SÃO
PAULO
TECH
SCHOOL

Engenharia de Software

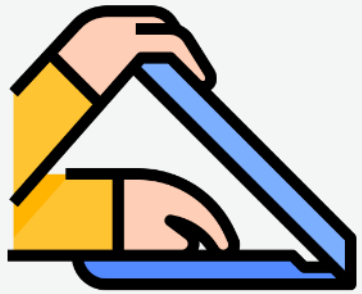
Qualidade de Software Pt2

Aula 11

Fábio Figueredo

Fabio.figueredo@sptech.school

Regras básicas da sala de aula



1. Notebooks Fechados: Aguarde a liberação do professor;

2. Celulares em modo **silencioso e guardado**, para não tirar sua atenção

- Se, caso haja uma situação urgente e você precisar **atender ao celular, peça licença para sair da sala** e atenda fora da aula.

3. Proibido usar Fones de ouvido: São liberados apenas com autorização do professor.

4. Foco total no aprendizado, pois nosso tempo em sala de aula é precioso.

- Venham sempre com o **conteúdo da aula passada em mente** e as atividades realizadas.
- Tenham caderno e caneta;
- **Evitem faltas** e **procure ir além** daquilo que lhe foi proposto.
- **Capricho, apresentação e profundidade** no assunto serão observados.
  - "**frequentar as aulas** e demais atividades curriculares aplicando a **máxima diligência no seu aproveitamento**" (Direitos e deveres dos membros do corpo discente - Manual do aluno, p. 31)

Regras básicas da sala de aula

As aulas podem e devem ser divertidas! Mas:

- **Devemos respeitar uns aos outros** – cuidado com as brincadeiras.
  - "observar e cumprir o regime escolar e disciplinar e comportar-se, dentro e fora da Faculdade, **de acordo com princípios éticos condizentes**" (Direitos e deveres dos membros do corpo discente - Manual do aluno, p. 31)

Boas práticas no Projeto

COMPROMISSO

COM VOCÊ: **ARRISQUE**, NÃO TENHA MEDO DE ERRAR

COM OS PROFESSORES: ORGANIZE A **ROTINA** PARA OS ESTUDOS

COM OS COLEGAS: **PARTICIPAÇÃO ATIVA** E PRESENTE

COM O PROJETO: **RESPEITO** E FLEXIBILIDADE

Boas práticas no Projeto

Reações **defensivas** não levam ao envolvimento verdadeiro!

Transforme cada problema e cada dificuldade em uma **OPORTUNIDADE** de aprendizado e crescimento.

EVITE:

- Justificativas e Desculpas
- Transferir a culpa
- Se conformar com o que sabe
- Se comparar com o outro

Dica: **Como ter sucesso** (*Maiores índices de aprovações*)

Comprometimento

- Não ter faltas e atrasos. Estar presente *(Não fazer 2 coisas ao mesmo tempo)*
- Fazer o combinado cumprindo os prazos

Atitudes Esperadas:

- **Profissionalismo**: Entender que não é mais ensino médio *(Atitude, comportamento, etc.)*
- **Não estar aqui só pelo** estágio ou pelo diploma
- Não ficar escondido: precisa **experimentar**
- **Trabalhar** em grupo e **participar** na aula
- **Não ser superficial** ou "achar que sabe"
- **Não se enganar** utilizando de "cola"
- Assumir a responsabilidade: Não colocar a culpa em outra coisa. **Não se vitimizar.**

