



Qualidade de Software

Investigating Severity Thresholds for Test Smells

Davide Spadini, Martin Schvarcbacher, Ana-Maria Oprescu, Magiel Bruntink,
Alberto Bacchelli.

Ítalo Lima Dantas (italolimad@alu.ufc.br) UFC

Autores

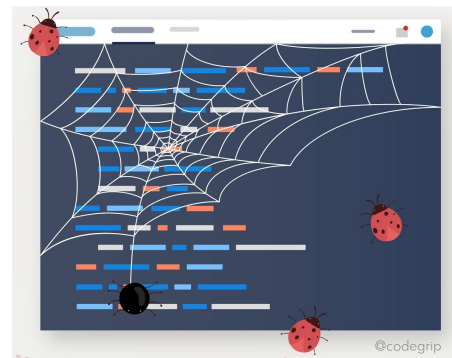
- Davide Spadini, *Delft University of Technology Amsterdam.*
- Martin Schvarcbacher, *Software Improvement Group, Amsterdam.*
- Ana-Maria Oprescu, *University of Amsterdam, Amsterdam.*
- Magiel Bruntink, *Software Improvement Group, Amsterdam.*
- Alberto Bacchelli, *University of Zurich, Zurich.*

Introdução

O que são Test Smells ?

Test Smells (cheiros de teste, em tradução livre) é um nome popular atribuído às más decisões de projeto implementadas no código de teste, sim, também existem erros no código de testes, assim como no código de produção, ambas violações dificultam e exigem um maior esforço na manutenibilidade do sistema.

Sim, existem erros no código responsável por identificar erros, **afinal, onde há código, haverá de ter erros.**



Types of Test Smells ☆ 📌 ☁️

Arquivo Editar Ver Inserir Formatar Dados Ferramentas Complementos Ajuda [A última edição foi feita há 7 minutos](#)




🔍 ↶ ↷ 🖨️ 📑 100% ▼ R\$ % .0 .00 123 ▼ Padrão (Ari... 10 ▼ B I S A 🔗 🏠 📄 📊 📈 📉 📋 📌 📎 📏 📐 📱 📲 📳 📴 📵 📶 📷 📸 📹 📺 📻 📼 📽 📾 📿 📰 📱 📲 📳 📴 📵 📶 📷 📸 📹 📺 📻 📼 📽 📾 📿 📰 📱 📲 📳 📴 📵 📶 📷 📸 📹 📺 📻 📼 📽 📾 📿 📰

	A	B
17		
1	Test smell	Description
2	Mystery Guest	A test that uses external resources (e.g., file containing test data)
3	Resource Optimism	A test that makes optimistic assumptions about the state/existence of external resources
4	Eager Test	A test that tries to verify too many functionalities which can lead to difficulty in understanding the test code
5	Assertion Roulette	A test that has multiple assertion statements that do not provide any description of why they failed
6	Conditional Test Logic	A test that has control flow statements inside a test
7	General Fixture	It occurs when the test setup method creates fixtures (class fields used by the test cases) and a portion of the tests use only a subset of the fixtures
8	Empty Test	It occurs when a test method does not contain executable statements
9	Magic Number Test	A test that contains so-called "magic numbers", which are numbers used in assertions without any explanation where they come from and their value may not be immediately clear from ju
10	Sleepy Test	A test which contains thread suspension calls
11	Verbose Test	A test that is too long and hard to understand
12	Ignore Test	A test method or class that contains the @Ignore annotation
13		
14		
15		




