SÃO
PAULO
TECH
SCHOOL

# Engenharia de Software

# Qualidade de Software

Aula 12

**Gerson Santos**

gerson.santos@sptech.school

# Antes de começar... Vamos alinhar

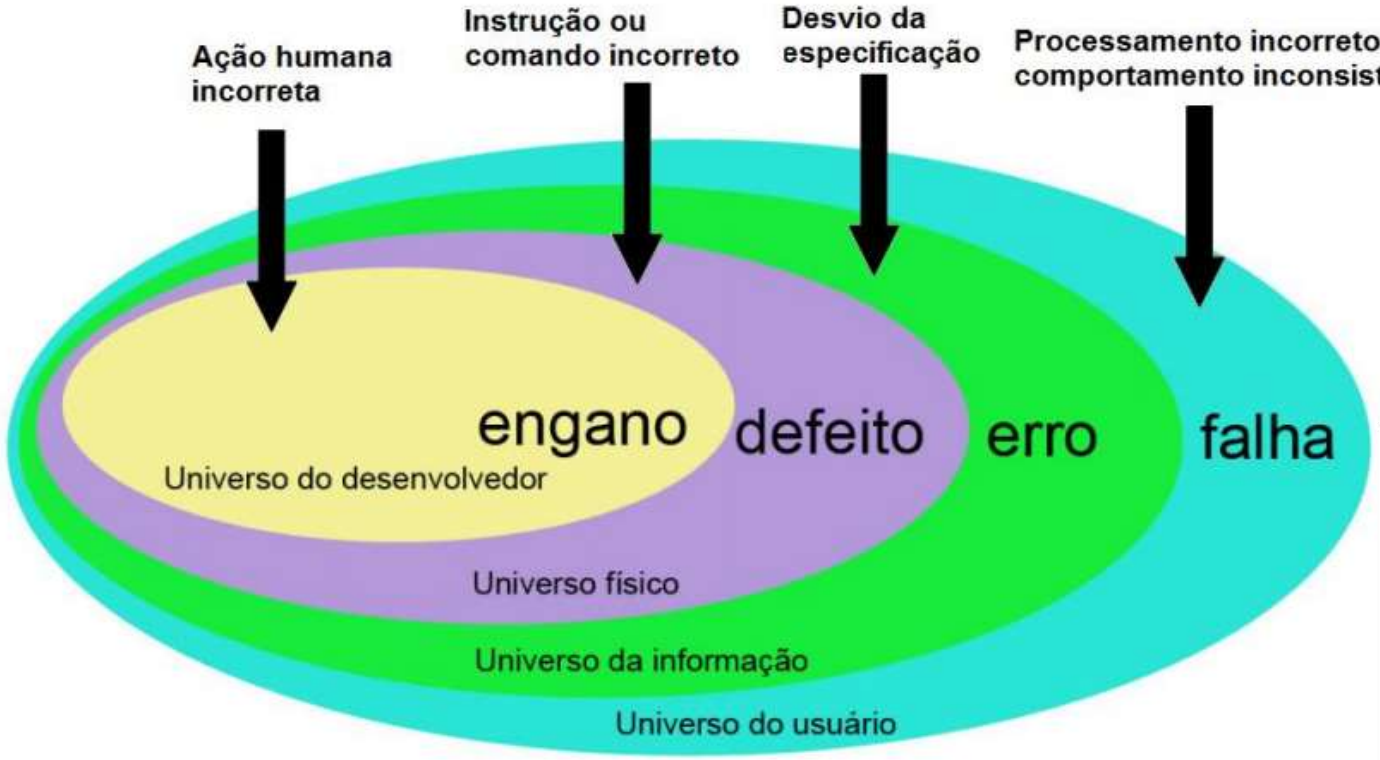
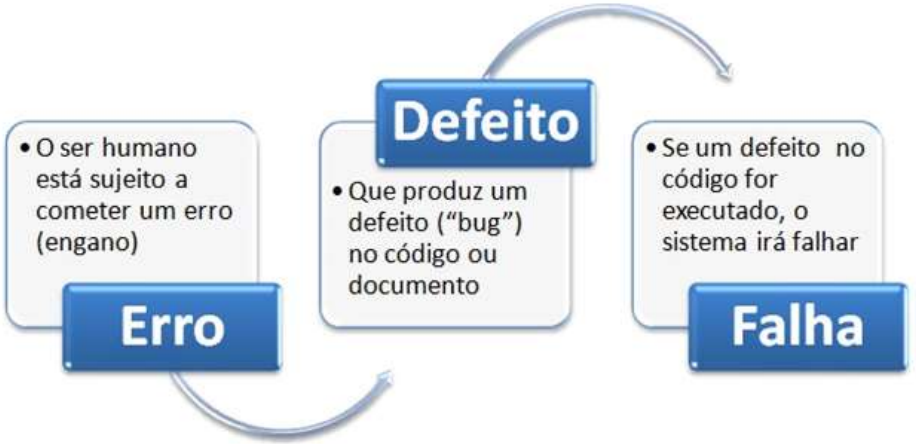



**Figura 1 - Engano x defeito x erro x falha**
**Fonte: Adaptado de Barbosa et al., 2000.**

# Vamos alinhar conceito

Não vamos entrar na discussão do que é BUG, defeito, erro, etc.

