Conceitos e Técnicas de Testes de Software

Prof. Evandro César Freiberger

Instituto Federal de Mato Grosso
Departamento da Área de Informática
evandro.freiberger@ifmt.edu.br

2023

Sumário

Teste de Software...

- Processo de executar um programa com objetivo de encontrar erros. (Myers, 1979)
- Processo de avaliar um software ou um componente de software para verificar se ele satisfaz os requisitos especificados ou identificar diferenças entre resultados esperados e obtidos. (ANSI/IEEE Standard 729)

Conceitos Básicos de Testes de Software

- Para que os erros possam ser descobertos antes de o software ser liberado para a utilização, existe um conjunto de atividades, que no geral são chamadas de “Validação, Verificação e Teste”, ou “VV&T”.

- O objetivo dessas atividades é garantir que tanto o modo pelo qual o software está sendo construído, quanto o produto em si, estejam em conformidade com o especificado.

- Atividades de VV&T não se restringem ao produto final. Ao contrário, podem e devem ser conduzidas durante todo o processo de desenvolvimento do software, desde a sua concepção.

Teste de Software Contribui...

- **Verificação**: Estamos construindo corretamente o produto?
  - O objetivo é verificar se o software atende aos requisitos funcionais e não funcionais especificados
  - Verificação inclui da realização de testes para encontrar erros

- **Validação**: Estamos construindo o produto certo?
  - A inexistência de erros não mostra a adequação operacional do sistema
  - Deve ser feita a validação com o cliente
  - A validação procura assegurar que o sistema atende as expectativas e necessidades do cliente

Padrão IEEE 1012 - *IEEE Standard for System, Software, and Hardware Verification and Validation*

Conceitos Básicos de Testes de Software

Teste de Software: O software tem defeitos?

O teste de software é considerado uma **técnica dinâmica de verificação e validação**, pois o software é **executado com dados de teste** e seu comportamento é analisado.

Diferentemente da inspeção de software ou, também chamada de revisão por pares, que é considerada uma técnica estática, pois não é necessário executar o software em um computador.

Obs: **Se o Teste de Software não encontra defeitos, não significa que o software não os têm.**

Porque Não Testar... (1)

Nove descupas para não Testar - *Fonte: 9 Excuses Why Programmers Don't Test Their Code*

1. Meu código funciona bem — por que eu deveria me incomodar em testá-lo?
   - Ninguém é perfeito — os erros acontecem com muito mais frequência do que deveriam.
   - Não há problema em confiar no seu código. Mas mesmo se outro desenvolvedor revisou seu código e o aprovou, você não deveria estar satisfeito . . . ainda.
   - Todas essas coisas dependem do fator humano. E é exatamente isso que você não deve querer, pois o fator humano falha de tempos em tempos.

"Na verdade, ninguém cria código perfeito da primeira vez, exceto eu. Mas só existe um de mim- Linus Torvalds

Porque Não Testar... (2)

2. Esse trecho de código não pode ser testado.
   - Quando você lida com um código que não pode ser testado, uma vez que está mal projetado e o código está abaixo do ideal, você tem alguns problemas maiores.
   - O código não testável é caro para manter e modificar. Além disso, o código não testável será difícil para os novos membros da equipe aprenderem.
   - Se você não estiver testando seu código, acabará enfrentando o problema em que os desenvolvedores não têm confiança para implantar o código.
   - A falta de testes os deixa ansiosos, pois não têm a confirmação de que suas alterações não quebraram nada.

Porque Não Testar... (3)

3. Não sei o que testar
   - Um ótimo começo seria testar os casos mais comuns (caminhos felizes). Isso informará quando essa parte do código será quebrada, com maior impacto.
   - Mas há muito mais a testar do que apenas clicar no cenário do caminho feliz. Você deve testar alguns casos extremos de partes do seu código que são as mais complexas ou de partes com maior probabilidade de erros.
   - Sempre que você encontrar um bug, é uma boa prática escrever um novo teste que cubra esse caso de teste.
   - Torne a escrita de testes parte da sua rotina — não os torne opcionais.

Porque Não Testar... (4)

4. O teste aumenta o tempo de desenvolvimento e o tempo está acabando.
   - Este é um dos maiores equívocos quando se trata de teste.
   - Sempre que você começa a escrever testes pela primeira vez, isso definitivamente aumenta o tempo de desenvolvimento.
   - Primeiro, você precisa aprender a escrever código testável e como fazer testes adequados.
   - Este é um investimento que você faz a longo prazo.
   - Quando o teste se tornar um hábito, você economizará bastante tempo.
   - Fazer testes poupará o problema de desenvolvedores que não têm confiança para implantar seu código.

Porque Não Testar... (5)

5. Os requisitos não são bons

   - Requisitos ruins não são uma desculpa para pular a escrita de testes. Você tem muitas opções para descobrir os requisitos. Infelizmente, requisitos não documentados são uma realidade que a maioria de nós enfrenta com mais frequência do que gostaria.
   - Você precisa trabalhar com qualquer pequeno pedaço de documentação que possa colocar em suas mãos.
   - Você também deve dar uma olhada nas versões atuais ou mais antigas do aplicativo — se possível, é claro. Você provavelmente encontrará muitas pistas ocultas aqui. Não se esqueça de dar uma olhada nos testes. Eles podem estar cheios de jóias escondidas que podem ajudá-lo.
   - Por último, mas não menos importante, tente conversar com alguns membros da equipe. Eles podem saber muito mais sobre os requisitos não documentados que você está procurando.

Porque Não Testar... (6)

6. Este pedaço de código não muda.
   - Não faz sentido testar o código que não muda. Isso é uma completa perda de tempo.
   - Bem, a única certeza que temos no desenvolvimento de software é que os requisitos mudarão. Às vezes, eles parecem estar sempre mudando.
   - As pessoas mudam de idéia por muitas razões e as fazem regularmente. A mudança de requisitos pode ser causada por muitas coisas. Algumas pessoas solicitam um recurso, mas na verdade não sabem o que querem. Outra razão pela qual os requisitos podem mudar é por causa da política da organização.
   - Sempre que você estiver lidando com alterações de requisitos, priorize o teste dos fluxos mais comuns.

Porque Não Testar... (7)

7. Posso testar mais rapidamente se fizer manualmente.
   - O problema com os testes manuais é que você deve executá-los repetidamente.
   - Sim, você pode testar um determinado recurso mais rapidamente ao fazê-lo manualmente uma vez. Porém, os testes manuais consomem muito tempo quando você precisa executá-los repetidamente.
   - A gravação de testes leva algum tempo, mas depois de fazer o teste, você pode executá-lo inúmeras vezes e custa apenas uma fração do tempo em comparação com o teste manual.

