TESTE DE SOFTWARE: INTRODUÇÃO, CONCEITOS BÁSICOS E TIPOS DE TESTES

Teste

Então, por que não testar tudo?

- Exemplo: cálculo do preço de um produto em uma loja

```
double getPreco (double preco, double  margem, boolean aVista) {
   ...
}
```

Teste

Então, por que não testar tudo?

- Exemplo: cálculo do preço de um produto em uma loja

```
double getPreco (double preco, double  margem, boolean aVista) {
   ...
}
```

- Valores possíveis: $2^{64} * 2^{64} * 2 = 6{,}8*10^{38}$

Teste

Então, por que não testar tudo?

- Exemplo: cálculo do preço de um produto em uma loja

```
double getPreco (double preco, double  margem, boolean aVista) {
   ...
}
```

- Valores possíveis: $2^{64} * 2^{64} * 2 = 6{,}8*10^{38}$
- Intel Core i7 7500U: 49.360 MIPS
  (https://en.wikipedia.org/wiki/Instructions_per_second)

Teste

Então, por que não testar tudo?

- Exemplo: cálculo do preço de um produto em uma loja

```
double getPreco (double preco, double  margem, boolean aVista) {
   ...
}
```

- Valores possíveis: $2^{64} * 2^{64} * 2 = 6{,}8*10^{38}$
- Intel Core i7 7500U: 49.360 MIPS
  (https://en.wikipedia.org/wiki/Instructions_per_second)
- Se cada teste fosse executado em 1 instrução: $1{,}3*10^{28}$s

Teste

Então, por que não testar tudo?

- Exemplo: cálculo do preço de um produto em uma loja

```
double getPreco (double preco, double  margem, boolean aVista) {
  ...
}
```

  - Valores possíveis: $2^{64} * 2^{64} * 2 = 6{,}8*10^{38}$
  - Intel Core i7 7500U: 49.360 MIPS
    (https://en.wikipedia.org/wiki/Instructions_per_second)
  - Se cada teste fosse executado em 1 instrução: $1{,}3*10^{28}$s
    - Universo tem "só" 10^{17}s

Teste

Técnicas de teste que ajudam a escolher os conjuntos de dados de teste

- Diminuem o número de casos de teste
- Maior probabilidade de existência de erros

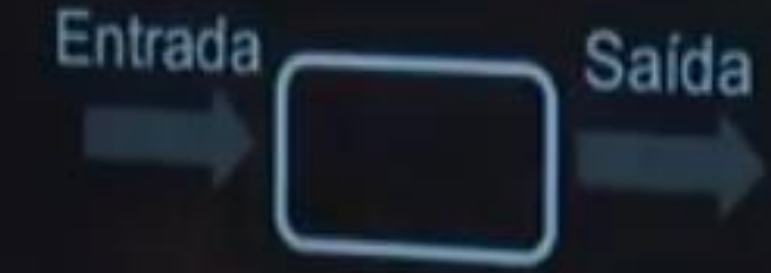
Duas técnicas principais:

- **Estrutural**: caixa branca
- **Funcional**: caixa preta

Teste funcional

Alvo do teste como uma caixa preta

- Não se usa nenhum detalhe interno



- Foco na especificação

Teste funcional

Existem alguns métodos:

- Partição equivalente
- Valores limite
- Tabelas de decisão
- Grafos de causa e efeito

Partição equivalente

Divide o domínio de entrada em classes de equivalência

- Resultado do teste de uma entrada nessa partição é equivalente aos demais

- **Como definir classes de equivalência?**
  - Dividir cada condição de entrada em grupos
    - Condições de entrada na especificação
    - Grupos válidos e inválidos

Exemplo

```
// Calcula o preço a ser cobrado por um produto ao considerar o
// preço real, a margem de lucro (entre 0 e 1) e um desconto para
// pagamentos à vista (de 5%)
double getPreco (double preco, double  margem, boolean aVista) {
  ...
}
```