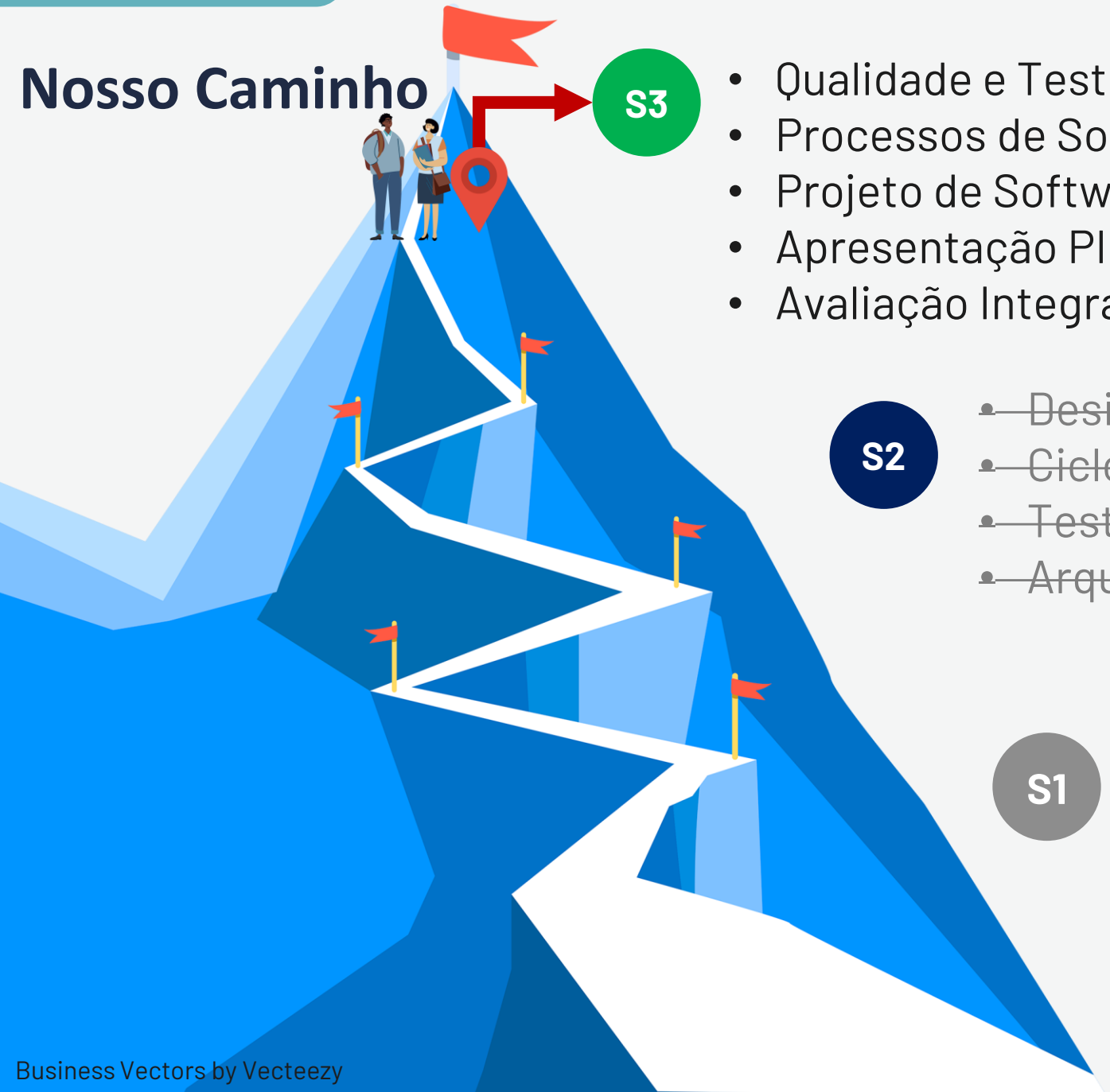
Break

> 10 minutos, definidos pelo professor.

Obs: Permanecer no andar, casos específicos me procurar.

Atenção: Atrasados deverão aguardar autorização para entrar na sala.

Nosso Caminho



S3

- Qualidade e Testes
- Processos de Software
- Projeto de Software
- Apresentação PI
- Avaliação Integrada

S2

- ~~Design Inclusivo~~
- ~~Ciclo de Vida de Produto~~
- ~~Teste de Usabilidade~~
- ~~Arquitetura de Software~~

S1

- ~~Introdução a Engenharia de Software~~
- ~~Conceitos de UI e UX~~
- ~~Fatores Humanos~~
- ~~Personas~~
- ~~Design de Interface Básico~~

Palavra-chave dessa Sprint:

PRAGMATISMO

prag·má·ti·co

. adjetivo

1. Relativo à pragmática ou ao pragmatismo.

2. **Que tem motivações relacionadas com a ação ou com a eficiência. = PRÁTICO**

. adjetivo e substantivo masculino

3. Que ou quem revela um sentido prático e sabe ou quer agir com eficácia.

Frase dessa sprint:

Aprender/Ensinar processos, métodos e ferramentas para construção e manutenção de **softwares profissionais**.

No episódio anterior...

O que é Qualidade?

No Aurélio

1 - Maneira de ser boa ou má de uma coisa.

6 - Aquilo que caracteriza uma coisa.

10 - Atributo, modalidade, virtude, valor.

Em Software

- **A qualidade engloba os requisitos, especificações e o design do sistema.**

Satisfação do Cliente =
Produto compatível + Boa qualidade (Produto útil)
+ Entrega dentro do orçamento e do cronograma

Produto Útil

Um produto útil fornece o conteúdo, funções e recursos que o usuário final deseja.

Um **produto útil** sempre **satisfaz** os **requisitos** que foram **explicitamente declarados** pelas partes interessadas.

Além disso, satisfaz um conjunto de **requisitos implícitos** (por exemplo, facilidade de uso, convenções, etc) **que são esperados de todos os softwares de alta qualidade**.

Por exemplo, o campo CEP em um formulário.

Atributos de Qualidade

1. Usabilidade
   - Fatores Humanos
   - Estética
   - Documentação
2. Confiabilidade
   - Frequência e severidade das falhas (* MTBF, * MTTR)
3. Desempenho
   - Velocidade de Processamento
   - Escalabilidade
   - Disponibilidade
4. Facilidade de Suporte
   - Extensibilidade – Capacidade de receber novas funcionalidades
   - Compatibilidade
   - Reparabilidade

Os Fatores da Qualidade de McCall.

*** MTBF** – Tempo médio entre falhas / **MTTR** – Tempo médio entre os reparos

Como medir a qualidade?

Você precisa estabelecer métricas!

As equipes de projeto precisam desenvolver um conjunto de perguntas específicas para avaliar o grau em que cada fator de qualidade do aplicativo foi satisfeito.

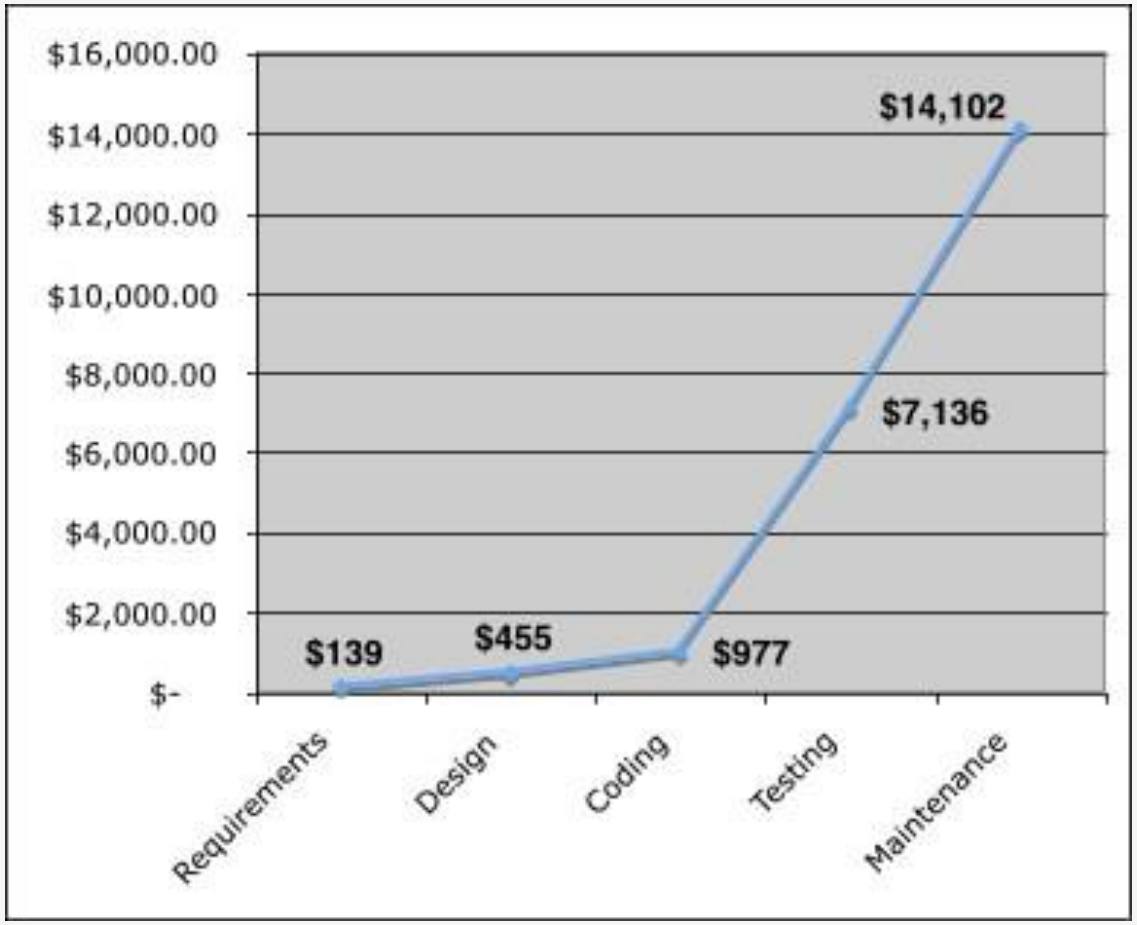
Medidas subjetivas de qualidade de software podem ser vistas como pouco mais que uma opinião pessoal.