Porque Não Testar... (8)

8. O cliente quer pagar apenas pelos produtos a serem entregues.
   - Todo cliente deseja ver o máximo de entregas possível, pois isso dá a sensação de que a equipe de desenvolvimento está realizando algum trabalho.
   - É exatamente isso que os desenvolvedores dizem a seus gerentes sempre que não desejam testar.
   - Os testes são uma parte essencial do desenvolvimento de software, e essa parte do código é tão importante quanto qualquer outra parte — portanto, você deve tratá-lo dessa maneira.
   - Inclua tempo para escrever testes nas estimativas que você faz.
   - Escrever testes reduz os custos de manutenção e os custos de desenvolvimento a longo prazo.

Porque Não Testar... (9)

9. Este pedaço de código é tão pequeno . . . não vai quebrar nada.
   - Todo desenvolvedor escreveu trechos de código tão pequenos que não poderiam quebrar. E, no entanto, ele ainda conseguiu quebrar alguma coisa.
   - Como você sabe que seu código funciona perfeitamente?
   - Por favor, deixe suas palavras serem apoiadas por alguns testes reais. O teste completo filtra os bugs críticos, garantindo que o código funcione da maneira pretendida.

Cenário Atual do Desenvolvimento de Software

- Prazos apertados para entregas
- Clientes menos tolerantes a atrasos nas entregas
- Necessidade de software de boa qualidade
- Clientes menos tolerantes a falhas
- Novas tecnologias e sistemas cada vez mais complexos
- O mercado têm poucos testadores especialistas
- Pouco investimento na área de testes
- Software testados apenas pelos próprios desenvolvedores
  - Sistemas perfeitos
  - Vício da Implementação já Conhecido

Conceitos Básicos de Testes de Software

A literatura tradicional estabelece significados específicos para diferentes termos relacionados como: falha, defeito, erro, engano.

- Definimos **defeito** (defect) como sendo um passo, processo ou definição de dados incorretos;
- **Engano** (mistake), como a ação humana que produz um defeito;
- A existência de um **defeito** pode ocasionar um **erro** (error) durante uma execução do programa, que se caracteriza por um estado inconsistente ou inesperado;
- Tal estado pode levar a uma **falha** (failure), ou seja, pode fazer com que o resultado produzido pela execução seja diferente do resultado esperado, ou até mesmo que a execução seja interrompida inesperadamente.

Conceitos Básicos de Testes de Software

Existe o consenso de que não é possível eliminar todas as falhas/erros de um Software.

Conceitos Básicos de Testes de Software

Domínio de Entrada e Domínio de Saída.

- O **domínio de entrada** de um programa P, denotado por D(P), é o conjunto de todos os valores possíveis que podem ser utilizados para executar P.

- O **domínio de saída** do programa, é o conjunto de todos os resultados possíveis produzidos pelo programa.

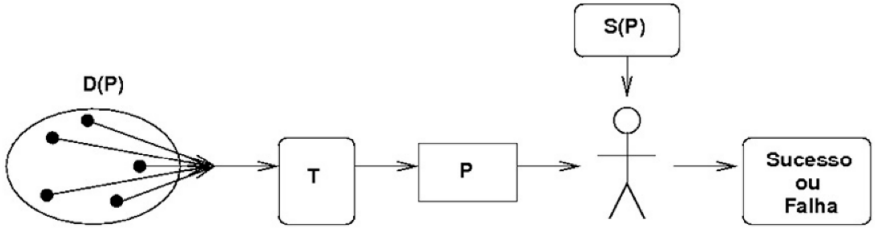
- Exemplo - um programa que recebe como parâmetros de entrada dois números inteiros x e y, com y >= 0, e calcula o valor de x elevado y , indicando um erro caso os argumentos estejam fora do intervalo especificado.
  - O domínio de entrada deste programa é formado por todos os possíveis pares de números inteiros (x, y).
  - O domínio de saída seria o conjunto de números inteiros e mensagens de erro produzidos pelo programa.

Conceitos Básicos de Testes de Software

- Um **Dado de Teste** para um programa **P** é um elemento do domínio de entrada de **P**.
- Um **Caso de Teste** é um par formado por um dado de teste e o resultado esperado para a execução do programa com aquele dado de teste.
- Ao conjunto de todos os **casos de teste** usados durante determinada atividade de teste costuma-se chamar **conjunto de teste** ou **conjunto de casos de teste**.
- Por exemplo, no programa que calcula x elevado y , temos os seguintes casos de teste:
  ((2, 3), 8),
  ((4, 3), 64),
  ((3, -1), Erro).

Conceitos Básicos de Testes de Software

Cenário Típico das Atividades de Testes.



Para um domínio de dados de um programa **D(P)**, aplicado em um conjunto de casos de testes **T**, executados pelo programa **P**, e comparados com as especificações do programa **S(P)** o resultado pode ser **sucesso** ou **falha**.

Conceitos Básicos de Testes de Software

- Testes não verificam completamente as saídas de um sistema, visto que as **ENTRADAS SÃO INFINITAS**
- Teste não garante a qualidade, pois a qualidade tem um conceito muito mais amplo
- Testes custam em média de 20% a 30% do custo de elaboração de um software (**ou pelo menos deveriam custar**)

Conceitos Básicos de Testes de Software

Característica	Sub-características
Funcionalidade O conjunto de funções satisfaz as necessidades explícitas e implícitas para a finalidade a que se destina o produto?	Adequação Acurácia Interoperabilidade Segurança de acesso Conformidade
Confiabilidade O desempenho se mantém ao longo do tempo e em condições estabelecidas?	Maturidade Tolerância a falhas Recuperabilidade
Usabilidade É fácil utilizar o software?	Inteligibilidade Apreensibilidade Operacionalidade
Eficiência Os recursos e os tempos utilizados são compatíveis com o nível de desempenho requerido para o produto?	Comportamento em relação ao tempo Comportamento em relação aos recursos
Manutenibilidade Há facilidade para correções, atualizações e alterações?	Analisabilidade Modificabilidade Estabilidade Testabilidade
Portabilidade É possível utilizar o produto em diversas plataformas com pequeno esforço de adaptação?	Adaptabilidade Capacidade para ser instalado Capacidade para substituir Conformidade

Tabela 2. Categorias de características de qualidade de software da ISO/IEC 9126 (NBR 13596).