Types of Test Smells ☆ 📌 ☁️

Link para a Planilha

Tipos de Test Smells

 Test Smell	 Description	 Problem
Mystery Guest	Um teste que utiliza recursos externos.	A utilização de recursos externos (como por exemplo arquivos que contém dados de teste) dificulta a compreensão e o rastreamento. Além disso, pode introduzir dependências (sujeitas à falhas) ocultas.
Resource Optimism	Um teste que faz suposições otimistas sobre o estado / existência de recursos externos.	Por tratar com otimismo a existência e/ou estado dos recursos externos pode implicar em um comportamento não determinístico dos testes, com variação de resultados.
Eager Test	Um teste que tenta verificar muitas funcionalidades.	Aumenta a dependência entre as funcionalidades avaliadas, torna-se difícil de ler, e se prejudica a legibilidade pode consequentemente prejudicar as etapas de documentação.
Assertion Roulette	Um teste que tem várias declarações que não fornecem nenhuma descrição do motivo da falha.	Dificulta a investigação do problema e pode consequentemente dificultar as etapas de correção/refatoração.
Conditional Test Logic	Um teste que tem instruções de fluxo de controle (condicionais) dentro de um teste.	Assim como no contexto de code smells, isso pode ocasionar problema para os testes, por terem vários pontos de ramificação.
General Fixture	Ocorre quando o método de configuração de teste cria campos de classe usados pelos casos de teste e uma parte dos testes usa apenas um subconjunto dos acessórios.	Pode causar problemas de espera, em uma analogia, faz muitas requisições, mas só usa realmente um subconjunto das respostas.

[Link para a Tabela](#)

 Test Smell	 Description	 Problem
Empty Test	Ocorre quando um método de teste não contém instruções executáveis.	Um teste vazio pode ser considerado problemático e mais perigoso do que não ter um caso de teste, já que JUnit indicará que o teste passa mesmo se não houver instruções executáveis presentes no corpo do método.
Magic Number Test	Um teste que contém "números mágicos", que são números usados em afirmações, sem nenhuma explicação de onde vêm e seus valores podem não ser imediatamente claros apenas olhando para o código de teste.	Os números mágicos devem ser substituídos por uma constante nomeada, onde o nome descreve de onde vem o valor ou o que ele representa.
Sleepy Test	Um teste que contém chamadas de suspensão de encadeamento.	O uso dessa chamada de método introduz um atraso adicional na execução do teste.
Verbose Test	Um teste muito longo e difícil de entender. Legibilidade prejudica.	Testes muito longos impedem que sejam usados como documentação e são mais difíceis de manter devido à sua complexidade.
Ignore Test	Um método de teste ou classe que contém a anotação @Ignore.	os métodos de teste que contêm @Ignore adicionam sobrecarga desnecessária em relação ao tempo de compilação e aumentam a complexidade e a compreensão do código.

Esses foram os selecionados, porque exigem a visualização do código fonte de teste completo.

Objetivos

Objetivo Geral

- **Investigar limites de gravidade para cheiros de teste.**

- **Investigar os test smells de forma análoga ao processo realizado para code smells, destacando a importância que esses têm para a manutenção de um software e a pouca atenção que recebem no cenário de pesquisa atual.**

Objetivos Específicos

- **Calibrar os limites de detecção de forma que os níveis de gravidade possam ser atribuídos a uma instância de test smells, permitindo que os desenvolvedores se concentrem nos de maior gravidade. (Priorização).**
- **Melhorar a acessibilidade da detecção automática de cheiros de teste, integrando-se às ferramentas de desenvolvedor existentes. (Automatização).**

Dados da Pesquisa

Tabelas com detalhamento descritivo dos projetos do github, analisados no estudo.

Projetos Eclipse	353
Projetos Apache	1136
Total	1489

Tabela 1 - Descrição dos projetos do github.

Description of Dataset

 Total de Projetos	 Linhas de Código (Total)	 Linhas de Código (Média)	 Linhas de Código (Mediana)
1489	25.356.827	31.617	6.650

Tabela 2 - Descrição dos projetos do github.

Comparativo com o Projeto utilizado no outro trabalho da Disciplina, número de linhas de código: 2798.

Trabalhos Relacionados

Linha Cronológica

- ❖ Van Rompaey et al. criou uma ferramenta de detecção de test smells baseada em métricas para Java para detectar General Fixtures e Eager Tests.
- ❖ Posteriormente, Breugelmans e van Rompaey desenvolveram a ferramenta TestQ, que funciona com código de teste C, C ++ e Java para detectar os seguintes odores de teste: Assertion Roulette, Eager Test, Empty Test, For Testers Only, General Fixture, Indented Test (Equivalente ao Conditional Test Logic), Indirect Test, Mystery Guest, Sensitive Equality e Verbose Test.