Exemplo

```
// Calcula o preço a ser cobrado por um produto ao considerar o
// preço real, a margem de lucro (entre 0 e 1) e um desconto para
// pagamentos à vista (de 5%)
double getPreco (double preco, double  margem, boolean aVista) {
  ...
}
```

	Válido	Inválido
Preço	> 0	<= 0
Margem	>= 0 e <= 1	< 0 > 1
À vista	Sim Não	

Partição equivalente

- **Classes válidas**
  - Caso de teste deve cobrir diversas classes válidas
- **Classes inválidas**
  - Caso de teste deve cobrir apenas uma classe inválida

Exemplo

	Válido	Inválido
Preço	> 0 (1)	<= 0 (2)
Margem	>= 0 e <= 1 (3)	< 0 (4) > 1 (5)
À vista	Sim (6) Não (7)	

Exemplo

	Válido	Inválido
Preço	> 0 (1)	<= 0 (2)
Margem	>= 0 e <= 1 (3)	< 0 (4) > 1 (5)
À vista	Sim (6) Não (7)	

1,3,6 ; 2,3,6
1,3,7 ; 1,4,7

Exemplo

	Válido	Inválido
Preço	> 0 (1)	<= 0 (2)
Margem	>= 0 e <= 1 (3)	< 0 (4) > 1 (5)
À vista	Sim (6) Não (7)	

Teste	Classes	Preço	Margem	À vista
1	1, 3 e 6	10	0.5	true
2	1, 3 e 7	10	0.5	false
3	2	-4	0.5	true
4	4	10	-1	true
5	5	10	10	true

1,3,6 ; 2,3,6

1,3,7 ; 1,4,7

Valores limite

Condições no limite das partições costumam ter mais defeitos

- Limite inferior e superior

Ao invés de escolher qualquer valor da classe, escolhe aqueles na fronteira

- Considera classes de equivalência de entrada e de saída

Exemplo

	Válido	Inválido
Preço	> 0 (1)	<= 0 (2)
Margem	>= 0 e <= 1 (3)	< 0 (4) > 1 (5)
À vista	Sim (6) Não (7)	

Teste	Classes	Preço	Margem	À vista
1	1, 3 e 6	10	0.5	true
2	1, 3 e 7	10	0.5	false
3	2	-4	0.5	true
4	4	10	-1	true
5	5	10	10	true

Exemplo

	Válido	Inválido
Preço	> 0 (1)	<= 0 (2)
Margem	>= 0 e <= 1 (3)	< 0 (4) > 1 (5)
À vista	Sim (6) Não (7)	

Teste	Classes	Preço	Margem	À vista
1	1, 3 e 6	10	0.5	true
2	1, 3 e 7	10	0.5	false
3	2	-4	0.5	true
4	4	10	-1	true
5	5	10	10	true

Teste estrutural

O teste é feito considerando o conhecimento da estrutura interna do código

- **Problema: número de caminhos**
- **Existem diferentes critérios**
  - Critério baseado em fluxo de controle
    - Todas as arestas

Teste estrutural

O teste é feito considerando o conhecimento da estrutura interna do código

- **Problema: número de caminhos**
- **Existem diferentes critérios**
  - Critério baseado em fluxo de controle
    - Todas as arestas

Exemplo

```
double getPreco (double preco, double  margem, boolean aVista) {
   double valor = preco * (1 + margem);

   if (aVista) valor *= 0.95;
   if (preco <= 0 || margem <= 0.1 || margem > 1)
   throw new ErroDeCalculo("Valor inválido");

   return valor;
 }
```