Perfil do Testador

Perfil do Testador

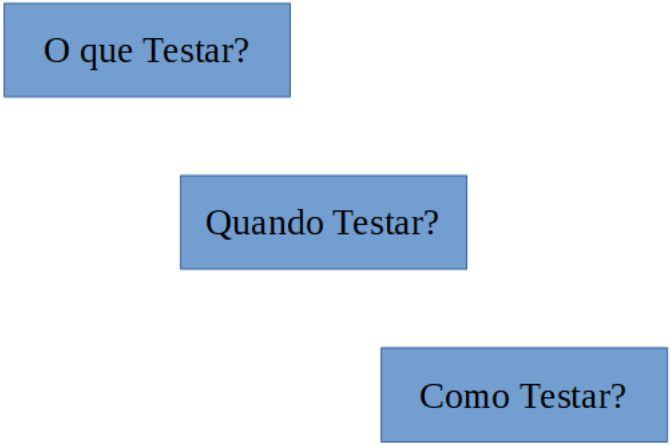
Testador: O Perfil de um Chato necessário!
Check list do testador: Características básicas do chato de plantão!

Ser o Sherlock Holmes da TI: investigativo, o testador deve ser um profissional curioso, aquele que procura nas combinações mais absurdas erros que podem comprometer o sistema.

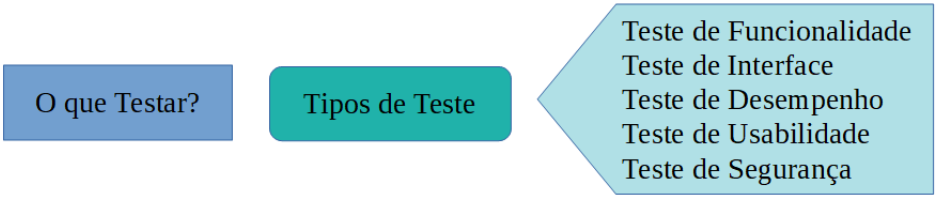
Usar de sensatez: fazer uso do bom senso é uma das principais características de um profissional da qualidade, afinal é este profissional que mensurará o nível de qualidade do sistema.

Ser o negociador: o testador deve usar a sua capacidade de negociação e persuasão a favor da qualidade, sugerindo melhorias e possíveis soluções. Convencer o desenvolvedor de que a lógica empregada está errada nem sempre é uma tarefa fácil.

Questões Chave de Teste de Software



Questões Chave de Teste de Software - O que Testar?



O que Testar? - Teste de Funcionalidade

Objetivo: verificar se as funcionalidades do sistema estão ocorrendo de forma correta (conforme regras de negócio e requisitos)

Forma de verificar: executar testes de Unidade, Integração, Sistema, Aceitação, Alfa, Beta e Regressão

Observação:

- precisa estar sempre atualizado em relação ao código em desenvolvimento

O que Testar? - Teste de Usabilidade

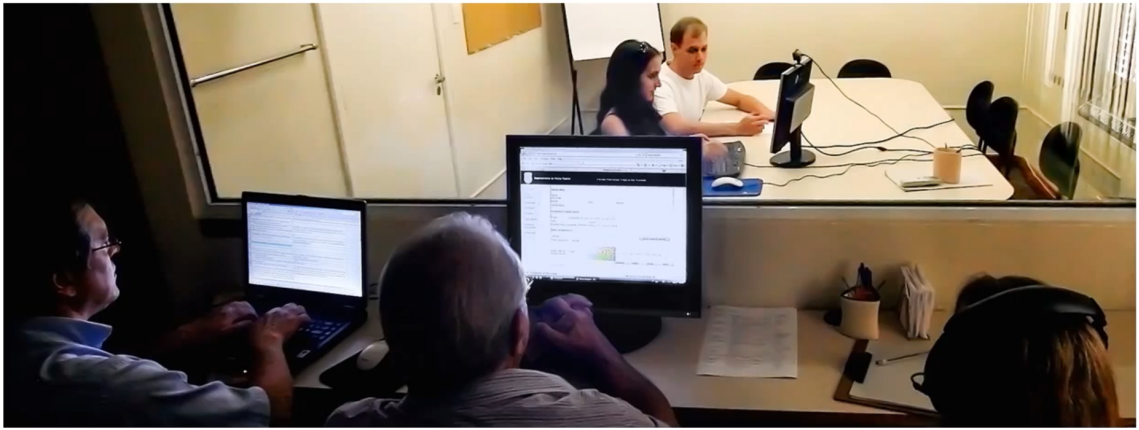
Objetivo: validar os aspectos que envolvem a experiência do usuário ao utilizar o sistema

Forma de verificar: monitorar o usuário usando o sistema, quer seja pessoalmente ou via ferramentas de apoio

Exemplos:

- Ergonomia de um website
- Definição e disposição de cores da interface
- Tipo da interface (menu, touch screen, gestos, voz, etc)

O que Testar? - Exemplo de Teste de Usabilidade



O que Testar? - Teste de Segurança

Objetivo: validar a proteção do sistema contra invasões ou acesso não autorizado à informações

Forma de verificar: depende do que será testado; é comum até a terceirização dessa tarefa; vão desde o desenvolvimento até a operação do sistema

Exemplos:

- Sites com acesso restritos
- Tráfego de informações criptografadas
- Senhas fracas, dedutíveis ou usuais

O que Testar? - Teste de Portabilidade

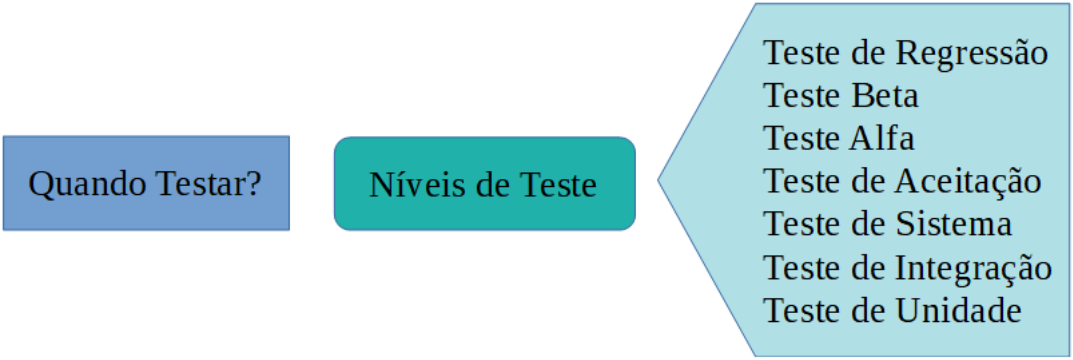
Objetivo: validar o funcionamento do sistema em diferentes plataformas, ambientes e dispositivos, para os quais o sistema está sendo proposto

