# Plano de teste
# Test plan

# Visão geral

# O plano de teste

- O plano de teste é um processo executado para encontrar erros e comportamentos inesperados em aplicações (BUGS).
- Consiste em uma sequência de testes; cada teste é uma sequência de passos a serem executados e os resultados esperados.
- É importante que o Testador siga os passos fielmente, e com atenção.
- O plano de testes também pode conter erros, tais como numeração fora de ordem ou comandos sem sentido (inconsistentes ou impossíveis de executar).
- Se o Testador encontrar algo que possa ser um erro, o Engenheiro de Teste precisa ser informado de forma a decidir o que fazer.
- Os planos de teste são divididos em seções para ajudar na identificação e registro dos erros (bugs) no final do processo.

# O processo de teste

- Durante a execução dos testes, o Testador precisa prestar atenção e seguir as instruções contidas no plano de testes. Ao mesmo tempo, precisa manter a mente aberta para possíveis novos casos de teste, e conversar com o Engenheiro de Teste para identificar a validade do nova caso de teste.
- O Testador deve relatar com precisão e em detalhes qualquer comportamento inesperado observado na aplicação durante os testes, registrando (escrevendo) a Seção, o Número do teste, e o Passo onde o bug foi identificado.

# O relatório de teste

- Ao concluir os testes, e com a identificação dos bugs em mãos, o Testador deve escrever o Relatório de Teste. O formato do relatório de teste é fornecido pelo Engenheiro de teste.
- O relatório precisa ser claro e simples de ler, de forma que todos consigam entender os problemas da aplicação sob teste. Deve ser informado a Seção, o Número do teste, e o Passo onde o bug foi identificado.
- O Relatório de Teste incluiu também o Relatório de Problemas do Plano de Teste, onde o Testador deve descrever os problemas encontrados no plano de teste, fornecendo a , where the tester will describe any test plan problem, giving the Seção, o Número do teste, e o Passo onde o problema foi identificado.

# Conteúdo de planos de teste

# Testes de verificação de ambiente

- Ambientes de rede são instáveis por natureza. Problemas em cabos placas, configuração de servidores etc. podem levar a comportamentos anômalos. Por esta razão, ao preparar testes para sistemas que executam em ambientes desse tipo é importante acrescentar alguns testes que visem verificar a integridade do ambiente.
- Realizar testes de conectividade entre os nodos da rede, efetuando pings entre eles e entre o gateway e cada um dos nodos (se for o caso).
- Testar a conexão via browser quando houver servidor web, verificando também o proxy (se for o caso).

# Testes de instalação

- Verificar  ambiente: testar o comportamento do software quando o ambiente não está completo tanto no que se refere a hardware ou software.
- Verificação de arquivos de instalação: testar o comportamento do software de instalação na ausência de algum arquivo de instalação ou no caso de algum deles estar corrompido.
- Verificar direitos de instalação: testar o comportamento do software no caso de tentativa de instalação por usuário não autorizado ou sem a devida licença.
- Verificar o fluxo do processo de instalação: verificar se o usuário consegue navegar pelas telas de instalação, desistir a qualquer momento, retroceder algum passo para alteração de opções etc.

# Testes de instalação (cont.)

- Verificar se a instalação detecta componentes previamente instalados: verificar se a instalação avisa sobre versões anteriores previamente instaladas ou sobre bibliotecas ou arquivos compartilhados pré-existentes em versões mais antigas ou mais novas ou em línguas diferentes.
- Verificar se a instalação foi executada corretamente: verificar se todos os arquivos foram copiados para os diretórios corretos, se as entradas de registro foram corretamente criadas, se os atalhos e entradas de menu foram corretamente criados
- Verificar o procedimento de desinstalação: verificar quais os procedimentos para a desinstalação e se os mesmos realmente removem a ferramenta da máquina limpando/eliminando diretórios, arquivos de DLLs no diretório do sistema, arquivos de configuração e entradas de registro.

# Testes de instalação (cont.)

- Verificar os procedimentos de atualização e reinstalação: verificar o comportamento dos procedimentos de instalação no caso de instalação “por cima” da mesma versão, instalação de nova versão “por cima” ou a partir de desinstalação, instalação de versão mais antiga sobre versão mais nova (regressão).
- Realizar testes de instalação com exceções, ou seja, espaço insuficiente em disco, restrição de acesso a arquivos/diretórios, arquivos/diretórios protegidos, diretórios inválidos/inexistentes, etc.