Métricas de software representam medidas indiretas de alguma manifestação de qualidade e tentam quantificar a avaliação da qualidade do software.

Exemplo de Métrica: Qualidade da aula baseado na quantidade de bocejos durante uma hora de aula.

Prevenir é melhor que remediar?

Quanto custa a qualidade? $$$$$



Veja no exemplo que o custo de reparação fica extremamente alto logo depois que sai da mão do desenvolvedor.

Conseguem pensar o por quê?

O PO está na empresa para garantir que isso não aconteça!

Quanto custa a Qualidade?

Custo de Prevenção

- Planejamento de qualidade (QA, Teste Unitário, PO, Scrum Master, ...)
- Equipamentos/Ambientes de teste
- Treinamento

Custos de Avaliação (Internos)

- Testes e Depuração
- Coleta de dados e métricas

Custo de Falha (Internos)

- Retrabalho e Correção
- Efeitos colaterais (altera uma coisa, estraga outra)
- Coleta de Dados e Métricas
- Desgaste da Equipe

Custos Externos

- Resolução de reclamações
- Retorno e substituição do produto
- Suporte (SAC)
- Reputação
- Satisfação do Cliente
- Responsabilidade Civil

https://www.google.com/url?sa=i&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwi6gODV6fXhAhXsGLkGHabKD8oQjRx6BAgBEAU&url=http%3A%2F%2Fwww.folhavitoria.com.br%2Feconomia%2Fblogs%2Fgestaoeresultados%2F2012%2F02%2F15%2Fcustos-da-qualidade-1a-etapa%2F&psig=AOvVaw2Ha92fZgovRT_oLGGyRDPC&ust=1556645269822925

Exemplos de Critérios de Aceitação

BDD (Behavior Driven Development)

Dado uma condição, Quando faço alguma ação,
Então espero algum resultado.

Para a aula de hoje...

Tópicos da Aula

- Bug e Defeito
- Tipos de Teste
- Atividade

Antes de começar... Vamos alinhar

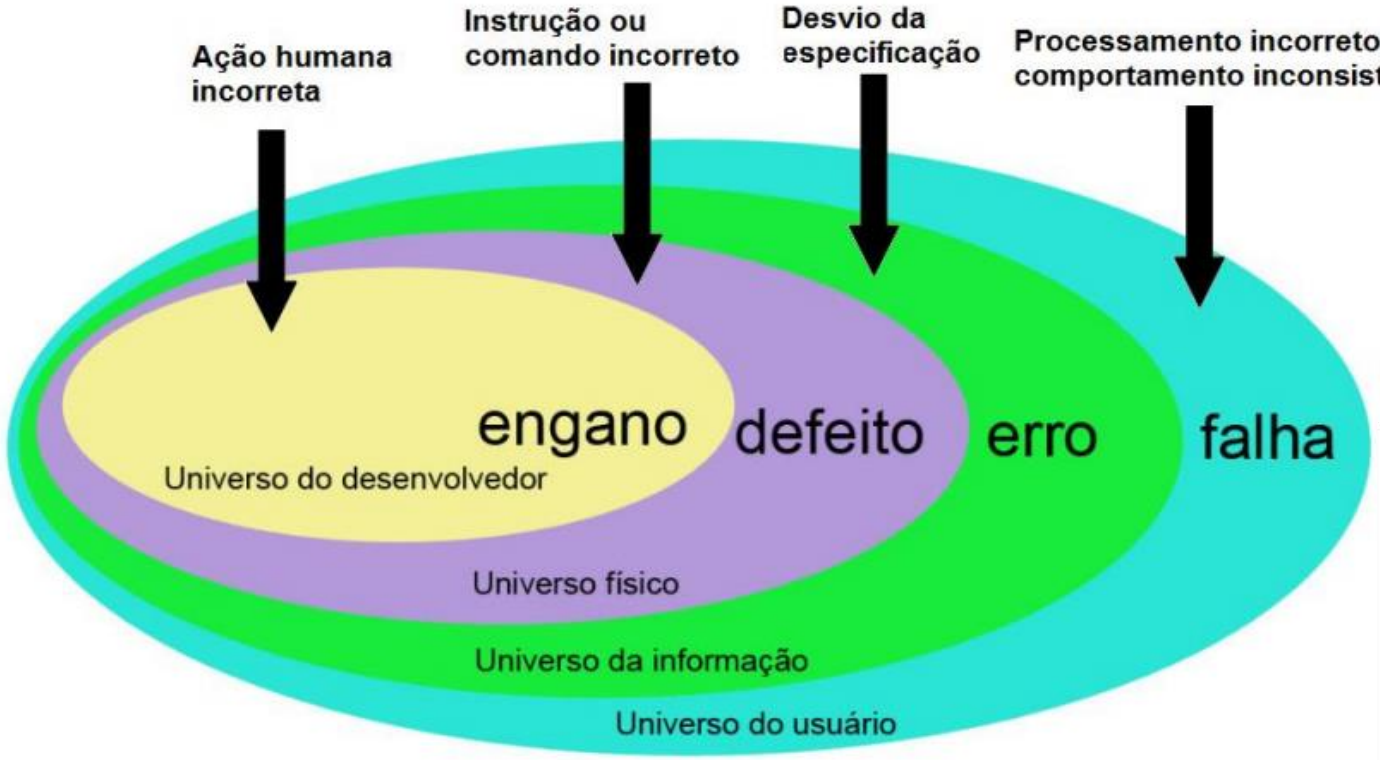
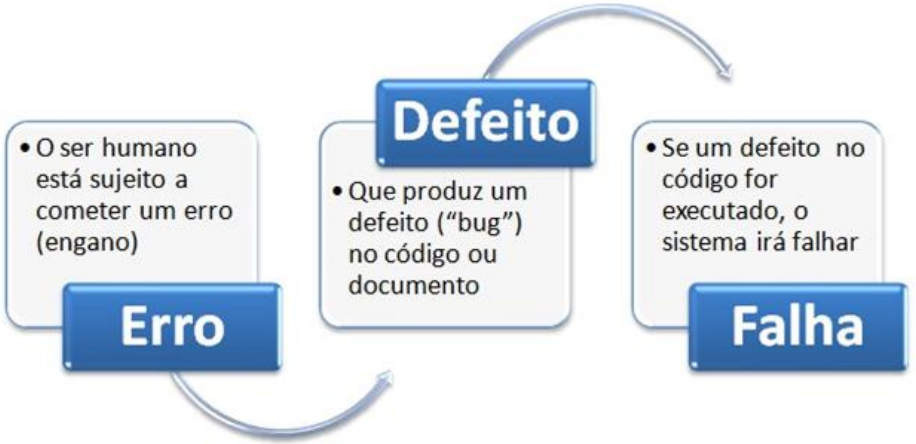