Forma de verificar: planos de testes que contemplem a execução em todos os ambientes propostos (funcionalidade e padrões de interface)

Exemplos:

- IOS, Android, Windows, Linux
- Computadores, notebooks, smartphones, tablets
- Firefox, Chrome, Opera, Edge

Questões Chave de Teste de Software - Níveis de Testes de Software



Níveis de Testes de Software - Teste de Unidade ou Unitário

Objetivo: encontrar falhas em unidades autônomas do sistema

O que testa: sub rotinas, métodos, classes, . . . , menores unidades

Realizado: pelos programadores

Forma: normalmente é automatizado por ferramentas (JUnit, PHPUnit, etc)

Observação:

- precisa estar sempre atualizado em relação ao código em desenvolvimento
- costuma dar muito trabalho para manter o isolamento (dados e requisições)

Níveis de Testes de Software - Teste de Integração

Objetivo: validar a comunicação entre componentes de um sistema

O que testa: a integração (envio/recebimento de dados) entre os componentes

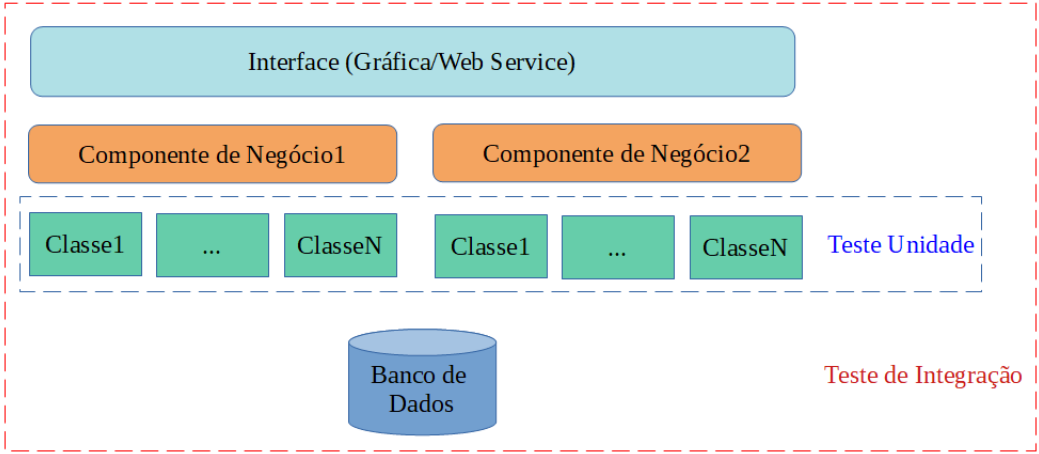
Realizado: pelos programadores

Forma: normalmente é automatizado por ferramentas (JUnit, PHPUnit, etc)

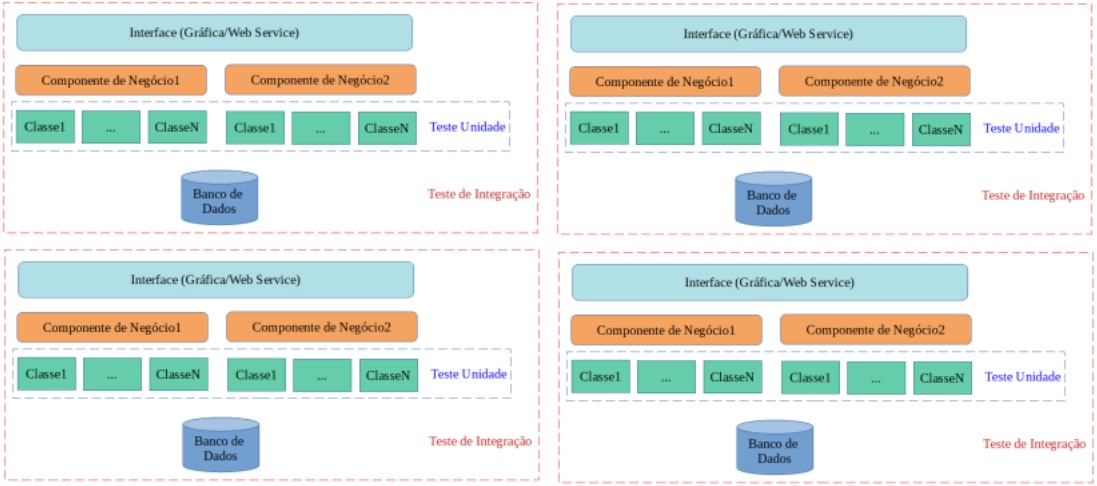
Observação:

- Vão desde a chamada de métodos entre classes, comunicação entre camadas, comunicação com banco de dados, chamada de serviços (Web Services/Microsserviços, API, Middleware), etc

Níveis de Testes de Software - Teste de Integração



Níveis de Testes de Software - Teste de Integração



Níveis de Testes de Software - Teste de Sistema

Objetivo: executar o sistema sob o ponto de vista de seu usuário final, buscando por falhas

O que testa: entradas e saídas do sistema (funcionalidade)

Realizado: equipe de teste/testador especialista

Forma: cenários de testes a partir de requisitos especificados, pode ser automatizado

Observação:

- A percepção de um bom testador irá explorar uma gama maior de possibilidade de encontrar erros;
- Embora a automatização seja possível, o teste com testadores reais pode ampliar os casos de testes;

Níveis de Testes de Software - Teste de Aceitação

Objetivo: executar o sistema sob o ponto de vista de seu usuário final, buscando atestar se aceita ou não o sistema

O que testa: principalmente funcionalidade conforme acordo solicitado

Realizado: número reduzido de usuário final do sistema (escolhidos), pessoas que acompanham o desenvolvimento do sistema

Forma: usuários finais simulam o uso do sistema verificando o comportamento do sistema, tendo como base os cenários mais críticos do sistema (requisitos)

Observação:

- Pode incluir recuperação de falhas, segurança e desempenho

Níveis de Testes de Software - Teste Alfa

Objetivo: executar o sistema sob o ponto de vista de seu usuário final, buscando dar feedback sobre o funcionamento do sistema