O livro mais simples é o do Pressman. Vamos por ele:

BUG : Antes de liberar o software

DEFEITO: Depois de liberar o software

# Verificação & Validação..mais divergências

A **Verificação** refere-se ao conjunto de tarefas que garantem que o software implemente corretamente uma função específica.

**Validação** refere-se a um conjunto diferente de tarefas que asseguram que o software que foi construído seja rastreável aos requisitos do cliente.

Boehm [Boe81] afirma isso de outra maneira:

**Verificação: "Estamos construindo o produto corretamente?"**

**Validação: "Estamos criando o produto certo?"**

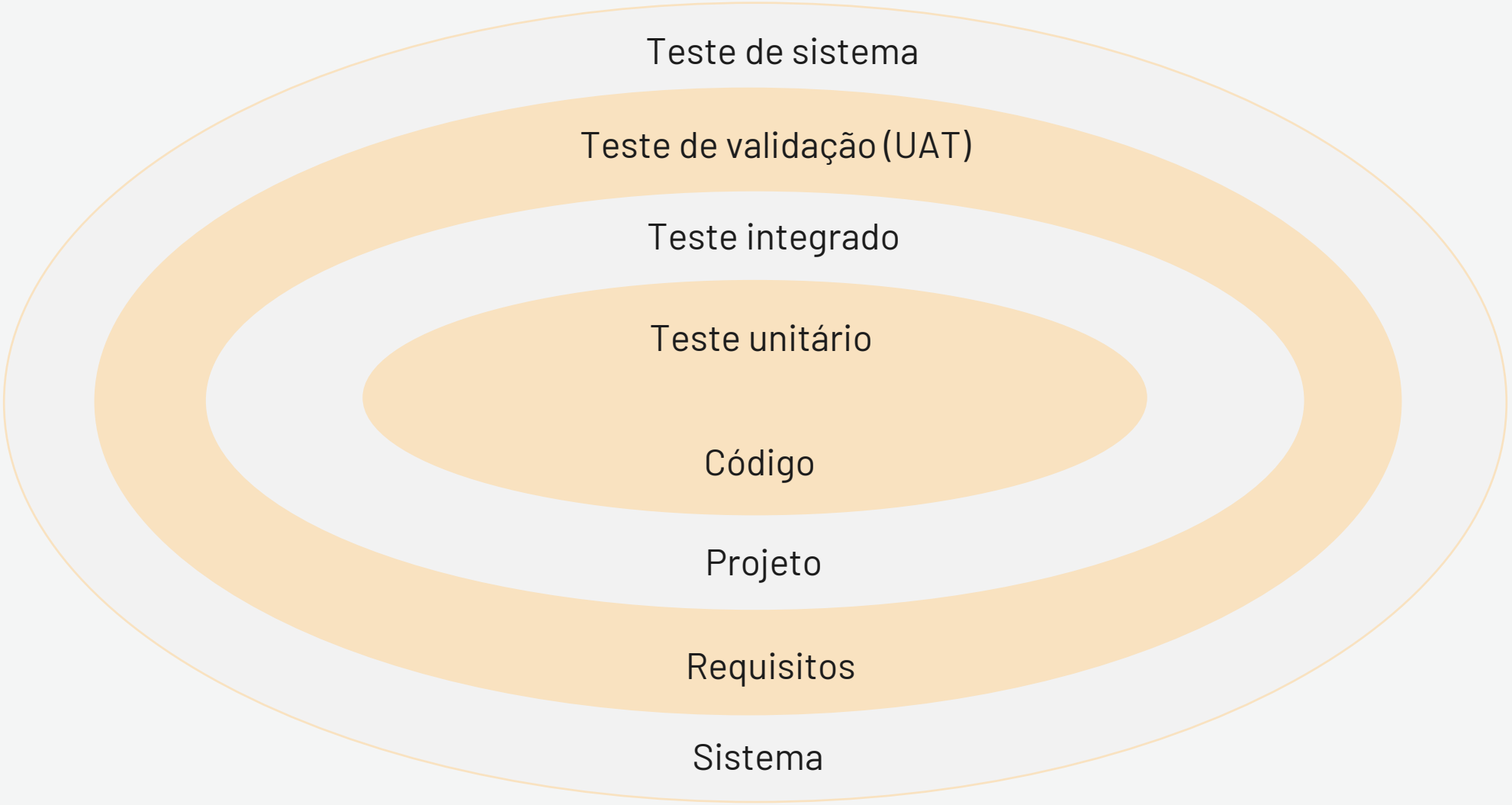
# Testes

“Teste é o processo de exercitar um programa com a intenção específica de encontrar erros antes de entregar o mesmo para o usuário final.”

