

# **CURSO DE TESTES AUTOMATIZADOS**

## **UM POUCO SOBRE TESTES**

**Wagner Costa**  
wcaquino@gmail.com

# VALE A PENA TESTAR?

- Se temos um software pequeno, cuja lógica pode ser facilmente entendida, será que realmente precisamos testar?

# O EFEITO BORBOLETA



# UM CENÁRIO COMUM

“O software já está pronto... só falta testar!”

- Existe uma regra básica sobre software que diz:
  - “Software que não foi testado é **garantidamente** software que **não funciona!**”
- Uma abordagem (usada por muitos) consiste em manualmente explorar o programa em busca de erros...

# COMO LIDAR COM MUDANÇAS?

- Mas como saber se um software continua funcionando após alguma mudança?
- O software é composto por várias entidades, as quais interagem entre si para atingir um objetivo mais amplo;
- Logo, estas entidades estão, de alguma maneira (direta ou indiretamente), ligadas;
- Alterar uma destas entidades pode “quebrar” uma outra que nem sabíamos que tinha relação com a entidade alterada;

# SUGESTÕES PARA CUIDAR DAS MUDANÇAS

- Assuma que sua mudança é tão pequena que nenhuma outra parte do código vai senti-la;
- Assuma que alguém irá testar o código por você;
- Assuma que alguém irá descobrir o problema por acaso;
- Coloque o software no ambiente de produção e espere pra ver se alguém reclama;

**Algumas dessas sugestões nos traz segurança de que o software funciona?**

# MURPHY É IMPLACÁVEL

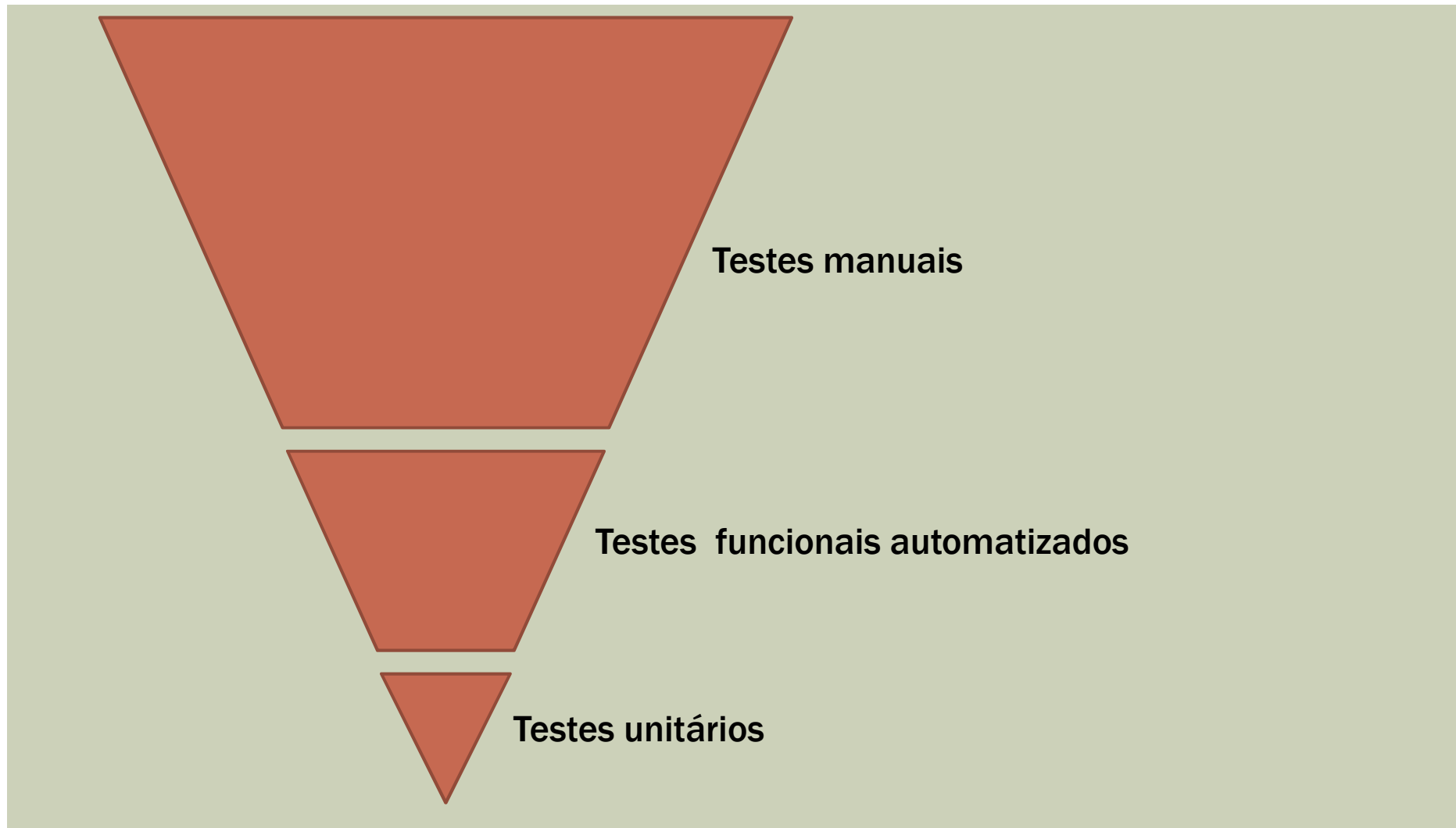


# COMO EVOLUIR COM SEGURANÇA?

- Testes trazem segurança à evolução do software:
  - A cada novo passo, você sabe que o que ficou para trás está funcionando corretamente e não vai lhe atrapalhar no futuro;
- Alterar código testado é mais seguro;
  - Se nossa mudança quebrou alguma parte do software, os testes dizem imediatamente o que quebrou e sob quais circunstâncias:
- Código testado pode ser modificado sem medo:
  - Nos dá coragem para mudar:
    - Em um projeto de software, mudança é a regra!
  - Garantia de um refactoring bem sucedido.



# PIRAMIDE INVERTIDA TESTES TRADICIONAIS



# PROBLEMA ORGANIZACIONAL

- Existe uma acomodação da equipe de desenvolvimento supondo que os erros serão detectados pela equipe de teste.