Linha Cronológica

- ❖ O trabalho de Greiler et al. focou na identificação de problemas comuns com acessórios de teste. Eles implementaram uma ferramenta chamada TestHound, que funciona no nível de bytecode JVM e pode detectar os seguintes odores de teste: General Fixture, Test Maverick, Dead Field, Lack of Cohesion of Test Methods, Obscure In-Line Setup e Vague Header.
 - Nessa ferramenta, eles mostraram aos desenvolvedores o código impactado pelos cheiros de teste junto com dicas de como refatorar o código de teste.
- ❖ Palomba et al. desenvolveu TASTE, que pode detectar General Fixtures, Eager Tests, and Lack of Cohesion of Methods (usando técnicas de recuperação de informação), evitando assim a necessidade de analisar totalmente o código de teste, sua ferramenta mostra uma melhor precisão e recall do que as ferramentas baseadas em AST TestQ e TestHound .

Linha Cronológica

- ❖ Bavota et al. estudou a difusão de test smells em projetos de código aberto e industriais. Para ajudar na pesquisa, eles criaram uma ferramenta de detecção de test smells para Resource Optimism, Indirect Testing, Test Run War, Mystery Guest, General Fixture, Eager Test, Lazy Test, Assertion Roulette, For Testers Only, Test Code Duplication e Sensitive Equality.
- ❖ Zhang et al. focou nas dependências entre os testes, investigando empiricamente os sistemas de rastreamento de problemas. Eles desenvolveram uma ferramenta que identifica dependências entre os testes em um nível de suíte de teste.
- ❖ Gambi et al. também investigou a detecção de dependências de teste e desenvolveu a ferramenta PraDeT, que também requer a execução de testes para encontrar dependências.

Ferramentas

- **tsDetect**

- **BetterCodeHub**

tsDetect

Ela trabalha com projetos Java JUnit e tem suporte para o JUnit 4 que pode até ser estendido para o JUnit 5.

Precision e Recall acima de 85 %.

Vantagens da Ferramenta:

- **Código Aberto**
- **Desenvolvido recentemente (2018)**
- **Usa uma detecção baseada em AST de cheiros de teste e tem suporte para a adição de novos tipos e as respectivas regras de detecção.**

tsDetect

Inovação

- Em 1 e 2, os arquivos de teste e produção são identificados a partir da estrutura do projeto.
- Em 3 e 4 o tsDetect verifica se os arquivos de teste exibem odores de teste.
- Em 5, os resultados do processo de detecção de test smells são salvos.