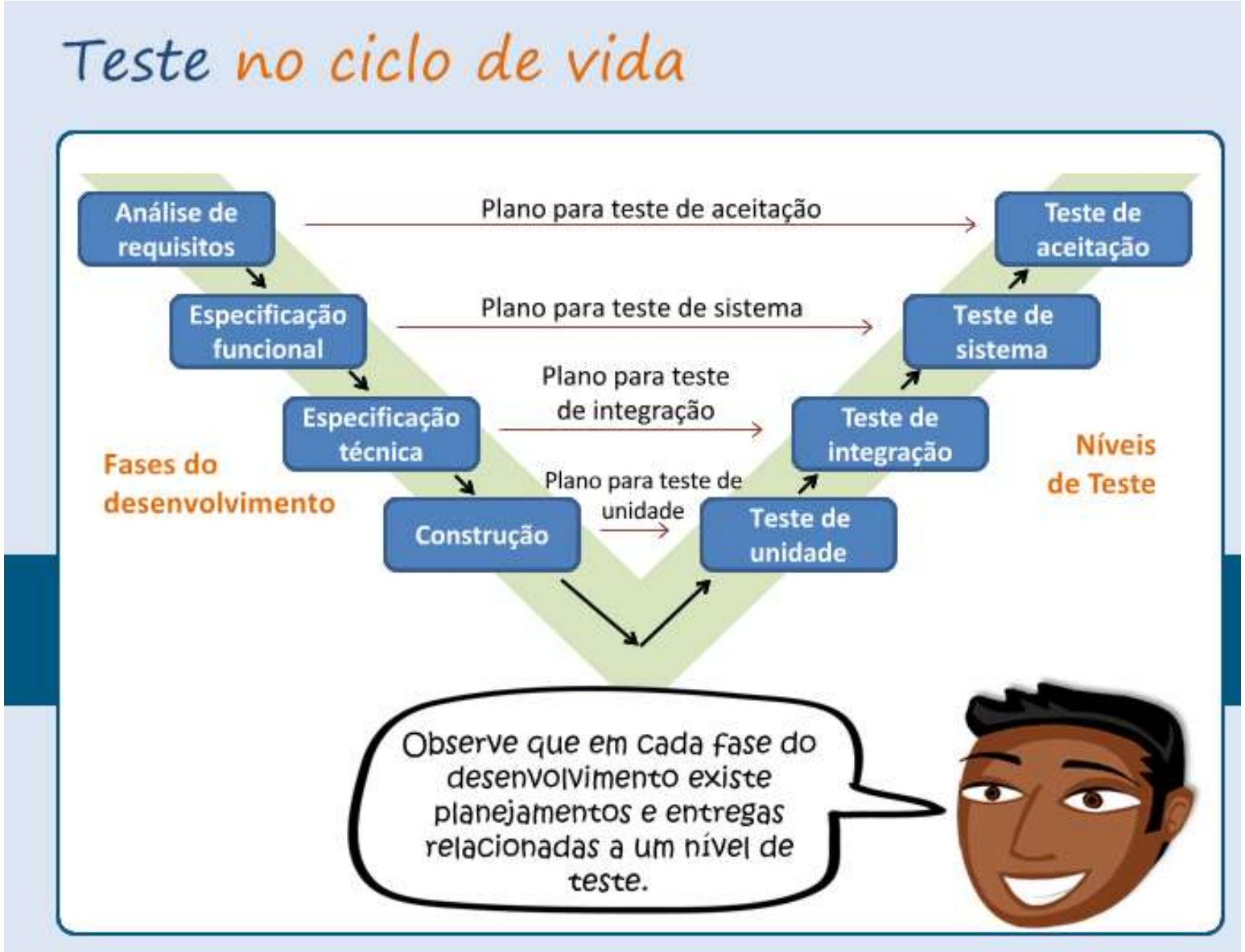
Sommerville


https://vidadeprogramador.com.br/2012/01/10/tester/

# Niveis de Teste/Estratégia de Teste

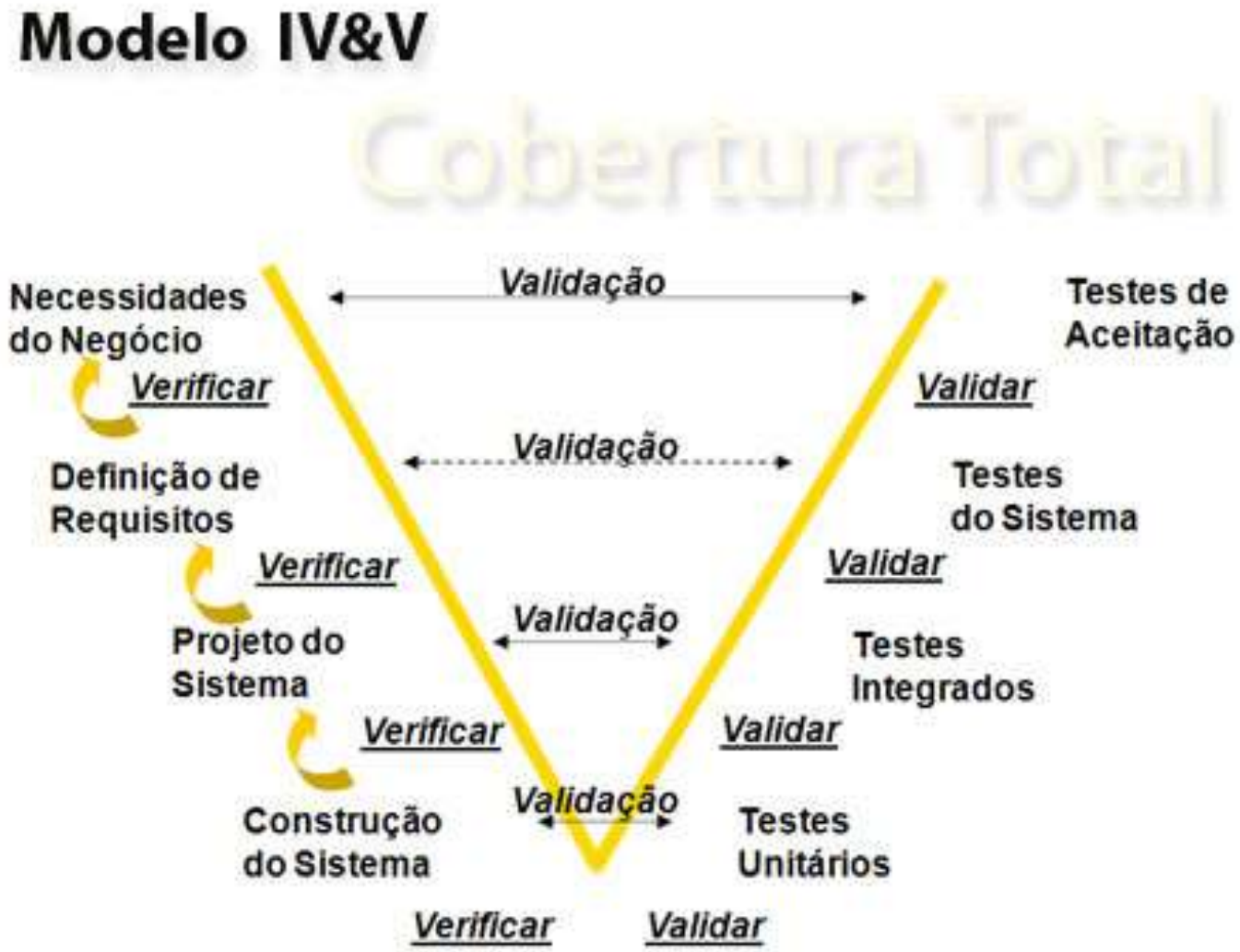


# Níveis de Teste/Estratégia de Teste




http://www.inf.ufsc.br/~fabiane.benitti/byebug/#

# Verificação & Validação



# Quem testa o software?

Se você acha que é o usuário...meu Deus!

ALFA – Ambiente controlado – dentro de casa – pessoas selecionadas

BETA – Externo com pessoas selecionadas. (innovators)

Normalmente quem são os responsáveis pelos Testes de Software:

- Desenvolvedor/Programador
- Tester
- Muito comum o Analista de Negócios também testar representando o cliente (ele simula também uma pré-validação/pré-homologação)

# Tipos de Testes

# Teste unitário

- Garantir que os problemas serão descobertos cedo!
- Facilitar a manutenção do código
- Servir como documentação
- Ajuda a melhorar o design do código e consequentemente você vira um desenvolvedor melhor
- Reduzir o medo
- Nas linguagens orientadas a objeto, você testa a classe, o método e até o objeto.
- Test Case é o teste para cada classe feito pelo desenvolvedor

# Teste Unitário – O que testar?

- O segredo dos Testes Unitários é saber o que testar, você precisa usar a criatividade para imaginar os testes possíveis, mas com o tempo treinando você consegue evoluir e até gerar uma lista mais comum de possibilidades, afinal, testar divisão por 0 não ocorre uma única vez. ☺
- Não faz sentido testar itens triviais como GET/SET a não ser que você tenha muito código neles;