**PODE FICAR TRANQUILO,  
O FURO ESTÁ DO LADO DELES !!!**



# MANIFESTO ÁGIL

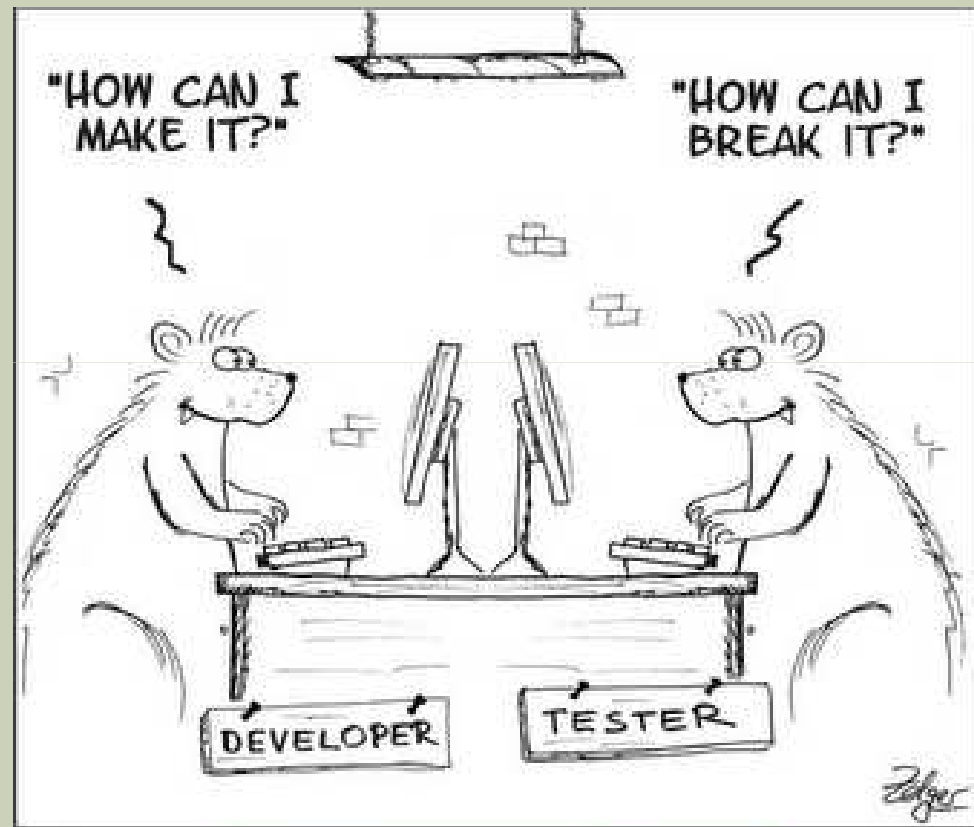
“(...) **Software em funcionamento**  
mais que documentação abrangente (...)”



© Scott Adams, Inc./Dist. by UFS, Inc.

# DESENVOLVEDOR X TESTADOR

- Desenvolvedor não gosta de testar e testador não gosta de “codificar”.
- Com os testes caixa branca (TDD principalmente), os desenvolvedores passaram a criar testes.
- Com as ferramentas de testes funcionais automatizados, os testadores passaram a escrever testes em linguagens de programação.



They are not so much different,  
but they have different path for the same goal,  
to improve quality!!

# TEST-INFECTED



# PIRAMIDE DE TESTE ÁGIL



Fonte: [eliasnogueira.com](http://eliasnogueira.com)

# 10 PRINCIPIOS DOS TESTADORES ÁGEIS

- Prover feedback contínuo
- Entregar valor ao cliente
- Disponibilizar comunicação Face a Face
- Ter coragem
- Tornar tudo simples (KISS)
- Praticar a melhoria contínua
- Responder à mudanças
- Auto organizar-se
- Foco nas pessoas
- Divirta-se

# DOCUMENTAR ATRAVÉS DE TESTES?





# COMO DOCUMENTAR UM BUG?

- Seja preciso.
- Seja claro – explique de forma que outros possam reproduzir o erro.
- Um bug por relatório.
- Nenhum bug é simples o suficiente para não ser reportado.
- Separe claramente fatos de especulações.
- Um *printscreen* vale mais que mil parágrafos.

# BUGS HAVE FEELINGS TOO

## BUGS HAVE FEELINGS TOO

IF YOU FIND A BUG:  
REPORT IT

BUGS DON'T LIKE  
TO BE FORGOTTEN



IF YOU FIND A BUG:  
GET TO KNOW THEM

BUGS LIKE TO BE  
UNDERSTOOD



This ladybird  
has 3 spots

IF YOU FIND A BUG:  
TAKE A PHOTO

BUGS LIKE TO KEEP MEMORIES  
OF THE OCCASION



IF YOU FIND A BUG:  
GET TO KNOW THEIR MATES

BUGS ARE SOCIALITES



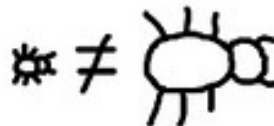
IF YOU FIND A BUG:  
REPORT IT QUICK

OTHERWISE BUGS SETTLE IN AND  
MAKE A HOME FOR THEM SELVES



IF YOU FIND A BUG:  
BE HONEST

BUGS DON'T LIKE  
GOSSIPS



IF YOU FIND A BUG:  
NOTE HOW YOU  
MEET THEM

BUGS ARE ROMANTICS



IF YOU FIND A BUG:  
DON'T IGNORE IT

BUGS CAN BITE IF  
NOT APPRECIATED



AG

**TESTAR OU NÃO TESTAR**

# O QUE GANHAMOS COM TESTES?

- Testes nos dão segurança para modificar código;
- Testes garantem que funcionalidades desejadas estão implementadas;
- Testes garantem que **bugs testados não existem**;
- Testes possibilitam uma rápida detecção de problemas;
- Testes são documentação extra para o projeto;
- Testes nos dão indícios da qualidade do software produzido;

# QUAL O INVESTIMENTO?

- Testar é chato!
- Testar é demorado:
- Os testes precisam evoluir junto com o projeto:
- Fazer bons testes é difícil:

# BENEFÍCIOS DO INVESTIMENTO EM TESTES

- Maior satisfação do usuário.
- Melhoria da imagem da empresa.
- Maior redução de incertezas que cercam o software.
- Redução do custo de manutenção em produção do produto entregue.

# TESTES AGREGAM VALOR O PRODUTO



# MAS... POR QUE NÃO TESTAMOS?

- Teoricamente, todo desenvolvedor sabe que deve desenvolver testes para seu software:
  - Na prática, poucos o fazem;
- Por que não?
  - “O prazo está muito apertado”;
  - “A pressão para que terminemos rápido é muito grande”;
- Será que prazo e pressão podem realmente ser usados como argumentos para não se testar?

