Luís Henrique Soares Monteiro

# Um plugin para a ferramenta ESLint que verifica Test Smells

Belo Horizonte, Minas Gerais
2024

Luís Henrique Soares Monteiro

# Um plugin para a ferramenta ESLint que verifica Test Smells

Relatório Final POC 1

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Orientador: Thiago Ferreira de Noronha
Coorientador: Natã Goulart da Silva

Belo Horizonte, Minas Gerais
2024

# Sumário

# 1 Introdução

No contexto da engenharia de software, as boas práticas de programação abrangem uma variedade de aspectos, incluindo convenções de nomenclatura, organização de código, padrões de projeto e práticas de teste. Ao aderir a essas práticas, os desenvolvedores podem melhorar a qualidade do código, facilitar a colaboração em equipe e reduzir a incidência de bugs e erros.

Detectar e corrigir problemas associados às práticas de teste, ou test smells, representa um desafio significativo na engenharia de software contemporânea. Os test smells são indicadores de problemas nos testes automatizados, como redundâncias, complexidade excessiva, baixa cobertura ou má estruturação. Identificar tais práticas é crucial, pois testes de baixa qualidade podem comprometer a confiabilidade do software e aumentar a carga de manutenção.

O ESLint (2013) é uma ferramenta de código aberto amplamente utilizada na comunidade de desenvolvimento de software JavaScript. Ele funciona como um linter, ou seja, uma ferramenta de análise estática que verifica o código-fonte em busca de problemas, erros ou padrões problemáticos, de acordo com um conjunto de regras configuráveis. Além disso, é altamente configurável e permite que os desenvolvedores personalizem as regras de verificação de acordo com as necessidades específicas de seus projetos. Ele pode ser integrado facilmente em fluxos de trabalho de desenvolvimento, tanto em ambientes locais quanto em sistemas de integração contínua.

De tal forma, é possível desenvolver uma ferramenta integrada ao ESLint (2013) que faça verificação de código fonte buscando especificamente problemas em código de teste, desde que esses problemas sejam passíveis de detecção estática.

## 1.1 Objetivos Gerais

Este projeto busca promover a importância das boas práticas de programação na engenharia de software e apresentar uma solução prática para aprimorar a qualidade dos testes automatizados em JavaScript por meio do desenvolvimento de um plugin para o ESLint (2013). Essa abordagem visa elevar o padrão de qualidade do código produzido, beneficiando tanto os desenvolvedores quanto os usuários finais das aplicações.

Neste contexto, o plugin que será desenvolvido para o ESLint (2013) desempenha um papel crucial, já que irá estender as capacidades da ferramenta para detectar "test smells", ou seja, indicadores de problemas nos testes automatizados.

## 1.2 Objetivos Específicos

Durante a etapa da POC 1, o objetivo é desenvolver os seguintes aspectos:

- Aprendizado das regras de implementação envolvidas no desenvolvimento do plugin.
- Definição de quais serão os casos de testes verificados pela ferramenta.
- Implementação dos casos de teste mapeados.
- Execução de testes para verificar o funcionamento do plugin em uma ou mais bases de testes.
- Documentação e descrição de como utilizar a ferramenta para facilitar o uso e garantir seu licenciamento adequado, tornando-a acessível como um projeto de código aberto. Essa etapa inclui sua publicação em um gerenciador de pacotes.

# 2 Referencial Teórico

Alguns estudos foram utilizados como base para o entendimento e a criação dessa proposta, os principais são (DAMASCENO..., 2018), que aborda a relação entre a qualidade do código e testes e (NGUYEN..., 2012), que aprofunda a pesquisa sobre detecção de práticas ruins de programação em aplicações web dinâmicas. Os principais conceitos empregados neste trabalho são:

- Test Smells: [Test... (2018)] são descritos como sendo vários padrões de refatoração comuns em testes de software. Essa referência explora os diferentes tipos de test smells, sendo estes os que serão abordados pela nova ferramenta:
  - *Assertion Roulette*: Ocorre quando um método de teste possui múltiplas asserções não documentadas. Múltiplas declarações de asserção em um método de teste sem uma mensagem descritiva impactam a legibilidade / compreensibilidade / manutenção, pois não é possível entender o motivo da falha do teste.
  - *Conditional Test Logic*: uso de condicionais (if-else e instruções de repetição) que constroem cenários onde blocos de código podem não ser executados.
  - *Exception Handling* descrito quando o caso de teste possui instruções de try, catch e throw
  - *Ignored Testo* caso de teste não é executado devido ao uso de uma propriedade ".skip" ou similiar.
  - *Redundant Print* O método de teste contém uma instrução que imprime o valor de uma variável no console. Esta é uma declaração redundante que pode ter sido adicionada por um desenvolvedor, para fins de depuração, no momento da escrita do método de teste
  - *Sensitive Equality* O uso do valor padrão retornado por um método toString() de objetos, para realizar comparações de strings, corre o risco de falha no futuro devido a mudanças na implementação de objetos do método toString().
- O VSCode... (2021) é uma extensão para o Visual Studio Code que verifica a ortografia do texto em arquivos de código-fonte e de texto. Ele destaca palavras que estão fora do dicionário do idioma configurado, permitindo que os desenvolvedores identifiquem e corrijam erros de digitação e ortografia enquanto escrevem código ou texto. Essa extensão é útil para manter a qualidade e legibilidade do código, além de melhorar a precisão e profissionalismo da comunicação escrita..

