

ALGUMAS PERGUNTAS QUE MOTIVAM ESTA AULA

- Como podemos avaliar se um produto está PRONTO?
- Como sei que um produto digital tem qualidade? Se é suficientemente BOM?
- Como podemos controlar a qualidade de entregas sucessivas de software?
- Como o próprio processo de desenvolvimento pode ser melhorado?
- Quando as atividades de garantia e controle da qualidade começam? Quando estão completas?
- Quais técnicas específicas devem ser aplicadas durante o desenvolvimento?

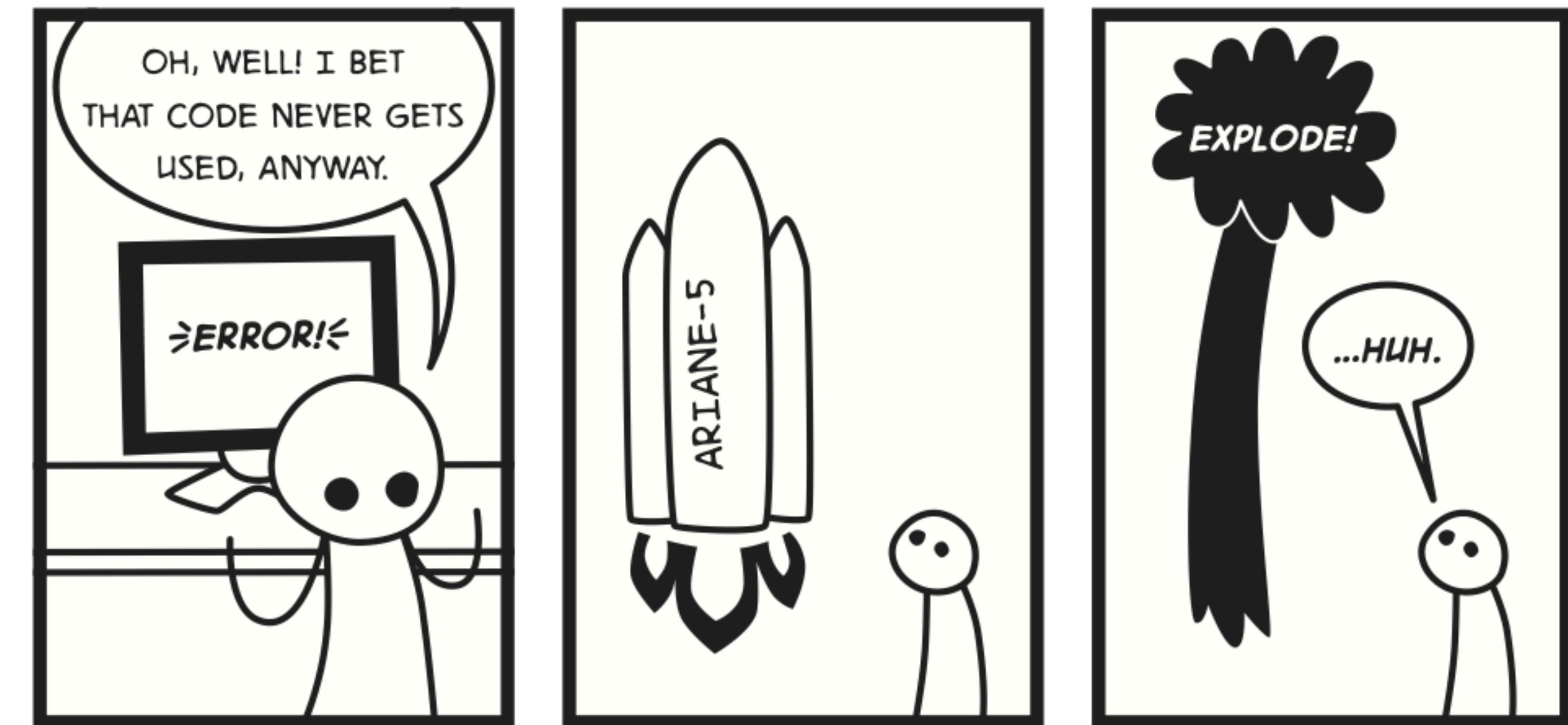
O QUE É QUALIDADE (DE SOFTWARE)?

O QUE É QUALIDADE DE SOFTWARE?

- “Um produto de software apresenta qualidade dependendo do grau de satisfação das necessidades dos clientes sob todos os aspectos do produto” [Sanders,1994].
- “Qualidade de software é a conformidade a requisitos funcionais e de desempenho que foram explicitamente declarados, a padrões de desenvolvimento claramente documentados, e a características implícitas que são esperadas de todo software desenvolvido por profissionais” [Pressman, 1994].

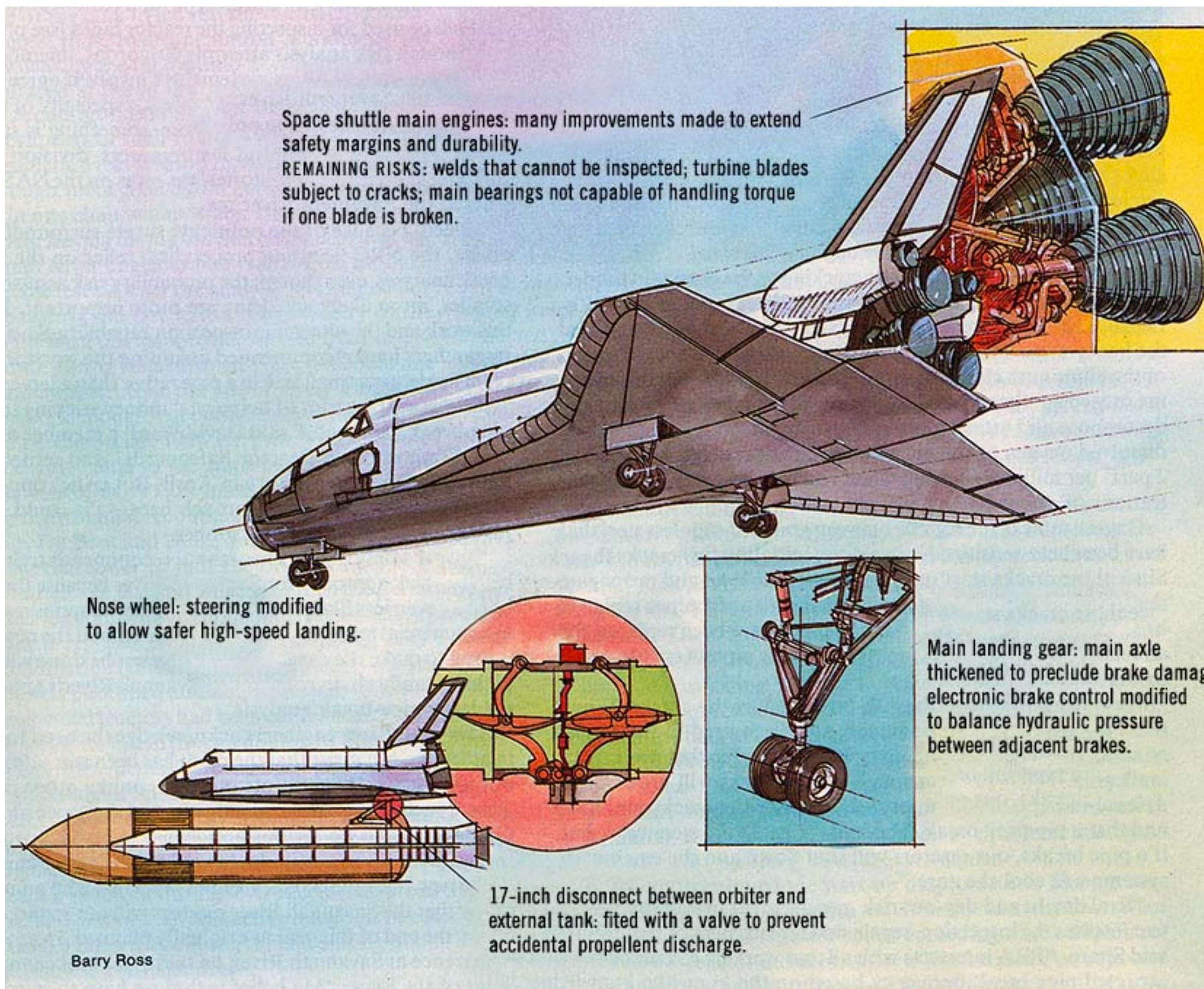
IMPORTÂNCIA DA QUALIDADE

- Seres humanos erram, por isso, introduzimos defeitos que geram inúmeras consequências negativas, de desconfortos nos usuários até catástrofes.