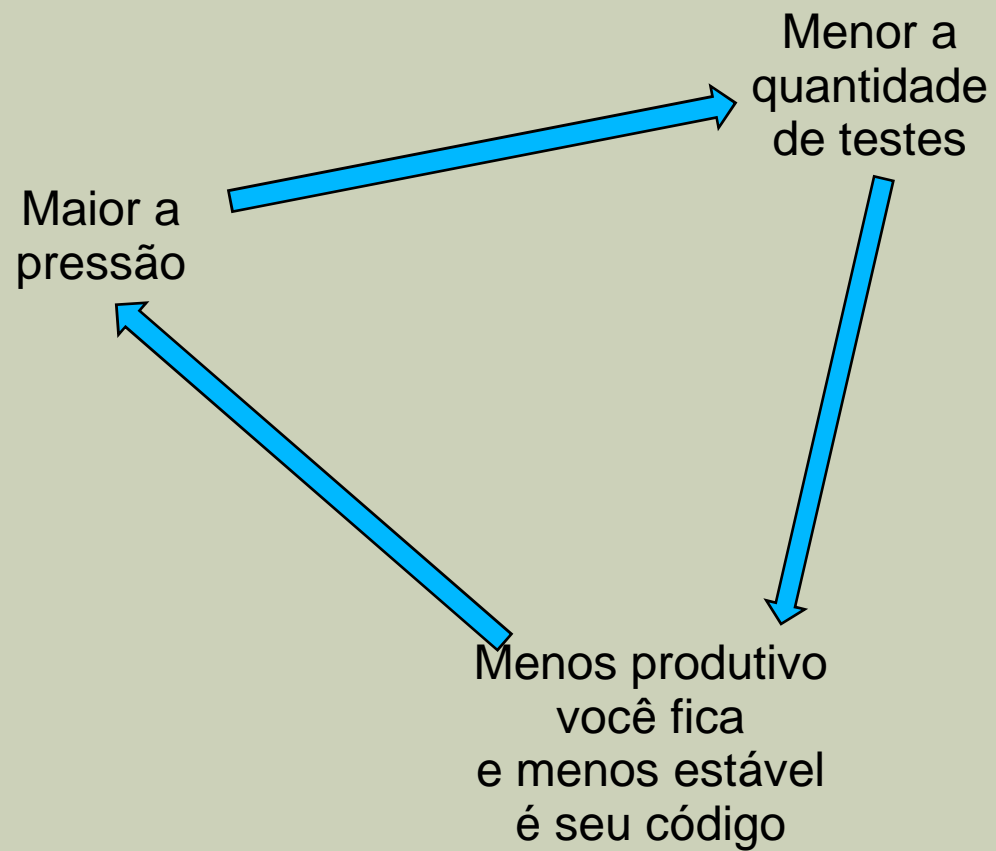


# SITUAÇÃO TÍPICA

- “A pressão é muito grande!”
- “Não temos tempo para testar!”
- “O trabalho que temos é muito maior do que aquilo que somos capazes de fazer!”



# SITUAÇÃO TÍPICA

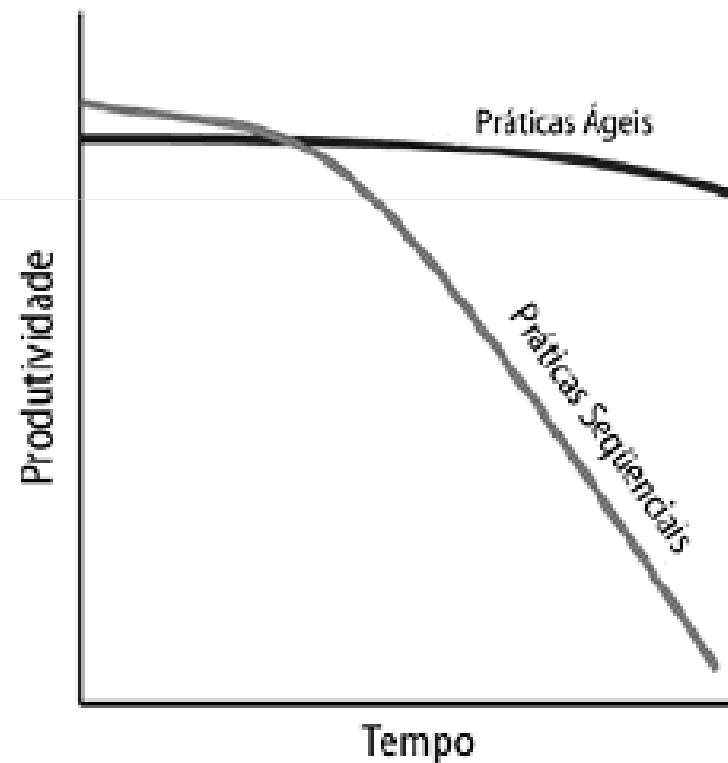


# QUEM VAI TER QUE CONSERTAR NO FINAL?



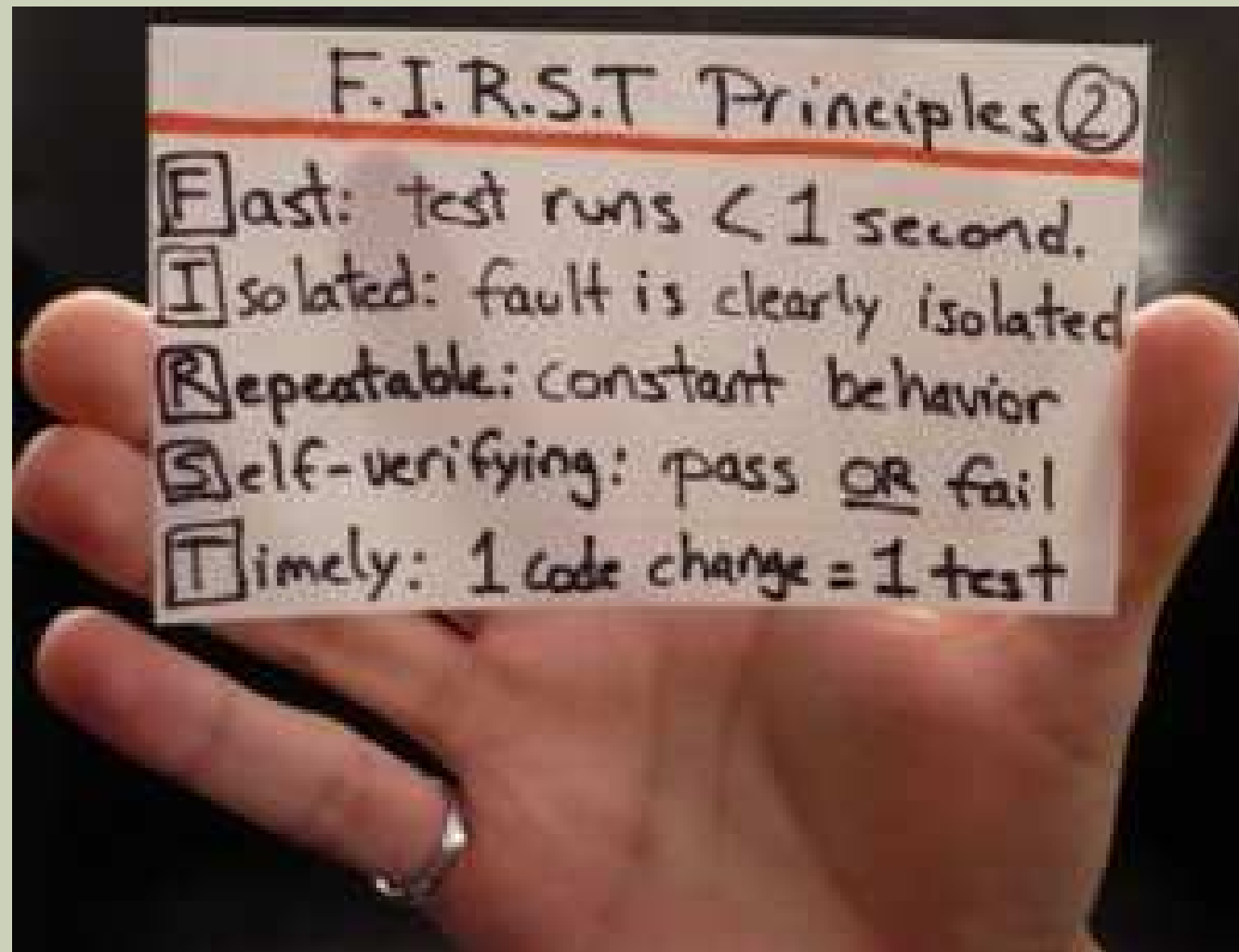
# COMPARATIVO DE PRODUTIVIDADE

COMPARATIVO ENTRE PRODUTIVIDADE PELO TEMPO ENTRE  
PRÁTICAS ÁGEIS E SEQUENCIAIS



FONTE: Disponível em: <<http://www.improveit.com.br/xp/dissertacaoXP.pdf>>

# FIRST



# TEORIA DA JANELA DE VIDRO



# BROKEN WINDOWS

**"Considere-se um edifício com algumas janelas quebradas. Se as janelas não são reparadas, a tendência é que vândalos quebrem mais janelas."**

