

Tópicos - Artigo Qualidade de Software

≡ Property

Tópicos de Discussão

1489 projetos de código aberto foram avaliados.

Cheiros de teste é um nome popular para más decisões de projeto implementadas no código de teste, sim, há também erros no código de testes, assim como no código de produção, ambas violações dificultam e exigem um maior esforço na manutenibilidade do sistema.

Cheiros de Teste, avaliação dos limites de gravidade para os cheiros de teste, investigação com desenvolvedores sobre a rigidez dos parâmetros atuais estabelecidos para as regras de detecção dos cheiros de teste (inclusive por ferramentas automatizadas).

Não há uma preocupação comparável entre os test smells e os code smells, por parte das equipes que desenvolvem software, geralmente, a qualidade dos códigos de teste não é priorizada e dificilmente erros nessa etapa são removidos através do processo de refatoração. Foi observado que os desenvolvedores nem sempre observam os cheiros de teste como algo problemático e isso pode se dar pela ausência de limites de gravidades bem estabelecidos, visto que há uma necessidade de maior investigação e pesquisa nessa área, assim como foram determinados os limites de severidade para o código em produção, se faz necessário priorizar esse tipo de pesquisa também os códigos de teste, este o foco do artigo.

Objetivo Geral:

Investigar limites de gravidade para cheiros de teste.

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 cheiros de maior gravidade. (Priorização).

Melhorar a acessibilidade da detecção automática de cheiros de teste, integrando-se às ferramentas de desenvolvedor existentes. (Automatização).

Alves et al definiu um método que calibra limites para métricas de qualidade de código (produção), baseada em benchmark, a metodologia usada nesse trabalho foi semelhante e consiste em: Coletar dados de sistemas de software (código de teste de unidade) de 1489 projetos java, do ecossistema Apache e Eclipse, aplicar esses dados na ferramenta de detecção tsDetect (ferramenta própria para a detecção de cheiros de teste), utilizar os sistemas como referência para derivação de limites de gravidade, estabelecidos como "médio", "alto", ou "muito alto".

Descobertas: quatro dos nove cheiros de teste que consideramos deveriam ter limiares mais altos do que o que foi relatado anteriormente na literatura. Passando para o segundo objetivo, os autores integraram a detecção de cheiros de teste fornecida por tsDetect em uma extensão de protótipo (back-end) do BetterCodeHub 1 (BCH), no qual envolveram usuários para interação com o protótipo, um total de 31 desenvolvedores, em 47 projetos diversos, responderam e forneceram 301 pontos de dados de test smells.

De acordo com os desenvolvedores:

Empty Test, Sleepy Test e Mystery Guest têm a maior prioridade como candidatos à refatoração.

Empty Test, Ignored Test e Conditional Test Logic são considerados os odores com maior impacto na manutenção do código.



As avaliações enviadas pelos usuários estão alinhadas com os limites estabelecidos pelos autores, com uma diferença estatisticamente significativa e um forte coeficiente de Spearman, sugerindo que os limites recém-definidos podem ser usados para priorizar instâncias de cheiros de teste.

Contribuições do Estudo:

- Limites de gravidade calibrados para detectores de cheiro de teste

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

Ferramentas de Trabalhos Relacionados

Van Rompaey et al. criou uma ferramenta de detecção de cheiro de teste 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 (the equivalent of Conditional Test Logic), Indirect Test, Mystery Guest, Sensitive Equality e Verbose Test

Greiler et al. focado 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. Eles concluíram que ter uma ferramenta para apontar os problemas com o conjunto de testes pode ajudar os desenvolvedores com a refatoração.

Palomba et al. desenvolveu TASTE (Textual AnalySis para detecção de smEll de teste), que pode detectar General Fixtures, Eager Tests, and Lack of Cohesion of Methods using Information Retrieval techniques, 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 [22]. (Definir o que é Recall e Precisão).

Bavota et al. estudou a difusão de cheiros de teste em projetos de código aberto e industriais [6]. Para ajudar na pesquisa, eles criaram uma ferramenta de detecção de cheiro de teste 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, não em um nível de caso de teste individual).

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.