<http://csfieldguide.org.nz/releases/2.4.1/en/chapters/software-engineering.html>

SOBRE DEFEITOS/ERROS ICÔNICOS DA HISTÓRIA



Leia mais:

<https://www.computerworld.com/article/2515483/enterprise-applications/epic-failures--11-infamous-software-bugs.html>

<http://www.cse.psu.edu/~gxt29/bug/softwarebug.html>

<http://news.bbc.co.uk/2/hi/science/nature/2570053.stm>

<https://spectrum.ieee.org/tech-history/heroic-failures/the-space-shuttle-a-case-of-subjective-engineering>

CUSTO DE NÃO TESTAR CEDO

Design and architecture	Implementation	Integration testing	Customer beta test	Postproduct release
1X*	5X	10X	15X	30X

*X is a normalized unit of cost and can be expressed in terms of person-hours, dollars, etc.

Source: National Institute of Standards and Technology (NIST)†

\$60B



The Bottom Line

According to the NIST (National Institute of Standards and Technology), software defects cost the U.S. economy nearly \$60 billion a year.

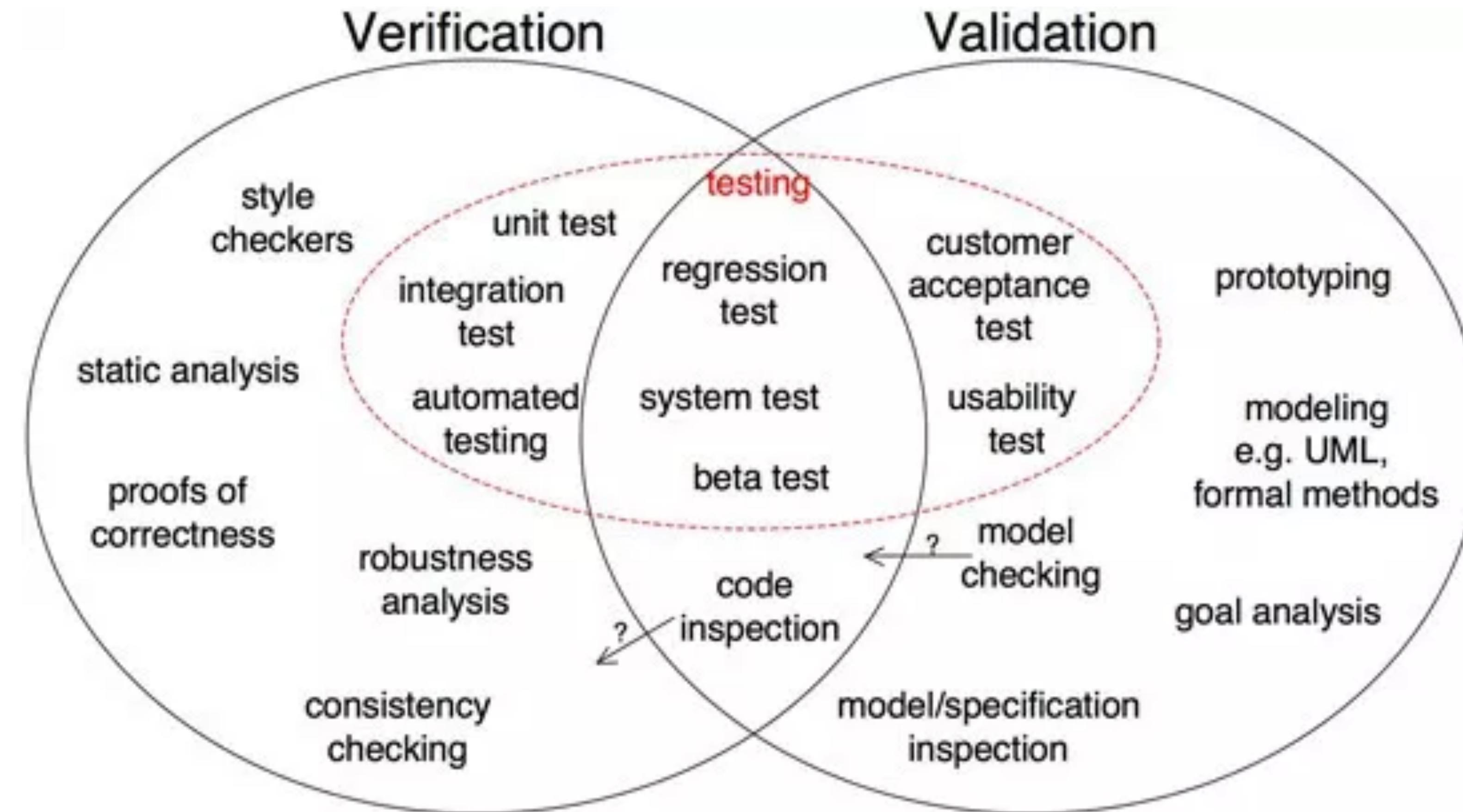
Correcting a bug after release costs more than fixing it during QA, which costs more than catching it during coding.



VERIFICAÇÃO E VALIDAÇÃO DE SOFTWARE

- V & V é considerada uma disciplina coerente: "V & V é uma disciplina de engenharia de sistemas que avalia o software em um contexto de sistemas, relativo a todos os elementos do sistema de hardware, usuários e outros softwares".
- (*from Software Verification and Validation: Its Role in Computer Assurance and Its Relationship with Software Project Management Standards, by Dolores R. Wallace and Roger U. Fujii, NIST Special Publication 500-165*)

V&V: TIPOS DE ATIVIDADES



V&V: PECULIARIDADES DO SOFTWARE

- O software possui algumas características que tornam o V & V particularmente difícil:
 - Muitos requisitos de qualidade diferentes
 - Evolução (e deterioração) da estrutura
 - Não-linearidade inerente
 - Distribuição irregular de falhas
- Exemplo
 - Se um elevador puder transportar com segurança uma carga de 1000 kg, ele também pode transportar com segurança qualquer carga menor;
 - Se um procedimento classificar corretamente um conjunto de 256 elementos, ele poderá falhar em um conjunto de 255 ou 53 ou 12 elementos, bem como em 257 ou 1023

QUALIDADE DE SOFTWARE: ANÁLISES ESTÁTICA E DINÂMICA

- Qualidade de Software é intimamente ligada às atividades de Verificação e Validação (Anderson, 2008).
- Atividades de **análise estática e dinâmica** podem ser empregadas conjuntamente para maximizar a detecção de defeitos em produtos de software.
- **Análise estática:** Processo de avaliar um sistema ou componente com base em sua **forma, estrutura, conteúdo ou documentação**.
- **Análise dinâmica:** Processo de avaliar um sistema ou componente com base em seu **comportamento durante a execução**".