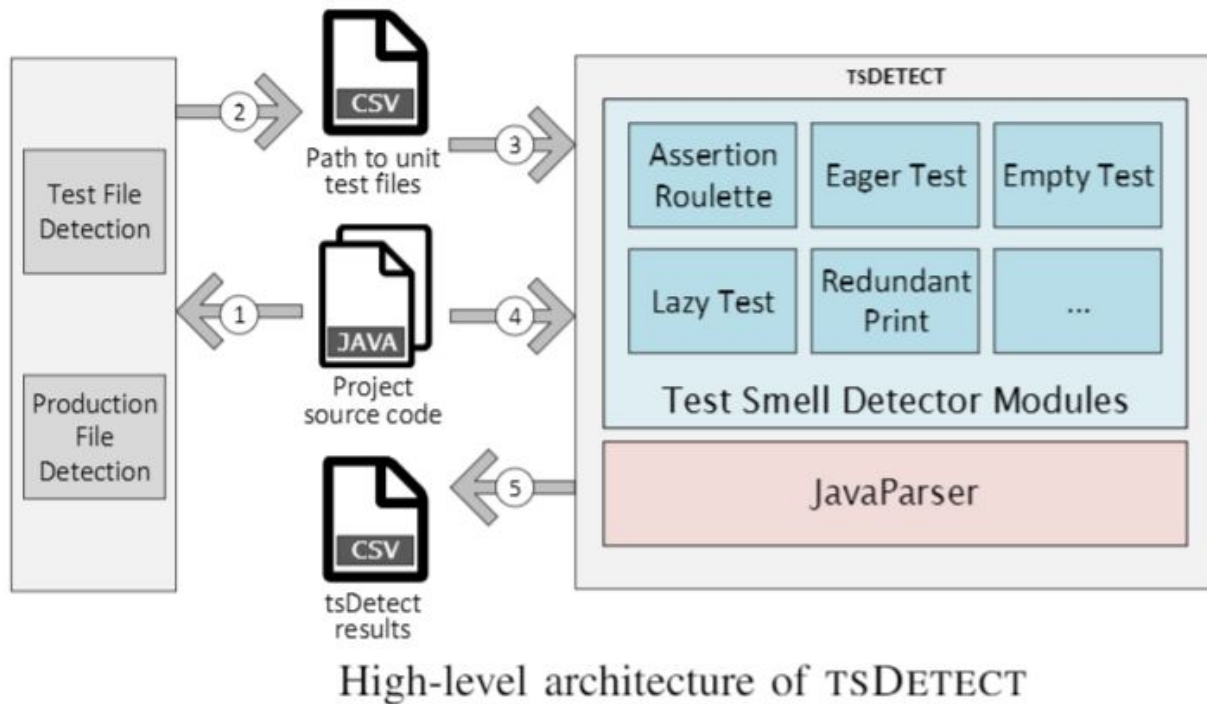


Figura 1. Arquitetura da ferramenta tsDetect.

O que é Precision ?

- Quando os algoritmos preveem que sim, em quantas ele está realmente correto?

(True Positives / predicted yes)

O que é Recall ?

- Quando é realmente sim, com que frequência o algoritmo prediz que sim?

(True Positives/Actual yes)

[Link para Exemplificação de Métricas](#)

💡 **TP = True Positives.** Aqueles que foram classificados e realmente pertencem à classe. No contexto dos tests smells: Um código que a ferramenta identificou que possui um test smell x, e o código realmente possui.

💡 **FP = False Positives.** Aqueles que foram classificados e não pertencem à classe. No contexto dos tests smells: Um código que a ferramenta identificou que possui um test smell x, e o código não possui.

💡 **FN = False Negatives.** Aqueles que não foram classificados e pertencem à classe. No contexto dos tests smells: Um código que a ferramenta não conseguiu identificar que possui um test smell x, e o código possui.

💡 **TN = True Negatives.** Aqueles que não foram classificados e realmente não pertencem à classe. No contexto dos tests smells: Um código que a ferramenta identificou que não possui um test smell x, e o código realmente não possui.

Figura 2. Definição de conceitos utilizados para as métricas.

BetterCodeHub

O que é ?

É uma ferramenta que auxilia o desenvolvedor na correção de bugs, a ponto de definir o estado de “done” do seu código.

- O Better Code Hub verifica sua base de código do GitHub em relação a 10 diretrizes de engenharia elaboradas pela autoridade em qualidade de software, Software Improvement Group (SIG).
- Analisa mais de 15 milhões de código todas as semanas, a empresa possui o maior benchmark da indústria, contendo mais de 10 bilhões de linhas de código em mais de 200 tecnologias.



1º Objetivo da Pesquisa

Definindo os Limites de Severidade

tsDetect

Para criar novas classificações de gravidade para os test smells, antes, os autores definiram métricas para os mesmos.

- Empty Test e Ignored Test foram excluídos dessa etapa de métricas, devido ao fato de ser de natureza binária (um teste é vazio ou não é) e os outros podem ser quantificados, portanto, os autores declararam qualquer ocorrência desses dois tipos supracitados como de prioridade máxima.
- Essas métricas são medidas em quantidades, exemplo: Quanto maior o número de arquivos externos utilizados, maior a tendência de aumentar a prioridade do test smell.

Metrics of Test Smells

<u>Aa</u> Test Smell	 Metric
<u>Assertion Roulette</u>	Afirmações sem descrição
<u>Conditional Test Logic</u>	Declarações condicionais
<u>Eager Test</u>	Chamadas de método de produção
<u>General Fixture</u>	Fixtures não usadas
<u>Magic Number Test</u>	Números mágicos
<u>Mystery Guest</u>	Arquivos externos utilizados
<u>Resource Optimism</u>	Arquivos não verificados quanto à existência
<u>Sleepy Test</u>	Threads de chamadas suspensas
<u>Verbose Test</u>	Declarações

Tabela X. Métricas selecionadas para os Test Smells definidos.

What the Smell? An Empirical Investigation on the Distribution and Severity of Test Smells in Open Source Android Applications.

O que aborda ?

- Ampla adoção de dispositivos móveis e a facilidade/larga escala de desenvolvimento.
- Fatores de qualidade para o sucesso de um aplicativo.
- Importância do estudo das más práticas de programação para os testes. O impacto na qualidade e manutenção.
 - Estende o conjunto existente de más práticas de código de teste, introduzindo novos cheiros de teste.
 - Fornece à comunidade de engenharia de software uma ferramenta de detecção de cheiro de teste de código aberto.