- **Artigo criado por James Q. Wilson e George L. Kelling em 1982 na área de Criminologia e Sociologia.**
- **Em 1996, lançaram o livro: *Fixing Broken Windows: Restoring Order and Reducing Crime in Our Communities* para o conter ou eliminar os crimes dos ambientes urbanos.**

# TESTES MANUAIS E AUTOMÁTICOS



# TESTES MANUAIS

- São testes onde pessoas precisam seguir um roteiro de passos para verificar o comportamento do artefato testado;
- Entrada de dados e interpretação de resultados são responsabilidade de humanos;
- Demoram para serem concluídos;
- Se tornam uma atividade repetitiva;
  - Suscetível ao erro;
  - Estressa as pessoas;
- São testes não determinísticos;
  - O humor e atenção dos testadores podem modificar o resultado dos testes.

# PROBLEMAS COM TESTES MANUAIS

- Os testes não são reaproveitáveis
- Todo o esforço investido para testar algo não vale de nada se precisarmos testar novamente;
- Quando os roteiros existem, a atividade de teste já é desgastante. Imagine o que acontece quando os roteiros não existem!

# TESTADOR SAINDO DA EMPRESA



# TESTES AUTOMATIZADOS

- Testes codificados, através de alguma linguagem de programação ou de script, que seguem um padrão para escrita e possuem semântica bem definida para os seus resultados;
- A concordância com padrões permite combinar testes em suítes;
- O uso de suítes permite incorporar a execução constante de testes ao processo de desenvolvimento:
  - Builds constantes durante a codificação;
  - Integração contínua;
- São totalmente reutilizáveis.

# TESTES AUTOMATIZADOS

- O esforço para criação dos testes é compensado durante o ciclo de vida do software:
  - Perde-se menos tempo com depuração:
  - Desgasta menos a equipe na hora de corrigir um determinado bug...
- Testes manuais dificilmente compensam o esforço...

# UM PEQUENO ESTÍMULO

Ciclo de Teste	Esforço despendido (horas) para testar Casos de Uso					
	Simples		Intermediário		Complexo	
	Manual	Automat	Manual	Automat	Manual	Automat
1	6,0	10,07	12,0	23,01	16,0	29,64
2	12,0	10,37	24,0	23,41	32,0	30,32
3	18,0	10,67	36,0	23,81	48,0	31,00
4	24,0	10,97	48,0	24,21	64,0	31,68
5	30,0	11,27	60,0	24,61	80,0	32,36

**Fonte:** Instituto Atlântico – Fortaleza – CE – 2006

# DICAS PARA AUTOMAÇÃO

- Projete os casos de testes para depois automatizar.
- Não pense em automatizar tudo.
- O erro pode estar no teste e não na aplicação.
- As interfaces de usuário mudam constantemente.
- Evite ao máximo lógicas complexas dentro dos scripts.
- Ao encontrar um erro, automatize o teste para o mesmo antes de corrigi-lo.
- Se tiver roteiro de testes, faça o que está no mesmo.
- Não espere encontrar muitos bugs através de testes de regressão, eles existem para garantir a estabilidade.

# CUIDADOS COM AUTOMAÇÃO

- Vícios atrapalham a criação de testes:
  - Desenvolvedores tendem a “pegar leve” quando estão testando o próprio código;
  - Só testam situações simples, onde sabem que o teste irá passar;
- Nesse caso, os testes mostram o que está funcionando, mas não capturam situações de erro;
  - Acrescentam pouco ao projeto (mas são necessários);
- Pode ser amenizado com programação em pares ou rodízio na criação dos testes.



# POR QUE NÃO ABOLIR OS TESTES MANUAIS?

- Disposição dos campos na tela;
- Layout;
- Elementos não desejáveis e, conseqüentemente, não previstos.

# AINDA SOBRE AUTOMAÇÃO DE TESTES

- Não substitui uma equipe de testes.
- Os testes precisam ser constantemente atualizados.
- Inicialmente, a automação custa tempo e dinheiro.
- Automação não é remendo, é investimento.
- Permite executar os testes mais vezes.
- Automação permite aproveitar recursos (computacionais) fora do horário normal de trabalho.

# CONCEITOS BÁSICOS SOBRE TESTES

# O QUE É UM TESTE?

- É a investigação do software a fim de fornecer informações sobre sua qualidade em relação ao contexto em que ele deve operar.
- Os testes não podem garantir que não existam erros num software.

# CLASSIFICANDO OS TESTES

- Caixa-preta;
- Caixa-branca;
- Caixa-cinza.

# TESTES CAIXA-PRETA

- Testam o comportamento externo do componente de software. Por exemplo:
  - Testes onde os resultados das ações testadas são comparados com as saídas esperadas;
- Estes testes são elaborados baseando-se no comportamento esperado do componente, e não na implementação usada;
  - Resumindo: Ao fazer o caso de teste, não temos interesse em saber qual o comportamento interno ou estrutura interna do programa. Nos concentramos apenas no comportamento geral do programa.
- Os casos de testes são derivados de especificação;
- Também chamado "data-driven" ou "input/output-driven";