Observação: esse código tem um defeito

- Margem <= 0.1
- O correto seria margem <= 0

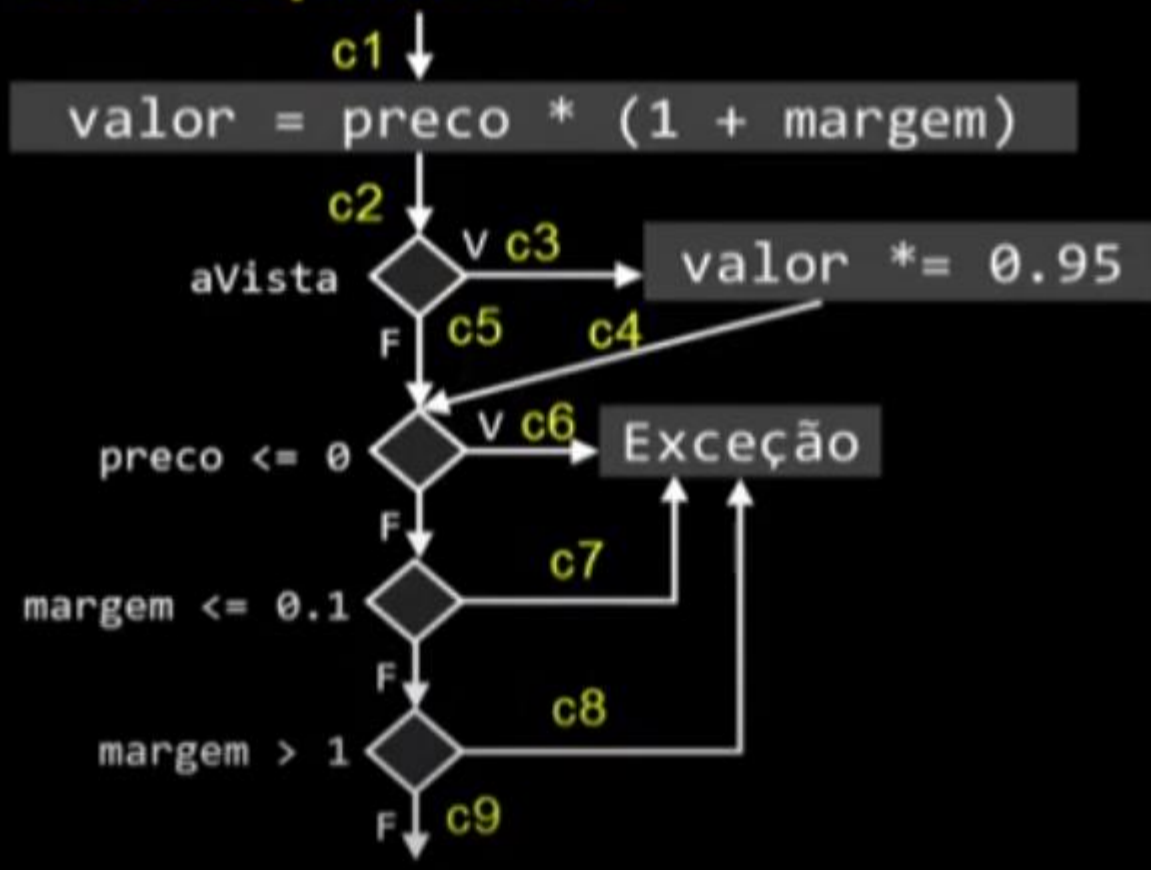
Exemplo

```
double getPreco (double preco, double  margem,
  boolean aVista) {
   double valor = preco * (1 + margem);

   if (aVista) valor *= 0.95;
   if (preco <= 0 || margem <= 0.1 || margem > 1)
   throw new ErroDeCalculo("Valor inválido");

   return valor;
}
```