2º Objetivo da Pesquisa

Automatizando a detecção automática de Test Smells

Ações Executadas

Nessa etapa para explorar a percepção humana sobre os test smells, os autores modificaram o Better Code Hub, com base nos limites descobertos no próprio trabalho e apresentaram test smells encontrados nas bases de código dos desenvolvedores que estavam participando da pesquisa.

- O front-end foi modificado para mostrar os resultados da análise junto com um feedback operacional para cada cheiro de teste encontrado.
- Isso facilita o alcance do 2º viés proposto pelos autores, que vai além de simplesmente exibir métricas para os testes, explorando um feedback maior que está ligado com a parte humana da etapa.

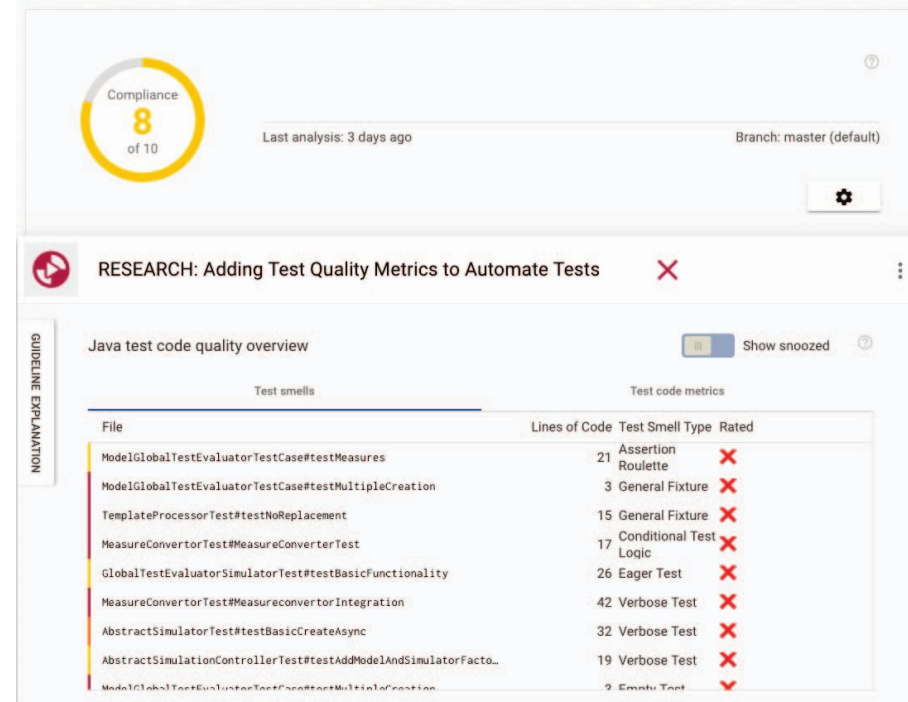



Figura 3. Exemplo de detecção de test smell.

Exemplo da Interação

 ModelGlobalTestEvaluatorTestCase#testMeasures GENERATE GITHUB ISSUE ×

Assertion Roulette

Occurs when a test method has multiple non-documented assertions. Multiple assertion statements in a test method without a descriptive message impacts readability/understandability/maintainability as it's not possible to understand the reason for the failure of the test. [Further reading](#)

Please rate the impact of this test smell instance on the test suite maintainability (*):

Don't know

False Positive

Very Low

Low

Medium

High

Very High

What action will you take with this test smell (*):

Please choose ▼

Estimated effort to refactor: person-hours

Additional remarks (how to refactor, ...):

Submit your rating

```
61
62
63  @Test
64  public void testMeasures() throws Exception {
65      modelParams.put("__inputCount__", 0);
66      modelParams.put("__MeasuresCount__", 1);
67      modelParams.put("__PermanentFailureEntry__", false);
68      modelParams.put("__GlobalMeasureIdMapping__", getResourceContent("GlobalTestEvaluator_GlobalMeasureIdMapping.json"));
69      modelParams.put("__GlobalTargetMeasures__", getResourceContent("GlobalTestEvaluator_GlobalTargetMeasures.json"));
70      model = new ModelGlobalTestEvaluator(modelParams);
71      //>20 && <= 100
72      Measure measure1a = new Measure(150);
73      Measure measure1b = new Measure(50);
74
75      model.setValue("name1", mapper.writeValueAsString(measure1a));
76      model.step(1);
77      assertThat(model.isPassed()).isFalse();
78      assertThat(model.getMessages()).isNullOrEmpty();
79
80      model.setValue("name1", mapper.writeValueAsString(measure1b));
```