Como os desenvolvedores percebem os test smells

Tufano et al. pediu a 19 desenvolvedores de vários projetos de código aberto que examinassem amostras de código de teste que continham instâncias de cheiros de teste. Em 82% dos casos os devs não conseguiram reconhecer nenhum problema com o código de teste, com isso, o autor enfatiza a importância da utilização de ferramentas para detecção automática, é importante relatar que o código analisado foi desenvolvido ou mantido pelos desenvolvedores correspondentes, antes da entrevista. Eles também descobriram que a maioria dos odores de teste são criados durante o desenvolvimento inicial do teste e não são removidos na refatoração subsequente.

Palomba et al. investigou a percepção do desenvolvedor sobre os cheiros do código, usando os autores originais e desenvolvedores independentes. Eles pediram aos desenvolvedores que identificassem os problemas de design (odores de código) e, se encontrados, dessem a eles uma classificação de gravidade. As classificações de gravidade foram aplicadas a toda a categoria de odores de

código e não a instâncias específicas de odores de código. Este estudo se concentrou em código de produção (o que não é o foco da pesquisa aqui em questão).

Kummer estudou se os desenvolvedores reconhecem cheiros de teste usando uma amostra de 20 desenvolvedores. O autor conclui que cheiros de teste podem ser refatorados pelos desenvolvedores sem que eles saibam que são instâncias específicas de cheiros de teste e sugere que ferramentas automatizadas de detecção de cheiros de teste podem ajudar os desenvolvedores a justificar sua remoção.

Diferencial:

Ambos os estudos diferem dos nossos na execução, pois apresentaram seções de código de teste de desenvolvedores sem contexto adicional e fora dos ambientes de trabalho habituais do desenvolvedor. Em contraste, nosso estudo pede aos desenvolvedores que trabalhem dentro do fluxo de trabalho normal de sua ferramenta de qualidade de código (BCH) e dentro do contexto de seu próprio projeto.

Test Smells e Qualidade de Código

Spadini et al. estudou a relação entre a presença de cheiros de teste e mudança de software e tendência a defeitos. Eles analisaram vários lançamentos de dez produtos de software e seus casos de teste, investigando os seguintes cheiros de teste: Mystery Guest, Resource Optimism, Eager Test, Assertion Roulette, Indirect Testing e Sensitive Equality. Entre cada versão, eles analisaram como o código de produção mudaram e quantas correções foram relatadas pelos sistemas de acompanhamento de problemas de acompanhamento e commits Git. Suas principais descobertas foram que “testes afetados por cheiros de teste estão associados a maior tendência a alterações e defeitos do que testes não afetados por cheiros”.

Diferencial:

Nesta pesquisa, foi investigado uma variedade mais ampla de cheiros de teste e utilizado um modelo de qualidade de código existente para comparação, no entanto, não foi investigado a tendência a alterações e defeitos.

Trabalhos Relacionados

Tiago L. Alves, Christiaan Ypma, and Joost Visser. 2010. Deriving Metric Thresholds from Benchmark Data. In Proceedings of the 2010 IEEE International Conference on Software Maintenance (ICSM '10). IEEE Computer Society, Washington, DC, USA, 1–10. <https://doi.org/10.1109/ICSM.2010.5609747>

PERUMA, Anthony Shehan Ayam. What the smell? an empirical investigation on the distribution and severity of test smells in open source android applications. 2018.

Ferramentas

- tsDetect
- BetterCodeHub

Inovação

Definição e exemplificação dos Test Smells.

Exibição de Ferramentas.

Investigar um trabalho não citado sobre Test Smells.

Mostrar Comparações dos Resultados - Montar uma Planilha/Tabela.

Exibir Lista de Projetos (1489) - 353 baseados no Eclipse e 1136 baseados no Apache.

Metodologia

Grande problema das ferramentas existentes: Fornecem um feedback binário para a presença/ausência de certos test smells, no código. Ou seja, apenas informam se existe ou não aquele determinado tipo de test smell no código investigado.

Surgem então, duas perguntas cruciais para a metodologia proposta:

- Como os cheiros de teste podem receber uma classificação de gravidade?
- Qual é a percepção dos desenvolvedores sobre os cheiros de teste em sua base de código?

Decisão sobre as ferramentas pesquisadas: Foco somente na linguagem (cheiros de teste) em Java.