Grafo equivalente

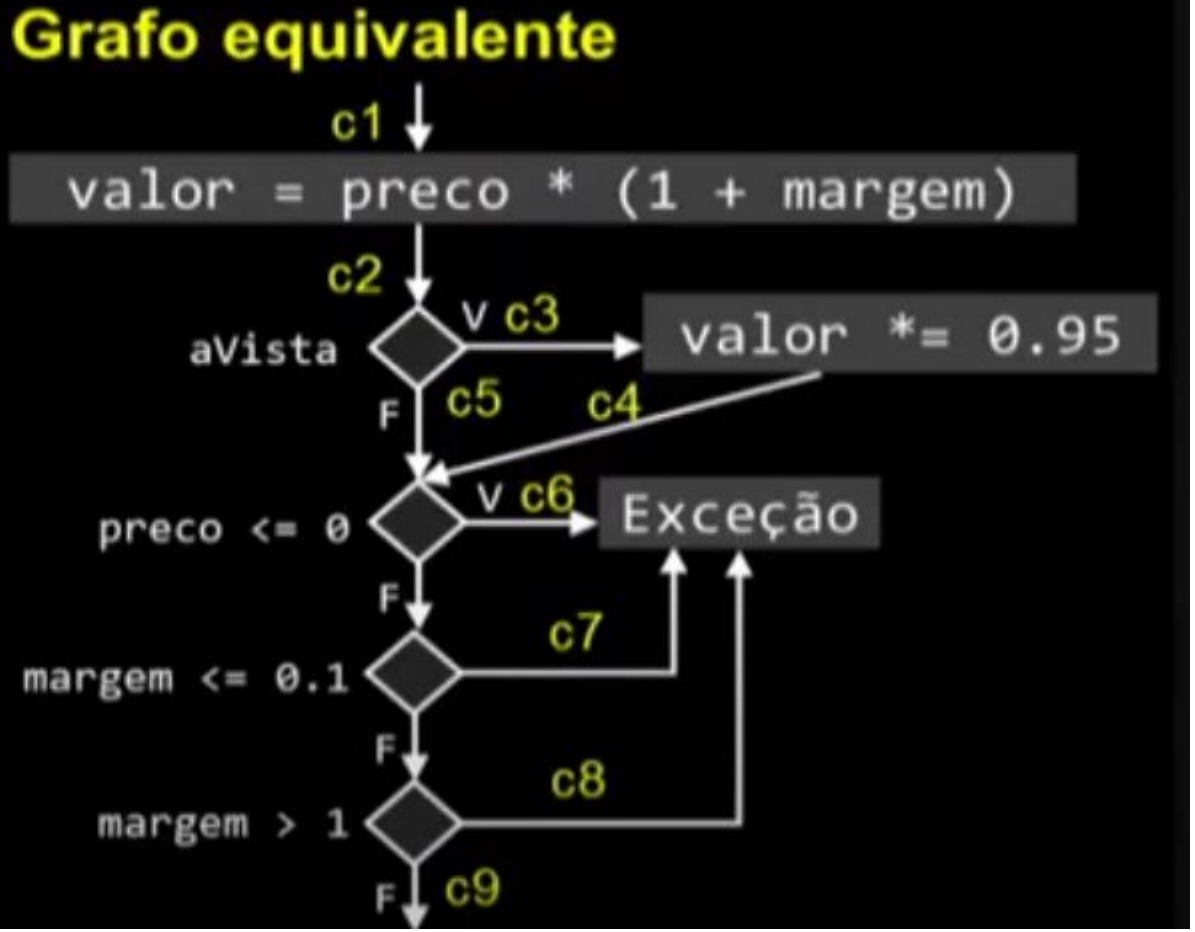


Exemplo

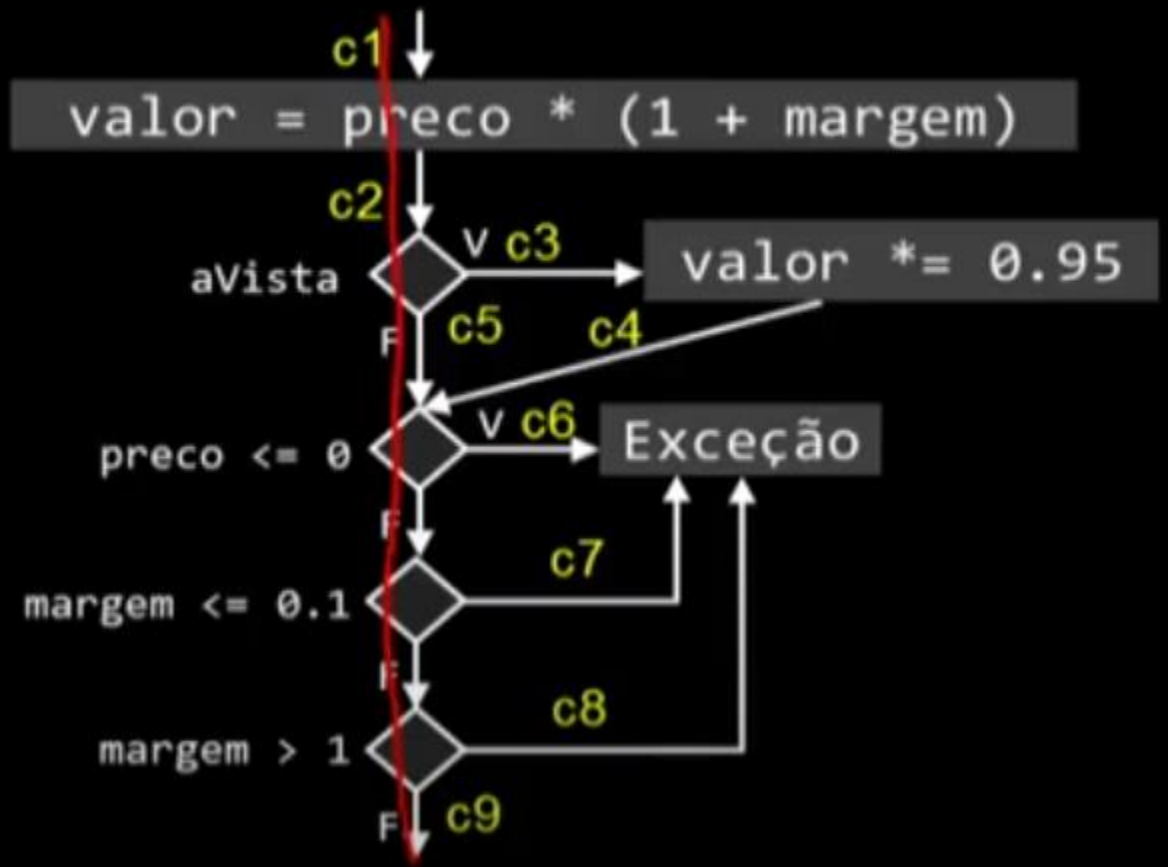
```
double getPreco (double preco, double  margem,
  boolean aVista) {
   double valor = preco * (1 + margem);

   if (aVista) valor *= 0.95;
   if (preco <= 0 || margem <= 0.1 || margem > 1)
   throw new ErroDeCalculo("Valor inválido");

   return valor;
}
```



Exemplo


c1
valor = preco * (1 + margem)
c2
aVista
V c3
valor *= 0.95
F c5
c4
preco <= 0
V c6
Exceção
F
c7
margem <= 0.1
F
c8
margem > 1
F c9

Exemplo

c1, c2, c9

Exemplo

Caminhos a testar
c1, c2, c3, c4, c9
c1, c2, c3, c4, c6
c1, c2, c3, c4, c7
c1, c2, c3, c4, c8
c1, c2, c5, c9
c1, c2, c5, c6
c1, c2, c5, c7
c1, c2, c5, c8

- Quais valores testar?