Figura 1 - Engano x defeito x erro x falha
Fonte: Adaptado de Barbosa et al., 2000.



Essas definições não são um consenso.

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.

Barry Boehm [Software Engineering Economics - 1981] afirma isso de outra maneira:

Verificação: "Estamos construindo o produto corretamente?"

Validação: "O resultado está (ou será) correto?"

Verifica o andamento e Valida o Resultado!

Testes

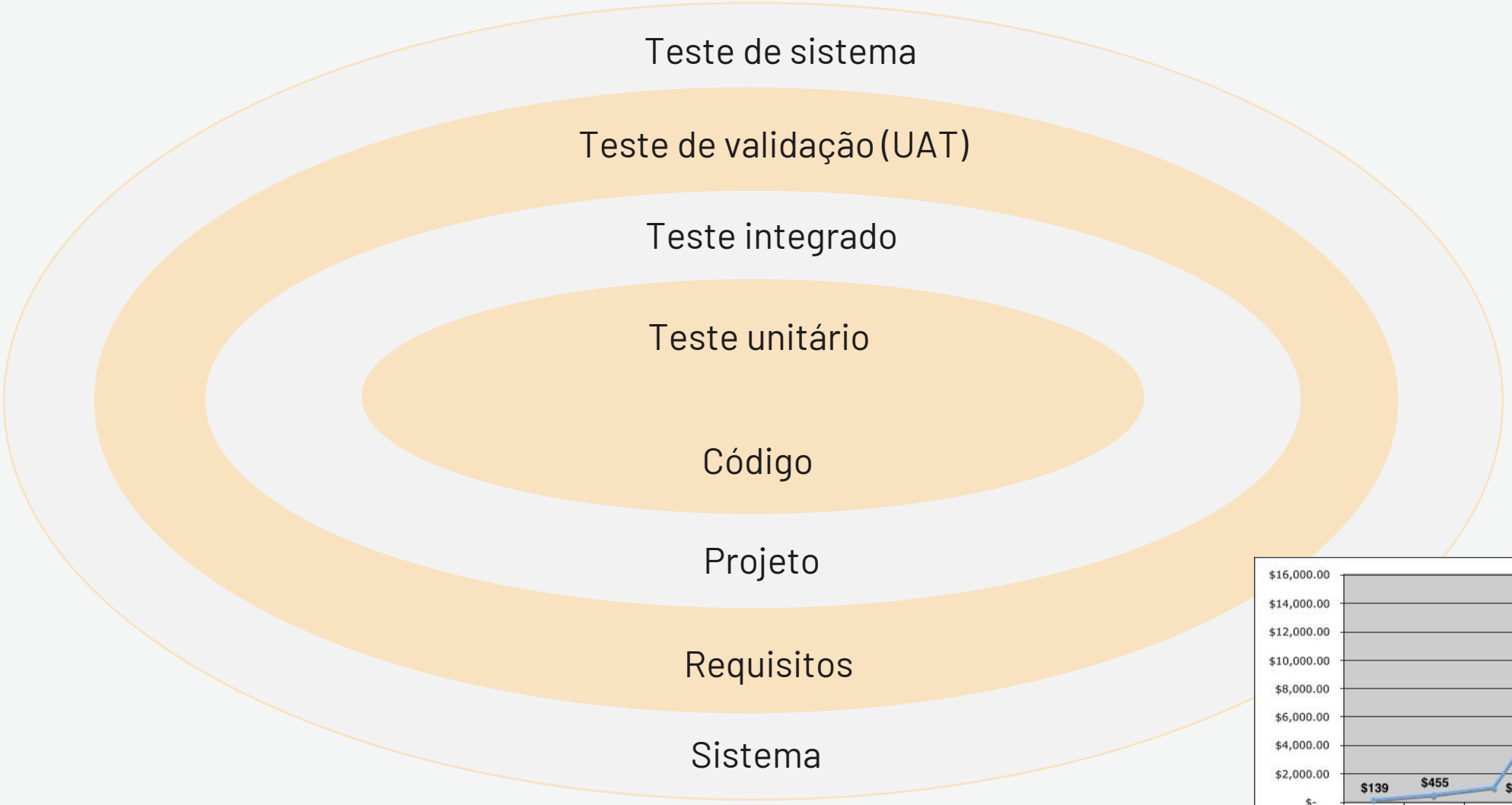
TESTES SÃO FEITOS PARA ENCONTRAR ERROS, NÃO ADIANTA TESTAR SOMENTE O CAMINHO FELIZ!