- JSNose (2013) é uma ferramenta de análise estática desenvolvida para JavaScript, focada na detecção de más práticas de codificação e problemas de segurança no código-fonte. Ele funciona examinando o código JavaScript em busca de padrões problemáticos, vulnerabilidades conhecidas e potenciais erros de programação. O JSNose ajuda os desenvolvedores a identificar áreas de melhoria no código JavaScript, como variáveis não utilizadas, funções inseguras, uso inadequado de APIs e práticas que podem levar a vulnerabilidades de segurança. Ao destacar esses problemas, o JSNose auxilia os desenvolvedores a escrever código mais seguro e robusto.

Tais tecnologias desempenham papéis cruciais na monitorização e análise de código estático, e serão utilizadas como base em conjunto com os casos de teste referenciados para implementar a ferramenta.

# 3 Metodologia

Como a proposta é desenvolver um plugin para uma ferramenta já existente neste projeto, a idéia é evoluir e aprimorar tanto a qualidade das possibilidades de avaliação do ESLint (2013) como a quantidade, já que os casos de teste abordados acima passarão a ser cobertos pela nova checagem.

Para a execução do projeto, os seguintes passos foram seguidos:

- Estudos sobre o desenvolvimento de linters: Identificar test smells que já são verificados por outras ferramentas, inclusive outras linguagens, de modo a entender quais estratégias são utilizadas para melhor adaptar a imlpementação do plugin.
- Implementação dos casos de teste escolhidos: Identificar e implementar os requisitos específicos da aplicação usando como base as ferramentas similares previamente analisadas.
- Teste individual de cada test smell: Nesta etapa, serão executados uma série de testes individuais em cada um dos casos de teste escolhidoa, de maneira à garantir que as validações fazem o que é esperado.
- Testes em larga escala: Nesta etapa, serão executados testes utilizando o plugin em grandes projetos de código aberto que possuam uma base grande e variada de testes automatizados, de maneira que seja possível observar e avaliar o uso da ferramenta em projetos reais da comunidade.
- Documentação e publicação: Descrever de maneira detalhada como utilizar a ferramenta, bem como disponibilizar em um repositório de código aberto toda a implementação realizada. Por fim será feita a publicação da ferramenta em um gerenciador de pacotes para que a comunidade possa utilizar.

# 4 Atividades Conduzidas

As atividades conduzidas para a realização do projeto tiveram o cronograma alterado em questão da greve realizada em todo o país. Abaixo seguem as atividades realizadas:

- Estudos e implementação do linter: A primeira etapa do projeto envolveu um estudo aprofundado sobre o ESLint e as regras a serem implementadas. Durante este período, o tempo foi dedicado a entender como o ESLint funciona, sua arquitetura de plugins e a Árvore... (), que é uma representação estruturada do código-fonte em uma forma de árvore, onde cada nó da árvore corresponde a uma construção sintática do código. O ESLint (2013) usa a AST para analisar o código de maneira eficiente, permitindo a aplicação de regras de linting ao percorrer e interpretar essa estrutura. Essa análise baseada em Árvore... () facilita a detecção de padrões de código, erros e "smells"de programação de forma precisa e flexível e as melhores práticas para criar regras personalizadas. Após compreender o funcionamento do ESLint, foi iniciada implementação das regras escolhidas. As regras foram definidas com base em "test smells"comuns que podem comprometer a qualidade dos testes. Elas são:
  - assertion-roulette: Detecta múltiplas asserções em um único teste.
  - conditional-test-logic: Detecta lógica condicional em testes.
  - ignored-test: Detecta testes ignorados usando .skip ou .only
  - exception-handling: Detecta uso inadequado de tratamento de exceções em testes.
  - redundant-print: Detecta uso desnecessário de declarações console.log em testes.
  - sensitive-equality: Detecta comparações sensíveis em strings que podem causar falhas inesperadas em diferentes versões da linguagem Javascript.
- Teste individuais: Para garantir a robustez das regras implementadas, foram programados testes unitários para cada regra. Utilizando o framework de teste Mocha (), forma criados cenários de teste que cobrem tanto casos válidos quanto inválidos para todas as regras. Esses testes asseguram que as regras funcionem conforme o esperado e ajudem a manter a qualidade do código
- Testes em larga escala: O plugin foi testado em projetos open-source já existentes. Este passo foi crucial para avaliar o desempenho e a eficácia do plugin em diferentes bases de código. A aplicação do plugin em projetos reais permitiu identificar possíveis melhorias e ajustes necessários para aumentar a precisão das regras. Abaixo seguem