Anderson, P. The use and limitations of static-analysis tools to improve software quality. CROSSTALK The Journal of Defense Software Engineering, v. 21, n. 6, p. 18{21, 2008.

ANÁLISE ESTÁTICA

- Dentre as principais atividades de análise estática estão (ISO/IEC/IEEE, 2010):
 - **Inspeção;**
 - **Revisão Informal;**
 - **Leitura Baseada em Perspectiva** (Basili et al., 1996)
 - **Walk-through;**
 - **Desk checking.**
- Em geral, também não é possível provar a correção do artefato por meio de técnicas de análise estática.
- Realizada sem a execução do programa;
- Utilizada na avaliação de artefatos como documento de requisitos, diagramas, código fonte e outros;
- Em geral, também não é possível provar a correção do artefato por meio de técnicas de análise estática.

DEFEITO, ERRO E FALHA

- **Defeito (fault)**: um passo, processo ou definição de dados incorreto
- **Erro (error)**: se caracteriza por um estado inconsistente ou inesperado
- **Falha (failure)**: um comportamento que difere do comportamento esperado.

```
public static int numZero (int [ ] arr)
{ // Effects: If arr is null throw NullPointerException
 // else return the number of occurrences of 0 in arr
    int count = 0;
    for(int i = 1; i < arr.length; i++)
    {
        if (arr [ i ] == 0)
        {
            count++;
        }
    }
    return count;
}
```

Fault: Should start searching at 0, not 1

Test 1
[2, 7, 0]
Expected: 1
Actual: 1

Error: i is 1, not 0, on the first iteration
Failure: none

Test 2
[0, 2, 7]
Expected: 1
Actual: 0

Error: i is 1, not 0
Error propagates to the variable count
Failure: count is 0 at the return statement

DEFEITOS (BUGS)

TOP DEFINITION



It's not a bug, it's a feature

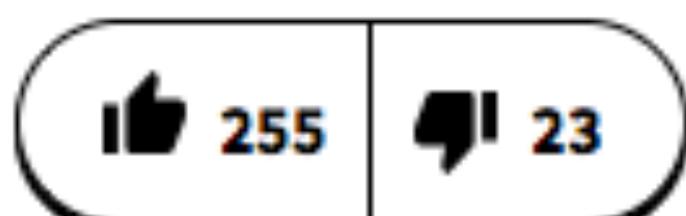
Excuse **made** by software developers when they try to convince the **user** that a flaw in their program is **actually** what it's supposed to be doing.

User: When I am on the item detail screen and **click** on "delete this item", the program returns to **to the** item overview screen and the item is still there.

Developer: Your access permissions don't allow you to delete items. It's not a bug, it's a feature.

#programmer #user #luser #bug #feature

by **crushnet** September 14, 2012



ABORDAGEM DE GESTÃO DA QUALIDADE

- A equipe deve:
 - escolher e realizar a combinação certa de técnicas para:
 - alcançar o nível de qualidade exigido
 - dentro de restrições de custo
 - projetar uma solução específica que atenda:
 - o problema
 - os requisitos
 - o ambiente de desenvolvimento

**QUALIDADE É MISSÃO DE TODOS.
QUALIDADE NÃO É UMA FASE DO SEU PROJETO.**

EVOLUÇÃO DE SOFTWARE

>

MANUTENÇÃO

ATIVIDADES DE V&V

- Não podemos revelar e remover todas as falhas
- Não podem durar indefinidamente: queremos saber se os produtos atendem aos requisitos de qualidade

V&V NO LONGO PRAZO

- Atividades de “manutenção”/evolução incluem:
 - análise de mudanças e extensões
 - geração de novos conjuntos de testes para as funcionalidades adicionadas
 - re-execuções de testes para verificar a não regressão de software
 - funcionalidades após mudanças e extensões
 - rastreamento e análise de falhas

TIPOS DE TESTE

TIPOS DE TESTE (VISÃO GERAL)

Caixa-preta (Funcional)



Caixa-branca (Estrutural)



Caixa-cinza



TIPOS DE TESTE

- Unidade
- Integração
- Interface de Usuário
- Aceitação
- Mutação
- Desempenho
- Estresse
- Segurança

TESTES DE UNIDADE

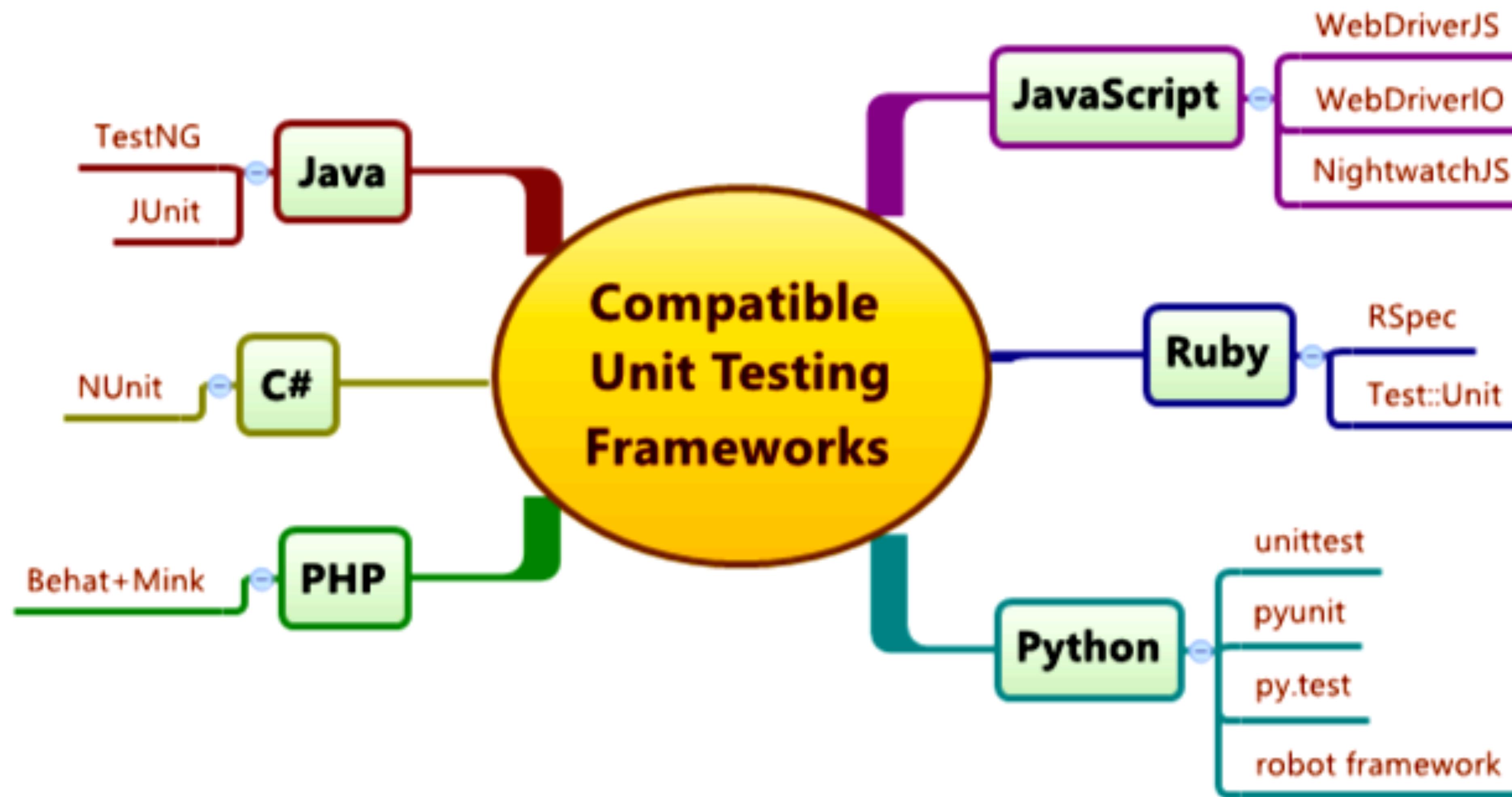
ESTRUTURA DE UMA CLASSE DE TESTES

- Os testes devem ser independentes:
 - Uns dos outros
 - Obs: padrão Chained Tests
 - Do número de vezes que é executado
 - De fatores externos
- Set Up: Prepara o ambiente para a execução do teste
- Tear Down: Limpa / Remove o ambiente

