invalido: %d\n", n);
        n = -1;
    }
    else {
        fatorial = 1;
        for (i = 1; i <= n; i++)
            fatorial *= i;
        printf ("%d! = %d\n", n, fatorial);
    }
    return n;
}
```



Exemplo de Cobertura de Código: Fatorial

- Quais valores de teste para n nós precisamos para cobrir todos os comandos?
 - $n < 0, n > 0$
- Eles nos dão cobertura de desvio?
 - Não, também precisamos $n = 0$
- Eles nos dão cobertura de condição?
 - Sim, não há condições compostas
- E sobre cobertura de caminho?
 - Necessário cobrir $n = 1$ e $n = \text{max}$

```
#include <stdio.h>

int main () {
    int n, fatorial, i;
    printf ("Digite o numero: ");
    scanf ("%d", &n);

    if (n < 0) {
        printf ("Invalido: %d\n", n);
        n = -1;
    }

    else {
        fatorial = 1;
        for (i = 1; i <= n; i++)
            fatorial *= i;
        printf ("%d! = %d\n", n, fatorial);
    }
    return n;
}
```

Cobertura de Código como Ferramenta de Projeto de Teste

- Por elas mesmas, as técnicas caixa preta podem deixar 75% ou mais dos comandos sem cobertura.
 - Isso é um problema?
 - Depende do que não está coberto!
- Ferramentas de cobertura de código podem instrumentar um programa para monitorar a cobertura de código durante o teste.
- Lacunas na cobertura de código podem levar a mais casos de teste para atingir níveis altos de cobertura.

Complexidade Ciclomática de McCabe

- A complexidade ciclomática de McCabe mede a complexidade do fluxo de controle.
 - Medida pelo desenho de um grafo dirigido.
 - Os nós representam entradas, saídas, decisões.
 - Arestas representam comandos que não realizam desvios.
- Ele possui algumas implicações úteis para o teste:
 - Módulos de alta complexidade são inherentemente cheios de bugs e passíveis de regressão.
 - O número de caminhos base pelo grafo é igual ao número de testes base para cobrir o grafo.
- Vamos ver como....

Exemplo de Complexidade Ciclomática: Fatorial

Programa

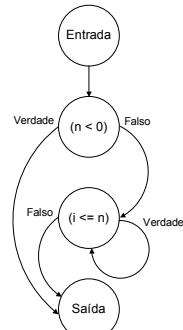
```
#include <stdio.h>

int main () {
    int n, fatorial, i;
    printf ("Digite o numero: ");
    scanf ("%d", &n);

    if (n < 0) {
        printf ("Invalido: %d\n", n);
        n = -1;
    }

    else {
        fatorial = 1;
        for (i = 1; i <= n; i++)
            fatorial *= i;
        printf ("%d! = %d\n", n,
               fatorial);
    }
    return n;
}
```

Diagrama de Fluxo



Complexidade Ciclomática

$$C = \#R + 1 \\ C = 2 + 1 = 3$$

ou

$$C = \#A - \#N + 2 \\ C = 5 - 4 + 2 = 3$$

Definições

C = Complexidade ciclomática
 R = Regiões fechadas
 A = Arestras (setas)
 N = Nós (balões)

Caminhos e Testes Base

Programa

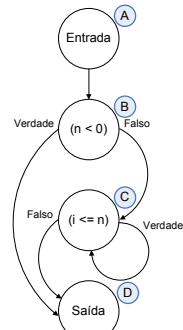
```
#include <stdio.h>

int main () {
    int n, fatorial, i;
    printf ("Digite o numero: ");
    scanf ("%d", &n);

    if (n < 0) {
        printf ("Invalido: %d\n", n);
        n = -1;
    }

    else {
        fatorial = 1;
        for (i = 1; i <= n; i++)
            fatorial *= i;
        printf ("%d! = %d\n", n,
               fatorial);
    }
    return n;
}
```

Diagrama de Fluxo



Caminhos Básicos

1. ABD
2. ABCD
3. ABCCD

Testes Base

Entrada	Esperado
1. -1	(inválido -1)
2. 0	0! = 1
3. 1	1! = 1

Capítulo 4: Técnicas de Modelagem de Teste
Seção 4.4 – Técnicas Baseadas em Estrutura ou Caixa Branca

- *Exercício – Conversor Hexadecimal*

- No próximo slide você encontrará um programa simples em C que aceita uma sequência com caracteres hexadecimais (entre outros caracteres não desejados). Ele ignora os outros caracteres e converte os caracteres hexadecimais na sua representação numérica.
- Se você testar com as sequências de entrada “059”, “ace”, e “ACD” qual nível de cobertura você atingiria?
- Quais sequências de entrada você poderia adicionar para atingir cobertura de comando e desvio? Elas seriam suficientes para testar esse programa?
- Discuta.

Capítulo 4: Técnicas de Modelagem de Teste
Seção 4.4 – Técnicas Baseadas em Estrutura ou Caixa Branca

```
int main() {
    int c;
    unsigned long int hexnum, nhex;
    hexnum = nhex = 0;
    while ((c = getchar()) != EOF) {
        switch (c) {
            case '0': case '1':
            case '2': case '3':
            case '4': case '5':
            case '6': case '7':
            case '8': case '9':
                /* Digito decimal */
                nhex++;
                hexnum *= 0x10;
                hexnum += (c - '0');
                break;
            case 'a': case 'b':
            case 'c': case 'd':
            case 'e': case 'f':
                /* Minusculo */
                nhex++;
                hexnum *= 0x10;
                hexnum += (c - 'a' + 0xa);
                break;
            case 'A': case 'B':
            case 'C': case 'D':
            case 'E': case 'F':
                /* Maiúsculo */
                nhex++;
                hexnum *= 0x10;
                hexnum += (c - 'A' + 0xA);
                break;
            default:
                /* Digitos nao hexadecimais */
                break;
        }
        printf("%d digitos hexadecimais:
%lx\n", nhex, hexnum);
    }
    return 0;
}
```

Capítulo 4: Técnicas de Modelagem de Teste
Seção 4.4 – Técnicas Baseadas em Estrutura ou Caixa Branca

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 4: Técnicas de Modelagem de Teste

Seção 4.5 – Técnicas Baseadas na Experiência

- Conceitos chave
 - Razões para escrever casos de teste baseados na intuição, experiência e conhecimento
 - Comparar técnicas baseadas na experiência com técnicas de teste baseadas na especificação

Testes Baseados na Experiência

- Testes baseados na experiência são baseados no testador...
 - ... habilidade e intuição.
 - ... experiência com aplicações similares.
 - ... experiência com tecnologias similares.
- Ao invés de serem pré-projetados, os testes baseados na experiência são geralmente criados durante a execução do teste (isto é, a abordagem de teste é dinâmica).
- Testes são frequentemente limitados no tempo ou “*time-boxed*” (isto é, breves períodos focados em condições de teste específicas).
- Exemplos incluem suposição de erros, caça ao *bug*, “quebra do software” baseada em listas de checagem ou taxonomias dos *bugs*, e teste exploratório.

Abordagens Comuns Baseadas na Experiência

- Muitas abordagens profissionais para teste baseado na experiência não criam testes totalmente durante a execução.
- Podem ser guiadas por:
 - Lista de checagem.
 - Taxonomia do *bug*.
 - Lista de ataques.
 - Abordagem de caça ao *bug*.
 - Conjunto de gráficos de teste.
- Estas linhas mestras são preparadas antes do teste com algum nível de detalhe.
- Teste puramente “*on-the-fly*” (teste ad hoc) é comum, mas trata-se usualmente de (ineficiente) teste manual aleatório.

Estratégias de Teste Dinâmico

Vantagens

- Efetivo na descoberta de *bugs*.
- Resiste ao paradoxo do pesticida devido à grande variedade.
- Eficiente (mantém registros pequenos).
- Boa checagem nos testes preparados.
- Divertido e criativo.

Desvantagens

- Cobertura com lacunas, especialmente quando sob pressão.
- Difícil de estimar.
- Sem prevenção de *bug*.
- Discussão extensiva não é adequada a grandes equipes.
- Nem todos os testadores têm o nível necessário de habilidade e experiência.

Estudo de Caso Teste Exploratório

Equipe	7 Técnicos	3 Engenheiros + 1 Gerente
Experiência	<10 anos no total	> 20 anos no total
Tipo de teste	Scripts precisos	Exploratório documentado
Hrs/Dia de testes	42	6
Bugs encontrados	928 (78%)	261 (22%)
Efetividade	22	44

Este estudo de caso mostra o teste exploratório como cerca de duas vezes mais efetivo em encontrar *bugs* numa base hora-por-hora. Ele é显著mente mais eficiente, porque os testes com scripts requerem esforço extensivo para criá-los.

Entretanto, em qual extensão o nível relativo de experiência é importante? O que o teste exploratório cobriu? Qual é o valor de reuso dos *scripts*?

Capítulo 4: Técnicas de Modelagem de Teste
Seção 4.5 – Técnicas Baseadas na Experiência

▪ *Exercício – Testes Dinâmicos vs. Pré-projetados*

Principais fatores ou preocupações	Reativo	Pré-projetado
Realizada através do teste de regressão		
Maximiza a eficiência de encontrar <i>bugs</i>		
Usa testadores com experiência limitada		
Automatiza metade dos casos de teste		
Entrega precisa, estimativa precisa		
Testa sem a preparação da equipe de teste		
Testa uma mudança rápida de UI		
Usa um esforço de teste distribuído		

Coloque um “+” na coluna onde o fator ou preocupação motiva a utilização da abordagem, e um “-” na coluna onde o fator ou preocupação desestimula a utilização da abordagem.

Capítulo 4: Técnica de Modelagem de Teste
Seção 4.5 – Técnicas Baseadas na Experiência

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 4: Técnica de Modelagem de Teste

Seção 4.6 – Escolhendo as Técnicas de Teste

- Conceito chave:

- Os fatores que influenciam a seleção das técnicas apropriadas de projeto de teste.

Fatores na Escolha das Técnicas

- Tipo de sistema.
- Normas regulatórias.
- Requisitos do cliente ou contratuais.
- Nível e tipo do risco.
- Objetivos de teste.
- Documentação disponível.
- Conhecimento dos testadores.
- Tempo e orçamento.
- Ciclo de vida do desenvolvimento.
- Experiência prévia nos tipos de defeitos encontrados.
- Outros?

O Espectro Dinâmico / Pré-projetado

Sem documentação; frequentemente associada a esforços amadores em testes.
Prós: barata e rápida.
Contras: sem medidas de cobertura

Usado por empresas fortemente regulamentadas; segue modelos específicos.