Teste

- A forma de realizar os testes depende do processo
- Normalmente, o teste é realizado em vários níveis

- Teste em V
  - Cada teste considera aspectos de uma atividade de desenvolvimento

- Por mais que se planeje a construção de um software, erros são passíveis de ocorrer. Pode ser um bug num game, uma falha que feche um programa ou um erro que impossibilite você salvar um arquivo.
- Quem já passou por esse tipo de situação sabe como é chato quando ficamos na mão por culpa de um programa com falhas. O teste de software serve justamente para tentar encontrar possíveis erros que um programa recém-desenvolvido possa apresentar, de modo a conseguir corrigi-lo antes que seja lançado no mercado, ficando disponível para uso do público.
- O teste de software geralmente é a última etapa na construção de um programa, visando checar o seu nível de qualidade. Os defeitos que um teste busca identificar incluem erro de compatibilidade, de algum algoritmo, de requisitos que não podem ser complementados, limitação de hardware etc. A lista é grande e aumenta com o tamanho do programa.

Quais os tipos de testes de software?

Existem diferentes tipos de testes que podem ser aplicados num software para identificar suas falhas, sendo as principais:

Teste da caixa branca

- utiliza o aspecto interno do programa/sistema, o código fonte, para avaliar seus componentes. Ele também é conhecido como teste orientado à lógica ou estrutural. Podem ser analisados itens como: fluxo dos dados, condição, ciclos etc. Na hora de implementá-lo é preciso verificar a criticidade, a complexidade, a estrutura e o nível de qualidade que se pretende obter do programa, envolvendo confiança e segurança;

Teste da caixa preta

– diferente do teste anterior, que prioriza os aspectos internos, o teste da caixa preta verifica aspectos externos. Os requisitos funcionais do sistema são avaliados. Não se observa o modo de funcionamento, sua operação, tendo como foco as funções que deverão ser desempenhadas pelo programa. Desse modo, avalia-se se um grupo de entrada de dados resultou nas saídas pretendidas, levando-se em consideração a especificação do programa. Ou seja, o que se esperava que o software deveria fazer. É conhecido também como técnica funcional;

Teste da caixa cinza

- – esse tipo de teste une os dois anteriores, por isso o termo “cinza”. Avalia tanto os aspectos internos quanto os externos, de entrada e saída. Pode utilizar-se de engenharia reversa;

Teste de regressão

- esse consiste em realizar testes a cada versão de um software, onde se modificam-se funcionalidades. Desse modo, evita-se que erros que foram corrigidos antes no software antes voltem a aparecer na hora de se incrementar algo novo a ele.

Teste de unidade

- testa-se unidades menores de um software, de modo isolado, para ver se todas funcionam adequadamente;

Teste de integração

- depois das unidades testadas, realiza-se uma verificação se elas funcionam juntas, integradas. Pode ocorrer delas apresentarem incompatibilidades ao funcionarem em conjunto, mesmo após terem sido aprovadas no teste de unidade;

Teste de carga

- – esse teste é feito para avaliar os limites de uso do software, o quanto ele suporta em volume de informações, tráfego etc. sem que apresente erros;

Teste de usabilidade

- – esse teste é feito por um pequeno grupo de usuários para ver se o software satisfaz as suas necessidades. Nesse teste analisa-se como o usuário usa o sistema, verificando onde ele tem mais dificuldade. Ouve-se também suas impressões, porém é preciso confrontá-las com as observações do avaliador;

Teste de stress

- aqui leva-se o software ao seu limite de potência e funcionamento, para mais ou para menos, de modo a avaliar em qual ponto ele deixa de funcionar adequadamente. Isso é feito para verificar se suas especificações máximas ou mínimas de uso estão corretas.

O que é um plano de teste de software?

- Um plano de teste é feito para colaborar com o desenvolvimento de um software. É por meio desse plano que os componentes técnicos, funcionais, estruturais etc. serão verificados e validados, de modo a garantir o bom funcionamento do programa junto ao usuário final. Sendo assim, **um plano de teste de software tem como foco garantir a confiabilidade e segurança de um software**, identificando possíveis erros e falhas durante a sua confecção, ou mesmo depois.
- Ele deve ser planejado em conjunto com a proposta do software, sendo aplicado em cada etapa do projeto e não somente no final.