TESTES DE UNIDADE

- Unidade: módulo/classe ou função/método
- Teste básico e muito importante
 - Encontra muitos erros
- Não deve se preocupar com as responsabilidades de outras unidades
- Caixa-branca
 - Focando na funcionalidade

FRAMEWORKS PARA TESTES AUTOMATIZADOS



QUANDO ESCREVER TESTES

- Antes de escrever o código (Test First)
- Durante a implementação
- Antes de uma refatoração
- Quando um erro for encontrado: Escreva um teste que simule o erro e depois corrija-o
- Quando um teste de aceitação falha, pode ser um sinal que testes de unidade estão faltando
- Quando um usuário ou cliente encontra um erro, é um sinal que estão faltando testes de unidade e de aceitação

QUANDO EXECUTAR TESTES

- A execução deve ser ágil
- Sempre que uma nova porção de código é criada
- À noite ou em uma máquina dedicada
- Assim que alterações são detectadas no repositório
- Lembre-se: é melhor escrever e executar testes incompletos do que não executar testes completos

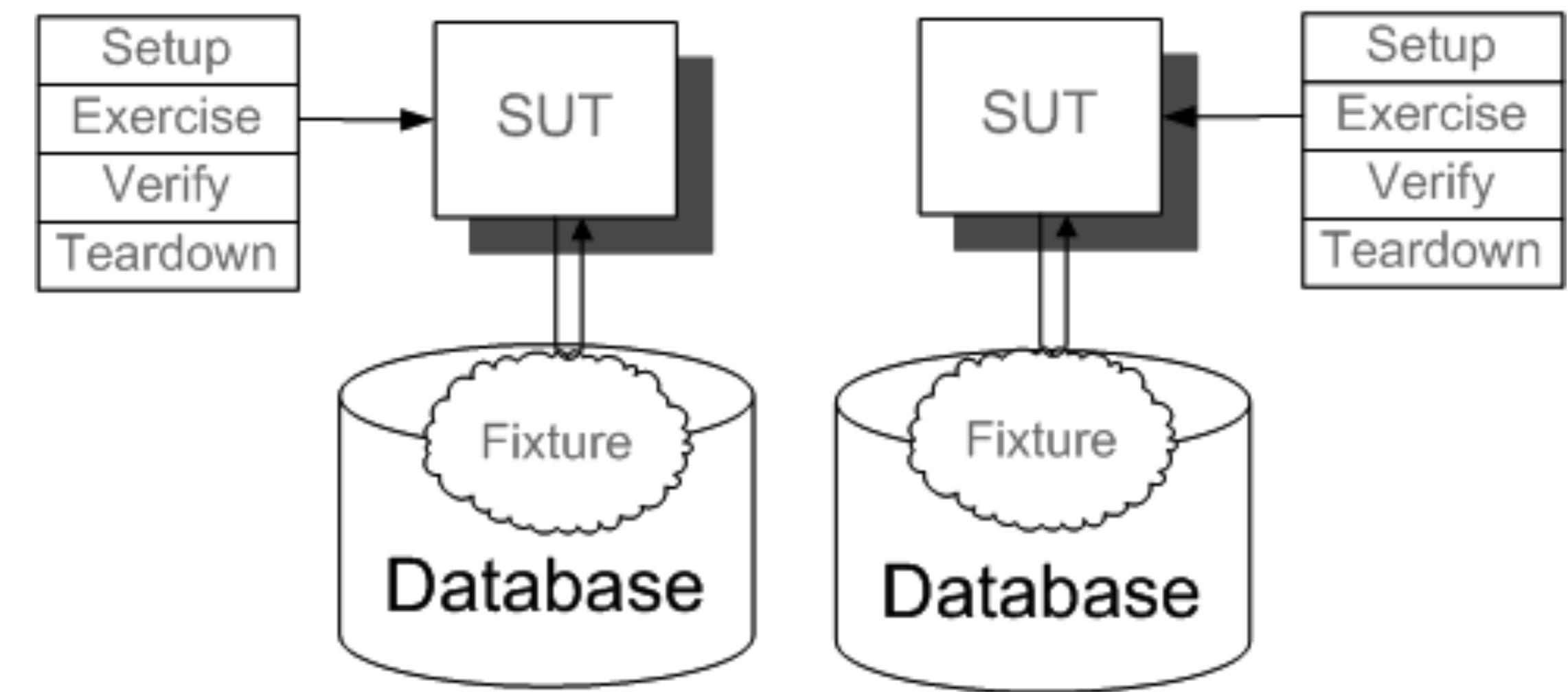
TESTES COM BANCOS DE DADOS

- DAOs, ORM, Queries, Stored Procedures, Triggers
- BDs de desenvolvimento/teste
 - Sujeira no BD:
 - Transaction Rollback Teardown
 - Table Truncation Teardown
 - Concorrência: Database Sandbox
 - BDs em memória
- Set Up e Tear Down são importantes:
 - Setup do teste: coloque sua base de dados em um estado conhecido antes de rodar os testes sobre ela.
 - Rode o teste: como em qualquer outro teste, mas agora com o apoio de alguma database regression testing tool
 - Cheque o resultado: é preciso ser possível obter os “dumps” das tabelas para que se compare os resultados reais com os esperados.

EX: DATABASE SANDBOX

P: Como desenvolvemos e testamos software que depende de um banco de dados?

R: Fornecemos um banco de dados de teste separado para cada desenvolvedor ou testador.



<http://xunitpatterns.com/Database%20Sandbox.html>

QUER SABER MAIS SOBRE TESTES EM BANCO DE DADOS?

SITES/LITERATURA RECOMENDADA:

<http://www.agiledata.org/essays/databaseTesting.html>

DICAS

- Particionamento de domínios
- Valores Limites:
 - Números: 0, -1, 1, muito pequenos, muito grandes
 - Strings: nula, vazia, pequenas, grandes
 - Coleções: nula, vazia, cheia
- Testes aleatórios
- Em geral, não testamos get/set (muito triviais)

DICAS (2)

- Internas: Testar a classe que usa é suficiente
- Abstratas: Instanciar uma subclasse e chamar os métodos da classe mãe
- Classes simples de armazenamento de dados não precisam ser testadas

OUTROS TIPOS DE TESTE

TESTES DE SOFTWARE

- **Teste de Software:**

- Executar o programa a ser testado com alguma entrada e conferir visualmente os resultados obtidos

- **Teste Automatizado:**

- Programa ou script executável que roda o programa a ser testado e faz verificações automáticas em cima dos efeitos colaterais obtidos.