Ferramenta escolhida: Ts Detect.

TsDetect:

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

Precision e Recall acima de 85 %.

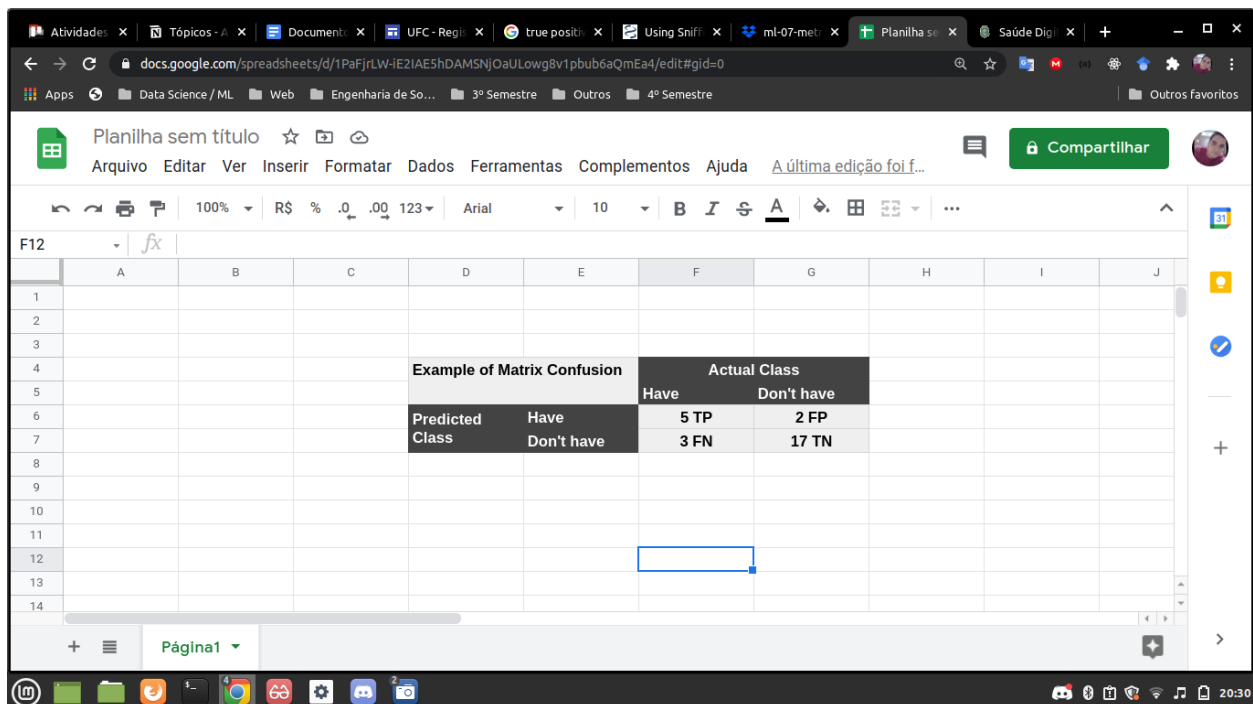
O que é Precision ?

Quando os algoritmos preveem que sim, em quantas ele está realmente correto?
(TP/predicted yes)

O que é Recall ?

Quando é realmente sim, com que frequência o algoritmo prediz que sim?
(TP/Actual yes)

Exemplo de Matriz de Confusão.



		Actual Class	
		Have	Don't have
Predicted Class	Have	5 TP	2 FP
	Don't have	3 FN	17 TN



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.

Link para Exemplo com Métricas de Aprendizado de Máquina, usando Sklearn.

Google Colaboratory

 <https://colab.research.google.com/gist/italolima04/2f07b28d5c6128f48b5ad9745594231d/exemplo-qualidade-de-software.ipynb>



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.

Tipos de Test Smells abordados no Artigo

Type of Test Smells

 Test Smell	 Description	 Problem
<u>Mystery Guest</u>	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.
<u>Resource Optimism</u>	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.
<u>Eager Test</u>	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.
<u>Assertion Roulette</u>	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.
<u>Conditional Test Logic</u>	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.