Ferramentas de teste de software

- Para garantir a qualidade de um programa, as desenvolvedoras realizam testes nele. Isso é necessário para que falhas sejam detectadas antes que o software seja colocado no mercado. Sabe aquele programa que vive travando, não roda direito ou que faz o PC ficar lento? Esse, provavelmente, deve ter passado pelo processo de desenvolvimento com essas imperfeições. Então, para evitar que isso aconteça, as empresas contratam profissionais (os testadores de software ou analistas de testes) para identificarem esses problemas e relatarem para que os desenvolvedores os corrijam. Mas, para fazer isso eles precisam realizar uma bateria de testes diferentes, que envolvem desde análise da estrutura interna do software até a avaliação da interface.

O que são ferramentas de teste de software?

- Para que esses testes possam ser realizados de modo mais rápido e com maior abrangência, existem ferramentas que automatizam alguns deles ou auxiliam na execução de outros. Essas são as ferramentas de teste de software.

O que é e o que faz um testador de software?

- Com a expansão do mercado de tecnologia da informação (TI), uma nova profissão surgiu e tem crescido nos últimos anos: o testador de software ou analista de testes. Esse profissional é contratado para encontrar erros, falhas, bugs e outros tipos de problemas que não foram detectados durante a confecção de um software.
- O mercado para esse tipo de profissão é amplo. Abrange desde a prestação de serviços de testes de softwares para programas gerenciais até aplicativos de smartphones voltados para o público. E a expectativa é de que ele fique cada vez maior, à medida em que clientes de desenvolvedoras de softwares passam a solicitar a avaliação desse profissional nos programas encomendados.

Automação de testes de software

- Num mundo cada vez mais interligado pela tecnologia, os planos de testes de softwares têm um peso importante, pois muitos negócios dependem de que esses estejam funcionando corretamente.
- Qualquer falha num programa de gerenciamento financeiro pode acarretar prejuízos grandes em termos monetários. Um erro num software de um equipamento médico pode custar a vida uma pessoa ou dificultar o atendimento a alguém que precisa.
- E não é só os casos mais extremos que precisam de testes de software de qualidade, pois uma simples falha de um programa de uso comum que ocorra com frequência pode fazer com que usuário dele abra queixa ou não o recomende para outros usuários, migrando para a concorrência. Quem perde é quem desenvolveu o software.

Processo de Teste de Software

- O Processo de Testes de Software representa uma estruturação de etapas, atividades, artefatos, papéis e responsabilidades que buscam a padronização dos trabalhos e ampliar a organização e controle dos projetos de testes.
- O Processo de Teste, como qualquer outro processo deve ser revisto continuamente, de forma a ampliar sua atuação e possibilitar aos profissionais uma maior visibilidade e organização dos seus trabalhos, o que resulta numa maior agilidade e controle operacional dos projetos de testes.

Etapa 1: Planejamento dos Testes

- Esta etapa caracteriza-se pela definição de uma proposta de testes baseada nas expectativas do Cliente em relação à prazos, custos e qualidade esperada, possibilitando dimensionar a equipe e estabelecer um esforço de acordo com as necessidades apontadas pelo Cliente.

Dinâmica das Macro-Atividades

- Este diagrama representa a seqüência das “macro-atividades” a serem executadas na etapa de “Planejamento dos Testes”.

Detalhamento das Macro-Atividades

- Esta lista representa o conjunto de atividades que deverão ser executadas para que cada macro-atividade seja considerada finalizada, funcionando como um “check-list” de execução da etapa de “Planejamento dos Testes”.