- Também não faz sentido fazer muitos testes na mesma condição e deixar outras condições para trás;
- Use apenas os dados necessários;
- Comece pelo mais simples e depois vá para o complexo;
- Achou um bug? Escreva classe que testa esse bug e depois conserte;

# Testes de Aplicações Web (Mobile é similar)

- **Conteúdo**
- **Banco de Dados**
- **Interface do usuário**
- **Usabilidade**
- **Compatibilidade**
- **Componentes**
- **Navegação**
- **Configuração**
- **Segurança**
- **Performance**
- **Carga**
- **Stress**

https://paulomonteiro.wordpress.com/cansado/

# Exercício em Grupo

Vamos utilizar o CASE da Última Prova Continuada e vamos definir a qualidade para ele!

Cada grupo vai tratar um item específico. 30 minutos para:

- Entender o tipo de teste;
- Listar os ganhos;
- Pesquisar uma ferramenta, template ou prática que podem ser utilizados;

1. **Conteúdo**
2. **Usabilidade**
3. **Compatibilidade**
4. **Regressão**
5. **Navegação**
6. **Configuração**
7. **Segurança**
8. **Performance**
9. **Carga e Stress**
10. **Fumaça**

# Teste de Conteúdo

**Conteúdo:** Avaliado em um nível sintático e semântico.

**nível sintático:** a ortografia, a pontuação e a gramática são avaliadas para documentos baseados em texto.

**nível semântico –** correção (da informação apresentada), consistência (em todo o objeto de conteúdo e objetos relacionados a informação é a mesma) e falta de ambiguidade.

# Teste de Conteúdo

Um  exemplo de CheckList:

A informação é factualmente precisa?