O que testa: principalmente funcionalidade

Realizado: número reduzido de usuário final do sistema, porém maior que no teste de aceitação, que não tem uma ligação direta com o desenvolvimento do sistema

Forma: usuários finais simulam o uso do sistema verificando o comportamento do sistema, mas sem um plano de testes

Observação:

- A equipe de desenvolvimento deve dedicar um tempo para absorver os feedbacks dos testes Alfa e retornar com alterações, ajustes ou justificativas

Níveis de Testes de Software - Teste Beta

Objetivo: executar o sistema sob o ponto de vista de seu usuário final, buscando dar feedback sobre o funcionamento do sistema

O que testa: principalmente funcionalidade

Realizado: número grade usuário final do sistema, desconhecidos da equipe de desenvolvimento

Forma: usuários finais usam o sistema e podem dar feedback pelo próprio sistema (formulário do próprio sistema)

Observação:

- A equipe de desenvolvimento não precisa dar feedback personalizado para os testes Beta
- Teste Beta funcionam como um pré-lançamento

Níveis de Testes de Software - Teste de Regressão

Objetivo: re-executar testes após alterações no sistema, para garantir que tudo continua funcionando corretamente (busca de efeitos colaterais)

O que testa: tudo o que existir de testes (automatizados ou não)

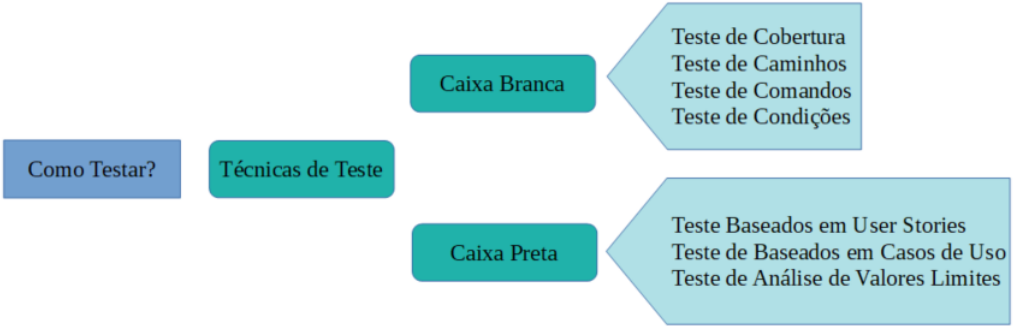
Realizado: depende do nível do teste que será repetido

Forma: depende do nível do teste que será repetido

Observação:

- A automação de testes e a documentação de cenários de testes facilitam a execução de testes de regressão.

Questões Chave de Teste de Software - Como Testar?



Como Testar? - Teste de Caixa Branca

Objetivo: validar as estruturas internas do sistema, analisando o código que realiza as ações

Níveis de teste: testes unitários, testes estáticos

Exemplos:

- Testes de métodos e classes
- Testes de comandos de repetição
- Testes de condições
- Análise de código (qualidade/padrão do código)

Como Testar? - Teste de Caixa Preta

Objetivo: Validar as funcionalidades do sistema, sem analisar o código que realiza as ações (analisa as saídas das operações mediante entradas submetidas)

Níveis de teste: Teste de Integração, Sistema, Aceitação, Alfa e Beta

Exemplos:

- Testes de integração entre componentes de negócio, com banco, com serviços
- Testes de uso na visão de usuário final (Sistema, Aceitação, Alfa e Beta)

Partição de equivalência

Nesta técnica, os valores de entrada do sistema são divididos em grupos que vão ter um comportamento parecido, para que possam ser processados da mesma maneira.

As partições de equivalência podem ser aplicadas para dados válidos e inválidos, bem como a valores de saída, valores internos, valores relacionados à eventos e para os parâmetros recebidos pelas interfaces.

Análise do valor limite

Este método parte do princípio de que o comportamento na borda de uma partição de dados tem maior probabilidade de apresentar erros.

Com isso, esta técnica se torna um complemento para a partição de equivalência.

Os valores máximos e mínimos de uma partição são seus valores limites e os testes podem ser feitos para dados válidos como inválidos.

Tabela de decisão

A tabela de decisão é um método importante para documentar regras de negócios que o sistema deve cumprir.

Estas são criadas a partir da análise da especificação funcional e da identificação destas regras de negócios.

A tabela de decisão contém as condições de disparo, combinações de verdadeiro e falso para cada entrada de dados, bem como a ação que resulta de cada combinação.

Transição entre status

Um sistema pode exibir diferentes comportamentos dependendo de seu status atual ou de eventos anteriores.

A elaboração de um diagrama permite que o testador visualize os status, transições, entradas de dados, eventos que os acionam e as ações que podem resultar.

Esta técnica pode ajudar a identificar possíveis transações inválidas.

Casos de uso

Os casos de uso descrevem as interações entre os atores, que podem ser usuários ou outros sistemas, os quais possuem pré-condições que devem ser atendidas para atingir o funcionamento correto do software.

Os casos de uso terminam com pós-condições, que são resultados observáveis e o estado do sistema após a execução. Esta técnica é útil para definir testes de aceitação, nos quais o usuário e/ou cliente participa.

Histórias de usuário

A história de usuário descreve um recurso - ou parte dele - que pode ser desenvolvido e testado em uma única iteração.

Esta técnica descreve a funcionalidade que será implementada, os requisitos não-funcionais e os critérios de aceitação.

A cobertura mínima de teste para uma história de usuário é composta pelos critérios de aceitação, portanto, os casos de teste são derivados desses critérios.

Como Testar? - Teste Estáticos

Objetivo: Analisar o código sem executá-lo e verificar se as boas práticas e padrões foram obedecidos