Prós: Rastreável e auditável
Contras: Altos custos iniciais e continuos, nem sempre seguido

Puramente exploratório	Exploratório documentado
------------------------	--------------------------

Muitas equipes de teste equilibram precisão e detalhamento, alcançando reprodutibilidade e rastreabilidade adequadas dentro de orçamentos realistas

IEEE 829	Puramente com scripts
----------	-----------------------

Pouca documentação; uma técnica sofisticada.
Prós: barata porém mensurável
Contras: necessita de testadores e gerentes habilidosos

Documentação padronizada; uma técnica bastante descrita.
Prós: reproduzível e rastreável
Contras: Recursos significantes para desenvolver e manter

Detalhe e Precisão do Caso de Teste

- *Trade-offs* no espectro da documentação do teste:
 - Testes precisos permitem que os testadores sejam menos habilidosos, mas não é muito flexível.
 - Testes imprecisos podem cobrir mais condições, mas não são muito reprodutíveis, especialmente entre diversos testadores.
 - Testes precisos provêm critérios transparentes de teste, mas são difíceis e caros para manter.
 - Testes imprecisos são rápidos de escrever, mas a cobertura pode ser difícil de definir e medir.
- O grau de quanto o esforço de teste é dinâmico pode ser medido contando o número de palavras da documentação (nos seus casos de teste e procedimentos de teste) por hora de teste da execução do teste.

Capítulo 4: Técnica de Modelagem de Teste
Seção 4.6 – Escolhendo as Técnicas de Teste

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 4: Técnica de Modelagem de Teste

Exercícios de Fixação

- Faça os exercícios no formato do exame para o capítulo 4.
- Discuta.

Capítulo 5: Gerenciamento de Teste

Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0

221 ctr.rsinet.com.br RSI®
2007 © Copyright RSI Informática Ltda

Capítulo 5: Gerenciamento de Teste

Seções:

1. Organização do Teste
2. Planejamento e Estimativa do Teste
3. Monitoração e Controle do Progresso do Teste
4. Gerenciamento de Configuração
5. Riscos e Teste
6. Gerenciamento de Incidente

Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0

222 ctr.rsinet.com.br RSI®
2007 © Copyright RSI Informática Ltda

Capítulo 5: Gerenciamento de Teste

Seção 5.1 – Organização do Teste

■ Conceitos chave:

- A importância do teste independente.
- Os benefícios e desvantagens do teste independente.
- Diferentes membros da equipe devem ser considerados para a organização da equipe de teste.
- Tarefas do típico líder de teste e do testador.

Qual é a Função da Equipe de Teste?

- Teste/controle de qualidade
 - Gerenciamento do risco.
 - Avaliação da qualidade.
- Garantia da Qualidade
 - Gerenciamento teste.
 - Garantindo a qualidade do produto pelo processo.
- Tenha certeza de que você é competente, tenha as pessoas necessárias, e suporte político para exercer o seu papel.



Don Quixote: campeão solitário da qualidade? Entendendo mal seu papel na empresa, você pode fazer um grande malefício político para você mesmo.

Variações de Independência

- Sem testadores independentes, os desenvolvedores testam seu próprio código.
- Testadores independentes dentro de equipes de desenvolvimento.
- Uma equipe ou grupo de teste independente dentro da organização, reportando à gerência do projeto ou à gerência executiva.
- Testadores independentes da organização de negócio ou usuário da comunidade.
- Especialistas de teste independente para alvos específicos de teste, tais como testadores de usabilidade, segurança ou certificação.
- Testadores independentes de fora ou externos à organização.

Por que Teste Independente?

- Dadas aplicações complexas e/ou críticas, é necessário:
 - Vários níveis de teste.
 - Alguns ou todos os níveis de teste sendo realizados por testadores independentes.
- O teste no desenvolvimento, apesar da sua importância, é normalmente menos efetivo em encontrar *bugs* (Jones aponta para 50% como máximo).
- Se testadores independentes querem ou têm autoridade para requerer e definir processos e regras, eles devem ter um direcionamento claro.

Considerações de Independência

Benefícios

- Ver mais, outros e diferentes defeitos.
- Se estiver em dúvida, é um *bug*.
- Verifica suposições de especificação e implementação.
- Credibilidade.
- Plano de carreira do testador.

Armadilhas Potenciais

- Isolamento da equipe de desenvolvimento.
- Visto como um gargalo.
- Programadores perdem o senso de responsabilidade pela qualidade.

Líder de Teste

- Conceber estratégias e planos de teste.
- Escrever ou revisar política de teste.
- Perguntar sobre teste para outras atividades de projeto.
- Estimar o teste.
- Aquisição de recursos de teste.
- Conduzir a especificação, preparação, implementação, e execução do teste.
- Monitorar e controlar a execução do teste.
- Adaptar o plano de teste baseado nos resultados do teste.
- Garantir gerenciamento de configuração do *testware*.
- Garantir rastreabilidade.
- Medir o progresso do teste, avaliar a qualidade do teste e do produto.
- Planejar qualquer automatização do teste.
- Selecionar ferramentas e organizar qualquer treinamento do testador.
- Garantir implementação do ambiente de teste.
- Programar os testes.
- Escrever relatórios de resumo do teste.

Testador

- Revisar e contribuir para os planos de teste.
- Analisar, revisar e avaliar requisitos do usuário, especificações.
- Criar suítes de teste, casos, dados e procedimentos.
- Iniciar o ambiente de teste.
- Implementar testes em todos os níveis de teste.
- Executar e registrar os testes, avaliar resultados e documentar problemas encontrados.
- Monitorar o teste usando as ferramentas apropriadas.
- Automatizar os testes.
- Medir o desempenho de componentes e sistemas.
- Revisar os testes uns dos outros.

Refinando a Posição do Testador

- Engenheiros de Teste
 - Pares técnicos dos programadores.
 - Escolhe teste como uma especialidade.
 - Escreve casos de teste, organiza suites de teste.
 - Cria, customiza e usa ferramentas avançadas de teste.
 - Possui habilidades únicas.
- Técnico de teste
 - Testador habilitado e experiente.
 - Pode ser um aspirante a engenheiro de teste.
 - Executa testes.
 - Reporta bugs.
 - Atualiza o estado atual do teste.
 - Auxilia os engenheiros de teste.
- Outros membros da equipe
 - Administradores de sistema e banco de dados.
 - Engenheiros de versão e configuração.
 - Ferramenteiros de teste.

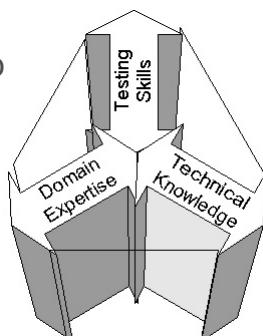
As posições corretas para sua equipe dependem das habilidades que você necessita.
Tenha certeza de balancear habilidades e posições na sua equipe de teste - habilidades são complementares e o todo pode ser maior do que a soma das partes.

Utilizando Testadores Amadores

- Em muitos projetos, testadores amadores (pessoas que não tem o teste como meio de vida) são usadas como parte (ou todo) da equipe de teste.
- Essas pessoas tipicamente incluem gerentes de projeto, gerentes de qualidade, programadores, especialistas no domínio e no negócio, ou operadores de infraestrutura ou TI.
- Uma equipe de teste como essa geralmente possui fortes habilidades em algumas áreas (domínio do negócio ou tecnologia), mas habilidades fracas em outros, e nenhuma habilidade ou experiência substancial com teste.
- Teste é um campo especial, com habilidades especiais (das quais as básicas são cobertas neste curso).

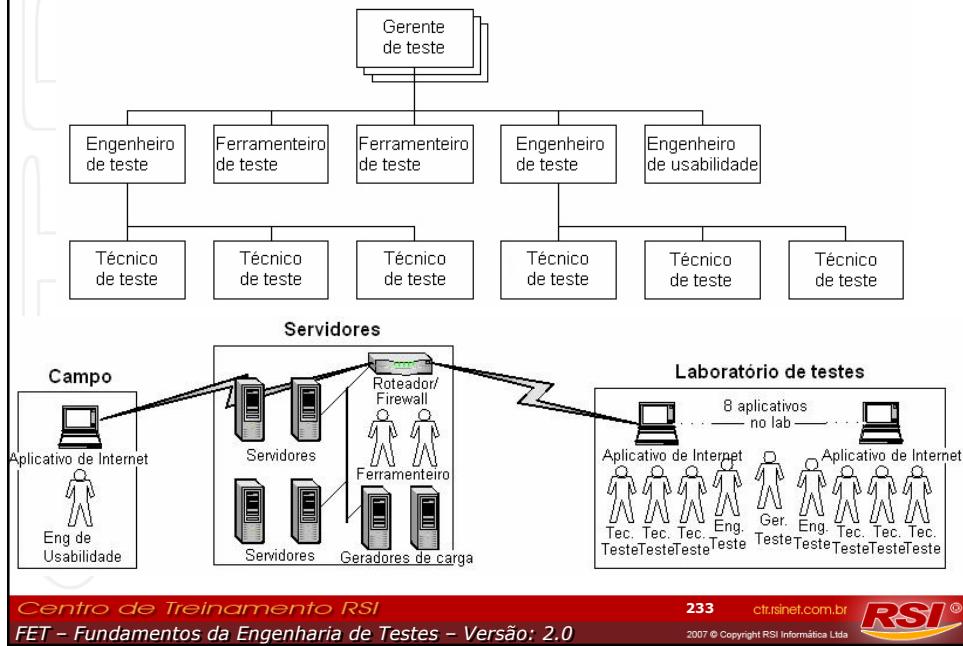
Balanceando as Habilidades

- Boas equipes de teste têm a mistura correta de habilidades baseada nas tarefas e atividades.
- Especialista no domínio da aplicação (negócio).
 - Entende o comportamento desejado.
- Testador habilitado.
 - Conhece riscos de qualidade e técnicas de teste.
- Guru técnico.
 - Consciente das questões técnicas e limitações.
- Qual é a mistura correta para...
 - ...teste conformidade de Internet?
 - ...teste de medicina nuclear?
 - ...seu projeto?



A profundidade e o comprimento apropriado de cada seta na figura depende do projeto, processo e produto

Estudo de Caso: Posições, Organização, Projeto



Capítulo 5: Gerenciamento de Teste Seção 5.1 – Organização do Teste

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 5: Gerenciamento de Teste

Seção 5.2 – Planejamento e Estimativa do Teste

■ Conceitos chave:

- Níveis e objetivos do planejamento do teste.
- Propósito e conteúdo do plano de teste.
- Estratégias de teste.
- Programação e priorização de casos de teste.
- Planejamento do teste para um projeto, níveis, alvos.
- Preparação e execução do teste pelo plano.
- Critérios de saída do teste.
- Estimativa através de métricas e experiência.
- Fatores de estimativa do teste.

Desenvolvendo Planos de Teste

- Por que escrever (e atualizar) planos de teste?