A informação é concisa e direta?

O layout do objeto de conteúdo é fácil para o usuário entender?

As informações incorporadas em um objeto de conteúdo podem ser encontradas facilmente?

Já foram fornecidas referências adequadas para todas as informações derivadas de outras fontes?

As informações apresentadas são consistentes internamente e consistentes com as informações apresentadas em outros objetos de conteúdo?

O conteúdo é ofensivo, enganoso ou abre as portas para litígios?

O conteúdo infringe direitos autorais ou marcas registradas existentes?

O conteúdo contém links internos que complementam o conteúdo existente? Os links estão corretos?

# Teste de Interface do Usuário

**Links**: Testar se os links estão corretos, ou seja, a navegação.

**Formulários**: Testar as Validações de Tela, ou seja, preenchimentos obrigatórios, campos especiais, caracteres especiais, campos incompletos, etc.

**Script**: Testar se o JavaScript está funcionando corretamente nos diversos browsers (engines). (Atenção para também testar itens como “applets”

**HTML dinâmico**: Testar se objetos de conteúdo que são manipulados no lado do cliente usando scripts ou folhas de estilo em cascata (CSS) estão corretos.

**Janelas pop-up do lado do cliente**: Testar o comportamento, se estão funcionando e as limitações.

**Streaming de conteúdo**: Testes o funcionamento dos vídeos, áudios ou similares.

**Cookies**: Testar o funcionamento e validar o conteúdo dos cookies.

# Teste de Banco de Dados

## CheckList

Testar a conectividade com o(s) banco(s) de dados (Driver, versão, etc)

Validar as Querys

Validar o “COLLATION”, ou seja, conjunto de caracteres utilizados pelo banco de dados (língua)

Validar os tipos de campos se estão adequados

Testar as cargas e exportações se existirem, assim como as transformações de dados

# Teste de Compatibilidade

- Navegadores disponíveis
- Sistemas Operacionais disponíveis: Android, Linux, Windows, IOS, etc.
- Velocidade de conexão com a Internet
- Responsividade
- Plug-ins (Flash, RealPlayer, etc)

# Teste de Navegação

***Links de navegação***: Conforme teste da interface

***Redirecionamentos*:** Como os links entram em ação quando um usuário solicita uma URL inexistente ou seleciona um link cujo destino foi removido ou cujo nome foi alterado. Testar as diversas formas de solicitar a URL (www, direto, etc).

***Marcadores***: Embora os marcadores sejam uma função do navegador, o WebApp deve ser testado para garantir que um título de página significativo possa ser extraído à medida que o marcador é criado.

***Quadros e conjuntos de quadros****: Testar* o conteúdo correto, layout e tamanho adequados, desempenho de download e compatibilidade com o navegador

***Mapas do* site**: Cada entrada do mapa do site deve ser testada para garantir que o link leve o usuário ao conteúdo ou à funcionalidade adequada.

***Mecanismos de pesquisa internos****: O* teste do mecanismo de pesquisa valida a exatidão e integridade da pesquisa, as propriedades de manipulação de erros do mecanismo de pesquisa e os recursos avançados de pesquisa, ou a integração com a pesquisa externa (Google).

# Teste de Configuração do Servidor/Nuvem

Os servidores/serviços estão com todas as bibliotecas, sdks,etc..na VERSÃO CORRETA?

O WebApp é totalmente compatível com o sistema operacional do servidor ou da cloud?

As medidas de segurança do sistema (por exemplo, firewalls ou criptografia) permitem que o WebApp execute e atenda os usuários sem interferência ou degradação do desempenho?

O WebApp foi testado com a configuração do servidor distribuído (se houver) que foi escolhido?

Os scripts WebApp do lado do servidor são executados corretamente?

Os logs de erros e da aplicação estão sendo gerados corretamente?

Se os servidores proxy/api gateway/etc forem usados, as diferenças em suas configurações foram tratadas com o teste no local?

# Mais Testes

**Testes de Segurança**: Simulações de ataque, análise de bugs e vulnerabilidades pré-existentes, seja do lado do cliente, seja do lado do servidor. Ex: Firewall, Autenticação, Criptografia.



**Testes de Performance**: Medir os tempos de respostas de acesso aos diversos módulos do sistema e entender como isso afeta a utilização do sistema.

# Teste de Carga vs Estresse

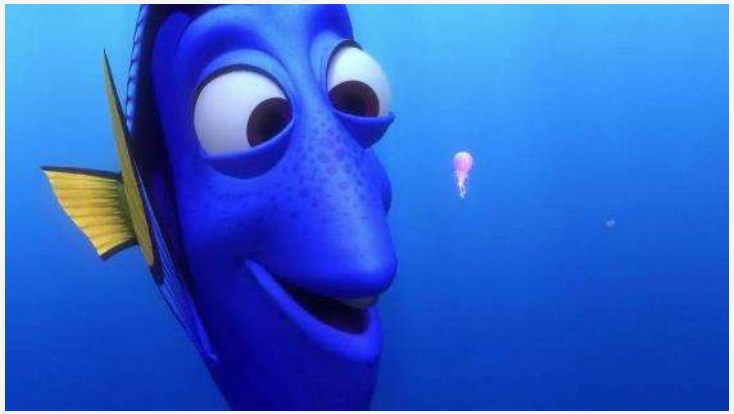
**Testes de Carga**: Simular a utilização massiva do sistema, quantidade conexões, volume de dados, volume de transações.