Estudo do Projeto:

- Estudar as modificações solicitadas pelo Cliente (novos requisitos);
- Estudar as modificações de arquiteturas dos aplicativos;
- Estudar as lições aprendidas dos Projetos Anteriores;
- Avaliar expectativas de custos, prazos e qualidade exigidas pelo Cliente;
- Avaliar os riscos envolvidos nos Projetos e seus impactos neste processo;

Avaliação de Impacto:

- Avaliar se o projeto exige a criação de casos de testes “progressivos”;
- Avaliar se o projeto exige modificações em casos de testes “regressivos”
- Avaliar se o projeto exige adequações na automação dos testes;
- Avaliar se o projeto exige adequação nas atuais ferramentas empregadas;
- Avaliar se o projeto exige a aquisição/construção de novas ferramentas;
- Avaliar se o projeto exige modificações na estruturação do ambiente;

Análise Interna de Esforço

- Levantar métricas históricas para auxiliar na elaboração das estimativas de esforço;
- Estimar esforço interno para absorção dos impactos da Arquitetura dos Testes;
- Demonstrar esforço externo para absorção dos impactos da Arquitetura dos Testes;

Análise Externa de Esforço:

- Avaliar disponibilidade de espaço físico e infra-estrutura para os Terceiros;
- Especificar as necessidades de adequações que serão repassadas a Terceiros;
- Especificar métricas de qualidade e produtividades esperadas;
- Especificar SLA’s de serviço e multas contratuais;
- Estabelecer concorrência e obter a melhor proposta (opcional);
- Receber Proposta de Trabalho (Cronograma, Prazos e Custos da Terceirização);
- Definição de Cenários Possíveis (Duração, Esforço, Custo e Qualidade):
- Levantar Lista de Projetos em Andamento e a serem Iniciados;

- Avaliar a disponibilidade de recursos internos para alocação no Projeto;
- Identificar Cenários Diversos (Terceirização, Redução de Escopo, Priorização de Projetos);
- Definir Cronograma-Macro para cada cenário identificado;
- Definir Riscos para cada cenário identificado e Planos de Ação Esperados;
- Estabelecer Propostas e Aguardar aprovação da Diretoria;
- Aprovação do Planejamento:
- Obter o Aceite das Propostas de Cenários Aprovados pela Diretoria;
- Obter o Aceite de uma das Propostas pelo Cliente;

- Divulgar do Cenário Aprovado do Projeto aos colaboradores e terceiros;
- Obter a Assinatura do CONTRATO-MESTE e elaborar os ANEXOS; (no caso de terceirização)
- Alocar Espaço Físico dos Terceiros; (no caso de terceirização)
- Comunicar a Finalização da Etapa de Planejamento dos Testes; (externo)
- Definição das Responsabilidades
- Neste diagrama, está a representação dos papéis e responsabilidades para cada grupo de atividades envolvido na etapa de “Planejamento dos Testes”.

Definição de Cenários Possíveis (Duração, Esforço, Custo e Qualidade):

- Levantar Lista de Projetos em Andamento e a serem Iniciados;
- Avaliar a disponibilidade de recursos internos para alocação no Projeto;
- Identificar Cenários Diversos (Terceirização, Redução de Escopo, Repriorização de Projetos);
- Definir Cronograma-Macro para cada cenário identificado;
- Definir Riscos para cada cenário identificado e Planos de Ação Esperados;
- Estabelecer Propostas e Aguardar aprovação da Diretoria;

Aprovação do Planejamento:

- Obter o Aceite das Propostas de Cenários Aprovados pela Diretoria;
- Obter o Aceite de uma das Propostas pelo Cliente;
- Divulgar do Cenário Aprovado do Projeto aos colaboradores e terceiros;
- Obter a Assinatura do CONTRATO-MESTE e elaborar os ANEXOS; (no caso de terceirização)
- Alocar Espaço Físico dos Terceiros; (no caso de terceirização)
- Comunicar a Finalização da Etapa de Planejamento dos Testes; (externo)