# TESTES CAIXA-BRANCA

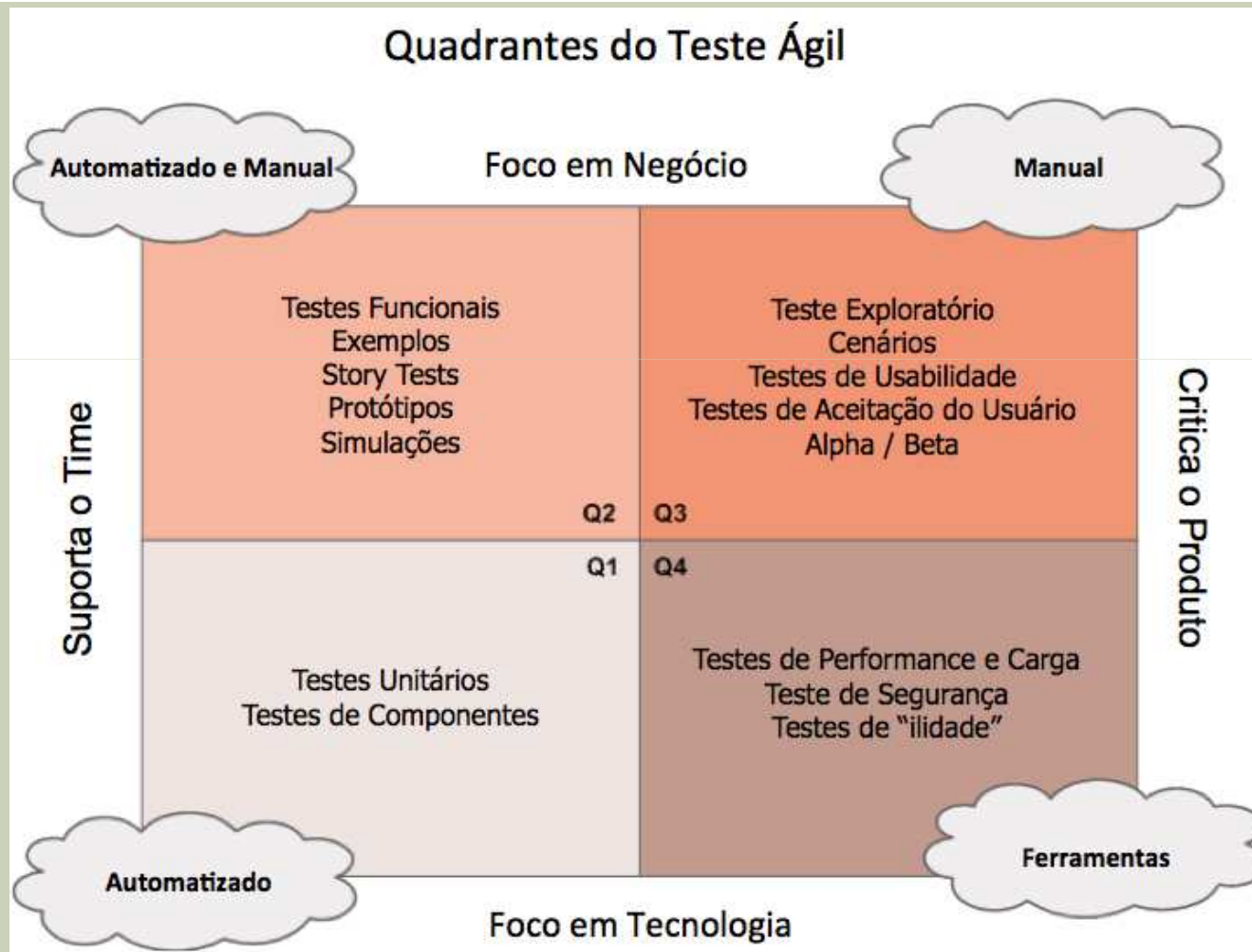
- Testam os componentes focando-se em suas implementações. Avalia o comportamento interno do componente. Por exemplo:
  - Testes tendenciosos que exercitam todos os “caminhos” dos componentes testados;
    - Blocos if-then-else, try-catch, switch;
- Também conhecidos como **logic-driven**;
  - Resumindo: Consiste em realizar testes em situações nas quais se conhece o comportamento interno do programa. Baseando-se neste conhecimento, o programador pode elaborar casos de testes mais direcionados e com maior cobertura de código.
- Recomenda-se que quem desenvolve a funcionalidade não crie o teste para a mesma.

# TESTES CAIXA-CINZA

- Consiste em uma “mistura” dos cenários caixa-preta e caixa-branca.
- Neste cenário, testes são escritos a partir das especificações das interfaces dos módulos testados, assim como testes caixa-preta;
- Entretanto, as entradas para os testes podem ser tendenciosas de forma a induzir algum estado ou condição dentro do módulo testado, assim como os testes caixa-branca.
- Isso envolve ter acesso a estruturas de dados e algoritmos do componente a fim de desenvolver os caso de testes, que são executados na técnica de caixa-preta.



# QUADRANTE DE TESTE ÁGIL

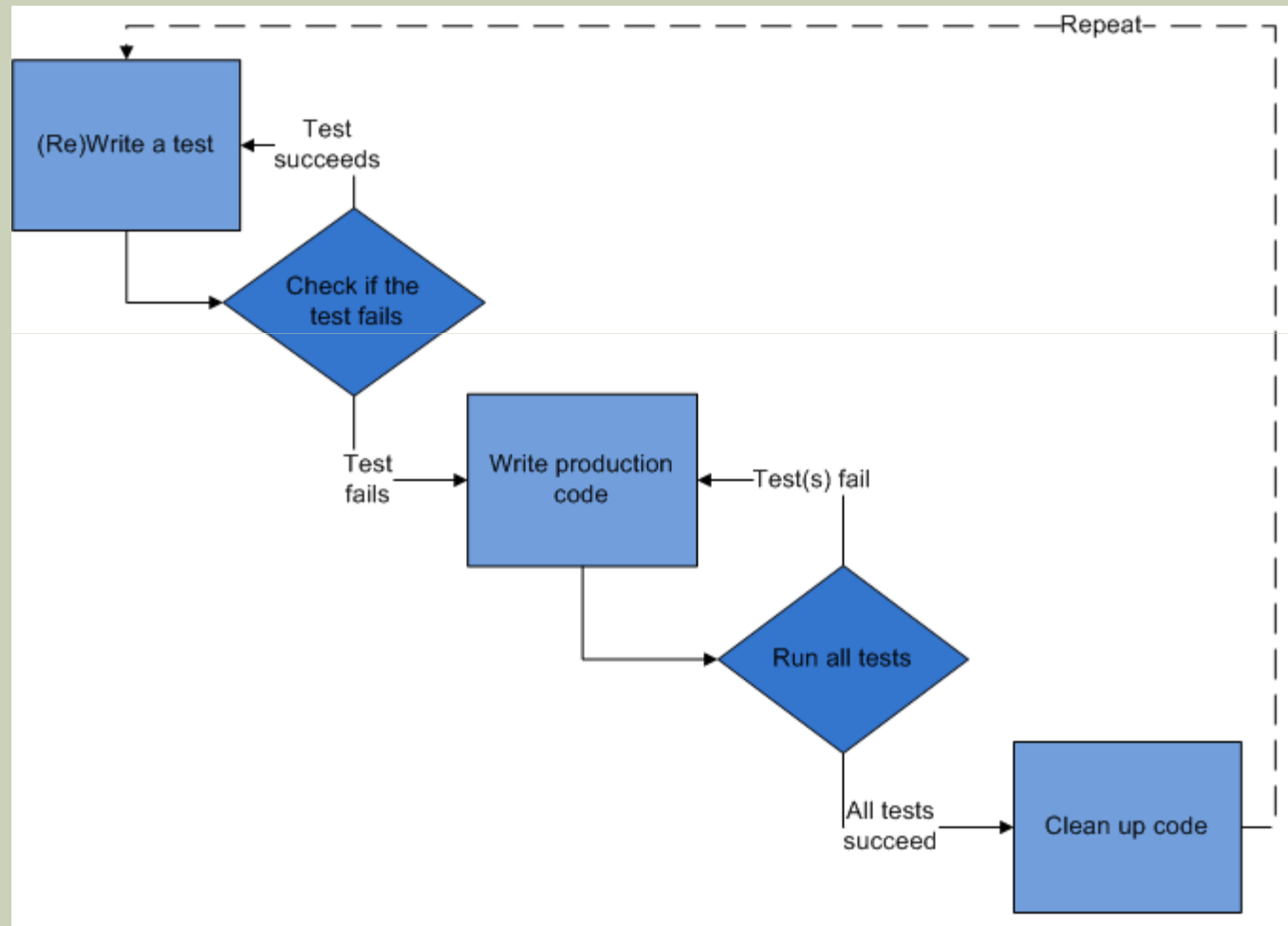


Fonte: [Adaptworks.com.br](http://Adaptworks.com.br)