É realizado com testes de Caixa Branca

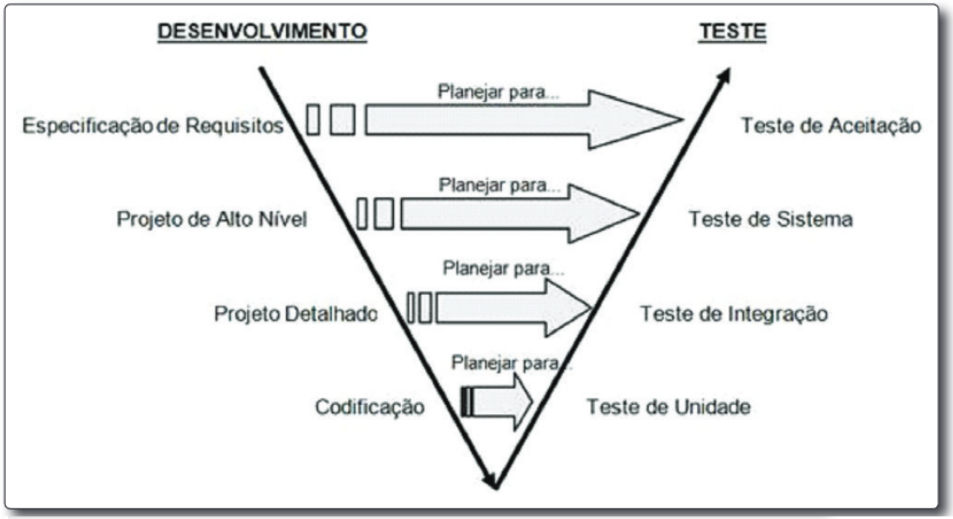
Existem ferramentas para a automação, mas exigem um esforço grande na customização e adequação para as características de cada projeto/código
Exemplos:

- Documentação de código
- Nomenclatura de identificadores
- Null Pointers são tratados
- Arquitetura definida foi seguida
- As conexões com banco de dados foram encerradas

Testes Manuais versus Automatizados

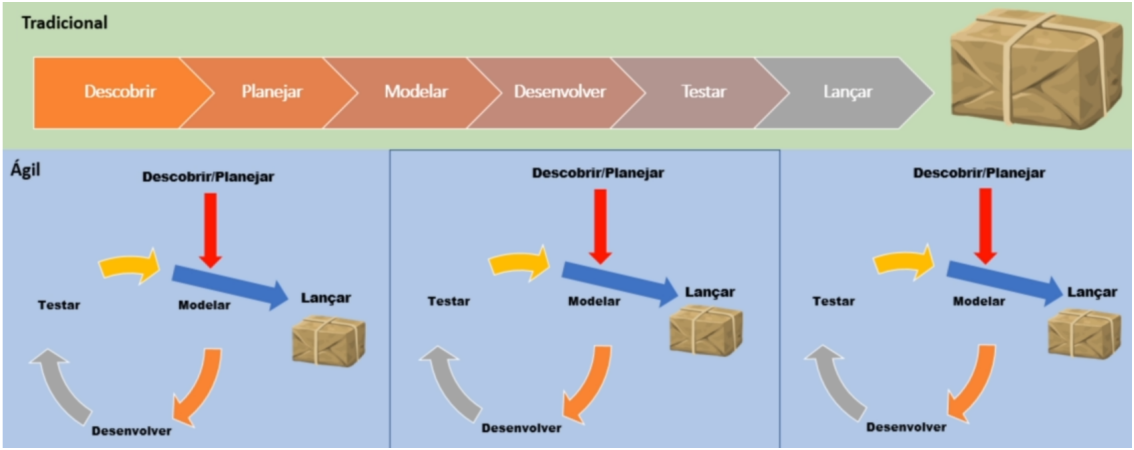
Testes Manuais	Testes Automatizados
Velocidade de execução baixa	Velocidade de execução alta
Repetitivo e cansativo	Pode ser executado quantas vezes necessário
Não exige tecnologia	Exige o domínio de tecnologias específicas
Alto custo a cada execução	Alto custo apenas na criação
Limitado em situações que envolve grande paralelismo e concorrência	Permite testar situações impossíveis de testes manuais
Podem explorar além do cenário de teste, quando necessário	Faz apenas o que foi programado para o teste
Podem avaliar questões visuais como cores e formas	Não avaliam questões visuais
Podem avaliar questões de usabilidade	Não avaliam questões de usabilidade

Testes no Modelo V de Desenvolvimento



Modelo V descrevendo o paralelismo entre as atividades de desenvolvimento e teste de software (CRAIG e JASKIEL, 2002)

Testes no Tradicional versus Modelos Ágeis

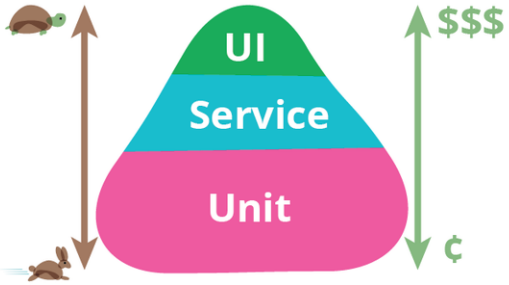


Documentação no Modelo Tradicional versus Modelo Ágil

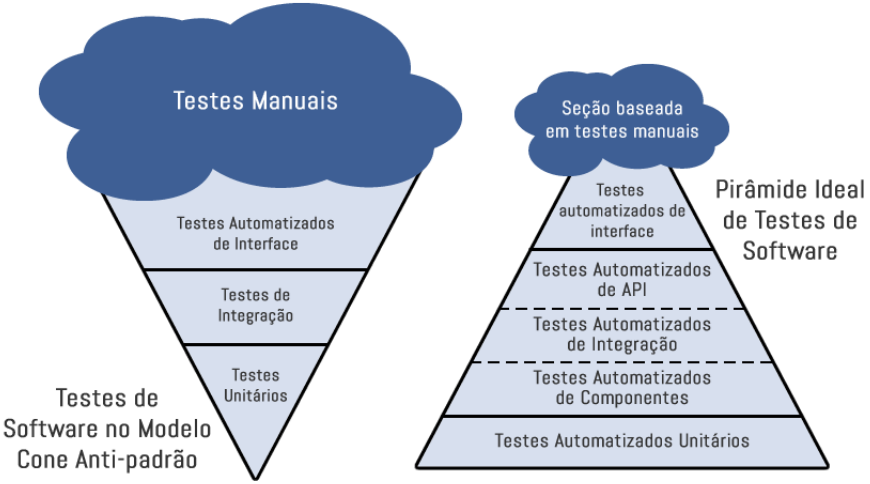
Pirâmide de Teste

Os testes Unitários são os mais rápidos para serem desenvolvidos e mais baratos.

Os testes de Interface com o Usuário são mais trabalhosos e mais caros para serem desenvolvidos.