 - Confrontar desafios, cristalizar pensamento, adaptar à mudança.
 - Comunique o plano para testadores, pares, gerentes.
- Considere múltiplos planos de teste quando os testes possuem...
 - Períodos de tempo diferentes (por exemplo, fases e níveis).
 - Metodologias e ferramentas diferentes (por exemplo, desempenho e funcionalidade).
 - Objetivos diferentes (por exemplo, teste de sistema e beta teste).
 - Audiências diferentes (por exemplo, teste de hardware e de software).
- ...mas então você pode querer um plano mestre de teste.
- Circule um ou dois rascunhos.
 - Promove realimentação e discussão cedo.
 - Previne desperdício de tempo se você está na direção errada.

Atividades do Planejamento do Teste

- Definir a abordagem do teste, níveis do teste.
- Integrar, coordenar o teste dentro do ciclo de vida.
- Decidir quem, o que, quando, como testar.
- Atribuir recursos para tarefas de teste.
- Definir a documentação do teste.
- Determinar o nível de detalhe para casos de teste, procedimentos em ordem para prover informação suficiente para apoiar preparação e execução reproduzível do teste.
- Selecionar métricas, gráficos, e relatórios (entregáveis) de monitoramento, controle, e reporte do teste.

Plano de Teste IEEE 829

- Um plano de teste é um plano de subprojeto para a parte a ser testada de um projeto, e inclui as seções mostradas no próximo slide.
- Você pode adaptar o esqueleto da IEEE 829 para uso em cada plano de teste detalhado (por exemplo, nível ou fase) bem como pelo plano mestre de teste.
- Você também pode criar seu próprio modelo ou esqueleto.
- O planejamento do teste influencia (e é influenciado por) política de teste da organização de teste, o escopo do teste, objetivos, riscos, restrições, criticidade, testabilidade, e disponibilidade de recursos.

Esqueleto do Plano de Teste da IEEE 829

- Identificador do plano de teste
- Introdução
- Itens de teste (isto é, o que é fornecido para teste)
- Funcionalidades a serem testadas
- Funcionalidades que não serão testadas
- Abordagem (estratégias, organização, extensão do teste)
- Critérios do item passar/falhar
- Critérios do teste (por exemplo, entrada, saída, suspensão e reativamento)
- Entregáveis do teste (por exemplo, relatórios, gráficos, etc.)
- Tarefas do teste (ou pelo menos os marcos chave)
- Necessidades de ambiente
- Responsabilidades
- Pessoas e necessidades de treinamento
- Programação
- Riscos e contingências (qualidade [produto] e riscos do projeto)
- Aprovações

Transições: Critérios de Entrada

- Critérios de entrada medem se o sistema está pronto para uma fase particular do teste.
 - Entregáveis prontos?
 - Laboratório prontos?
 - Equipes prontas?
- Estes tendem a se tornar cada vez mais rigorosos conforme as fases prosseguem.

Amostra de Critérios de Entrada

O teste de sistema pode iniciar quando:

1. Sistemas de rastreamento de *bugs* e teste estão prontos.
2. Todos os componentes estão sob controle formal, configuração automatizada e controle de gerenciamento de versão.
3. A equipe de operações configurou o ambiente servidor do Teste de Sistema, incluindo todos os componentes de hardware e subsistemas alvo. A Equipe de Teste recebeu o acesso apropriado a esses sistemas.
4. As Equipes de Desenvolvimento completaram todas as funcionalidades e correções de *bugs* programados para a versão.
5. As Equipes de Desenvolvimento testaram unitariamente todas as funcionalidades e correções de *bugs* programadas para a versão.
6. Menos do que 50 *bugs* necessário-corrigir (por Vendas, Marketing, e Serviço ao Cliente) estão em aberto para a versão.
7. As Equipes de Desenvolvimento enviam o software para a Equipe de Teste 3 dias úteis antes de iniciar o Teste do Sistema.
8. A Equipe de Teste completa um "smoke test" de 3 dias e relata os resultados na reunião de Entrada na Fase de Teste de Sistema.
9. A Equipe de Gerenciamento do Projeto concorda em prosseguir na reunião de Entrada na Fase de Teste de Sistema. Os seguintes tópicos serão resolvidos na reunião:
 - Se o código está completo.
 - Se o teste unitário está completo.
 - Determinar uma data alvo para a correção de quaisquer *bugs* necessário-corrigir conhecidos (nunca mais de uma semana após a Entrada na Fase de Teste de Sistema).

Transições: Critérios de Continuação

- Critérios de continuação medem se o teste pode prosseguir eficientemente e efetivamente.
 - Problemas no ambiente de teste.
 - *Bugs* bloqueando o teste no sistema sob teste.
- “Critério de Continuação” é uma forma polida de dizer “critério de parada” ao contrário.
 - Parar uma fase de teste raramente é popular.

Amostra de Critérios de Continuação

O Teste de Sistema continuará se:

1. Todo o software entregue à Equipe de teste é acompanhado de Notas da Versão.
2. Nenhuma mudança é feita no sistema, seja no código fonte, arquivos de configuração, ou outras instruções ou processos de inicialização, sem um relatório de *bug* acompanhando. Se uma mudança deve ser feita sem um relatório de *bug*, o Gerente de Teste abrirá um relatório urgente de *bug* pedindo informação e escalando ao seu gerente.
3. O *backlog* de *bugs* abertos ("lacuna de qualidade") continua com menos de 50. Os períodos para fechamento continuam menores do que 14 dias (em média, *bugs* são corrigidos dentro de dois ciclos de versão semanais).
4. Reuniões de revisão de *bugs* ocorrem duas vezes por semana até a Fase de Saída do Teste de Sistema para gerenciar o *backlog* de *bugs* abertos e os tempos de encerramento dos *bugs*.

Transições: Critérios de Saída

- Critérios de saída medem se a fase de teste pode ser considerada completa.
 - Medidas completas, como cobertura de código, funcionalidade ou risco.
 - Medidas de estimativas de densidade de defeitos ou confiabilidade.
 - Custo.
 - Riscos residuais, tais como defeitos não corrigidos ou falta de cobertura de teste em certas áreas.
 - Programações como aquelas baseadas no tempo para o mercado.
- Lembre que essas são decisões de negócio.

Amostra de Critérios de Saída

O Teste do Sistema terminará quando:

1. Nenhuma mudança (projeto/código/funcionalidades), exceto para tratar de defeitos identificados pelo Teste de Sistema, que ocorreram nas últimas 3 semanas.
2. Nenhum pânico, queda, travamento, "catástrofe", término inesperado de processo, ou outra forma de parada do processamento não tenha ocorrido em qualquer hardware ou software de servidor nas últimas 3 semanas.
3. Nenhum sistema cliente ficou inoperante devido a falha de atualização durante o Teste do Sistema.
4. A Equipe de Teste executou todos os testes planejados para o software candidato-GA.
5. As Equipes de Desenvolvimento resolveram todos os *bugs* definidos como "necessário corrigir" por Vendas, Marketing, e Serviço ao Consumidor.
6. A Equipe de Teste verificou que todas as questões no sistema de rastreamento de *bugs* ou estão fechadas ou adiadas, e, onde apropriado, verificadas por teste de regressão e confirmação.
7. As métricas de teste indicam: estabilidade e confiabilidade do produto; todos os testes planejados foram completados; cobertura adequada dos riscos críticos de qualidade.
8. A equipe de Gerenciamento de Projeto concorda que o produto, como definido durante o ciclo final do Teste de Sistema, satisfará as expectativas razoáveis de qualidade do cliente.
9. A equipe de Gerenciamento de Projeto preside uma Reunião de Saída da Fase de Teste de Sistema e concorda que nós completamos o Teste de Sistema.

Desenvolvendo uma *Work-Breakdown-Structure*

- Quais são os estágios principais de um subprojeto de teste?
 - Planejamento.
 - Recrutamento (se aplicável).
 - Aquisição e configuração do ambiente de teste.
 - Desenvolvimento do teste.
 - Execução do teste.
- Quebre em tarefas distintas dentro de cada estágio.
- Quais suítes de teste são requeridas pelos riscos críticos?
 - Por exemplo, funcionalidade, desempenho, tratamento de erro, etc.
- Para cada suíte de teste, quais tarefas são requeridas?
- Tarefas devem ser curtas em duração (por exemplo, poucos dias).

Estimativa

- Há duas abordagens gerais para estimativa (incluindo estimativa do teste):
 - Estimando tarefas individuais pelo dono dessas tarefas ou por especialistas (*bottom-up* através de *work-breakdown-structure*).
 - Estimando o esforço de teste baseado em métricas de projetos anteriores ou similares ou baseado em valores típicos (*top-down* ou *bottom-up* através de modelos paramétricos, princípios básicos, etc.).
- Embora ambos sejam úteis, extraíndo conhecimento da equipe para criar uma *work-breakdown-structure*, em seguida aplicando modelos e princípios básicos para checar e ajustar a estimativa, é frequentemente mais preciso (e mais defensável).

Fatores a Considerar na Estimativa do Teste

- Testar é complexo, e influenciado por:
 - Fatores de processo: teste pervasivo, controle de mudanças, maturidade de processo, ciclo de vida, processos de desenvolvimento e teste, primeiras fases de teste, (estimado vs atual) níveis e correções de *bugs*.
 - Fatores materiais: ferramentas, teste do sistema, ambientes de teste e depuração, documentação do projeto, similaridade.
 - Fatores humanos: habilidades, expectativas, suporte, relacionamentos
 - Fatores de atraso: complexidade, muitos *stakeholders*, muita renovação, distribuição geográfica, necessidade de documentação detalhada do teste, logística enganadora, dados de teste frágeis.
- Entenda técnicas de estimativa e esses fatores.
- Desvio da estimativa de teste pode surgir de fatores externos e eventos antes ou durante o teste.

Estratégias de Teste

- Análítica: por exemplo, baseada em risco e baseada em requisitos.
- Baseada em modelo: por exemplo, teste estocástico ou estatístico.
- Abordagem metódica: segue uma metodologia pré-planejada, por exemplo, baseada em falha, baseada em lista de checagem, e baseada em qualidade.
- Compatível com processo/norma: por exemplo, IEEE 829, ou metodologia ágil como desenvolvimento orientado a teste.
- Dinâmica ou heurística: uma abordagem mais reativa a eventos e condições do que pré-planejada ou pré-projetada.
- Baseada em conselhos: cobertura de teste dirigida primariamente por orientação externa (não-testador), guiada.
- Regressão: reuso do material existente de teste, automatização extensiva dos testes, suítes padrão de testes.

Capítulo 5: Gerenciamento de Teste Seção 5.2 – Planejamento e Estimativa do Teste

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 5: Gerenciamento de Teste

Seção 5.3 – Monitoração e Controle do Progresso do Teste

■ Conceitos chave:

- Métricas comuns usadas para monitoramento da preparação e execução do teste.