# TESTES DE UNIDADE



# TDD



# TESTES DE COMPONENTES



# COMO ISOLAR UM MÉTODO?

**REAL SYSTEM**



**Green** = class in focus  
**Yellow** = dependencies  
**Grey** = other unrelated classes

**CLASS IN UNIT TEST**



**Green** = class in focus  
**Yellow** = mocks for the unit test

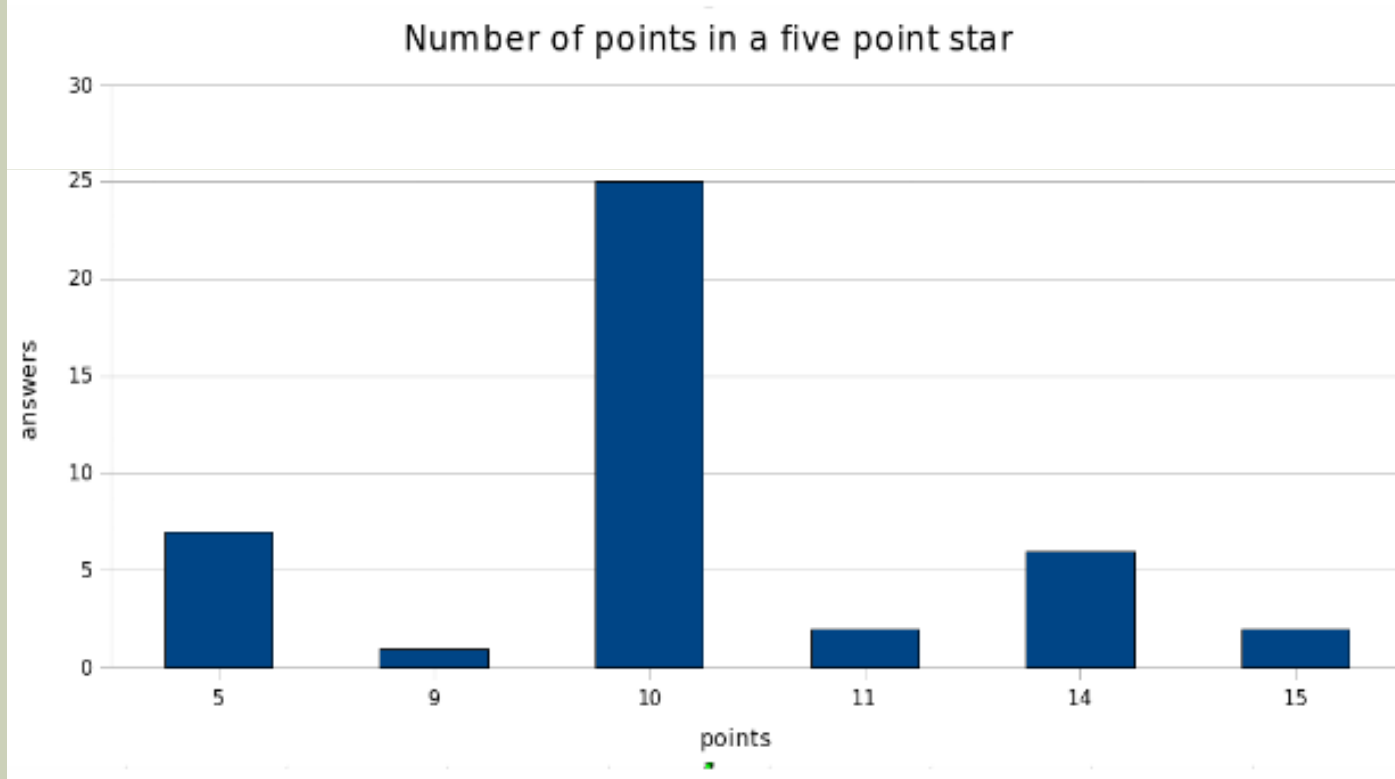
# TESTES FUNCIONAIS

- Testes funcionais são aqueles que procuram testar as funcionalidades de sua aplicação, verificando a integração entre as diversas partes que a compõe. A idéia é simular a interação de um usuário real com o sistema, sem se preocupar com os detalhes de implementação da funcionalidade.
- É um dos testes mais importantes, pois ele se caracteriza por mostrar se a aplicação funciona ou não em tudo aquilo que ela se propõe a atender em termos de funcionalidades.
- Podem ser manuais ou automatizados.

**QUANTOS PONTOS?**



# How many points are there?

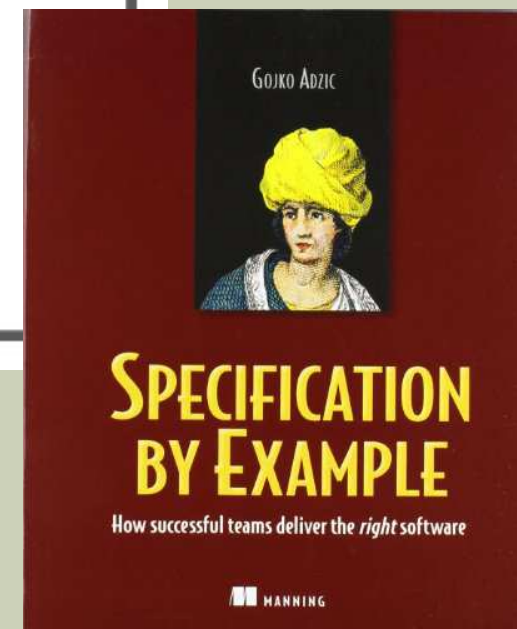


Fonte: Gojko Adzic



# EXEMPLOS

if Moped top	Tryung	Spoken D	Drugs?	Doctor	friend of family	good grade
NO	YES	all	Yes	Yes	-	-
NO	YES	all	No	Yes	-	-
NO	YES	Friends	Yes	Yes	-	-
NO	YES	Friends	No	No	-	+
NO	NO	all	Yes	Yes	-	-
NO	NO	all	NO	Yes	-	-
NO	NO	Friend	Yes	Yes	-	-
NO	NO	Friend	NO	NO	-	-
YES	-	-	-	YES	-	-
NO	NO	-	NO	NO	+	+



## STORY TEST

As **who** I want  
**what** so that  
**why**

# BDD

**Funcionalidade:** Relatório de vendas.

*Anna deseja um relatório de vendas do mês.*

*Como responsável pelo planejamento de marketing da empresa, Anna está interessada em resumos, de alto nível, das vendas considerando produtos, região, ticket médio e relação com investimentos em promoção.*

*Anna valoriza interfaces simples (com poucas opções), fáceis de aprender e usar.*