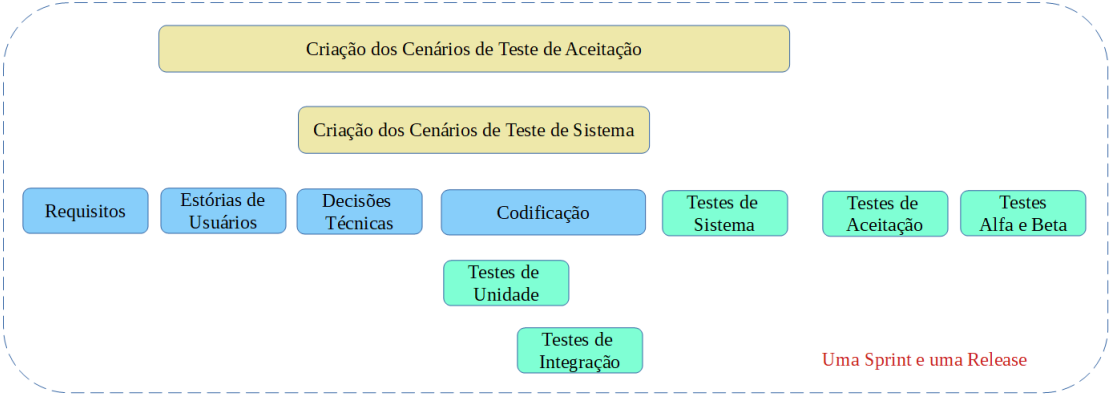
Pirâmide de Teste



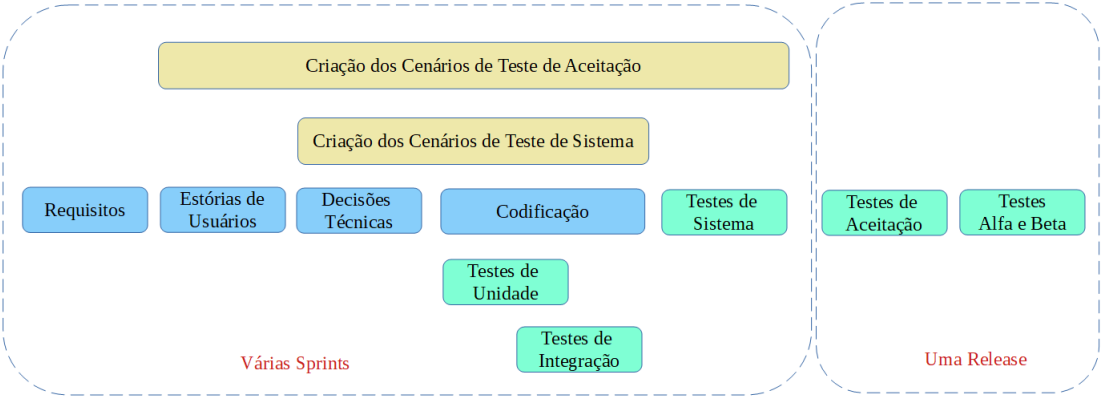
Responsabilidades nos Testes

- **Desenvolvedores**
  - criar e executar os testes de Unidade
- **Testadores**
  - Dar suporte a implementação dos testes de unidade
  - Criar, executar, monitorar e manter os testes (com apoio dos desenvolvedores)
    - Testes de Integração de componentes
    - Testes de Sistema
    - Testes de Integração de Sistemas
- **Partes Interessadas (Stakeholders)**
  - Testam as estórias
  - Podem usar casos de testes previamente escritos

Visão Geral dos Testes em um Modelo Ágil (versão 1)



Visão Geral dos Testes em um Modelo Ágil (versão 2)



Produtos Típicos dos Testadores

- **Plano de Testes**
  - Testes Automatizados
  - Testes Manuais
- **Relatórios de Defeitos**
- **Logs de Resultado de Testes**
- **Métricas de Testes**
  - Estatísticas
  - O que foi encontrado versus o que foi resolvido
  - Recorrência de defeitos

Gestão de Testes e Configuração

- Verificar de forma automatizada e contínua as alterações de código;
- Manter histórico de ações efetuadas
- Permitir a viabilidade de integração contínua
- Controlar versões de testes, relacionadas com versões de código
- Permitir a rastreabilidade de alterações de documentos, códigos, scripts de testes e etc
- Viabilizar as atualizações dos testes, frente às mudanças no sistema
  - Ponto crítico de testes

Critérios de Aceitação Definidos

- Comportamento Funcional
- Características de Qualidade
- Cenários (Casos de Uso e User Stories)
- Regras de Negócio
- Restrições
- Definição de Dados
- Interfaces Externas

O que é um plano de teste?

Um plano de teste é um documento que descreve os planos de um projeto para testar um determinado software.

Este documento normalmente inclui informações como cronograma do projeto, estratégia, metas, prazos e estimativas sobre os resultados dos testes ou possíveis desafios.

Os planos de teste também podem ajudar a priorizar os objetivos do próprio software, não apenas o processo de teste. Isso ajuda a especificar quais práticas e estratégias os profissionais usam durante a fase de teste.

Por exemplo, se uma prioridade do software é um recurso que salva arquivos automaticamente, um objetivo principal do plano de teste pode ser garantir que esse recurso funcione corretamente testando diferentes tipos de arquivo.

Componentes de um Plano de Teste (1)

Normalmente um plano de teste é composto por estes componentes:

Escopo do projeto: O escopo do projeto é um resumo dos objetivos e detalhes primários do processo de teste.

Cronograma: O cronograma do plano de teste inclui prazos importantes e estimativas de quanto tempo cada tarefa pode levar.

Recursos: Incluir uma lista de recursos no plano de teste permite que os profissionais considerem quais recursos possuem e quais podem querer adquirir.

Ferramentas: Esta seção de um plano de teste inclui as ferramentas que os profissionais podem usar para testar o software ou aplicativo.

Gestão de obstáculos: A gestão de obstáculos envolve o reconhecimento dos riscos que podem ocorrer durante o projeto e pode ajudar os profissionais a se prepararem para a situação.

Componentes de um Plano de Teste (2)

Gerenciamento de defeitos: A seção de gerenciamento de defeitos de um plano de teste inclui o protocolo caso um profissional de garantia de qualidade encontre um defeito no software. Isso inclui informações de contato e práticas de relatórios.

Detalhes finais: Os detalhes finais de um plano de teste explicam a extensão dos testes que a empresa espera realizar. Isso pode ajudar os profissionais a entender quando o software está pronto para aprovação.

Cenário de Teste VS Casos de Teste (1)

Cenário de Teste => indica o que testar (pontos do sotftware a serem testados)

Caso de Teste => indica como testar um cenário

Exemplo:

Cenário de Teste: funcionalidade de login de um sistema

Caso de teste 1:

- Na tela de login, preencher o nome do usuário e senha com valores válidos e pressinar o botão login;
- Sistema deve realizar a autenticação do usuário, fechar a tela de login e atualizar a área de usuário logado.