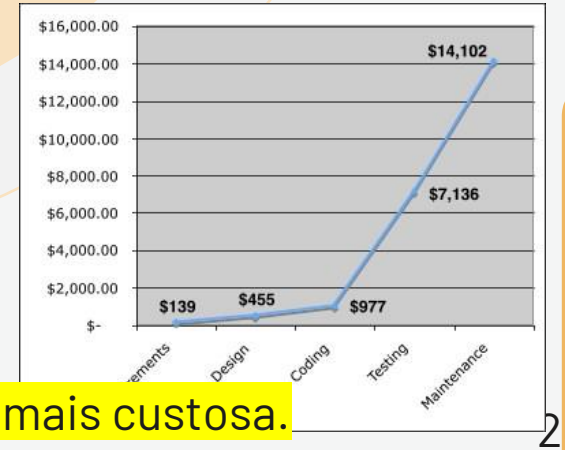
“Teste é o processo de exercitar um programa com a intenção específica de encontrar erros antes de entregar o mesmo para o usuário final.”

Sommerville

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

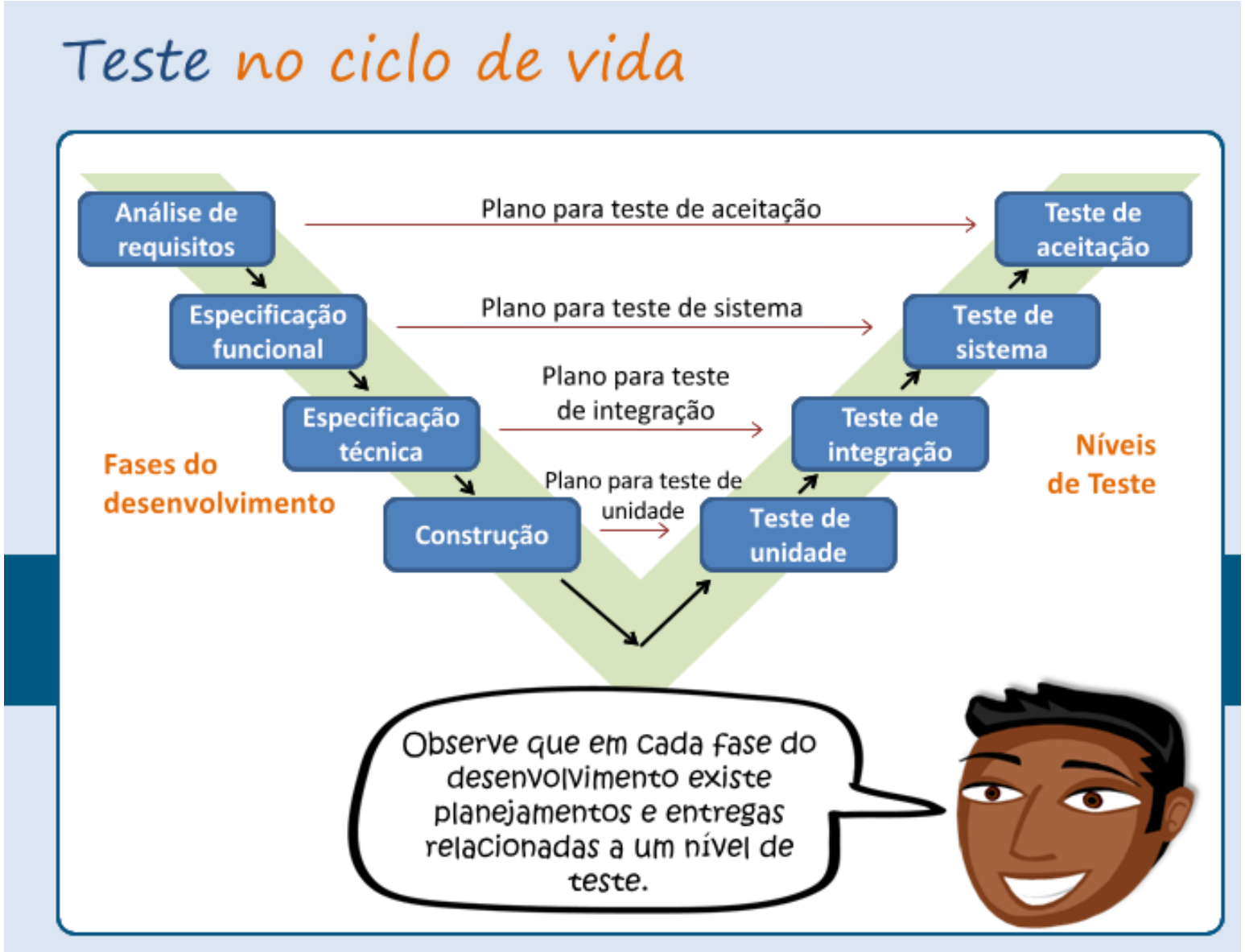
Níveis de Teste/Estratégia de Teste





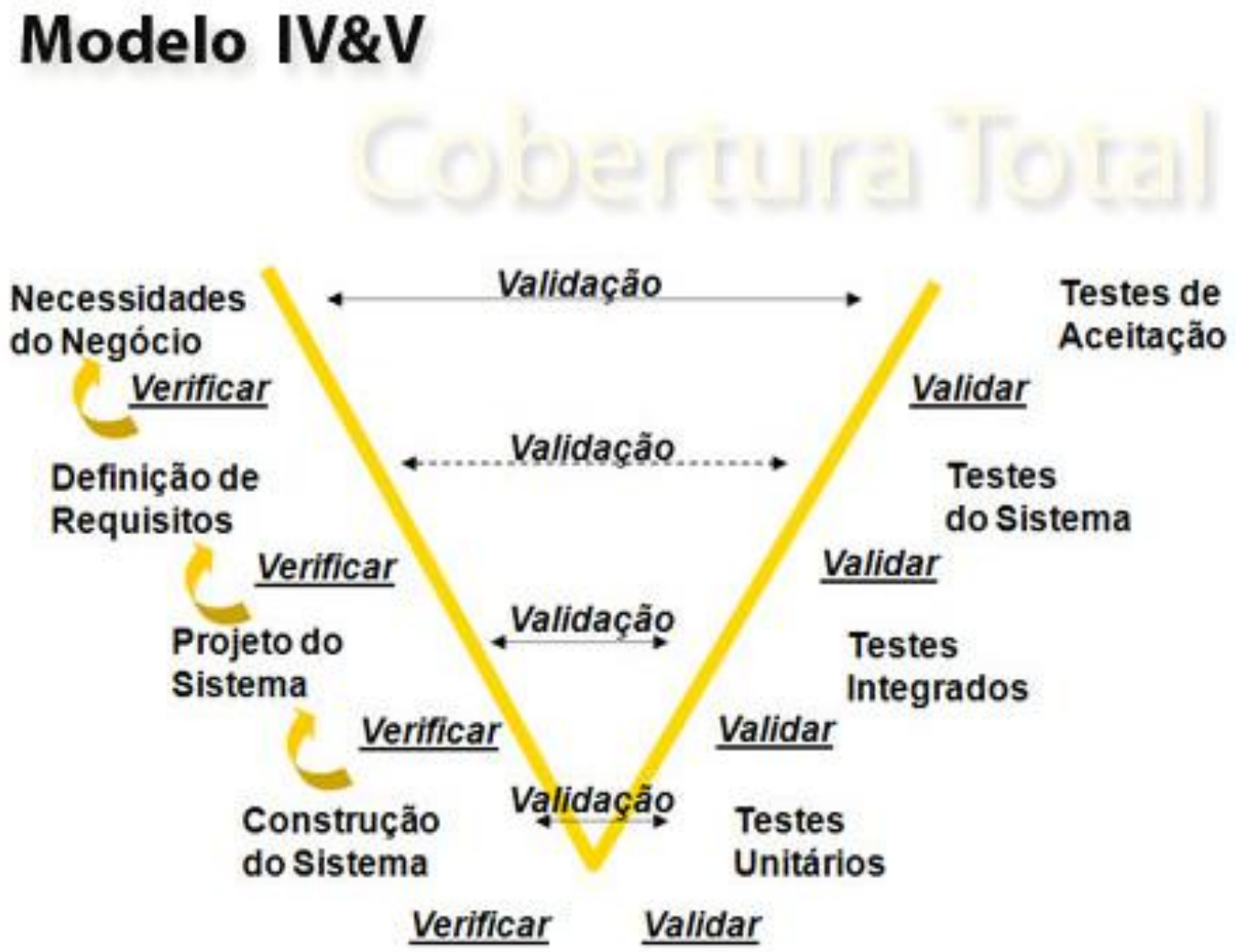
Não fazer algum dos teste, se encontrado erro nas etapas posteriores a correção será bem mais custosa.

Níveis de Teste/Estratégia de Teste



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

Verificação & Validação



Verifica o andamento e Valida o Resultado!

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 (Equipe de QA)**
- Muito comum o **Analista de Negócios/PO** também testar representando o cliente (ele simula também uma pré-validação/pré-homologação)

Tipos de Teste

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 – Quem faz, fica mais confiante com a entrega!**
- 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 – **Crie o teste enquanto está dando erro, antes de resolver, resolva e teste novamente;**

Testes de Aplicações Web (Mobile é similar)

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

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. **Interface do usuário (usabilidade)**