**Cenário:** Anna deseja relatório de vendas, para um mês anterior

Dado que Ana está logada no sistema

E acessou a página para emissão do relatório de vendas por mês

Quando informou um mês de referência

E confirmou a geração do relatório

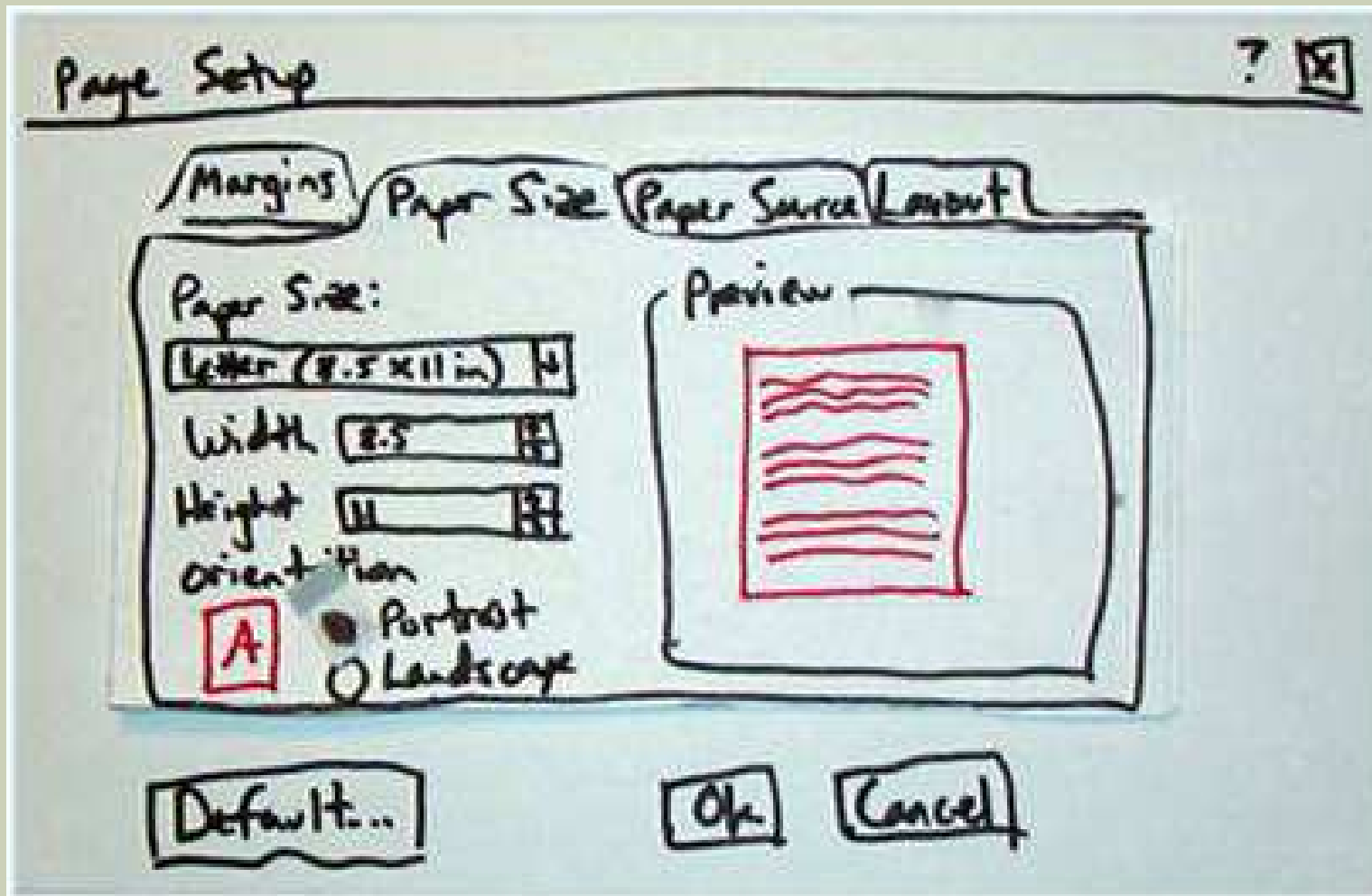
Então obtém um relatório sumarizado por região

E com o resumo dos investimentos em promoção na região

E com a relação dos 10 produtos mais vendidos

E com o ticket médio de cada um desses produtos

# PROTÓTIPOS



# TESTES EXPLORATÓRIOS



# CENÁRIOS



# CENÁRIOS DE NOVELAS

## The Rented and the Wrecked

BY BRIAN MARICK

A customer named Marick hires a car for a three-day business trip. Midway through the rental, he extends it for another week. (This, by the way, gives him enough rental points to reach Preferred status.) Several days later, he calls to report that the car has been stolen. He insists that the Preferred benefit of on-site replacement applies, even though he was not Preferred at the start of the rental. A new car is delivered to him. Two days after that, he calls to report that the "stolen" car has been found. It turns out he'd mis-remembered where he'd parked it. He wants one of the cars picked up and the appropriate transaction closed. Oh, and one other thing: the way he discovered the mislaid car was by backing into it with its replacement, so they're both damaged.

This scenario would test the following conditions

- Upgrading to Preferred status (during rental)
- Extending a rental
- Handling a stolen car
- On-site replacement
- Undoing on-site replacement
- Undoing handling of stolen car
- Return of a damaged car

functioning buttons, they can be a challenge to execute. To say it in soap opera terms: It doesn't make sense to write about a Hollywood wedding in a beautiful church if the lock on the church door

### Give It a Try

Try to write scenarios that are (1) based on real life, (2) exaggerated, and (3) condensed into a limited number of events. Cover functional



ited number of test cases—fewer, more efficient tests are easier to manage, can be executed faster, and produce fewer results to be analyzed. Use

# TESTES DE USABILIDADE





# 10 HEURÍSTICAS DE USABILIDADE DO NIELSEN

- Visibilidade de Status do Sistema
- Relacionamento entre a interface do sistema e o mundo real
- Liberdade e controle do usuário
- Consistência
- Prevenção de erros
- Reconhecimento ao invés de lembrança
- Flexibilidade e eficiência de uso
- Estética e design minimalista
- Ajude os usuários a reconhecer, diagnosticar e sanar erros
- Ajuda e documentação

TesteUsabilidade  
.com.br



*Com apenas 5 usuários, você identificará  
85% dos problemas de usabilidade.\*  
Envie seu site e tenha feedback gratuito!*

\* Fonte: Jakob Nielsen - guru de usabilidade

# TESTE DE ACEITAÇÃO DO USUÁRIO

ACCEPTANCE TESTS



PASSED

memegenerator.com

Story

1

*Done when passes*

1..\*

Acceptance Test

# TESTES ALFA



# TESTES BETA



# RELEASE CANDIDATE



*WordPress 3.4 Release Candidate 3*

# TESTES GAMA





# TESTES DE DESEMPENHO



# TESTES DE CARGA





# TESTE DE SEGURANÇA

- Confidencialidade
- Integridade
- Disponibilidade



# TESTE DE \*ILIDADE

Lines of code

**2.348** ▲

6.308 lines ▲

Classes

**80**

9 packages

281 methods ▲

+39 accessors

Complexity

**1,8** / method

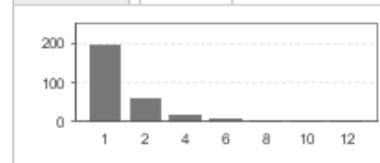
**6,3** / class

502 cmpx ▲

744 statements ▲

Methods

Classes



Comments

**34,0%**

1.209 lines ▲

0 commented LOCs

Duplications

**0,0%**

0 lines

0 blocks

0 files

Code coverage

**91,4%** ▲

93,5% line coverage ▲

84,7% branch coverage

92 tests ▲

4.5 sec ▲

Test success

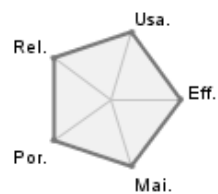
**100,0%**

0 failures

0 errors

Rules compliance

**99,9%** ▲



Violations

**6**

🚫 Blocker 0

🚫 Critical 0

🚫 Major 0

📉 Minor 3

📉 Info 3

[Add a measure](#)