# Testes de funcionalidade

- Verificação de todos os “pontos de entrada de dados” do programa: verificação de todos os possíveis pontos de entrada de dados tais como campos de janelas, arquivos de dados, arquivos de configuração, parâmetros de linha de comando (neste caso testar apenas os valores, deixar os teste de sintaxe para os testes de interface) etc.
  - Identificar as classes de equivalência. Para cada classe de equivalência gerar um caso de teste
  - Gere casos de teste para os valores limites de cada classe de equivalência

# Testes de funcionalidade (cont.)

- Identificar as combinações de dados de entrada: Identifique os conjuntos de entradas de dados que são levados em conta em conjunto e crie diferentes combinações de casos de teste
  - Gere casos de teste que forcem as saídas para seus valores limites
  - Crie casos de teste para forçar diferentes combinações de resultados

# Testes de funcionalidade (cont.)

- Verifique as saídas do programa:
  - Verifique se o conteúdo das saídas esta correto e coerente com as entradas
  - Verifique se o formato das saídas esta correto e coerente com as entradas
  - Verifique se os resultados são gerados na seqüência e tempo corretos
  - Verifique se os resultados estão completos
    - Verifique se não estão sendo gerados resultados de menos
    - Verifique se não estão sendo gerados resultados a mais

# Testes de funcionalidade (cont.)

- Nos casos de saída em arquivo, criar casos para verificar:
  - Se estão sendo gerados os arquivos corretos
  - Se os arquivos estão sendo gerados nos lugares corretos
  - Se os arquivos estão sendo gerados com o conteúdo correto
  - Se o conteúdo dos arquivos não é apenas correto mas coerente com as solicitações do usuário;
  - Se é possível gravar dados no disco/mídia;
  - O que acontece se não há espaço suficiente para a geração dos dados no disco/mídia;
  - O que acontece se há espaço suficiente apenas para gerar uma parte do arquivo no disco/mídia;
  - Se o sistema suporta nomes de arquivo longos fora do padrão DOS.
  - Verifique a localização dos arquivos de saída e se os dados estão sendo gravados nos arquivos corretos

# Testes de interface

- Verificar se as entradas corretas são aceitas
  - Verificar se são aceitos nomes longos como parâmetro.
- Verificar se as entradas incorretas são rejeitadas
  - Verifique se é feita consistência de valores nos campos para os quais existem limites especificados.
  - Verifique se os campos read-only estão corretamente configurados
  - Verifique se os campos de entrada de dados não “estouram”
  - Verifique os casos de parâmetros omitidos ou em excesso

# Testes de interface (cont.)

- Verificar a aparência e o sequenciamento das telas
  - Verificar a aparência das telas: verificar se a aparência das telas confere com a especificação. Se a localização e a quantidade de componentes está correta e se o texto das mensagens confere
  - Verificar se o sequenciamento das telas está correto: verificar se ao navegar pelas telas o usuário encontra a sequência especificada. Desenhe um diagrama de estados dos menus e telas e crie testes para verificar a cobertura do mesmo.
  - Verifique se existem teclas ou recursos de “escape” disponíveis em todas as situações.

# Testes de interface (cont.)

- Para os casos de interface gráfica:
  - Verifique se conjuntos de “check box” ou “ radio button” que devem ser mutuamente exclusivos estão funcionando corretamente.
  - Verifique se as “scroll bars” funcionam corretamente principalmente nos extremos
  - Verifique se o tamanho das listas e campos de edição/exibição de textos ou imagens esta adequado e se não “estoura” próximo aos limites
  - Verifique se os strings estão corretos e se os componentes (especialmente botões) não estão sobrepondo-os.
- Para os casos de interface por comandos com sintaxe própria:
  - Crie casos de teste baseados na BNF ou na especificação disponível da sintaxe

# Testes de help/documentação

- Verifique o conteúdo das telas de help contra as passagens correspondentes da especificação

- Se existir sistema de “help” sensível ao contexto, verificar se as telas corretas estão sendo ativadas conforme os diferentes contextos.