Path: testframework-simulators/src/test/java/testframework/mosaik/models/ModelGlobalTestEvaluatorTestCase.java

Formulário:

1. Avalie o impacto desta instância de test smell na capacidade de manutenção do conjunto de testes.
2. O que você fará com este test smell?
3. Esforço em horas estimado para refatorar (uma pessoa).
4. Observações adicionais.

Figura X. Detalhes do BCH sobre um teste smell encontrado, com o formulário de pesquisa correspondente.

Resultados

Definindo os Limites de Severidade

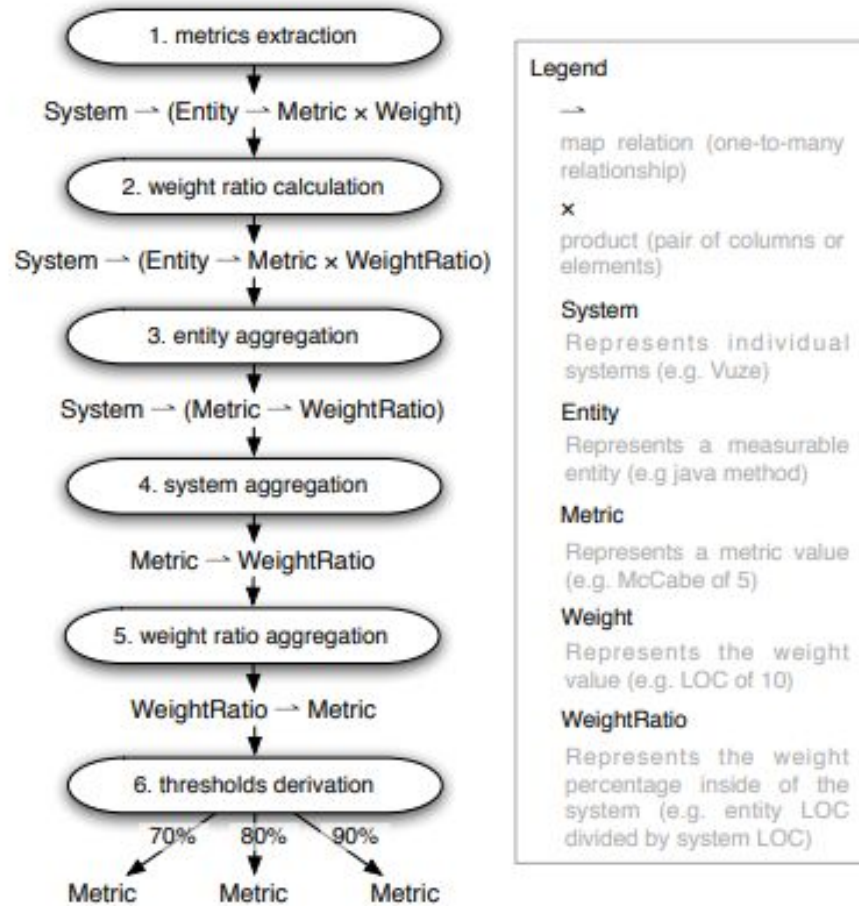
Os test smells foram classificados de acordo com a sua frequência nas bases de código e incluídos em percentis de (70, 80 e 90). Para cada test smell é definido um intervalo que baseará a sua classificação de gravidade, podendo ser: Média, alta, muito alta.

Test Smell	Severity Threshold		
	Medium	High	Very High
Assertion Roulette	3	5	10
Eager Test	4	7	39
Verbose Test	13	19	30
Conditional Test Logic	0	1	2
Magic Number Test	0	0	1
General Fixture	0	0	0
Mystery Guest	0	0	0
Resource Optimism	0	0	0
Sleepy Test	0	0	0

Exemplo para o Assertion Roulette

- Se ocorrência for inferior a 3: Pertence aos melhores 70% do corpus.
- Se ocorrência estiver entre 5 e 10: Considerado de alta severidade (Pertence aos 20% piores do corpus).
- Se ocorrência estiver acima de 10 deve ser considerada gravidade muito alta (Pertence aos piores 10% do corpus).