3. **Compatibilidade**
4. **Regressão**
5. **Navegação**
6. **Configuração**
7. **Segurança**
8. **Performance**
9. **Carga**
10. **Stress**

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?

O estilo estético do conteúdo está em conflito com o estilo estético da interface?

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 (validações gerais de conteúdo)*: 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.

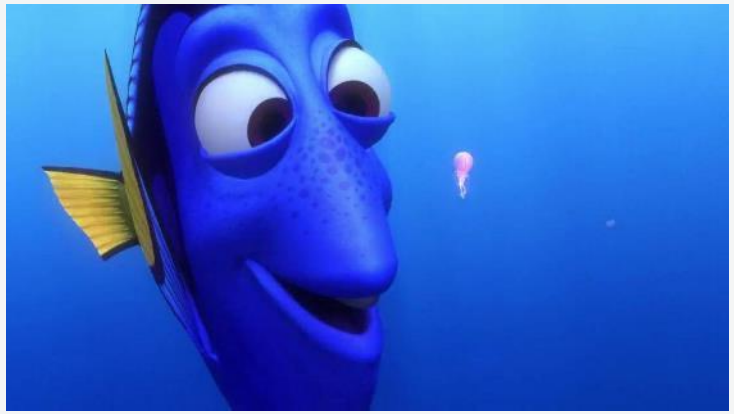
Teste de Carga vs Estresse

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)

Hardening

Hardening é um **processo de mapeamento das ameaças**, mitigação dos riscos **e execução das atividades corretivas**, **com foco na infraestrutura** e **objetivo principal** de torná-la preparada para **enfrentar tentativas de ataque**.

Normalmente, o processo inclui **remover** ou **desabilitar** nomes ou **logins** de **utilizadores** que **não estejam mais em uso**, além de serviços desnecessários.

Exemplo Linux:

Criar volumes separados para /var, /var/log, e /home.

Quaisquer diretórios em que usuários não administradores tenham acesso de gravação devem ser separados do volume raiz para limitar o

impacto desses volumes sendo preenchidos.

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.