- Entender e interpretar métricas de teste.
- O propósito e conteúdo do documento de relatório de teste de acordo com a norma IEEE 829.

Métricas Comuns de Teste

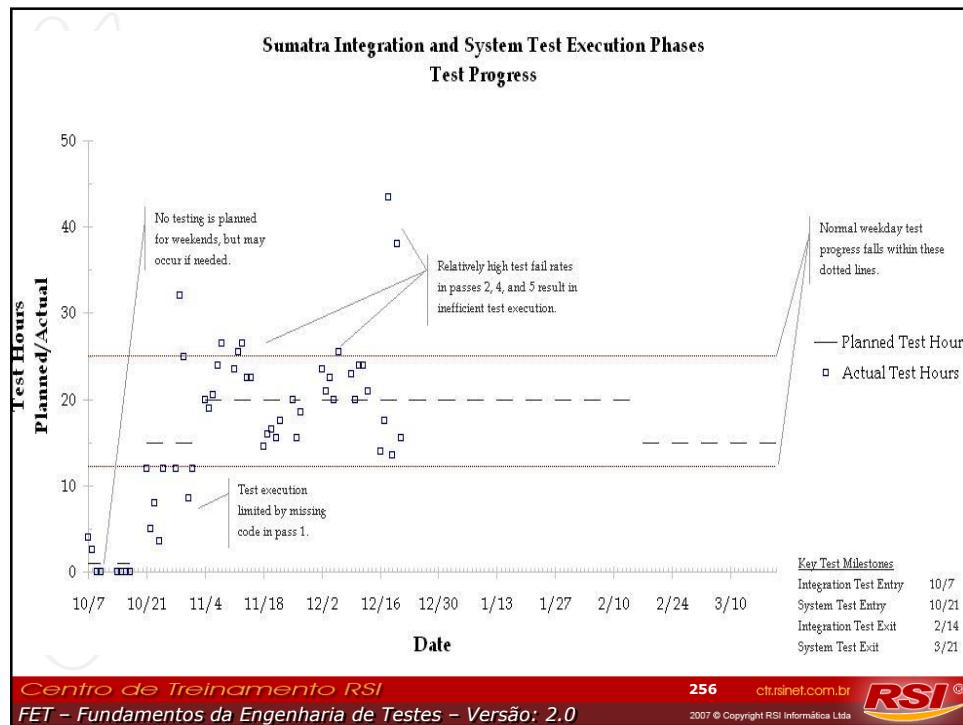
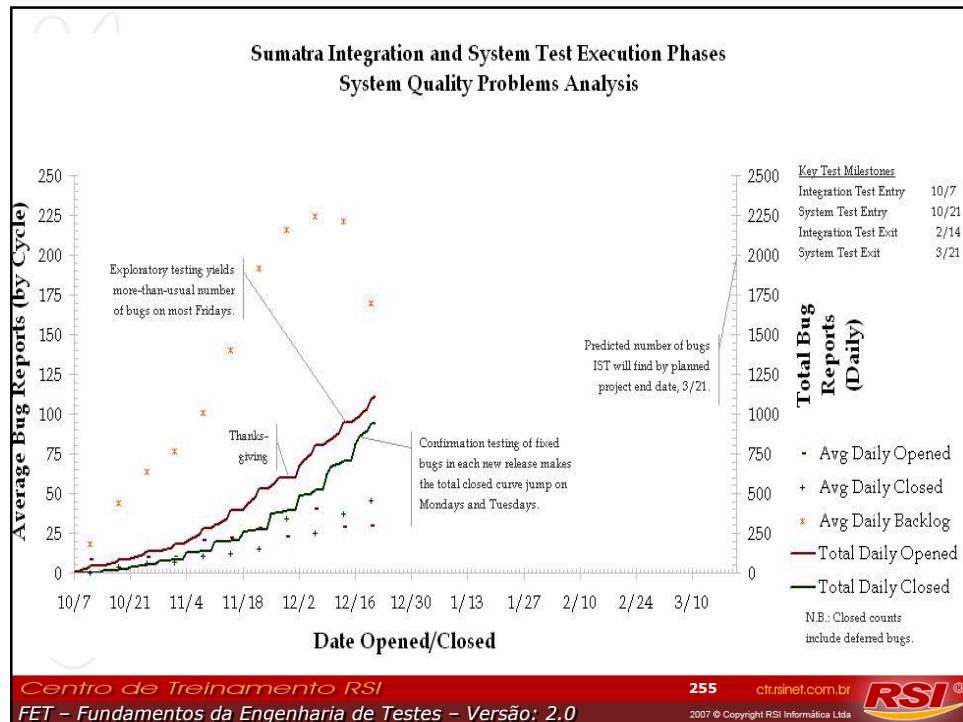
- Porcentagem de quanto da preparação de casos de teste está completa (ou porcentagem de casos de teste planejados que estão preparados).
- Porcentagem de quanto da preparação do ambiente de teste está completo.
- Execução de casos de teste (por exemplo, número de casos de teste executados/não executados, e número de casos de teste que passaram/falharam).
- Informação sobre defeitos (por exemplo, densidade de defeitos, defeitos encontrados e corrigidos, taxa de falha, e resultados de repetição de teste).
- Cobertura de requisitos, riscos ou código por testes, incluindo resultados passa/falha.
- Nível de confiança (subjetivo) dos testadores no produto.
- Datas dos marcos de teste.
- Custos do teste, incluindo o custo de descobrir o próximo defeito ou executar o próximo teste comparado com o benefício.

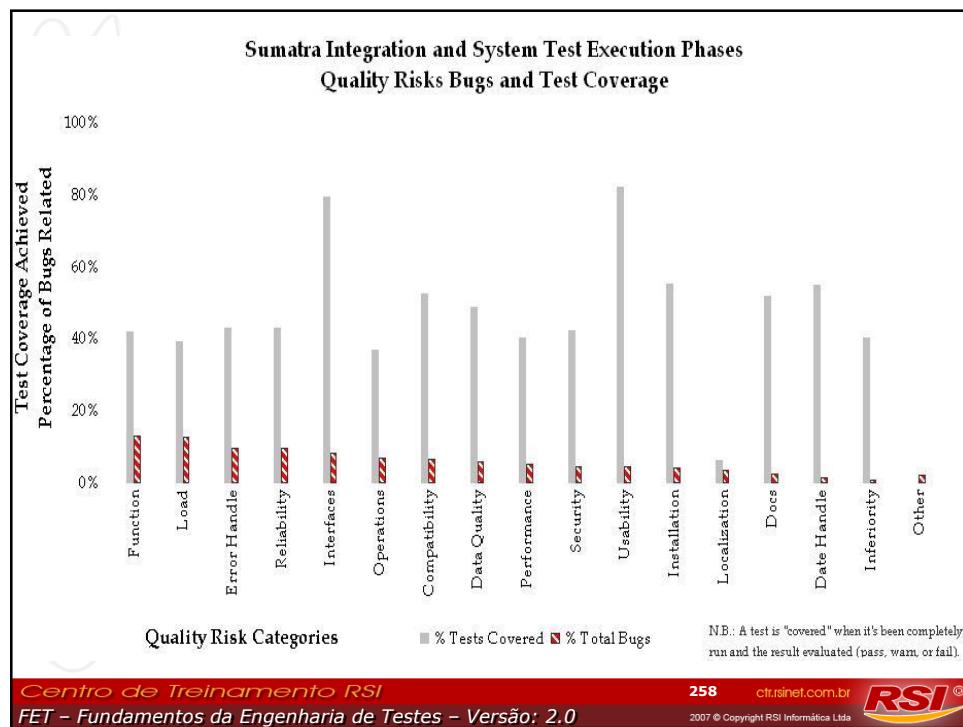
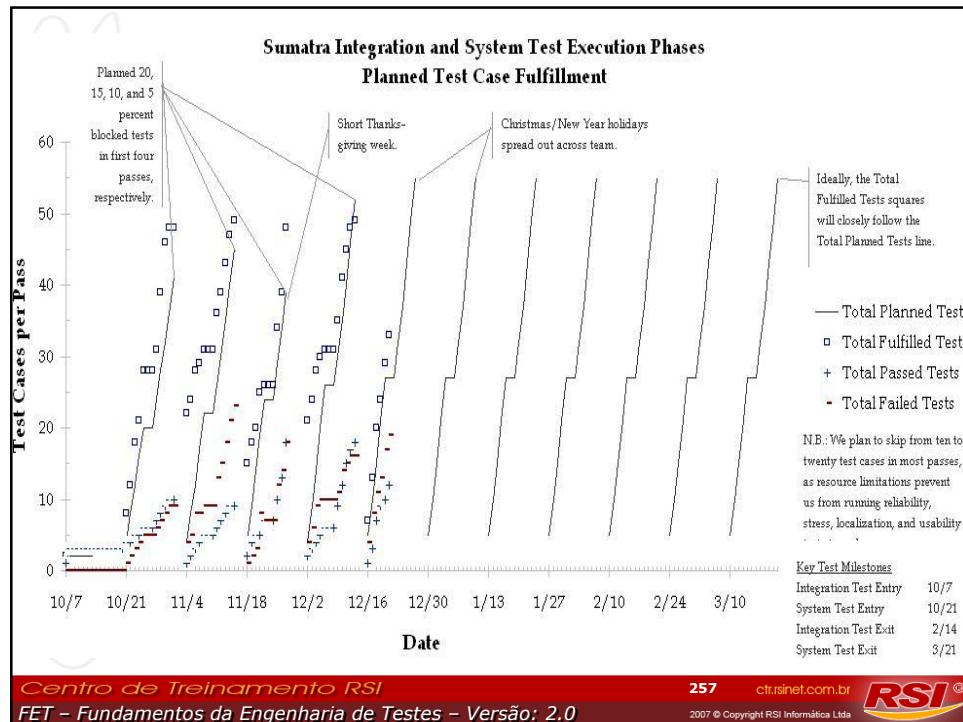
Relatório de Teste

- Resumir/analisar os resultados do teste.
 - Eventos chave (por exemplo, atingindo os critérios de saída).
 - Análise (para recomendações, guia) de...
 - ...defeitos remanescentes.
 - ...custo / benefício de mais testes.
 - ...riscos consideráveis.
 - ...nível de confiança.
- Métricas coletadas para avaliação:
 - Adequação dos objetivos de teste para nível do teste.
 - Adequação das abordagens de teste.
 - Efetividade do teste por objetivos.

Controle do Teste

- Guia e ações corretivas devido a informação e métricas do teste, as quais podem afetar as atividades de teste – ou outras atividades do ciclo de vida do software.
- Exemplos de ações de controle do teste:
 - Nova priorização do teste por gatilho/disparador de risco.
 - Ajustes na programação do teste devido a disponibilidade do ambiente de teste.
 - Colocar um critério de teste requerendo testar novamente as correções de *bugs* pelos desenvolvedores antes da integração em um *build*.