# Testes de stress

- Verificar as condições de “stress” ou de limite do software (quando aplicável) e gerar casos de teste sob estas condições.
  - Nos casos de rotinas cíclicas, forçar períodos mínimos de intervalo de forma a forçar erro por repetição exaustiva.

# Erros de teste clássicos

Classical Testing Errors

Brian Marick, Testing Foundations

# Introdução

- Classic mistakes cluster usefully into five groups, which I've called "themes":
  - **No propósito da atividade de teste.** The Role of Testing: who does the testing team serve, and how does it do that?
  - **No planejamento dos testes.** Planning the Testing Effort: how should the whole team's work be organized?
  - **No pessoal contratado para testar.** Personnel Issues: who should test?
  - **Na execução dos testes.** The Tester at Work: designing, writing, and maintaining individual tests.
  - **Na aplicação da tecnologia aos testes.** Technology Rampant: quick technological fixes for hard problems.

# O propósito de testar: Erros comuns

- **Atribuir a responsabilidade pela qualidade unicamente à equipe de teste**
  A responsabilidade de qualidade é de todos os envolvidos no projeto em todas as etapas; não é motivante culpar a equipe de teste pelo resultado de se auditar a qualidade do produto desenvolvido.

- **Achar que a tarefa de equipe de testes é simplesmente encontrar erros.**
  A tarefa mais importante da equipe de testes é encontrar os bugs *importantes*; o comum, no entanto, é focar na quantidade de bugs e não na relevância dos erros implicar em riscos importantes para o projeto.

- **Não encontrar os erros importantes.**
  Fundamentalmente importantes são os erros que trazem risco significativo à viabilidade do projeto com relação ao usuário final — seja do ponto de vista de execução do sistema em si, seja do ponto de vista da usabilidade e aplicabilidade do sistema ao problema do cliente.

- **Não informar sobre erros de usabilidade.**
  Se o teste de usabilidade não informa sobre a qualidade das interfaces do sistema, e existirem problemas de usabilidade, o produto será visto como ruim pelo cliente, não importando se é um problema formalizado pelo teste aplicado ou não.

# O propósito de testar: Erros comuns

- **Não focar em estimar a qualidade.**
  Um objetivo importante de fazer teste é estimar a qualidade do produto sendo criado (e estimar a qualidade desta estimativa). A técnica mais efetiva para avaliar a qualidade geral é fazer uma análise 'em largura' do produto completo.

- **Oferecer estatísticas de erros sem o contexto relevante**
  A estatística associada ao erro tende a ser analisada de forma muito otimista se não são oferecidos dados adicionais referentes ao volume de testes executados e à experiência anteriormente obtida; estes dados podem dar um parecer mais realista da situação na qual o projeto se encontra.

- **Começar a testar tarde demais**
  Se os testes são preparados e executados em todas as fases do projeto — começando antes da implementação em si — a chance de se encontrar erros de especificação em uma etapa inicial é muito maior.

# Planejamento dos testes: Erros comuns

- **Concentrar exageradamente em teste funcional**
  O teste funcional não testa a interação entre as diferentes funções. Produtos de software são orientados a tarefas, e como estas envolvem uma série de funções em conjunto, o teste deve verificar como estas funções diferentes interagem.

- **Não enfatizar o teste de configuração**
  Testar a interação de diversos tipos de software e hardware com o produto garante que este software realmente funcionará em qualquer ambiente; isto tem relevância muito maior para produtos de software comerciais com aplicação geral.

- **Deixar o teste de carga para o final do processo**
  Escalabilidade é uma questão de alto impacto em um software; deixar o teste para o final implica em sobrar pouco tempo para consertar problemas que possam surgir.

- **Não testar a documentação** e **não testar a instalação**
  Má documentação e instalações imperfeitas são ótimas formas de se fazer uma má impressão inicial do produto, e são problemas que atenção direcionada pode resolver de forma relativamente simples.

- **Confiar exageradamente no teste beta**
  Os usuários de um beta teste não são os usuários mais comuns. Em geral não usam o produto a fundo, e não informam sobre problemas de usabilidade em geral. Muitas vezes, não reportam os bugs que encontram, e os bugs reportados tem um custo alto de processamento e compreensão.