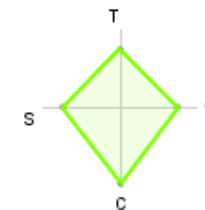
**SIG Maintain. Model** ⓘ

(A)nalsability +

(C)hangeability ++

(S)tability +

(T)estability +



Events

All ▼

19/02/2010	Version	0.5-SNAPSHOT	<a href="#">Edit</a> <a href="#">Delete</a>
11/02/2010	Version	0.4-SNAPSHOT	<a href="#">Edit</a> <a href="#">Delete</a>
19/01/2010	Version	0.3-SNAPSHOT	<a href="#">Edit</a> <a href="#">Delete</a>

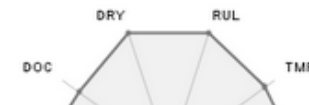
[Add an event](#)

Total Quality

**96,2%** ▲

91,4% tests ▲

100,0% arch ▲



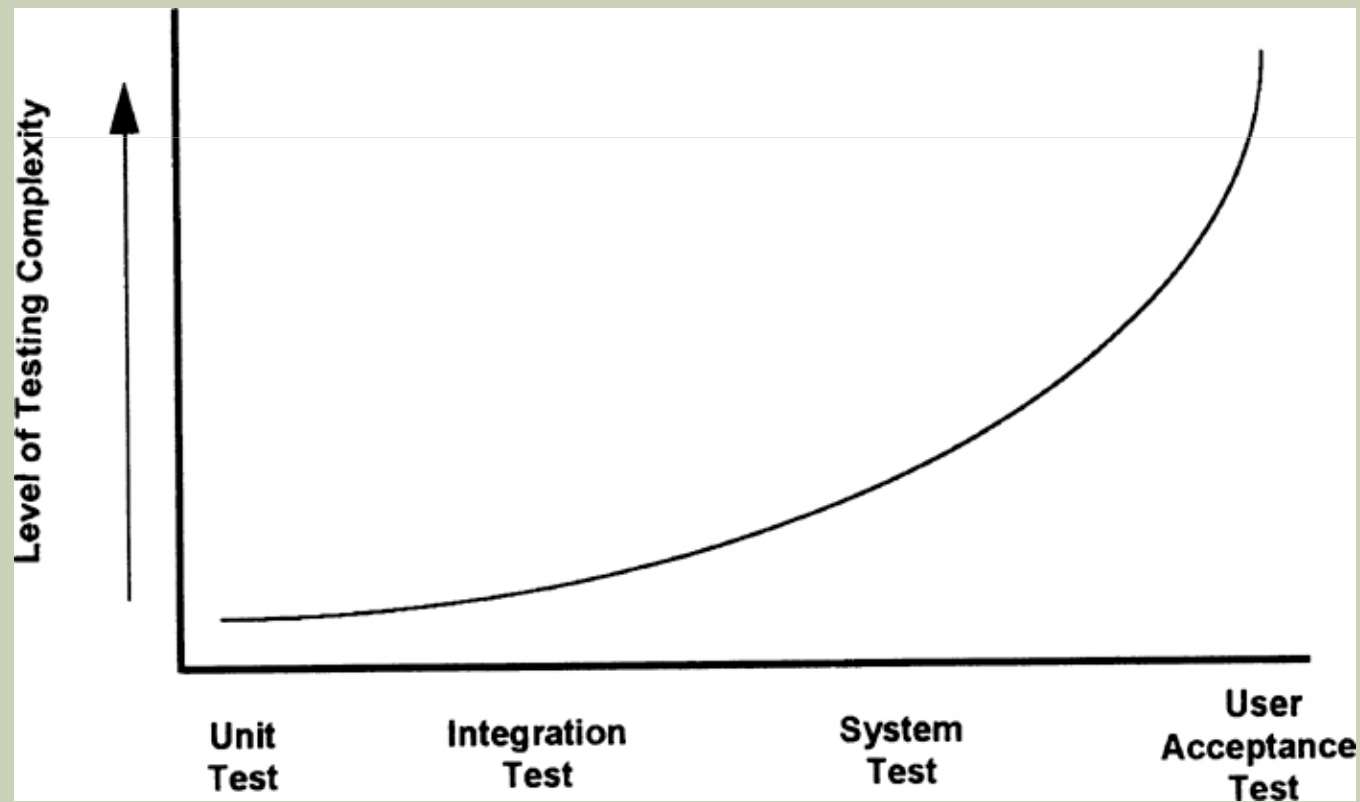
# 7 PRINCÍPIOS DO TESTE DE SOFTWARE

- Testes demonstram a presença de defeitos
- Teste exaustivo é impossível
- Testes devem iniciar o quanto antes e erros encontrados tarde custam mais para corrigir
- Agrupamento de defeitos
- Paradoxo do pesticida
  - Os mesmos testes aplicados repetidamente deixam de encontrar defeitos no local. Revise os testes por que os bugs estão em outro ponto não testado.
- Teste é dependente do contexto
- A ilusão da ausência de defeitos

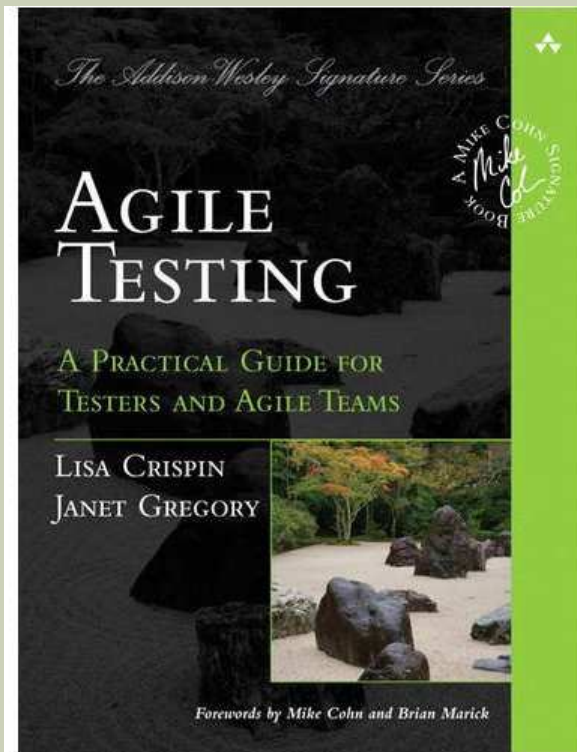
# A REGRA 10 DE MEYERS

“Quanto mais cedo descobrimos o erro, menor é o seu valor para o projeto. Esse custo em correções cresce 10 vezes para cada estágio que o projeto avança.”


Myers



# MAIS CONTEÚDO...



- Agile Testing – A practical guide for testers and agile team



- Every bug you take  
Every patch you fake  
Every build you break  
Every log you erase  
I'll be watching you

- **The DevOps Song**

- <https://gist.github.com/brunoborges/8559729>