- **Testar NÃO é depurar**

- Testar é verificar a presença de erros
- Depurar é seguir o fluxo para identificar um erro conhecido

TESTES DE INTEGRAÇÃO

- Testa a integração entre unidades ou componentes
- Duas unidades podem funcionar perfeitamente individualmente, mas podem haver falhas quando elas se comunicam

TESTES DE INTERFACE

- Simula eventos do usuário e verifica o estado da interface (FEST e FITNESSE)
- Módulos distintos e isolados podem alterar o estado da interface
 - Estado pode ficar inconsistente
 - Sujeira na tela
- Integração ou Aceitação

TESTES DE ACEITAÇÃO E ATDD

- Testa as funcionalidades do sistema a partir do ponto de vista do cliente
- Caixa-preta
- Data-Driven Test
- Teste de alto nível
- “Prova” para o cliente de que o software funciona
- O cliente pode especificar os testes e os desenvolvedores os implementam

TESTES DE REGRESSÃO

- Mesmo após liberado o software precisa ser testado
- A cada manutenção é preciso verificar se o software mantém suas características
- Se nenhum efeito colateral foi introduzido
- Técnicas de teste de regressão reutilizam subconjuntos do conjunto de teste existente
- Ao longo do tempo, muito mais viáveis se automatizados

TESTES FUMAÇA (SMOKE TESTS)

- Testes superficiais e abrangentes
- Realizados após a instalação do software
- Exemplo 1:
 - Acesse uma página Web e verifique que não aparece o número 404

TESTES DE DESEMPENHO

- Otimizar somente quando necessário
- Código claro traz mais benefícios do que milésimos de segundo de otimização
- Começar pelos gargalos da aplicação
- Utilizar Profilers para detectar os gargalos
- Arcabouços mostram o tempo de duração dos testes

TESTES DE ESTRESSE

- Testar com grandes quantidades de dados gerados automaticamente (ex: JMeter)
- Simulação de muitos usuários simultâneos
- Erros comuns:
 - Overflow
 - Falta de memória
- Alguns benefícios:
 - Encontrar vazamento de memória
 - Avaliar a capacidade de um ambiente

TESTES DE SEGURANÇA

- Útil principalmente para aplicações Web
 - Aplicações expostas a usuários mal-intencionados
- Testar valores inesperados:
 - Null
 - Grande quantidade de dados
 - Tipos de dados não esperados. Ex: string em um campo que espera um inteiro
- Testes de estresse para lidar com ataques de negação de serviço (DoS)
- Leia mais sobre testes de segurança no contexto Web na OWASP (OWASP TOP 10 - você precisa estudar isso!)



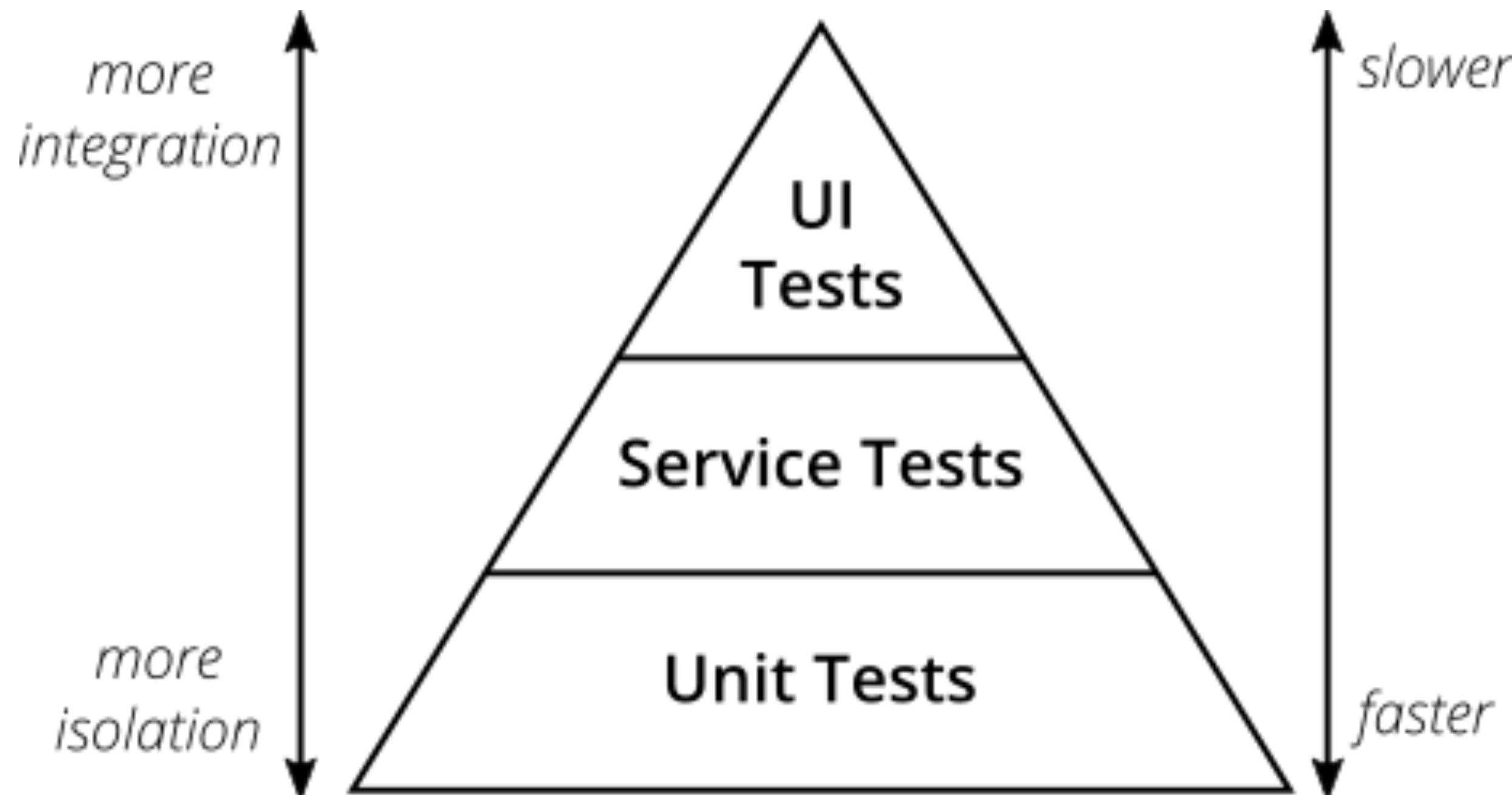
OWASP
Open Web Application
Security Project 2017

AUTOMAÇÃO DE TESTES: POR QUE ISSO IMPORTA TANTO?

POR QUE AUTOMAÇÃO IMPORTA EM AGILE?