Tabela 1. Limites de Gravidade Calculados.



- A metodologia para classificação de gravidade utilizada foi baseada na derivação de limiar baseada em benchmark propostas por Tiago L. Alves, Christiaan Ypma e Joost Visser.
- **Definição dos percentis: 70, 80 e 90.**

Figura 4. Arquitetura da metodologia utilizada para a derivação de benchmark.

Comparando os Limites de Severidade

Na tabela anterior, observa-se que apesar do percentil definido como 90, ainda há métricas que apresentaram 0, nesses casos, se obtivermos uma única ocorrência, já será definida como gravidade muito alta (caso da métrica Magic Number Test). É notório que apenas 4 test smells foram significativamente impactados, e esses são os que compõem os resultados a serem comparados.

Table 4: Test smells' distribution across systems (N=1,489)

Test Smell	Old Thresholds		Derived Thresholds	
	Impacted	Not impacted	Impacted	Not impacted
Assertion Roulette	56,177 (50.9%)	54,278 (49.1%)	34,156 (30.9%)	76,299 (69.1%)
Cond. Test Logic	29,077 (26.3%)	81,378 (73.7%)	20,095 (18.2%)	90,360 (81.8%)
Empty Test	1,280 (1.2%)	109,175 (98.8%)	1,280 (1.2%)	109,175 (98.8%)
General Fixture	8,829 (8.0%)	101,626 (92.0%)	8,829 (8.0%)	101,626 (92.0%)
Mystery Guest	6,071 (5.5%)	104,384 (94.5%)	6,071 (5.5%)	104,384 (94.5%)
Sleepy Test	4,500 (4.1%)	105,955 (95.9%)	4,500 (4.1%)	105,955 (95.9%)
Eager Test	29,333 (26.6%)	81,122 (73.4%)	14,284 (12.9%)	96,171 (87.1%)
Ignored Test	3,105 (2.8%)	107,350 (97.2%)	3,105 (2.8%)	107,350 (97.2%)
Resource Optimism	7,345 (6.6%)	103,110 (93.4%)	7,345 (6.6%)	103,110 (93.4%)
Magic Num. Test	18,920 (17.1%)	91,535 (82.9%)	18,920 (17.1%)	91,535 (82.9%)
Verbose Test	70,461 (63.8%)	39,994 (36.2%)	36,080 (32.7%)	74,375 (67.3%)

- Assertion Roulette
- Conditional Test Logic
- Eager Test
- Verbose Test

Tabela 2. Comparação entre os novos e antigos limites de gravidade.

Avaliando Estatisticamente os Limites de Severidade

- Verbose Test e o Conditional Test Logic mostraram uma relação estatisticamente significativa alta ($p < 0,001$) e um coeficiente de Spearman forte ($0,6 \leq r_s \leq 0,79$), enquanto o Eager Test e a Assertion Roulette mostraram uma relação estatisticamente significativa inferior ($p < 0,05$) e um coeficiente de Spearman mais fraco ($0,2 \leq r_s \leq 0,39$).

Test Smell	Responses	p	r_s
Eager Test	42	0.027	0.342
Conditional Test Logic	36	<0.001	0.753
Verbose Test	51	<0.001	0.679
Assertion Roulette	47	0.016	0.350

Tabela 3. Avaliação estatística para os test smells impactados.

Percepções dos Desenvolvedores

Para este segundo objetivo, os autores pediram que os desenvolvedores indicassem para cada instância de test smell se eles a classificariam como uma instância válida a ser removida, qual prioridade eles dariam para a refatoração, e quanto tempo levaria de acordo com eles.

Test Smell	Responses	Immediate Refactor	Long-term Refactor	No Action
Empty Test	14	71.43%	21.43%	7.14%
Sleepy Test	13	61.54%	23.08%	15.38%
Mystery Guest	10	60.00%	10.00%	30.00%
Resource Optimism	12	58.33%	16.67%	25.00%
Cond. Test Logic	39	53.85%	33.33%	12.82%
Verbose Test	54	48.15%	40.74%	11.11%
Magic Number Test	23	47.83%	34.78%	17.39%
Assertion Roulette	55	29.09%	50.91%	20.00%
Eager Test	51	21.57%	50.98%	27.45%
Ignored Test	10	20.00%	50.00%	30.00%
General Fixture	20	20.00%	40.00%	40.00%

Tabela 4. Ações realizadas pelos usuários, por cheiro de teste identificado.