Cenário de Teste VS Casos de Teste (2)

Caso de teste 2:

- Na tela de login, preencher o nome do usuário válido e senha válida e pressinar o botão login;
- Sistema deve emitir uma mensagem de usuário não autenticado e manter a tela de login aberta.

Caso de teste 3:

- Na tela de login, não preencher o nome do usuário e senha e pressinar o botão login;
- Sistema deve emitir uma mensagem que o usuário e a senha são obrigatórios e manter a tela de login aberta.

Especificação de Casos de Teste (1)

Especificação clássica, tende a ser mais extensa e mais textual. É um modelo interessante para projeto de testes de sistema ou aceitação, cujo objetivo principal são as funcionalidades ponta-a-ponta.

São considerados como um padrão de construção de casos de testes os seguintes intens:

- **ID** – Todo e qualquer caso de teste deve ser representado com um ID exclusivo.
- **Descrição** – Cada caso de teste deve ter uma descrição adequada que consiste em detalhes importantes, como qual recurso, unidade ou função está sendo testada e o que deve ser verificado.
- **Pré-condições** – Quais situações e condições que devem ser observadas antes do início da execução do teste.
- **Etapas de teste** – Para poder executar um teste corretamente, você deve entender corretamente como executar esse caso de teste específico. Portanto, escreva as etapas para executar o caso de teste em uma linguagem fácil e compreensível.

Especificação de Casos de Teste (2)

- **Informações de teste** – Reúna os dados necessários para a execução do caso de teste de forma precisa e concisa.
- **Resultado desejado** – Qual o resultado, situação ou estado deve ser atingido com a execução do teste.
- **Pós-condições** – Essas são as condições que devem ser atendidas após a execução bem sucedida do caso de teste.
- **Resultado real** – Qual o resultados obtido de fato após a execução do caso de teste.
- **Estado** – Por último, qual o estado do caso de teste: (aprovado ou reprovado), (passou ou não passou)

Especificação de Casos de Teste

O padrão **Give-When-Then** ou **Dado-Quando-Então**

Foi desenvolvido com o **objetivo de apoiar a codificação dos testes unitários** na organização do código do teste de forma clara e concisa.

Esse padrão **divide o teste em três seções distintas**, cada uma delas com um papel na estruturação e legibilidade do teste.

Contudo, podemos usar o padrão **Give-When-Then** para **apoiar a definição de casos de testes**, tanto para testes **Unitários**, como para **Testes de Integração** e **Sistema**.

Exemplos Give-When-Then

Exemplo01:

ID do Teste	quandoInstanciadoComRaioValido
Nome da Unidade	Circulo/getArea()
Given (Estado Inicial)	Criar uma instância com raio 2
When (Quando Executar)	Invocar o método getArea()
Then (Então Espera-se)	Resulta em 12,56

Exemplos Give-When-Then

Exemplo02:

ID do Teste	quandoInstanciadoComRaioNegativo
Nome da Unidade	Circulo/getArea()
Given (Estado Inicial)	Criar uma instância com raio -1
When (Quando Executar)	Invocar o método getArea()
Then (Então Espera-se)	Resulta em uma exceção RuntimeException

Exemplos Give-When-Then

Exemplo03:

Cenário: testar a integração entre o AlunoDAO e o banco de dados
na operação de inclusão

ID do Teste	incluirAlunoComDadosCorretos
Descrição da Integração	Este teste deve testar a integração entre o objeto AlunoDAO e o banco de dados, na operação incluir.
Given (Estado Inicial)	Dado que existe uma instância de AlunoDAO conectada ao banco Dado uma instância de AlunoVO com os seguintes dados: Nome: Aluno Teste Nome da mãe: Mãe do Aluno Teste Nome do pai: Pai do Aluno Teste ...
When (Quando Executar)	Executar o método inserir da classe AlunoDAO
Then (Então Espera-se)	Inclusão do aluno no banco sem exceção
Resultado Obtido	
Situação (passou ou não)	

Exemplos Give-When-Then

Exemplo04:

Cenário: testar a funcionalidade de inclusão de aluno

ID do Teste	IncluirAluno
Descrição da Integração	Esse teste testará a funcionalidade de inclusão de alunos com dados que devem ser válidos e resultam em inclusão do aluno.
Given (Estado Inicial)	Dado que o usuário esteja logado e autorizado para essa funcionalidade Escolher a opção Cadastros no menu principal Escolher a opção Cadastro de Aluno O Sistema deve exibir a tela de Cadastro de Aluno Clicar no botão Novo O Sistema deve exibir a tela de inclusão de alunos
When (Quando Executar)	O usuário preenche os campos com os seguintes dados: Nome: Aluno Teste Nome da mãe: Mãe do Aluno Teste Nome do pai: Pai do Aluno Teste ... Clicar no botão Salvar
Then (Então Espera-se)	Deve incluir o aluno no banco de dados e emitir uma mensagem: Aluno incluído com sucesso!!
Resultado Obtido	
Situação (passou ou não)	

Exemplos Tabela de Decisão

Exemplo01:

Cenário: Teste de cálculo de área do cilindro (getArea())

Tipo	Variáveis	Partições	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
Entradas	Objeto Circulo	Raio > 0	X	X	X									
		Raio = 0				X	X	X						
		Raio < 0							X	X	X			
		Nulo										X	X	X
	Altura	> 0	X			X			X			X		
		0		X			X			X			X	
		< 0			X			X			X			X
Saídas	Área		X											
	Exceção			X	X	X	X	X	X	X	X	X	X	X

Exemplos Tabela de Decisão

Exemplo02:

Cenário: Teste de cálculo de área do cilindro (getArea())

Tipo	Variáveis	Partições	T1	T2	T3	T4	T5	T6
Entrada	Objeto Circulo	Raio > 0	X	X				
		Raio <= 0			X	X		
		Nulo					X	X
	Altura	> 0	X		X		X	
		<=0		X		X		X
Saída	Área		X					
	Exceção			X	X	X	X	X

Exemplos Tabela de Decisão

Exemplo03:

Cenário: Teste de cálculo de área do cilindro (getArea())

Tipo	Variáveis	Partições	T1	T2	T3	T4	T5	T6
Entrada	Objeto Circulo	Raio 2	X	X				
		Raio -1			X	X		
		Nulo					X	X
	Altura	4	X		X		X	
		-4		X		X		X
Saída	Área		75,39					
	Exceção			X	X	X	X	X