Relatório Resumido de Teste da IEEE 829

- Um relatório resumido de teste descreve os resultados de uma dada fase ou nível do teste, e inclui as seguintes seções:
 - Identificador do relatório de resumo do teste.
 - Resumo (isto é, o que foi testado, quais são as conclusões, etc.).
 - Variações (do plano, casos, procedimentos).
 - Avaliação compreensiva.
 - Resumo dos resultados (por exemplo, métrica final, contagem).
 - Avaliação (de cada item de teste face a face com critérios passa/falha).
 - Sumário das atividades (uso de recursos, eficiência, etc.).
 - Aprovações.
- Pode ser entregue dentro ou no final de um nível de teste.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	
DataRocket System Test Case Summary																								
1																								
2																								
3																								
4																								
5																								
6																								
7																								
8																								
9																								
10																								
11																								
12																								
13																								
14																								
15																								
16																								
17																								
18																								
19																								
20																								
21																								
22																								
23																								
24																								
25																								
26																								
27																								
28																								
29																								
30																								
31																								
32																								
33																								
34																								
35																								
36																								
37																								
38																								
39																								
40																								
41																								
42																								
43																								
44																								
45																								
46																								
47																								
48																								
Test Case Summary																								
Roll Up Columns																								
T S P F FW Q I B																								
8 0 8 0 0 0 0 0																								
1 0 0 1 0 # 0 0 0																								
1 0 1 0 # 0 0 0																								
1 0 1 0 # 0 0 0																								
1 0 1 0 # 0 0 0																								
1 0 1 0 # 0 0 0																								
1 0 1 0 # 0 0 0																								
1 0 1 0 # 0 0 0																								
1 0 1 0 # 0 0 0																								
1 0 1 0 # 0 0 0																								
1 0 1 0 # 0 0 0																								
1 0 1 0 # 0 0 0																								
1 0 1 0 # 0 0 0																								
1 0 1 0 # 0 0 0																								
1 0 1 0 # 0 0 0																								
1 0 1 0 # 0 0 0																								
1 0 1 0 # 0 0 0																								
1 0 1 0 # 0 0 0																								
1 0 1 0 # 0 0 0																								
1 0 1 0 # 0 0 0																								

DataRocket System Test Suite Summary															
Pass One															
Suite	Total Cases	Planned Tests Fulfilled					Planned Tests Unfulfilled				Earned Value				
	Count	Skip	Pass	Fail	Failure		Count	Queued	IP	Block	Plan Eff	Act Eff	% Effort	% Exec	
Environmental Test	8	8	0	8	0	0.00	0	0	0	0	22.00	22.00	100%	100%	
Load, Capacity and Volume	8	8	0	7	1	0.67	0	0	0	0	32.00	34.00	106%	100%	
Basic Functionality	12	12	0	12	0	0.00	0	0	0	0	18.00	17.00	94%	100%	
Standards	3	3	0	3	0	0.04	0	0	0	0	24.00	24.00	100%	100%	
Total	31	31	0	30	1	0.71	0	0	0	0	96.00	97.00	101%	100%	
By Percentage		100%	0%	97%	3%		0%	0%	0%	0%					

*Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0*

261 ctr.rsinet.com.br 

Registro de Teste da IEEE 829

- Um registro de teste grava os detalhes relevantes sobre a execução do teste, e inclui as seguintes seções:
 - Identificador do registro do teste.
 - Descrição (itens sob teste [com números de versão], ambiente de teste, etc.).
 - Entradas de atividades e eventos (lista teste por teste, evento por evento, informação sobre o processo de execução do teste, os resultados dos testes, mudanças ou questões ambientais, *bugs*, incidentes, ou anomalias observadas, testadores envolvidos, qualquer suspensão ou bloqueio do teste, mudanças no plano, impacto da mudança, etc.).
- Planilhas de rastreamento que capturam esses e muitos outros detalhes são frequentemente usadas para esse propósito.

Capítulo 5: Gerenciamento de Teste
Seção 5.3 – Monitoração e Controle do Progresso do Teste

- Isto Ajuda?



Capítulo 5: Gerenciamento de Teste
Seção 5.3 – Monitoração e Controle do Progresso do Teste

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 5: Gerenciamento de Teste

Seção 5.4 – Gerenciamento de Configuração

- Conceito chave:
 - Como o gerenciamento de configuração apóia o teste.

Teste e Gerenciamento de Configuração

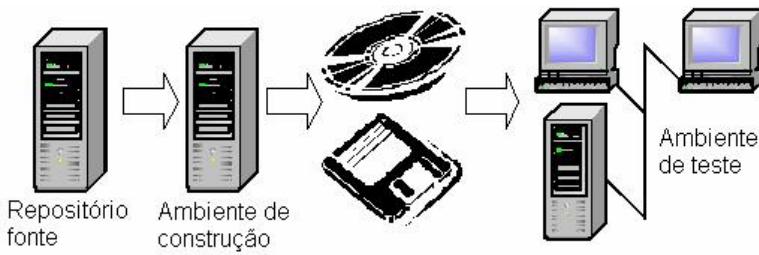
- O gerenciamento de configuração estabelece, mantém a integridade dos itens que constituem o software ou sistema através do projeto e ciclo de vida do produto.
- Para o teste o gerenciamento de configuração:
 - Permite o gerenciamento do *testware* e resultados.
 - Garante que cada item no teste pode ser relacionado a componentes conhecidos do sistema.
 - Suporta a entrega de uma versão de teste no laboratório de teste.
- Durante o planejamento do projeto e do teste, os procedimentos e infraestrutura (ferramentas) do gerenciamento de configuração devem ser escolhidos, documentados, e implementados, de forma que não ocorram surpresas na hora de execução do teste.

Tarefas Chave do Gerenciamento de Configuração

- Armazenar e controlar o acesso aos itens que constituem o sistema (também conhecidos como, controle de código fonte, apesar dele ir além do código).
- Identificar e documentar os itens sob controle.
- Permitir mudança nos itens controlados através de um processo organizado (por exemplo, comissões de controle de mudança).
- Relatar as mudanças pendentes, a caminho, e completas.
- Verificar a completude da implementação.

Teste e Gerenciamento da Versão

- Programação da versão (isto é, semanal? diário? horário?).
- Aplicar atualização (processo para instalar novo *build*).
- Retirar atualização (processo para remover um *build* ruim).
- Nomear o *build* (nível de revisão); ex. X.01.017.
- Interrogação (processo para determinar nível da revisão).
- Sincronizar com banco de dados, outros sistemas, etc.
- Papéis e responsabilidades para cada passo.



Relatório de Transmissão de Item de Teste da IEEE 829

- Um relatório de transmissão de item de teste descreve os itens sendo entregues para teste, e inclui as seguintes seções:
 - Identificador do relatório de transmissão do item de teste.
 - Itens transmitidos (inclui nomes do item e números de revisão).
 - Localização (onde, que mídia, rotulagem, etc.).
 - Status (*bugs* corrigidos, mudanças introduzidas, etc.).
 - Aprovações.
- Mais comumente usadas são as notas de versão, que incluem algumas dessas informações, geralmente informalmente.

Capítulo 5: Gerenciamento de Teste Seção 5.4 – Gerenciamento de Configuração

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 5: Gerenciamento de Teste

Seção 5.5 – Riscos e Teste

■ Conceitos chave:

- Perigos e riscos potenciais.
- Risco como um possível problema ameaçando a realização dos objetivos dos *stakeholders*.
- Projeto como oposto a riscos de produto.

Riscos de Projeto

- Um capítulo anterior introduziu a idéia do teste como uma forma de gerenciar riscos do produto ou de qualidade.
- O teste também é sujeito a riscos.
- Um risco é a possibilidade de um resultado negativo, e que poderia incluir eventos como versões de teste atrasados, problemas no ambiente, etc.
- Para descobrir riscos ao esforço de teste, pergunte a você mesmo e a outros *stakeholders*:
 - O que poderia dar errado no projeto que atrasaria ou invalidaria seu plano de teste e / ou estimativa?
 - Com que tipo de resultado inaceitável de teste você ficaria preocupado?

Riscos de Projeto e Riscos de Produto

- Riscos de projeto
 - Comprometem a capacidade de o projeto alcançar seus objetivos
- Riscos de produto
 - Áreas potenciais de falha no sistema são riscos para a qualidade do produto final

Tratando dos Riscos de Projeto

- Para cada risco de projeto, você tem quatro opções:
 - Minimização: Reduzir a probabilidade ou impacto através de passos preventivos.
 - Contingência: Tenha um plano pronto para reduzir o impacto.
 - Transferência: Obtenha algum outro partido para aceitar as consequências.
 - Ignore: Não faça nada sobre ele (melhor se ambos probabilidade e impacto são baixos!).
- Outra típica opção de gerenciamento de risco, compra de seguro, geralmente não está disponível.

Amostra de Riscos, Minimizações e Contingências

- Problemas de logística ou qualidade do produto bloqueiam os testes:
 - Planejamento cuidadoso, boa triagem de *bug*, projeto robusto de teste.
- Entregáveis de teste não instalam:
 - Teste de fumaça (*smoke test*), *builds* noturnos, processo para desinstalar.
- Mudanças excessivas invalidam resultados, requerem atualizações dos testes:
 - Bons processos de controle de mudanças, projeto robusto de teste, documentação limitada de teste, escalamento para gerência.
- Ambiente(s) de teste insuficiente ou irreal:
 - Explique os riscos para a gerência, terceirize teste de desempenho.
- Suporte ao ambiente de teste não confiável:
 - Bom processo de escalamento, habilidade de administração de sistema na equipe.
- Lacunas na cobertura do teste reveladas durante a execução do teste:
 - Teste exploratório, melhoria contínua do teste.
- Tropeços nas datas de inicio do teste e/ou entregas para testar:
 - Prioridade dos riscos para desistir de testes, escala para gerência.
- Cortes no orçamento e/ou recrutamento:
 - Prioridade dos riscos para desistir de testes, equipe bem sintonizada, terceirização de teste
- Depuração no ambiente de teste:
 - Escala para gerência, prioridade dos riscos para desistir de testes.

Outros Riscos de Projeto para Considerar

- Fatores organizacionais:
 - Carências de habilidades e pessoas.
 - Questões pessoais e de treinamento.
 - Problemas de comunicação.
 - Falta de acompanhamento nos resultados do teste, incluindo falha em usar esses resultados para melhoria do processo.
 - Atitude ou expectativas impróprias para o teste, ou por testadores ou por outros no projeto.
 - Organização complexa (por exemplo, distribuída).
- Questões do fornecedor:
 - Falha de um terceiro, incluindo um fornecedor de ferramenta de teste.
 - Questões contratuais.
- Questões técnicas:
 - Problemas em definir os requisitos corretos.
 - Requisitos inatingíveis (levando a testes bloqueados / não executáveis).