Observações sobre os resultados

- O test smell Empty Test apresentou o maior índice proporcional (71.43%) de avaliações para refatoração imediata de acordo com os desenvolvedores.
- Os test smells General Fixture e Ignored Test apresentaram o menor índice proporcional de avaliações refatoração imediata (20.00%), de acordo com os desenvolvedores e os maiores índices proporcionais para a opção de não refatorar (30.00% e 40.00%) respectivamente.

Percepções dos Desenvolvedores

Test Smell	Responses	Very Low	Low	Medium	High	Very High
Assertion Roulette	47	17.0%	27.7%	38.3%	14.9%	2.1%
Conditional Test Logic	36	2.8%	25.0%	30.6%	22.2%	19.4%
Eager Test	42	31.0%	26.2%	21.4%	16.7%	4.8%
Empty Test	13	0.0%	15.4%	7.7%	46.2%	30.8%
General Fixture	13	23.1%	30.8%	38.5%	7.7%	0.0%
Ignored Test	8	12.5%	12.5%	12.5%	37.5%	25.0%
Magic Number Test	20	15.0%	20.0%	30.0%	30.0%	5.0%
Mystery Guest	8	0.0%	12.5%	37.5%	50.0%	0.0%
Resource Optimism	10	10.0%	0.0%	20.0%	60.0%	10.0%
Sleepy Test	11	0.0%	9.1%	18.2%	63.6%	9.1%
Verbose Test	51	13.7%	23.5%	23.5%	27.5%	11.8%

Tabela X. Avaliações de impacto na capacidade de manutenção do conjunto de testes

Observações sobre os resultados

- O test smell Empty Test apresentou o maior índice proporcional de avaliações muito alto, para impacto na manutenção, de acordo com os desenvolvedores.
- O test smell General Fixture apresentou o maior índice proporcional de avaliações muito baixo, para impacto na manutenção, de acordo com os desenvolvedores.

Contribuição da Pesquisa

- Limites de gravidade calibrados para detectores de Test Smells.
- Um mecanismo para classificar a gravidade de certos test smells, permitindo que as ferramentas os classifiquem em categorias distintas com base em sua gravidade. Isso possibilita que os desenvolvedores se concentrem apenas nos mais críticos.
- Uma integração de um detector de test smells automático dentro de uma ferramenta de qualidade de código baseada no GitHub (BCH).
- Uma avaliação das percepções dos desenvolvedores de test smells dentro de suas próprias bases de código, integrando os novos limites em uma ferramenta de qualidade de código baseada no GitHub (BCH), validando assim os níveis de gravidade estabelecidos.

Ameaças a validade

Um ponto interessante ao final do artigo é que o autor cita 3 fatores que “ameaçam” a qualidade do estudo produzido. São esses:

- **Construct validity: Instrumentos de pesquisa. Ferramenta tsDetect.**
- **Internal validity: Fatores que podem afetar as variáveis e as relações que estão sendo investigadas. Experiência dos desenvolvedores com o BCH**
- **External validity: Generalização dos resultados. Diz respeito sobre as conclusões obtidas, mas ressalva a validade e o tamanho do corpus utilizado.**

Conclusão

Remember

- O estudo realiza uma investigação e classificação dos test smells com base em sua gravidade.
- Com isso, os autores:
 - Definiram métricas para cada test smell.
 - Aplicaram a derivação de limite com base em benchmark em uma amostra de projetos open source (Apache e Eclipse).
 - Classificação de gravidade (de baixa a muito alta).
 - Para os test smells Assertion Roulette, Eager Test, Verbose Test and Conditional Test Logic os limites de gravidade definidos são não binários e com diferença estatística significativa, se comparado com o antigo modelo.
 - Estudo com os desenvolvedores para verificar a opinião dos mesmos referente a necessidade e urgência de refatoração dos test smells e do impacto dos mesmos na manutenção do código.
 - Feedback mais completo (humanização do processo) para os test smells produzidos. Integração da detecção com a qualidade do código.

Dúvidas?

italolimad@alu.ufc.br

[Repositório do Trabalho no Github](#)