**Testes de Estresse**: Simular a utilização massiva do sistema mas buscando os limites para entender como se dá a degradação do sistema e o que ocorre quando os limites são atingidos, ou seja, se ocorrem falhas, lentidões, etc.

# Teste de Regressão

A medida que novos módulos são inseridos, novos “caminhos” são criados e desta forma é necessário testar novamente um conjunto de cenários para garantir que tudo continua funcionando corretamente.





“Não pode ter erro antigo voltando!”

“Cliente não quer ficar toda hora recomeçando...”

Teste de Regressão, RETESTA tudo que JÁ estava funcionando antes da alteração;

Cuidado! Teste de regressão precisa ser automatizado, caso contrário, faça apenas dos itens principais.

# Teste de Fumaça (Compilações Diárias]

Uma compilação inclui todos os arquivos de dados, bibliotecas, módulos reutilizáveis e componentes projetados necessários para implementar uma ou mais funções do produto. Uma série de testes é projetada para expor erros que impedirão a construção de executar adequadamente sua função.

A intenção deve ser descobrir erros bloqueadores que tenham a maior probabilidade de atrasar as entregas.  A construção é integrada a outras construções e todo o produto e testado diariamente, normalmente de forma automatizada.

**INTEGRAÇÃO CONTÍNUA (Guarde esse nome)**

# Exemplo de Arquivos de Testes

# Teste integrado

A maioria dos testes que vimos, são testes integrados.

Entender se as partes somadas funcionam corretamente. Caso não seja feita o teste unitário, a complexidade/volume de erros nesta fase pode aumentar muito:

A forma de construção da Aplicação afeta a forma de realizar o teste integrado.

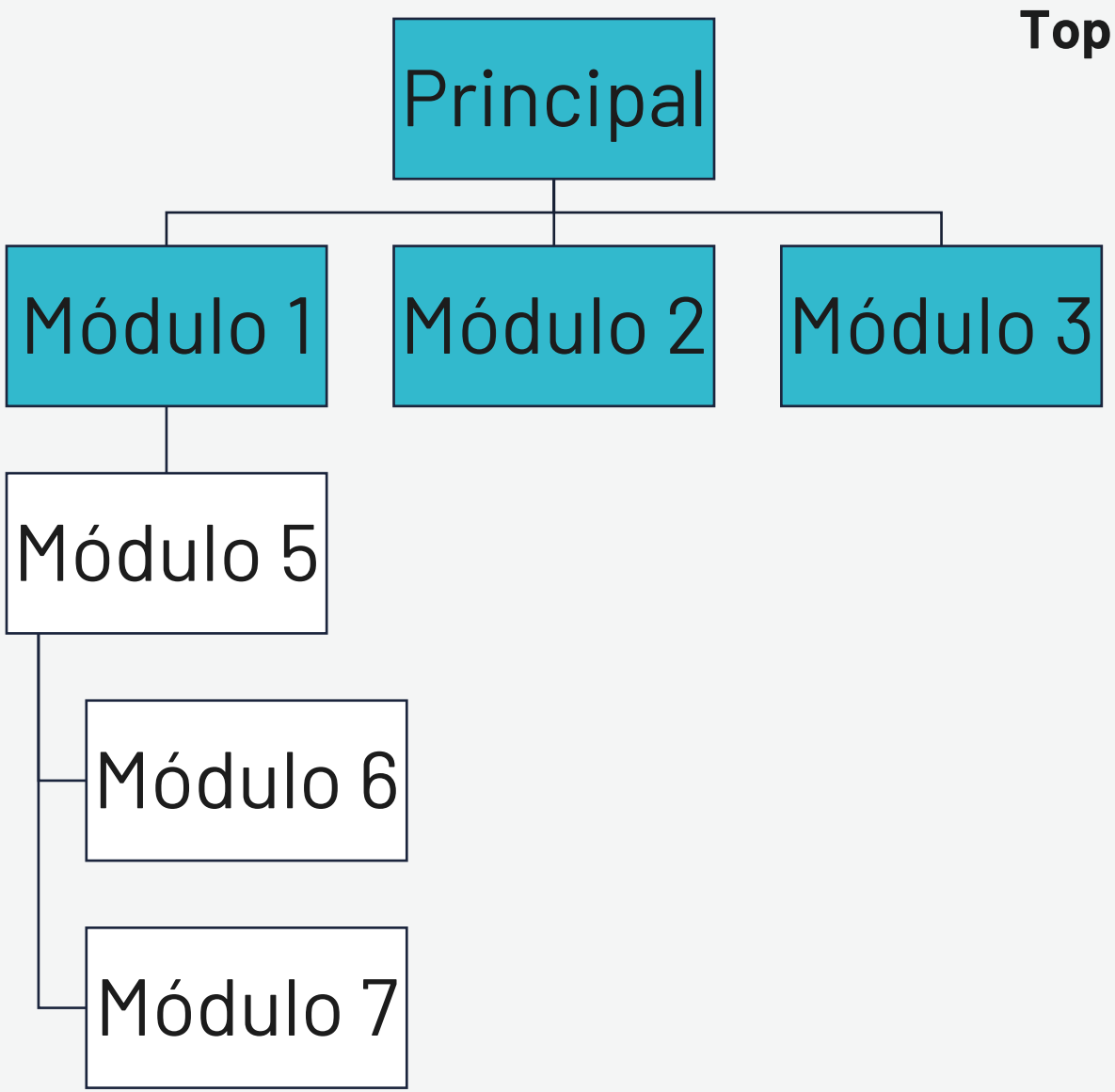
Formas:

- Big Bang
- Incremental

# Teste integrado: Big Bang

- Desenvolve toda a aplicação e depois começa a testar!
- Normalmente se testam as partes isoladas
- Por mais estranho que pareça, é bem comum!

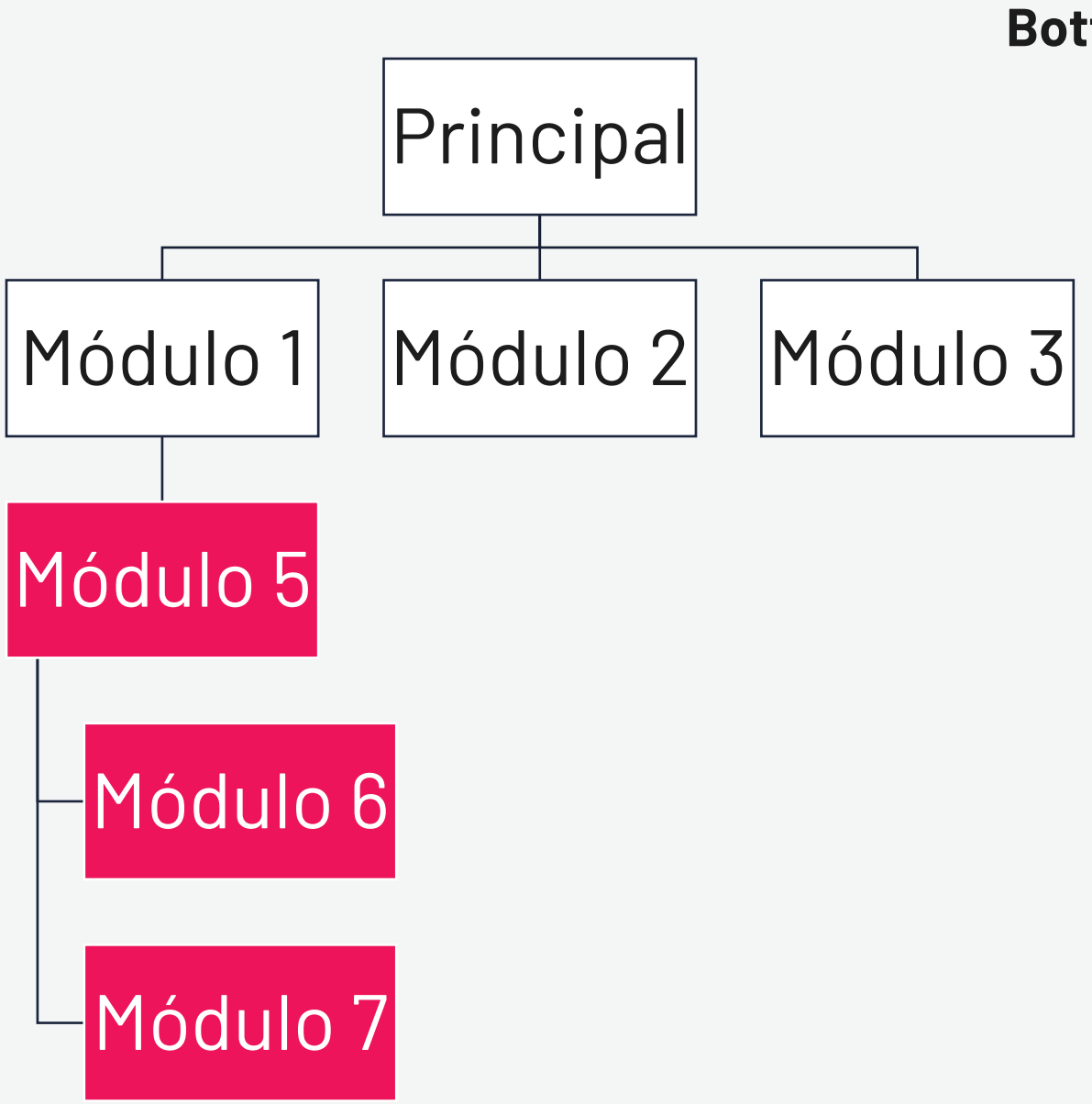
# Teste integrado: Integração descendente