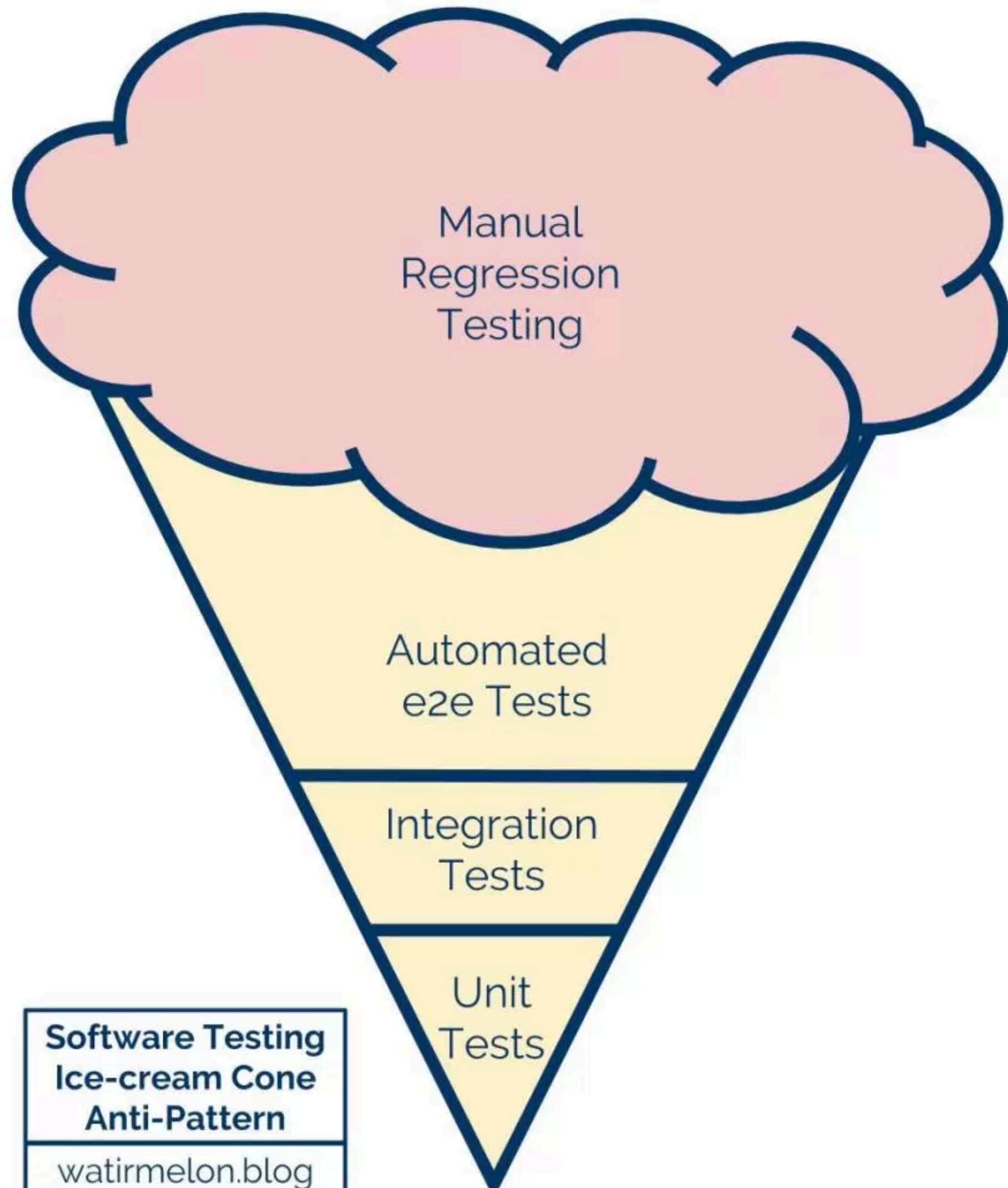
- Entre as práticas mais importantes
- A mais importante em muitos contextos
- Muitas das práticas dependem de testes automatizados:
 - Refatoração
 - Integração Contínua
 - Propriedade Coletiva do Código
 - Ainda: Design Simples, Releases Pequenos

O QUANTO TESTAR? O QUANTO AUTOMATIZAR?



- De maneira simplista e didática, começamos pela pirâmide de testes (Conceito do Mike Cohn).
- O importante é manter em mente: 1) escreva muitos pequenos testes de unidade e 2) escreva alguns testes de granularidade mediana e 3) escreva poucos testes de alto nível que teste o sistema de ponta a ponta.

ALGUNS TIMES CAEM NO ANTI-PADRÃO CONE DE SORVETE



- Pesadelo para qualquer time ágil
- Impossível confiar na estabilidade da sua base de código (com inúmeros *commits* e *refactorings*) sem uma malha considerável de teste automatizável)

MÉTRICAS E TESTES

MÉTRICAS (ISO/IEC/IEEE, 2010)

- **Métrica:** medida quantitativa do grau em que um sistema, um componente, ou processo possui um determinado atributo.
- **Métrica de Produto:** "Métrica usada para medir uma característica de qualquer produto intermediário ou na resultante do processo de desenvolvimento de software".
- **Métrica de Processo:** "Métrica usada para medir características de métodos, técnicas e ferramentas empregadas no desenvolvimento, implementação e manutenção de sistemas de software".

AUTOMAÇÃO DE MÉTRICAS

- Existem iniciativas de automação da coleta de diferentes tipos de métricas.
- Maioria destinada à coleta de métricas de produto a partir do código fonte.
 - Uma dessas iniciativas é o projeto **Mezuro** (<http://mezuro.org/>) (Meirelles, 2013)
 - Outra ferramenta bastante conhecida (métricas estáticas e dinâmicas) em um mesmo ambiente: **SonarQube** (<http://www.sonarqube.org/>).

AUTOMAÇÃO DE ANÁLISE ESTÁTICA

- Eficaz para:
 - Verificações rápidas e baratas de propriedades simples
 - Exemplo: análises de fluxo de dados simples podem identificar padrões anômalos
 - Verificações caras necessárias para propriedades críticas
 - Exemplo: ferramenta de verificação de estado finito para encontrar falhas de sincronização

MÉTRICA E FERRAMENTA: COBERTURA DE CÓDIGO

- Métrica Cobertura de Código (*code coverage*)
- Útil para guiar que testes fazer e os que não fazer

EXEMPLO DE ANÁLISE DE COBERTURA

```
public class EntryDataToAccountEntryConverter implements Converter<EntryData, AccountEntry> {  
  
    @Override  
    public AccountEntry convert(EntryData source) {  
        LocalDate dueDate = LocalDate.parse(source.getDueDate(), DateTimeFormatter.ofPattern("dd/MM/yyyy"));  
  
        AccountEntryType entryType = AccountEntryType.valueOf(source.getEntryType());  
        final AccountEntry accountEntry;  
  
        if (entryType == AccountEntryType.EXPENSE) {  
            accountEntry = new Expense();  
        } else if (entryType == AccountEntryType.RECEIPT) {  
            accountEntry = new Receipt();  
        } else {  
            throw new IllegalArgumentException("unsupported entry type");  
        }  
  
        accountEntry.setDueDate(dueDate);  
        accountEntry.setAmount(new BigDecimal(source.getAmount()));  
        accountEntry.setCategory(new Category(source.getCategory()));  
        accountEntry.setDescription(source.getDescription());  
  
        return accountEntry;  
    }  
}
```

- Verde: linha de código testada por algum teste automatizado;
- Amarela: um cenário não foi testado (ex: uma condição de um if);
- Vermelha: código foi testado por testes automatizados.

OUTRAS MÉTRICAS

- Defeitos nos testes
- Defeitos após liberação
- Problemas abertos
- Questões abertas
- Tempo gasto corrigindo defeitos
- Defeitos incluídos após correção
- Cobertura de código
- Complexidade de código
- Custo do retrabalho

Leia mais sobre métricas de código na tese do Paulo Meirelles: <http://www.teses.usp.br/teses/disponiveis/45/45134/tde-27082013-090242/pt-br.php>

EFEITO DISFUNCIONAIS DE MÉTRICAS

- Ao se criar uma **métrica**, é comum seres humanos se esquecerem do objetivo original e passarem a **perseguir** um certo número/alvo: **efeito disfuncional de métricas**.
- Este fenômeno também é conhecido como *You Get What You Measure* (YGWYM) e *Vanity Metrics* (*métricas que servem à vaidade*)
- Exemplo: perseguir cobertura de testes de 95% → programadores/as começam a codificar testes *sem assertion* (!!!) apenas para ver o número aumentar → péssima qualidade de código! → oposto do desejado com a métrica. <https://martinfowler.com/bliki/AssertionFreeTesting.html>
- Leia mais em: <http://www.claudiameiro.org/wp-content/uploads/2016/10/9maneirasdefalharcomKPIs.pdf>