 - A qualidade do projeto, código, testes e dados de teste.
 - Sistema altamente complexo.

Capítulo 5: Gerenciamento de Teste
Seção 5.5 – Riscos e Teste

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 5: Gerenciamento de Teste

Seção 5.6 – Gerenciamento de Incidente

- Conceitos chave:
 - O conteúdo de um relatório de *bug* ou incidente.
 - Escrevendo um relatório de *bug* ou incidente.

Metas e Audiência

- Relatórios de incidentes ou *bug* têm as seguintes metas:
 - Prover informação detalhada sobre o incidente ou *bug* para todos os que precisam dela.
 - Ser parte dos dados agregados para analisar (ex., gráficos de *bug*).
 - Levar à melhoria dos processos de teste e de desenvolvimento.
- Leitores típicos dos relatórios de *bug* incluem:
 - Desenvolvedores: aqueles que devem corrigir o problema relatado.
 - Gerentes: aqueles que devem fazer alocação de recursos / priorização de decisões sobre o problema.
 - Pessoal de suporte técnico: aqueles que devem estar cientes de questões adiadas / não corrigidas ao tempo de entrega.
 - Testadores: aqueles que devem conhecer o estado corrente do sistema.

Dez Passos para um Bom Relatório de *Bug*

1. Estruturar: teste cuidadosamente.
2. Reproduzir: teste novamente.
3. Isolar: teste de forma diferente.
4. Generalizar: teste em outro lugar.
5. Comparar: revise resultados de teste similar.
6. Resumir: relate o teste aos clientes.
7. Condensar: corte informação desnecessária.
8. Tirar ambiguidade: use palavras claras.
9. Neutralizar: expresse o problema de maneira imparcial.
10. Revisar: tenha certeza.

Bons relatórios de *bug* sobre um problema podem diferir em estilo e conteúdo.

Passo 1: Estruturar

- Teste estruturado é a base dos bons relatórios de *bug*.
 - Use uma abordagem cuidadosa e ponderada para o teste.
 - Siga casos de teste escritos ou execute casos automatizados gerados por processos padronizados ou escritos.
 - Use gráficos para estruturar o teste exploratório.
 - Tome notas cuidadosas.
- O relato do *bug* começa quando os resultados esperados e observados diferem entre si.
- Um teste medíocre resulta em relatórios de *bug* medíocres.

Passo 2: Reproduzir

- Sempre cheque a reprodutibilidade da falha como parte da escrita do relatório de *bug*.
 - Três vezes é uma boa regra básica.
- Documente uma sequência complicada de ações que reproduzirá a falha.
- Relate falhas intermitentes, difíceis de repetir.
 - Anote a taxa de incidência de falha (por exemplo, 1 em 3 tentativas).
 - O resumo deve mencionar intermitência.
- Um conjunto claro de passos para reproduzir trata diretamente da questão “irreprodutível”.

Passo 3: Isolar

- Mude as variáveis que podem alterar o sintoma.
 - Faça mudanças uma a uma.
 - Requer raciocínio e entendimento do sistema sob teste.
 - Pode não ser imediatamente obvia.
- Pode ser extensiva.
 - Compatibilize a quantidade de esforço com a severidade do problema.
 - Evite entrar nas atividades de depuração.
- Bom isolamento mostra adequada diligência e dá aos desenvolvedores um ponto de partida para a depuração.

Passo 4: Generalizar

- Procure por falhas relacionadas no sistema sob teste.
 - A mesma falha ocorre em outros módulos ou locais?
 - Existem sintomas mais severos da mesma falha?
- Evite problemas confusos não relacionados.
 - O mesmo sintoma pode surgir de *bugs* diferentes.
- Generalizar reduz relatórios duplicados de *bugs* e refina o entendimento da falha.

Passo 5: Comparar

- Examine resultados de testes similares.
 - O mesmo teste executado para versões anteriores.
 - Condições similares, outros testes, mesma versão.
- A falha é uma regressão?
 - A mudança não introduz defeitos em versões anteriores.
 - Geralmente encontrada quando falham testes que já haviam passado.
- Nem sempre possível.
 - Teste bloqueado anteriormente, reinstalação impraticável.
 - Funcionalidade testada não disponível em versões anteriores.

Passo 6: Resumir

- Coloque uma pequena “linha de chamada” em cada relatório.
 - Capture a falha e o impacto no cliente.
 - Analogia: manchete de jornal.
 - Mais difícil do que parece.
 - Testadores devem investir tempo no resumo.
- ★ Vantagens de bons resumos:
- Obter atenção da gerência.
 - Ajuda na definição de prioridades.
 - Nomear relatório de erro para desenvolvedores.

O resumo é geralmente a única parte
do relatório de bug que é lida.

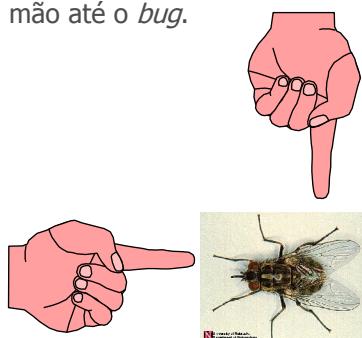
Passo 7: Condensar

- Elimine palavras ou passos estranhos.
 - Releia cuidadosamente o relatório.
 - Evite tanto comentários enigmáticos como monótonos.
- Existem detalhes ou ações irrelevantes?
- O tempo de cada um é precioso...
 - ...portanto não o desperdice com verbosidade desnecessária...
 - ...mas também não corte nada essencial.



Passo 8: Tirar Ambiguidade

- Remover, reescrever, ou expandir declarações e palavras vagas, equivocadas, ou subjetivas.
- Tenha certeza de que o relatório não está sujeito a má interpretação.
- Meta: declarações claras e inquestionáveis do fato.
 - Leve o desenvolvedor pela mão até o bug.



Passo 9: Neutralizar

- Entregue gentilmente as más notícias.
- Seja justo com as palavras e implicações.
- Evite:
 - Atacar desenvolvedores.
 - Criticar o erro básico.
 - Tentativa de humor.
 - Uso de sarcasmo.
 - Temperamento explosivo?
- Limite os relatórios de *bug* em declarações do fato.



△ Você nunca sabe quem irá ler os relatórios.

Passo 10: Revisar

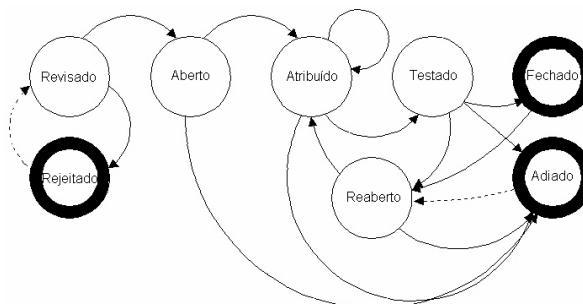
- Cada testador deveria submeter cada relatório de *bug* para um ou mais pares de teste para revisão.
- Pares de revisão deveriam:
 - Fazer sugestões para melhorar o relatório.
 - Perguntar questões esclarecedoras.
 - Até contestar a existência do *bug* se apropriado.
- A equipe de teste deveria somente submeter os melhores relatórios de *bug* possíveis, dadas restrições de tempo apropriadas à prioridade do *bug*.

Relatório de Incidente de Teste (*Bug*) da IEEE 829

- Um relatório de incidente de teste (ou *bug*) descreve um evento de teste que necessita de investigação adicional, particularmente um *bug*, e inclui as seguintes seções:
 - Identificador do relatório de incidente de teste.
 - Resumo (uma linha, especialmente de impacto nos *stakeholders*).
 - Descrição do incidente (entradas, resultados esperados, resultados reais, anomalias, data e tempo observado, ambiente, teste em progresso, reproduzibilidade, relatado por, e outros detalhes pertinentes).
 - Impacto (no teste, projeto, produto, *stakeholders*, etc.).
- Estritamente, relatórios de incidente descrevem qualquer comportamento questionável, enquanto que relatórios de *bug* descrevem comportamento conhecido devido a erros do que a maus testes ou dados de teste, erros de testadores, problemas no ambiente de teste, e similares.

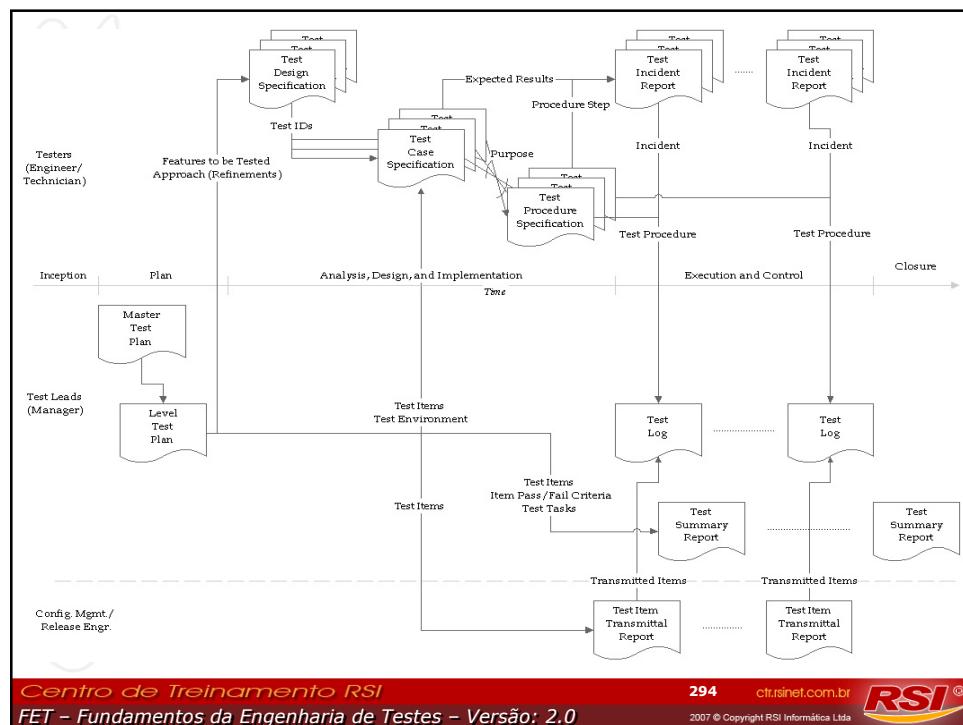
Ciclo de Vida ou *Workflow* do Relatório de *Bug*

- Relatórios de *bug* se movem através de uma série de estados (ciclo de vida ou *workflow*) até a resolução.
- Em cada estado não terminal, um gerente ou o comitê de triagem de *bugs* especifica um responsável que moverá o *bug* até o próximo estado.
- Sistemas de rastreamento de *bugs* podem e deveriam implementar e automatizar estes *workflows*, mas o apoio da equipe de projeto e da gerência é que fazem o *workflow* funcionar!