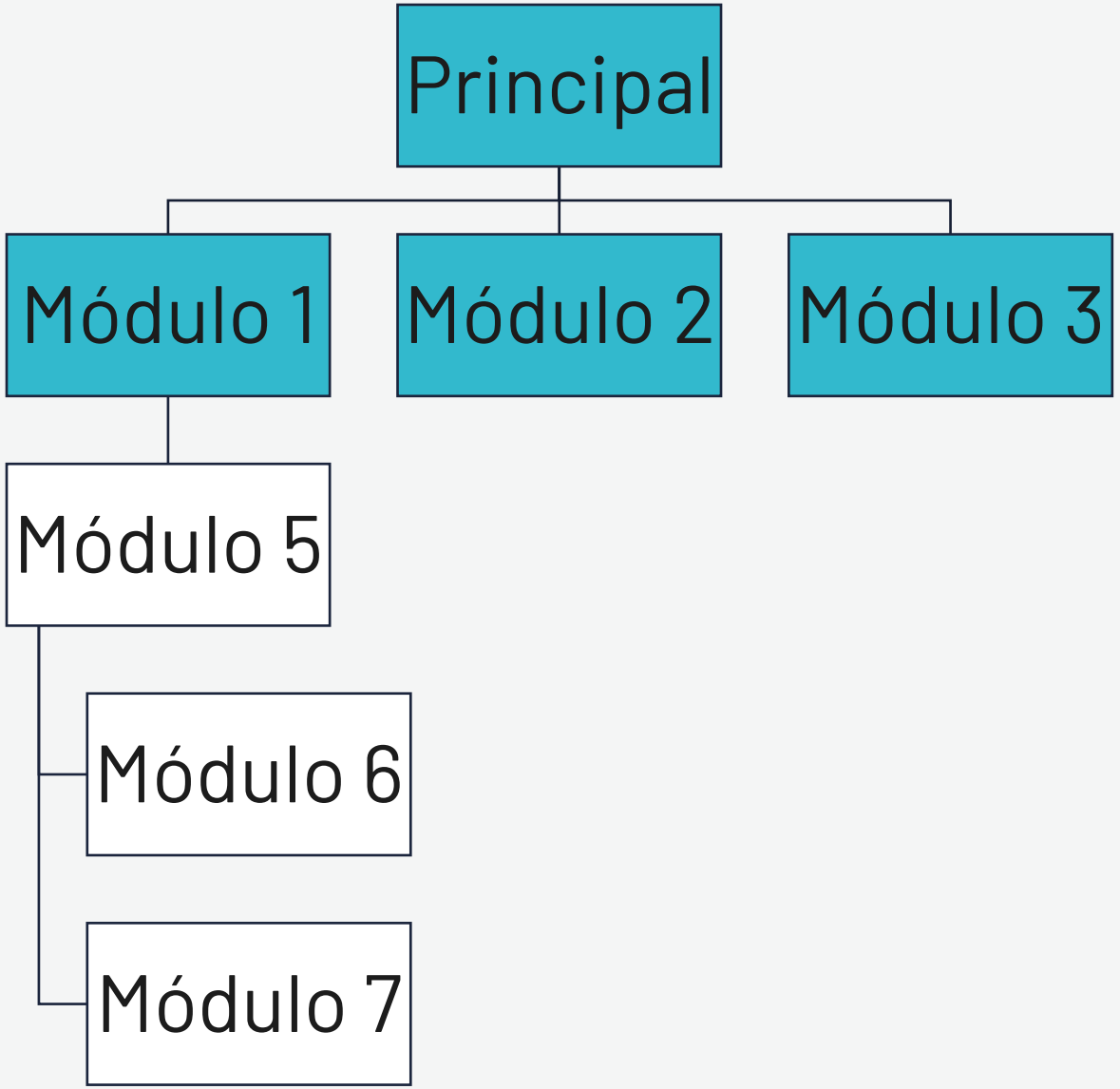
Formas:

- 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

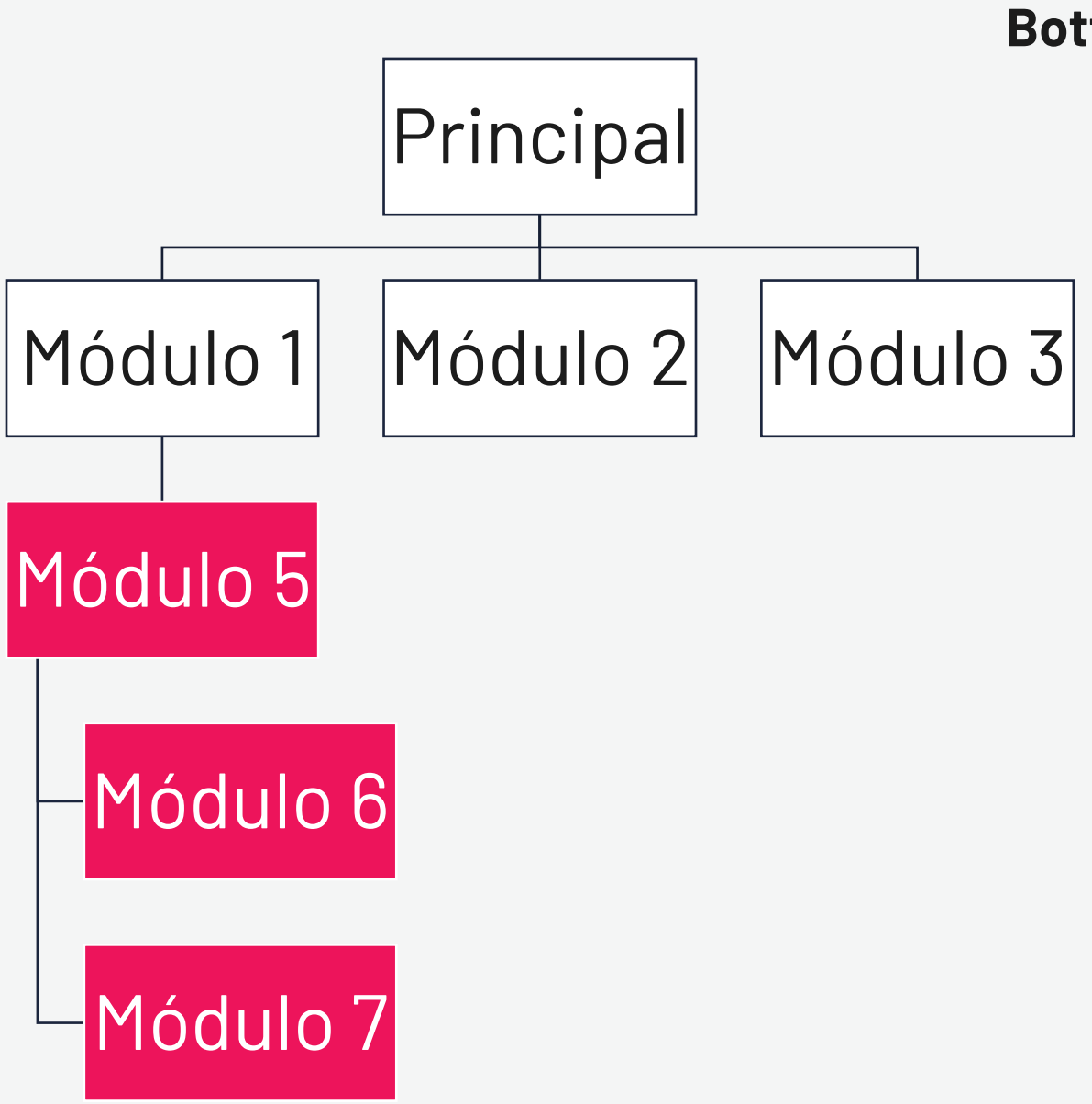


1. O teste começa com o módulo de alto nível e vai integrando os componentes internos aos poucos.
2. Os módulos dependentes são substituídos por mocks ou stubs, que simulam a funcionalidade dos módulos.

Mocks simulam comportamentos – por exemplo, em uma transação de cartão de crédito, um mock pode simular o pagamento com sucesso, falha, etc.

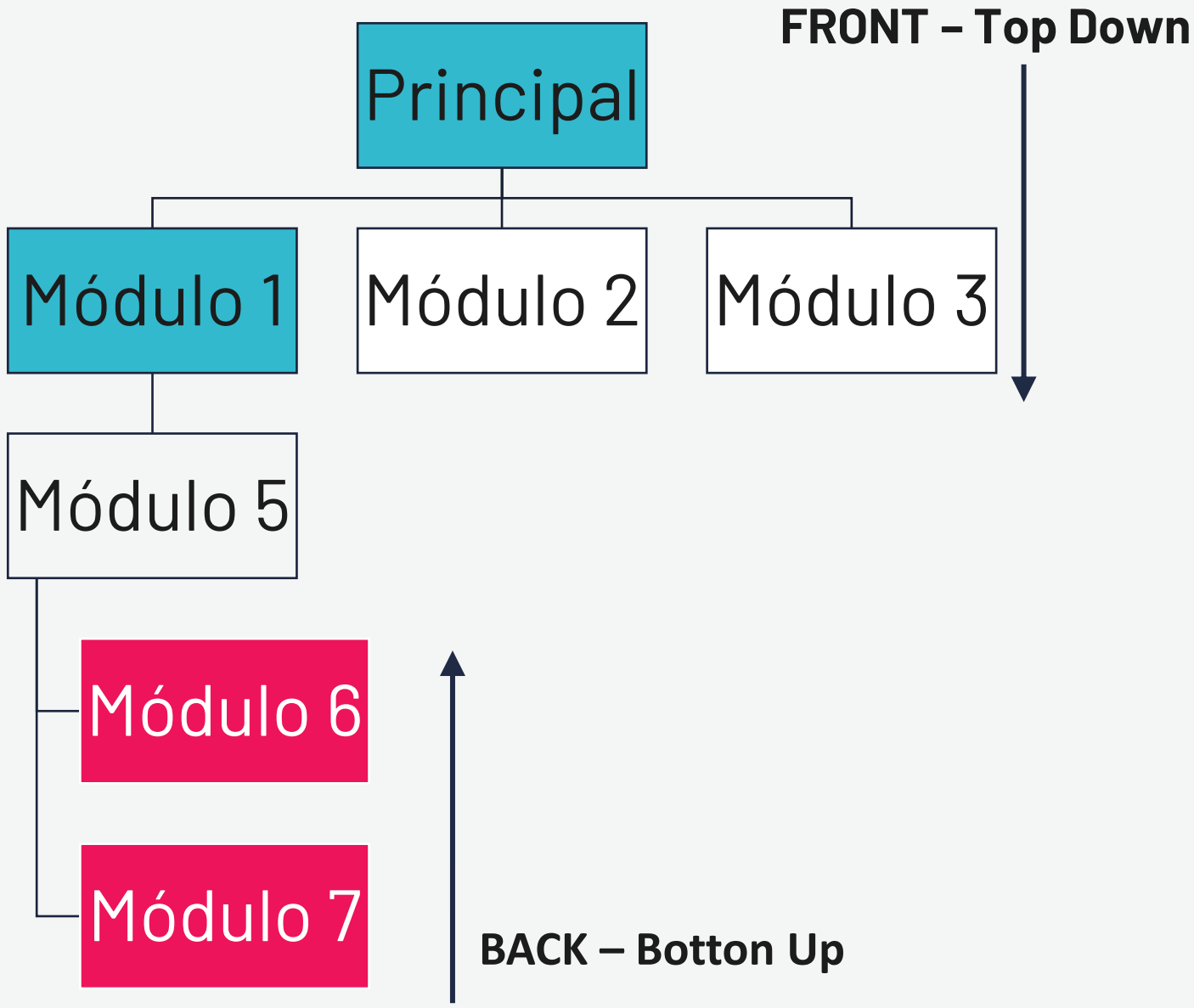
Stubs retornam valores fixos – por exemplo, em uma transação de envio de e-mail, um stub pode responder true quando o método é chamado. Permite testar a funcionalidade sem a necessidade de um servidor de e-mail.

Teste Integrado: Integração ascendente



1. Começa com as unidades de software mais baixas, como funções individuais ou módulos, e as integra gradualmente em grupos maiores, testando cada grupo à medida que são construídos e integrados.
2. As unidades mais básicas do software são testadas primeiro, em seguida, as unidades dependentes são adicionadas gradualmente e testadas em conjunto. Isso é feito até que todo o sistema esteja integrado e testado.

Teste Integrado: Sandwich



Combina a integração ascendente e descendente. Inicia com o teste ascendente e finaliza com o descendente garantindo que o sistema funcione corretamente em cada nível de integração.

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

Kahoot!

Agradeço

a sua atenção!

Fábio Figueredo

fabio.figueredo@sptech.school

SÃO PAULO TECH SCHOOL