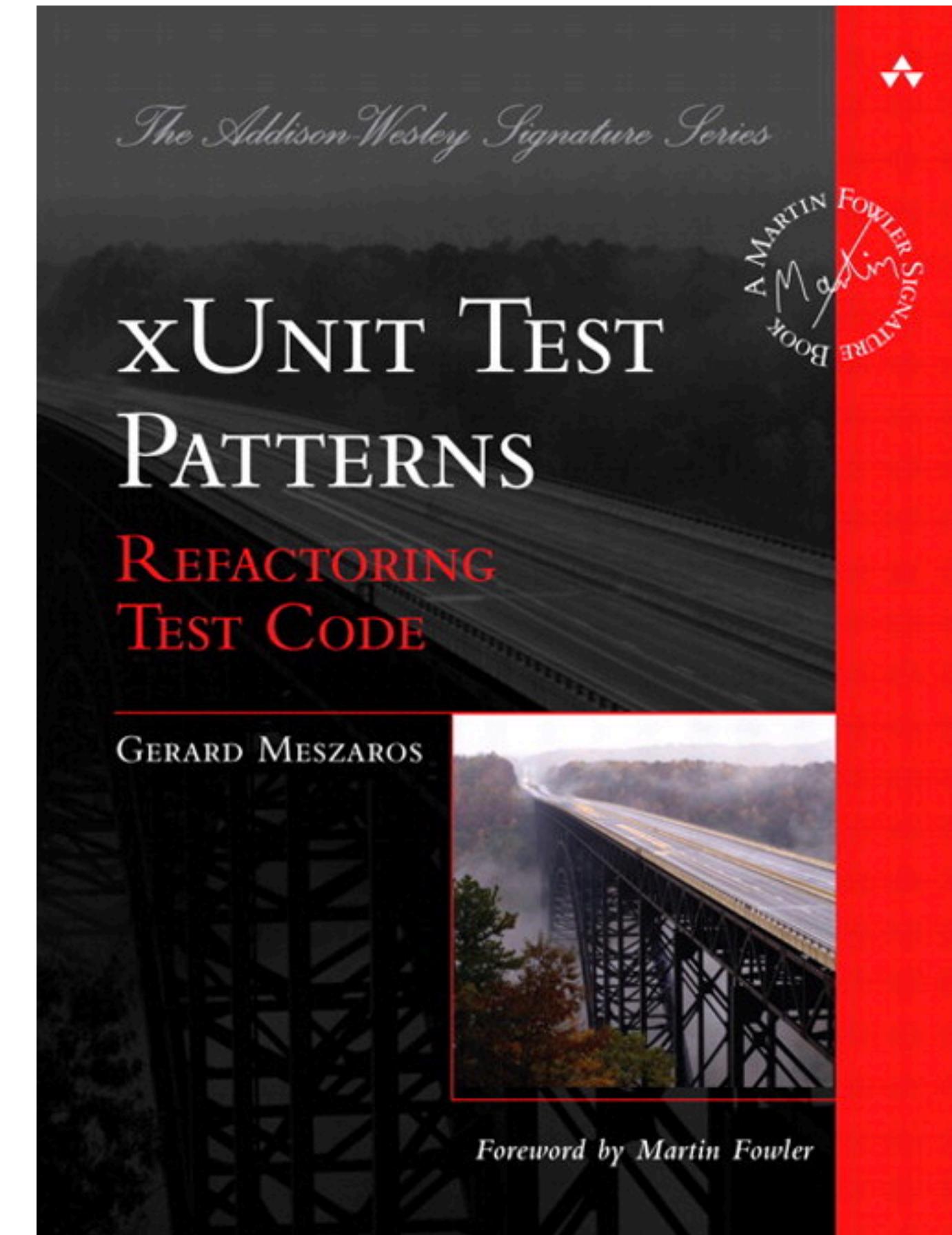
MÉTRICAS ACIONÁVEIS VS MÉTRICAS DE VAIDADE

- Uma boa métrica conduz ação, conduz mudança positiva no time.
- Quando a métrica deixar de dirigir mudança, está na hora de deixá-la.
- Métricas acionáveis também podem se tornar disfuncionais ao longo do tempo, observar e evoluir a forma de medição.
- Há muito debate sobre esse tema, busque **actionable and vanity metrics** para manter-se atualizada/o!

PADRÕES, ANTI-PADRÕES, BOAS PRÁTICAS

PADRÕES DE TESTES DE UNIDADE

- *Data-Driven Tests*
- *Generated Value*
- *Testcase Class*
- *Testcase Class per Class*
- *Testcase Class per Feature*
- *Unfinished Test Assertion*
- ...



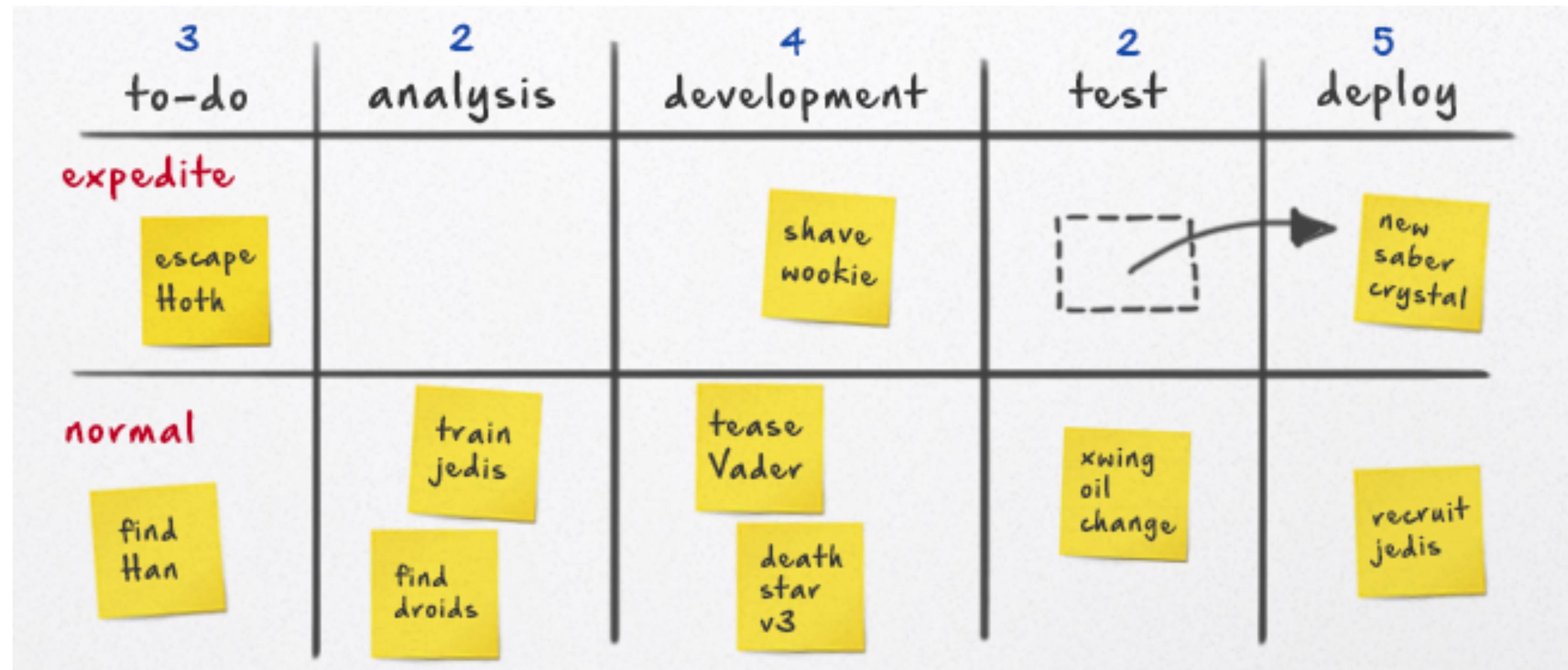
ANTI-PADRÕES DE TESTES

- Código dos Testes:
 - Testes Obscuros
 - Código Complexo: condições
 - Difícil de testar
 - Replicação de código de testes
 - Lógica de testes em produção
- Cheiros de comportamento:
 - Testes frágeis
 - Intervenções manuais
 - Testes lentos