Outras Informações para Incluir

- Informação adicional sobre a configuração do software ou sistema.
- A fase de introdução, detecção, e remoção do *bug*.
- Urgência/prioridade para correção.
- Conclusões e recomendações.
- Riscos, custos, oportunidades, e benefícios associados com corrigir/não corrigir o *bug*.
- Histórico da mudança, especialmente para cada mudança de estado.
- Data do relato, reporte de organização, e autor.
- Resultados esperados e obtidos.
- Identificação do item de teste e do ambiente.
- Processo do ciclo de vida no qual o incidente foi observado.
- Escopo ou grau de impacto nos interesses dos *stakeholders*.
- Severidade do impacto no sistema.



Capítulo 5: Gerenciamento de Teste
Seção 5.6 – Gerenciamento de Incidente

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 5: Gerenciamento de Teste

Exercícios de Fixação

- Faça os exercícios no formato do exame para o capítulo 5.
- Discuta.

Capítulo 6: Ferramentas de Suporte a Teste

Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0

297 ctr.rsinet.com.br RSI®
2007 © Copyright RSI Informática Ltda

Capítulo 6: Ferramentas de Suporte a Teste

Seções:

1. Tipos de Ferramentas de Teste
2. Uso Efetivo das Ferramentas: Riscos e Benefícios em Potencial
3. Implementando uma Ferramenta na Organização

Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0

298 ctr.rsinet.com.br RSI®
2007 © Copyright RSI Informática Ltda

Capítulo 6: Ferramentas de Suporte a Teste

Seção 6.1 – Tipos de Ferramentas de Teste

- Conceitos chave:
 - Diferentes tipos de ferramentas de teste.
 - Ferramentas de teste para programadores.

Classificação das Ferramentas de Teste

- Ferramentas podem ser classificadas pelas atividades primárias de teste que elas apóiam.
- Para muitas classificações de ferramentas, existem opções disponíveis tanto comerciais como *freeware*.
- Ferramentas automatizam tarefas repetitivas, tornam o teste mais eficiente.
- Ferramentas melhoram a confiabilidade do teste; por exemplo, automatizando grandes comparações de dados, gerando carga.
- Ferramentas são invasivas ou não-invasivas:
 - Ferramentas invasivas têm efeitos investigativos.
 - Ferramentas não-invasivas são normalmente mais caras.
- Algumas ferramentas são primariamente para desenvolvedores.

Ferramentas de Gerenciamento de Teste

- Fornece rastreabilidade dos testes, resultados dos testes e incidentes para a base de teste.
- Permite o registro de resultados de teste, e geração de relatórios de progresso.
- Auxilia a gerenciar os testes e o processo de teste.
- Interface para a execução do teste, rastreamento de *bugs*, e ferramentas de gerenciamento de requisitos.
- Fornece controle de versão ou interface com uma ferramenta externa de gerenciamento de configuração.
- Executa análise quantitativa (métricas) relacionada aos testes.

Ferramentas de Gerenciamento de Requisitos

- Armazena declarações de requisitos.
- Verifica consistência e requisitos indefinidos (faltantes).
- Permite que os requisitos sejam priorizados.
- Permite que testes individuais sejam rastreáveis para (e em termos de) requisitos, funções, e/ou funcionalidades.

Ferramentas de Rastreamento de *Bug* (ou Defeito ou Incidente)

- Armazena e gerencia relatórios de *bug*.
- Facilita a priorização / classificação do *bug*.
- Provê *workflow* baseado em estados, incluindo atribuição de ações para pessoas (por exemplo, correção, teste de confirmação, etc.).
- Permite monitoramento dos *bugs* de um projeto e status do *bug* ao longo do tempo.
- Provê apoio para análise estatística (geralmente através de exportação; por exemplo, para o Excel).
- Cria relatórios e gráficos (apesar de geralmente ser de utilidade limitada devido à variação nas necessidades).

Ferramentas de Gerenciamento de Configuração

- Armazena informação sobre versões e *builds* do software e *testware*.
- Permite rastreabilidade entre *testware* e produtos do trabalho de software e variantes do produto.
- Ajuda no desenvolvimento e no teste em múltiplas configurações de ambientes de hardware/software.

Ferramentas de Apoio ao Processo de Revisão

- Armazena informação sobre processos de revisão.
- Armazena e comunica comentários de revisão.
- Relato de defeito e esforço.
- Gerencia referências para regras de revisão e/ou lista de checagem.
- Provê ajuda para revisões *online*.
- Viabiliza rastreabilidade entre documentos e código fonte.

Ferramentas de Análise Estática

- Primariamente usadas por desenvolvedores.
- Encontra defeitos antes do teste dinâmico.
- Força uso de padrões de codificação.
- Analisa estruturas e dependências.
- Ajuda no entendimento do código fonte.
- Calcula métricas a partir do código.

Ferramentas de Modelagem e Projeto

- Primariamente usadas por desenvolvedores.
- Ajuda a criar modelos do sistema.
- Valida modelos.
- Ajuda na geração de alguns casos de teste baseados no modelo.

Ferramentas de Projeto de Teste

- Gera entradas de teste ou testes reais a partir de:
 - Requisitos.
 - Interface Gráfica com Usuário.
 - Projeto de modelos.
 - Código.
- Gera resultados esperados (apesar de que a confiabilidade desse tipo de oráculos de teste ser frequentemente limitada).
- Gera *frameworks* de teste, modelos, e *stubs*.

Ferramentas de Dados de Teste

- Manipula ou cria banco de dados, arquivos ou dados para uso durante a execução do teste.
- Cria grandes volumes de dados de teste úteis.
- Valida dados de teste de acordo com regras específicas.
- Analisa os dados para frequência de condições, etc.
- Mistura ou torna anônimos dados reais ou de clientes.

Ferramentas de Execução de Teste

- Executa testes (isto é, submete entradas por um *script* automatizado e compara resultados obtidos e esperados).
 - Usado apropriadamente, o *script* é um procedimento de teste independente da entrada que permite a você repetir os testes com dados diferentes, executar testes similares, e facilmente manter os testes.
 - Usados inapropriadamente – por exemplo, típica *captura-playback* – os *scripts* são misturados com os dados e/ou resultados esperados, possuem atrasos codificados diretamente neles, e são muito frágeis.
 - *Captura-playback* pode ser útil durante o teste exploratório.
- Inclui comparadores.
- Produz registros analisáveis.
- Trabalha na GUI, API, ou CLI.

Frameworks de Teste, Ambiente preparado, Simuladores

- Trocar ou substituir peças de hardware faltantes e/ou potencialmente problemáticas.
- Facilitar o teste de unidade(s) gerando e / ou suportando *drivers*, *stubs*, e/ou objetos simulados que substituem porções do sistema, que são indisponíveis ou removidos para isolar a unidade.
- Prover *frameworks* de execução no *middleware* para testar linguagens, sistemas operacionais ou hardware.

Comparadores de Teste

- Checa arquivos, banco de dados, ou resultados do teste mediante expectativas ou durante a execução do teste ou subsequentemente.
- Pode incluir um oráculo de teste automatizado ao invés de comparar mediante linhas de base estáticas.
- Pode ser tornado esperto através da sua programação para tratar variáveis dinâmicas como datas e tempo, ou ignorar a ordem dos registros quando a ordem de seleção em relatórios ou consultas é ambígua.

Ferramentas de Medida de Cobertura

- Primariamente usadas por desenvolvedores.
- Tanto invasiva como não-invasiva.
- Mede a porcentagem de tipos específicos de estrutura de código que foram exercitados:
 - Comandos.
 - Desvios ou decisões.
 - Objetos.
 - Chamadas de função.
- Checa quanto fundo um conjunto de testes executou o tipo de estrutura medido.

Ferramentas de Segurança

- Checa por vírus de computador.
- Simula várias formas de ataques.
- Simula várias condições de segurança.
- Checa o código para procurar violações de segurança.

Desempenho, Monitoramento, Análise Dinâmica

- Primariamente usadas por desenvolvedores, apesar de muitos testadores usarem ferramentas de desempenho e monitoramento.
- Ferramentas de análise dinâmica frequentemente são usadas para checar por dependências de tempo ou não liberação de memória.
- Monitora e relata como um sistema se comporta sob condições de uso simuladas.
- Gera várias (esperançosamente realistas) condições de carga para a aplicação, um banco de dados, rede ou servidor, geralmente por algum *script* ou procedimento programado.
- Monitora, analisa, verifica e relata o uso de recursos específicos do sistema, e fornece avisos de problemas possíveis.

Outras Ferramentas

- Algumas ferramentas são focadas em aplicações particulares.
 - Ferramentas de teste de desempenho baseada na *Web*.
 - Ferramentas de análise estática específicas da linguagem.
 - Ferramentas de teste de segurança.
- Algumas áreas alvo específicas da aplicação como sistemas embarcados.
- É claro, testadores também usam planilhas (muitas delas!) e banco de dados.
- Muitos desenvolvedores usam ferramentas de depuração.

Capítulo 6: Ferramentas de Suporte a Teste
Seção 6.1 – Tipos de Ferramentas de Teste

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 6: Ferramentas de Suporte a Teste

Seção 6.2 – Uso Efetivo das Ferramentas: Riscos e Benefícios em Potencial

- Conceitos chave:
 - Diferentes técnicas de escrever *scripts* para ferramentas de execução de teste, incluindo dirigida por dados e dirigida por palavra chave.
 - Benefícios e riscos potenciais da automatização do teste.

Oportunidades de Automatização e Riscos

Oportunidades

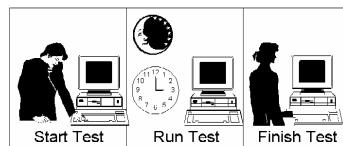
- Reduz o trabalho repetitivo.
- Consistência e capacidade de repetição melhoradas.
- Avaliação objetiva (especialmente cobertura, desempenho, etc.).
- Relatos simplificados.

Riscos

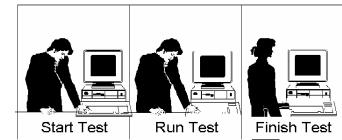
- Expectativas não realistas.
- Subestimativa do tempo, custo e esforço do desenvolvimento, execução, e manutenção.
- Uso da ferramenta para testes inadequados.

Teoria e Prática do Teste Automatizado

- Automatização é tipicamente desenvolvimento de software.
 - Um processo não trivial requerendo tempo, habilidade, e dinheiro significantes.
 - ROI da automatização tipicamente leva mais tempo do que um projeto.
 - Exceção: testes simples de unidade, componente, integração baseado em API.
 - Exceção: alguns testes não funcionais.
- Usualmente, o que é automatizado é apenas a execução do teste, não a análise dos resultados.
 - A automatização pode complicar a análise (e aumentar o número) de testes que falharam.