# Planejamento dos testes: Erros comuns

- **Executar os testes de forma estritamente consecutiva**
  Esperar um teste terminar por completo para iniciar outro retarda a descoberta de erros; idealmente, é melhor saber um pouco sobre todos os componentes de um sistema do que saber muito sobre os poucos que foram testados até agora.
- **Não identificar áreas de risco**
  O objetivo principal do teste é, mais especificamente, identificar o risco que problemas desconhecidos podem trazer. Utilizar dados históricos e analisar opiniões de uma variedade boa de usuários oferece uma base melhor para avaliar estes riscos.
- **Teimosia em aplicar um plano de teste ineficaz**
  O problema com o teste é que a equipe tende a se concentrar em projetar e implementar testes, e não encontrar os problemas importantes. Se o teste não encontra estes problemas, acaba sendo fundamentalmente irrelevante.

# Recursos humanos: Erros comuns

- **Usar o teste como um trabalho temporário para programadores novos**
  Testar requer prática e experiência; programadores novos têm desejo de sair logo da área de teste para uma área de maior visibilidade: o desenvolvimento.
- **Recrutar ex-programadores (ruins) para a equipe de teste**
  Maus programadores muitas vezes têm hábitos ruins – falta de atenção, desorganização – que os levarão a ser maus testadores.
- **Usar testadores que não conhecem o domínio da aplicação**
  Muitas vezes, o testador não conhece a fundo o domínio do problema, e os problemas que serão encontrados por ele serão menos relevantes que os problemas encontrados por um usuário experiente.
- **Não contratar pessoal do suporte técnico e documentação**
  O pessoal destes departamentos tem experiência em contato com os problemas do usuário, e sabem em geral o que é importante, o que é confuso, e o que é difícil de explicar — aspectos que estão intimamente ligados ao surgimento de problemas importantes.

# Recursos humanos: Erros comuns

- **Insistir que testadores sejam programadores**
  O bom testador não necessariamente é um programador, e uma equipe de testadores que é composta unicamente de programadores tem o problema de baixa diversidade, o que leva ao problema seguinte.
- **Não utilizar uma equipe de teste diversificada**
  Uma equipe de testes pouco variada tende a encontrar bugs concentrados em áreas específicas. O objetivo do teste é encontrar problemas de forma ampla, e não em profundidade.
- **Separar fisicamente testadores e programadores**
  Uma relação de trabalho boa e eficiente é importante se queremos obter o máximo proveito da equipe de teste; muito do trabalho será depuração feita em conjunto, que é muito complicada quando se separam as equipes envolvidas.
- **Acreditar que programadores não podem testar seu próprio código**
  Em efeito, programadores testam o seu código todo o tempo, e encontram muitos problemas. No entanto, encontram problemas de tipo bem selecionado, e a ação conjunto do teste de programadores com uma equipe de teste variada tende a ser a mais eficiente.
- **Não treinar e nem motivar os programadores para testar**
  Até que se valorize o teste feito por desenvolvedores, a equipe de teste terá que concentrar-se em executar os testes que os desenvolvedores fazem *e* os testes que só a equipe de teste tem capacidade de executar.

# Execução dos testes: Erros comuns

- **Dar mais atenção à execução dos testes do que ao seu projeto**
  Um testador que não é sistemático e não planeja o seu teste deixará de encontrar erros importantes, e não observará casos especiais.

- **Não rever os projetos de teste**
  Assim como programadores, testadores podem se beneficiar de análise conjunta do projeto dos testes. Testar em isolamento é uma técnica pouco eficiente de se conduzir a avaliação de qualidade.

- **Ser específico demais nas entradas e procedimentos do teste**
  Embora o teste deva conter uma descrição da configuração, entradas e saídas esperadas, ser muito específico na descrição dos passos a ser tomados torna a manutenção do teste mais cara, a sua elaboração mais complexa, e reduz a chance de se encontrar um problema associado à atividade mas não exatamente dentro do caso analisado.

- **Não notar e nem explorar irregularidades peculiares**
  Muitas vezes, irregularidades são esquecidas como irrelevantes, enquanto a verdade é que freqüentemente são fontes de problemas. Um bom teste utiliza os casos de teste como um ponto de partida para uma exploração do produto.