**Top Down**

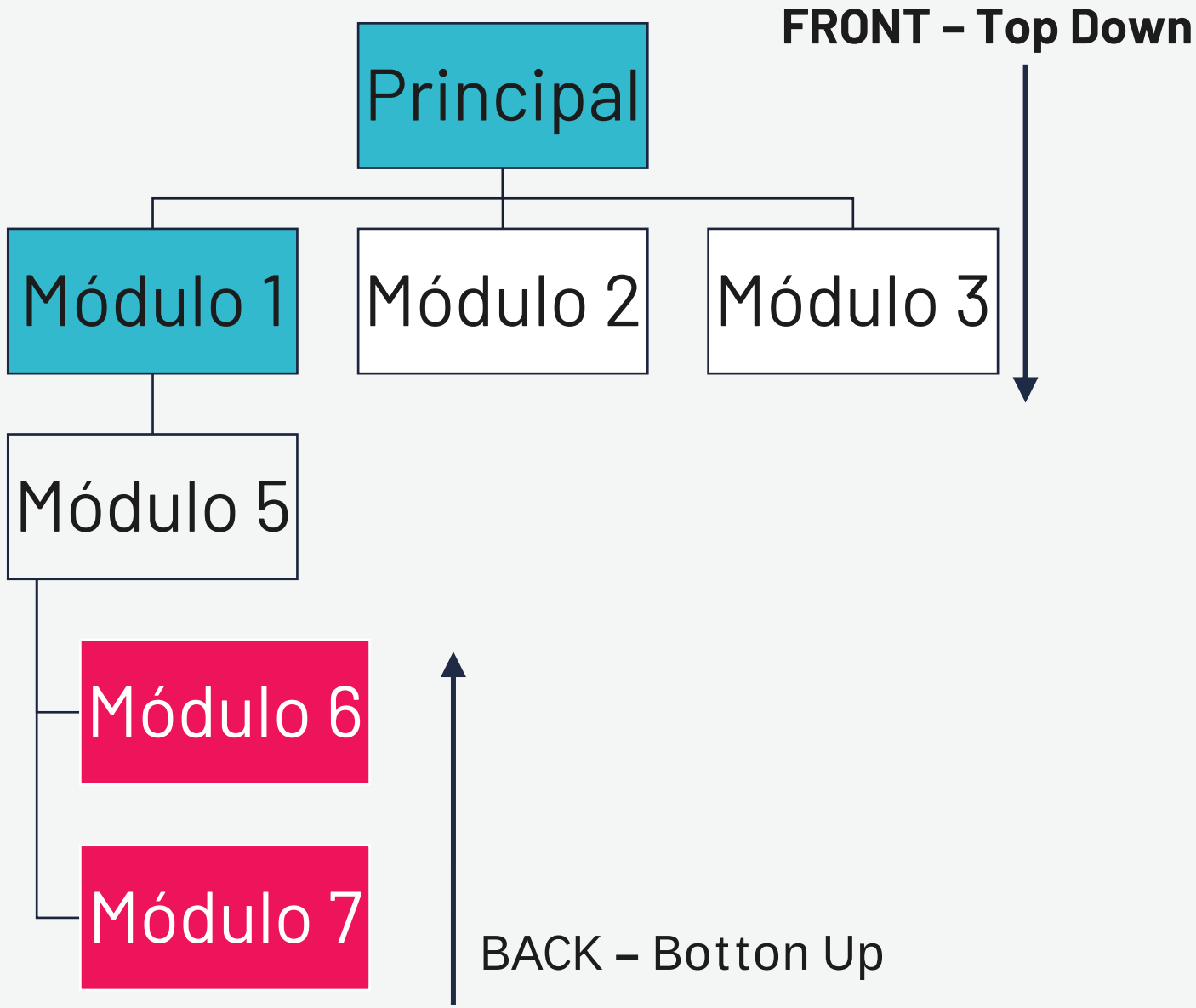
1. **Módulo de Controle Principal é utilizado como um testador**
2. **Os módulos podem ser substituídos por módulos pseudocontroladores dependendo da abordagem (stubs/mocks)**
3. **Os testes são realizados a medida que cada módulo é integrado.**
4. **Quando um novo módulo é integrado um conjunto de testes e realizado novamente (pode ser regressão)**

# Teste integrado: Integração ascendente




1. **O Módulos de baixo nível são agrupados (builds) conforme a função que executam, ou seja, dependências. (Exemplo em azul)**
2. **Precisa ser escrito um PseudoControlador para coordenar a entrada e saída de teste. (drivers/mocks)**

# Teste Integrado: Sandwich




# Comparação dos Testes

| | Bottom-up | Top-down | Big Bang | Sandwich |
|---|---|---|---|---|
| Integração | Cedo | Cedo | Tarde | Cedo |
| Tempo para o funcionamento básico do programa | Tarde | Cedo | Tarde | Cedo |
| Drivers (mocks) – dublê do front | Sim | Não | Sim | Sim (Alguns) |
| Stubs (mocks) – dublê do back | Não | Sim | Sim | Sim (Alguns) |
| Trabalho em paralelo no início | Médio | Baixo | Alto | Médio |
| Capacidade de Testar caminhos particulares | Fácil | Difícil | Fácil | Médio |

https://qualidadebr.wordpress.com/2009/05/10/tecnicas-de-integracao-de-sistema-big-bang-e-sandwich/

https://pt.stackoverflow.com/questions/157330/qual-o-conceito-de-stubs-e-de-drivers-em-testes-de-integra%C3%A7%C3%A3o

# Agradeço
a sua atenção!

**Gerson Santos**

gerson.santos@sptech.school

SÃO PAULO TECH SCHOOL