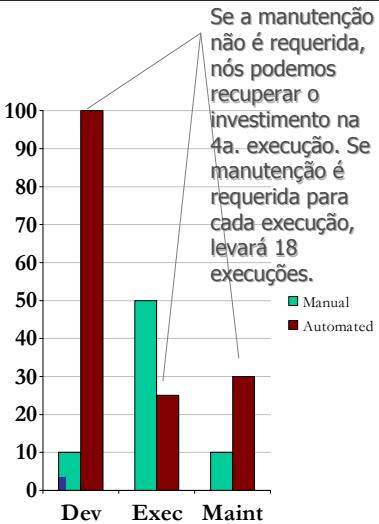
Se você foca apenas na execução do teste, automatização do teste parece muito eficiente, mas não esqueça que os testes automatizados tiveram que ser automatizados por alguém para serem automatizados



Automatização inteligente de tarefas de teste apropriadas é uma boa idéia.

ROI da Automatização de Teste

- Um retorno da maioria dos esforços de automatização de teste requer:
 - Risco de regressão suficientemente altos para justificar a repetição de testes.
 - Um projeto de teste de sistema que mantenha os custos de executar e manter teste automatizado bem abaixo do custo de teste manual equivalente.
 - Um sistema relativamente estável.
- Necessita recuperar altos custos de desenvolvimento de teste por muitos projetos.



Escolhendo Teste Manual ou Automatizado

- Testes bastante apropriados para teste manual:
 - Operações e manutenção.
 - Configuração e compatibilidade.
 - Tratamento e recuperação de erro.
 - Localização.
 - Usabilidade.
 - Instalação e configuração inicial.
 - Documentação e ajuda.
- Manual, automatizado, ou combinado:
 - Funcional.
 - Casos de uso (cenários do usuário).
 - Interface do usuário.
 - Tratamento de data e horário.
- Testes bastante apropriados para teste automatizado:
 - Regressão e confirmação.
 - Macaco (*monkey test*) ou aleatório.
 - Carga, volume, e capacidade.
 - Desempenho e confiabilidade.
 - Concordância com padrões.
 - caixa branca, particularmente unidades, componentes e integração baseados em API.
 - Complexidade estática e análise de código.

Teste manual não apropriado pode enganar as pessoas a respeito da cobertura de teste. Automatização não apropriada normalmente falha.
Alguns testes misturam as técnicas.

Dicas para Sucesso com Ferramentas de Execução de Teste

- *Captura-replay* parece diminuir custos de desenvolvimento, mas os custos de execução e manutenção para um grande número de testes são altos.
- Testes dirigidos a dados separam entradas (os dados) dos procedimentos (*scripts*) que usam os dados.
 - Especialistas (não em automatização) podem criar dados de teste
- Testes dirigidos a palavra chave usam palavras chave (ou palavras de ação) em um arquivo lido pelo *script*
 - Especialistas (não em automatização) podem criar arquivos de palavras chave.
- Especialistas em automatização criam os *scripts*.

Dicas para Sucesso com Outras Ferramentas

- Ferramentas de teste de desempenho requerem especialização em desempenho para projetar os testes e interpretar os resultados.
- Ferramentas de análise estática aplicadas ao código fonte pode forçar padrões de codificação, mas uma apresentação em fases para o código existente é necessária para gerenciar o volume de erros encontrados.
- Ferramentas de gerenciamento de testes frequentemente têm poucas facilidades de preparar relatórios, então exportar dados para uma planilha é usualmente a melhor forma de produzir os relatórios desejados.

Capítulo 6: Ferramentas de Suporte a Teste
Seção 6.2 – Uso Efetivo das Ferramentas: Riscos e Benefícios em Potencial

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

*Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0*

325 ctr.rsinet.com.br 
2007 © Copyright RSI Informática Ltda

Capítulo 6: Ferramentas de Suporte a Teste

Seção 6.3 – Implementando uma Ferramenta na Organização

- Conceitos chave:
 - Introduzindo uma ferramenta em uma organização.
 - Os objetivos de uma prova de conceito para avaliação da ferramenta.
 - Fatores requeridos para bom suporte da ferramenta.

*Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0*

326 ctr.rsinet.com.br 
2007 © Copyright RSI Informática Ltda

Derrubando Barreiras para Automatização

- Os seguintes problemas deveriam ser retificados antes de iniciar um esforço de automatização:
 - Processo de teste caótico, reativo, lento... primeiro tenha o processo manual de teste sob controle, então automatize.
 - Pessoas insuficientemente habilitadas... contrate ou treine.
 - Sistema instável, mudando rápido... estabilize o sistema primeiro – especialmente nas interfaces testadas como GUI e APIs.
 - Expectativas não realistas da gerência... comunique efetivamente os benefícios da automatização, limitações, casos de negócios (ROI), e custos.
 - Nenhuma ferramenta apropriada ou oráculo prático disponível para os importantes riscos de qualidade... construa ferramenta ou oráculo próprio – ou espere.

Quando esses problemas permanecem, a automatização usualmente falha, criando resistência organizacional para subsequentes projetos de automatização.

Objetivos de um Projeto Piloto de Automatização

- Aprender mais sobre a ferramenta e como usá-la.
- Adaptar a ferramenta e/ou processos e práticas para se moldar à organização e sistemas.
- Padronizar formas de uso, gerência, armazenagem e manutenção da ferramenta e dos itens de teste.
- Avalie o retorno sobre o investimento.

Fatores de Sucesso para Implantação da Ferramenta

- Implante a ferramenta no resto da organização de forma incremental.
- Adapte e melhore os processos de teste para se moldarem ao uso da ferramenta.
- Forneça treinamento, acompanhamento, e mentoramento para novos usuários.
- Defina guias de uso da ferramenta.
- Aprenda formas de melhorar continuamente o uso da ferramenta.
- Monitore o uso e benefícios da ferramenta.

Armadilhas de Craig para Ferramentas de Teste

- Sem estratégia clara.
- Expectativas não realistas.
- Falta de um *stakeholder*.
- Ferramenta de treinamento inadequada ou de baixa qualidade.
- Automatização das coisas erradas (por exemplo, testes errados).
- Escolha de ferramenta errada.
- Problemas de usabilidade com a ferramenta.
- Escolha de fornecedor errado.
- Sistema sob teste instável.
- Fazer demais, muito rápido.
- Subestimar as necessidades de tempo e de recursos.
- Ambiente de teste inadequado ou único.
- Abordagem correta, temporização errada.
- Custo excessivo.

Fonte: Rick Craig and Stefan Jaskiel's Systematic Software Testing, capítulo 6

Capítulo 6: Ferramentas de Suporte a Teste
Seção 6.3 – Implementando uma Ferramenta na Organização

Exercícios de Fixação

- Faça os exercícios no formato do exame para esta seção.
- Discuta.

Capítulo 6: Ferramentas de Suporte a Teste

Exercícios de Fixação

- Faça os exercícios no formato do exame para o capítulo 6.
- Discuta.

Referências

Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0

333 ctr.rsinet.com.br
2007 © Copyright RSI Informática Ltda 

Referências

- ISTQB, *Foundation Level Syllabus*, versão 2007br.
- RBCS (Rex Black Consulting Services Inc.): *Test Engineering Foundation, Essential Knowledge for Test Professionals*. Curso de treinamento preparatório para certificação CTFL ISTQB.
- Spillner, Andreas; Linz, Tilo; Schaefer, Hans: *Software Testing Foundations*, 2^a. ed. Rockynook, 2005.
- Kaner, Cem; Falk, Jack; Nguyen, Hung Quoc: *Testing Computer Software*, 2^a. ed. Wiley, 1999.
- Pressman, Roger S.: *Engenharia de Software*, 6^a. ed. McGraw Hill, 2005.

Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0

334 ctr.rsinet.com.br
2007 © Copyright RSI Informática Ltda 

Referências – Livros Referenciados no Syllabus

- Beizer, Boris: *Software Testing Techniques*, 2^a. ed, Van Nostrand Reinhold, 1990.
- Black, Rex: *Managing the Testing Process*, 2^a. ed , John Wiley & Sons, 2001.
- Buwalda, Hans et al.: *Integrated Test Design and Automation*, Addison Wesley, 2001.
- Chrissis, Mary-Beth et al.: *CMMI: Guidelines for Process Integration and Product Improvement*, Addison Wesley, 2004.
- Copeland, Lee: *A Practitioner's Guide to Software Test Design*, Artech House, 2004.
- Fewster, Mark and Graham, Dorothy: *Software Test Automation*, Addison Wesley, 1999.
- Hetzel, Bill: *Complete Guide to Software Testing*, QED, 1998.
- Kaner, Cem at al.: *Lessons Learned in Software Testing*, Wiley, 2002.
- Myers, Glenford: *The Art of Software Testing*, John Wiley & Sons, 1979.
- van Veenendaal, Erik, ed.: *The Testing Practitioner*, UTN Publishers, 2004.

Referências – Normas Referenciadas no Syllabus

- Glossário ISTQB para termos de teste, versão 1.3 br.
- CMMI: *Guidelines for Process Integration and Product Improvement*.
- Norma IEEE 829-1998 – Norma IEEE para Documentação de Teste de Software.
- Norma IEEE 1028-1997 – Norma IEEE para Revisão de Software.
- Norma ISO/IEC 9126-1:2001. Engenharia de Software – Qualidade de Produto de Software.
- Norma IEEE 12207 / ISO/IEC 12207-1996. Processos de Ciclo de Vida de Software.

Simulado



Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0

337 ctr.rsinet.com.br  2007 © Copyright RSI Informática Ltda

Avaliação do Treinamento

Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0

338 ctr.rsinet.com.br  2007 © Copyright RSI Informática Ltda



CTR
Centro de Treinamento RSI

Obrigado.

www.rsinet.com.br
(11) 3721-5292

ctr.rsinet.com.br
FET – Fundamentos da Engenharia de Testes – Versão: 2.0

2007 © Copyright RSI Informática Ltda



Advanced Level Syllabus ISTQB

Centro de Treinamento RSI
FET – Fundamentos da Engenharia de Testes – Versão: 2.0

340 ctr.rsinet.com.br

2007 © Copyright RSI Informática Ltda



Advanced Level Syllabus ISTQB

- O ***Advanced Level Syllabus*** ISTQB foi publicado no final de 2007.
- Contempla os seguintes capítulos:
 1. Aspectos Básicos do Teste de Software
 2. Processos do Teste
 3. Gerenciamento de Teste
 4. Técnicas de Teste
 5. Testes de Características de Software
 6. Revisões
 7. Gerenciamento de Incidentes
 8. Normas e a Melhoria do Processo de Teste
 9. Ferramentas de Teste e Automatização
 10. Habilidades de Pessoas – Composição de Equipe
- Mais informações www.bstqb.org.br/