<u>Aa</u> Test Smell	Description	Problem
<u>General Fixture</u>	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.
<u>Empty Test</u>	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.
<u>Magic Number Test</u>	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.
<u>Sleepy Test</u>	Um teste que contém chamadas de suspensão de encadeamento.	O uso dessa chamada de método introduz um atraso adicional na execução do teste.
<u>Verbose Test</u>	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.
<u>Ignore Test</u>	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, por que exigem a visualização do código fonte de teste completo.

"Com base em testes feitos em um conjunto de dados selecionado, tsDetect é altamente confiável na detecção de instâncias desses odores de teste"

Trabalho Relacionado

O F-Score para o tsDetect no subconjunto selecionado de cheiros de teste está entre 87% e 99%.

O que é F-Score ?

É uma média ponderada entre recall e precision.



$$F_1 = \frac{2}{\frac{1}{\text{recall}} \times \frac{1}{\text{precision}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$
$$= \frac{\text{tp}}{\text{tp} + \frac{1}{2}(\text{fp} + \text{fn})}$$



Estudo Preliminar

Definindo os Limites de Severidade

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

Metrics of Test Smells

 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

 Test Smell	 Metric
<u>Resource Optimism</u>	Arquivos não verificados quanto à existência
<u>Sleepy Test</u>	Threads de chamadas suspensas
<u>Verbose Test</u>	Declarações

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.

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.

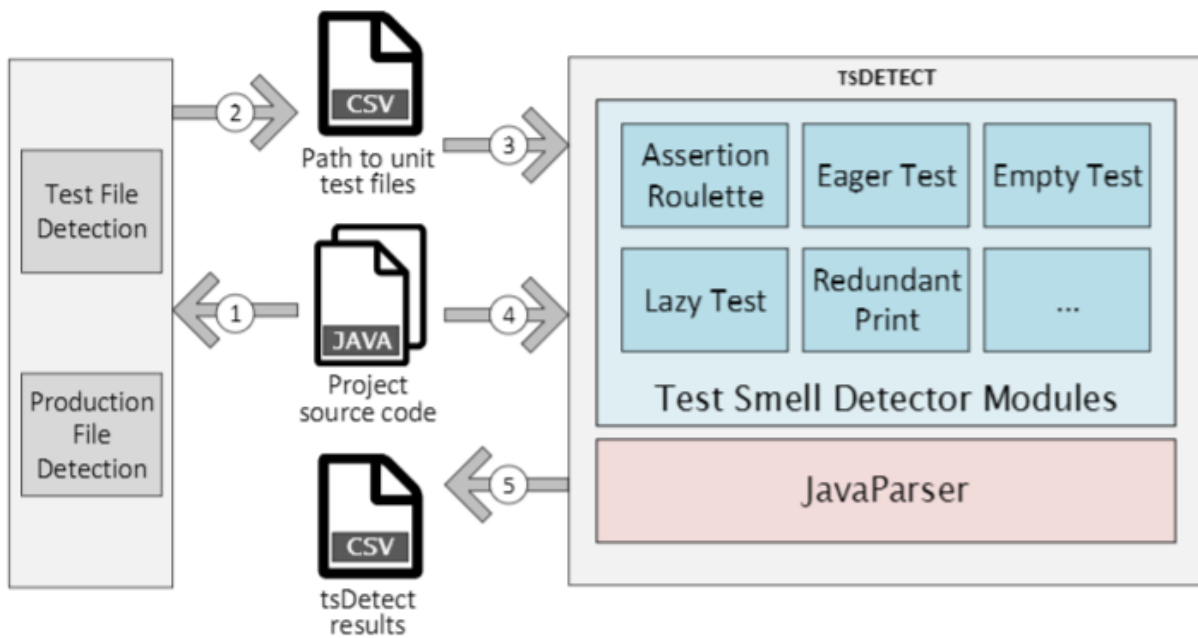
Description of Dataset

 Total de Projetos	 Linhas de Código (Total)	 Linhas de Código (Média)	 Linhas de Código (Mediana)
<u>1489</u>	25.356.827	31.617	6.650

Comparativo com o Projeto Utilizado no Outro Trabalho da Disciplina: 2798.

Isso implica que os projetos selecionados foram geralmente projetos considerados grandes, apesar do fato de que a média pode ter sido afetada por outliers.

Arquitetura da Ferramenta



High-level architecture of TSDETECT