- **Checar que o produto faz o que deve, mas não verificar se ele *não* faz o que *não* deve**
  O teste deve, além de focar na execução correta da ação, verificar se nada mais foi alterado além do estritamente desejado. Este ponto é muitas vezes esquecido durante a execução dos testes.

# Execução dos testes: Erros comuns

- **Elaborar suítes de teste compreendidas apenas por seus criadores**
  Se os testes são muito particulares, grande custo é implicado na saída de seus desenvolvedores, e alto custo de sua manutenção ocorre como conseqüência.

- **Testar apenas através da interface com o usuário**
  É uma boa técnica oferecer uma interface associada específica para executar os testes, porque muitos problemas não são encontrados diretamente através da interface com o usuário.

- **Informar problemas incompleta ou ineficientemente**
  Encontrar os problemas é uma das metades do trabalho da equipe de teste. Informar os testes é muito importante, mas muitas vezes não há informação de como reproduzir o erro, do que exatamente ocorreu de errado, do nível de prioridade do erro, e se oferece pouco apoio para fazer a depuração em si.

- **Adicionar apenas testes de regressão quando problemas são encontrados**
  Problemas tendem a se concentrar em áreas do produto; problemas encontrados em uma área determinada do produto implicam na necessidade de se fazer uma análise mais profunda daquela região, e não apenas em acrescentar à regressão.

- **Não manter um histórico de anotações para os próximos testes**
  Os produtos criados por uma empresa em geral tem grande semelhança entre si, e os problemas serão muitas vezes parecidos. Manter anotações a respeito dos problemas encontrados em um produto serve como base importante para análises futuras.

# Uso da tecnologia: Erros comuns

- **Tentar automatizar todos os testes**
  Automatizar os testes, embora possa a longo prazo reduzir o custo total do teste, tem aplicação específica. Muitas vezes, um teste não justifica o esforço requerido na sua automação.

- **Esperar re-executar testes manuais**
  Se a maior parte dos testes terá de ser re-executado manualmente, o tempo gasto no seu desenvolvimento e documentação será desperdiçado.

- **Usar ferramentas de automação de interface para reduzir o custo do teste**
  Como mudanças de interface são comuns, e há grande uso de *widgets* customizados, é difícil aplicar testes de interface automatizados, e há custo alto na sua implementação. Usar scripts para testar a interface é uma alternativa interessante que pode ser utilizada de forma mais proveitosa.

- **Esperar que testes de regressão encontrem uma grande proporção dos defeitos introduzidos**
  Embora os testes de regressão capturem problemas em partes já existentes do software causados por mudanças no sistema, a maior parte dos problemas estão justamente relacionados à funcionalidade nova criada em si, o que terá de ser analisada em novos testes.

- **Abraçar exageradamente a análise de cobertura do código**
  A análise de cobertura de código é uma observação de quanto do código está coberto por casos de teste. No entanto, problemas múltiplos podem acontecer em uma única linha do programa, e dependem de *como* é executado aquela parte do código. Apoiar-se apenas em cobertura do código, portanto, não oferecerá uma análise de qualidade sólida.

# Uso da tecnologia: Erros comuns

- **Remover testes da suíte de testes de regressão porque não aumentam a cobertura**
  Testes de regressão não têm como objetivo cobrir todo o código, e sim observar se mudanças no software implicaram na introdução de erros. Removê-los indiscriminadamente significará reduzir o poder da suíte de teste.

- **Usar cobertura como um objetivo primário para a equipe de teste**
  Usar a cobertura para avaliar a equipe é uma forma numericamente simples de se verificar a sua performance. O problema com avaliar a equipe de teste através da cobertura é que a equipe passa a se focar neste objetivo, e não no objetivo real da tarefa. Como a cobertura não é uma avaliação real da qualidade do software, não pode ser usada para avaliar a performance da equipe.

- **Não analisar a cobertura de código em absoluto**
  Tendo dito tudo isso, a análise da cobertura oferece uma análise simples da avaliação do código, e, embora frustrante se utilizada como a ferramenta única de qualidade, oferece uma base importante o suficiente para não ser abandonado por completo.