GESTÃO DA QUALIDADE

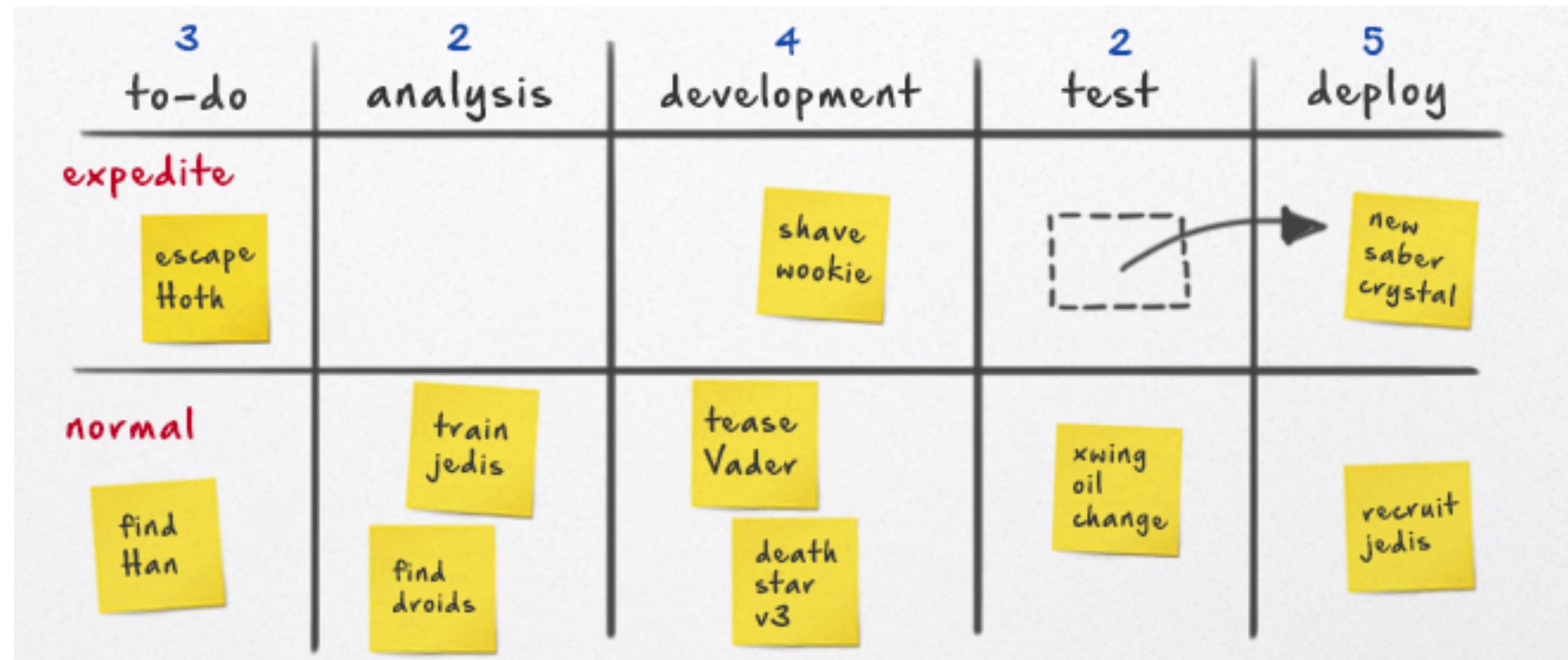
**QUALIDADE É MISSÃO DE TODOS.
QUALIDADE NÃO É UMA FASE DO SEU PROJETO.**

QUALIDADE AO LONGO DO DESENVOLVIMENTO



BACKLOG	KICK-OFF DA HISTÓRIA	DURANTE O DESENVOLVIMENTO:	DESK CHECKING
<ul style="list-style-type: none">• História INVEST foi priorizada	<ul style="list-style-type: none">• BAs DEVs QAs	<ul style="list-style-type: none">• DEVs• QAs	<ul style="list-style-type: none">• BAs DEVs QAs checam se a história está pronta para QA

QUANDO UM DEFEITO/ERRO/FALHA FOR IDENTIFICADO



ENCONTREI DURANTE O DESENVOLVIMENTO:

- resolva aqui mesmo, não precisa “registrar” um bug

ENCONTREI DURANTE TESTES/QA:

- volte a história para DEV
- se for um bug mais simples e o cliente concordar, pode ser registrado e deixado para depois.

ENCONTREI DEPOIS DE PRONTO:

- registre o defeito e gerencie

GERÊNCIA DE DEFEITOS

- Registro:
 - *Unique identifier (generated automatically by the tool)*
 - *Product / project name*
 - *Defect description*
 - *Priority*
 - *Type / classification*
 - *State*
 - *Name of the author of this entry*
 - *Product version*
- Classificação
 - *Requirements incorrect or misinterpreted*
 - *A Design error*
 - *An Implementation error*
 - *Misunderstanding of external environment*
 - *Error due to previous miscorrection of an error*
- Acompanhamento (Tracking)
 - Priorização
 - Visibilidade aos interessados
 - Foco em entregar

FERRAMENTAS PARA GERÊNCIA DE DEFEITOS

- Apóiam o registro e o tracking
- Possibilitam a geração de métricas para melhoria de processo
- Exemplos:



JIRA Software



REDMINE
flexible project management

trac
Integrated SCM & Project Management

REFERÊNCIAS

- Tech Talk ministrada na UnB (2018/01): COMO EU APRENDEI QUE TESTAR É IMPORTANTE? Prof. Dr. Maurício Aniche, TU Delft. www.mauricioaniche.com/talks/2018/unb.html
- Lisa Crispin, Janet Gregory. Agile Testing, "Agile Testing: A Practical Guide for Testers and Agile Teams", Addison-Wesley Signature Series (Cohn), 2008.
- Gerard Meszaros, "xUnit Test Patterns, Refactoring Test Code", Addison-Wesley, 2007.
- Authors Mauro Pezzè, Michal Young. "Software testing and analysis: process, principles, and techniques". Wiley, 2008. Livro online: <http://ix.cs.uoregon.edu/~michal/book/>
- Robert V. Binder, "Testing Object-Oriented Systems", Addison-Wesley Professional, 1999.
- L. Crispin, T. House, "Testing Extreme Programming", Addison-Wesley, 2005.
- R. Mugridge, W. Cunningham, "Fit for Developing Software", Prentice Hall, 2006.
- M. Delamaro, J. Maldonado, M. Jino, "Introdução ao Teste de Software", Campus, 2007.

MAIS MATERIAIS SOBRE CODE REVIEW/INSPEÇÃO

- Examples of **Code Review Checklists and Guides** <https://medium.com/@andregridnev/examples-of-code-review-checklists-and-guides-2dfed082a86d>
- **How We Do Code Review** <https://blogs.msdn.microsoft.com/vsappcenter/how-the-visual-studio-mobile-center-team-does-code-review/>
- **Characteristics of Useful Code Reviews: An Empirical Study at Microsoft** <https://www.microsoft.com/en-us/research/publication/characteristics-of-useful-code-reviews-an-empirical-study-at-microsoft/>
- Veja também a apresentação: <https://www.slideshare.net/mgreiler/on-to-code-review-lessons-learned-at-microsoft>
- Exemplo de **Desk Check**: https://sites.google.com/a/campioncollege.com/it_eveningschoool/problem-solving-and-programming/desk-check-guide
- **Inspeção de Fagan**: M. E. Fagan, "Design and code inspections to reduce errors in program development," in *IBM Systems Journal*, vol. 38, no. 2.3, pp. 258-287, 1999
- **Exemplo de checklist de inspeção**: <http://teaching.csse.uwa.edu.au/units/CITS2220/readings/JavalnspectionCheckList.pdf>