as etapas e scripts que foram utilizados para acelerar a coleta e execução de testes em larga escala. O código fonte dos scripts pode ser encontrado na Seção 7:

- Seleção de repositórios com a API do Github: Utilizando a API do github, um script 7.1 que permite encontrar repositórios foi desenvolvido. Ele busca por repositórios que possuam ao menos 500 estrelas, 100 forks e utilizem a linguagem "Javascript". Após encontrar essa lista de repositórios, o script procura, nos repositórios retornados, por arquivos de configurações que identifiquem frameworks de teste. Os frameworks abrangidos pela ferramenta até o momento são: Jest, Mocha e Karma. Além dos arquivos de teste, o script também filtra por repositórios que possuam variações da palavra 'test'. Foram selecionados os 30 primeiros repositórios, dadas limitações de armazenamento, já que muitos são projetos grandes e mantidos por toda a comunidade de desenvolvimento, e tempo de processamento até que eles estivessem configurados para rodar o plugin. Esse respositórios serviram de base para validar o funcionamento da ferramenta. Abaixo a lista:
  1. **Nome**: next.js, **Estrelas**: 121887, **Forks**: 26069, **Referência**: (NEXT..., 2024)
  2. **Nome**: create-react-app, **Estrelas**: 102050, **Forks**: 26635, **Referência**: (CREATE-REACT-APP, 2024)
  3. **Nome**: three.js, **Estrelas**: 99680, **Forks**: 35156, **Referência**: (THREE..., 2024)
  4. **Nome**: iptv, **Estrelas**: 79286, **Forks**: 1885, **Referência**: (IPTV, 2024)
  5. **Nome**: mermaid, **Estrelas**: 68025, **Forks**: 5946, **Referência**: (MERMAID, 2024)
  6. **Nome**: reveal.js, **Estrelas**: 67113, **Forks**: 16684, **Referência**: (REVEAL..., 2024)
  7. **Nome**: github-readme-stats, **Estrelas**: 65735, **Forks**: 21202, **Referência**: (GITHUB-README-STATS, 2024)
  8. **Nome**: webpack, **Estrelas**: 64262, **Forks**: 8730, **Referência**: (WEBPACK, 2024)
  9. **Nome**: express, **Estrelas**: 64091, **Forks**: 14196, **Referência**: (EXPRESS, 2024)
  10. **Nome**: Chart.js, **Estrelas**: 63736, **Forks**: 11873, **Referência**: (CHART..., 2024)
  11. **Nome**: markdown-here, **Estrelas**: 59516, **Forks**: 11267, **Referência**: (MARKDOWN-HERE, 2024)
  12. **Nome**: jquery, **Estrelas**: 58972, **Forks**: 20622, **Referência**: (JQUERY, 2024)

13. **Nome**: angular.js, **Estrelas**: 58907, **Forks**: 27550, **Referência**: (ANGULAR. . . , 2024)
14. **Nome**: gatsby, **Estrelas**: 55160, **Forks**: 10333, **Referência**: (GATSBY, 2024)
15. **Nome**: uptime-kuma, **Estrelas**: 53266, **Forks**: 4805, **Referência**: (UPTIME-KUMA, 2024)
16. **Nome**: prettier, **Estrelas**: 48809, **Forks**: 4234, **Referência**: (PRETTIER, 2024)
17. **Nome**: cypress, **Estrelas**: 46500, **Forks**: 3145, **Referência**: (CYPRESS, 2024)
18. **Nome**: dayjs, **Estrelas**: 46368, **Forks**: 2269, **Referência**: (DAYJS, 2024)
19. **Nome**: serverless, **Estrelas**: 46323, **Forks**: 5693, **Referência**: (SERVERLESS, 2024)
20. **Nome**: marktext, **Estrelas**: 45798, **Forks**: 3408, **Referência**: (MARKTEXT, 2024)
21. **Nome**: meteor, **Estrelas**: 44174, **Forks**: 5160, **Referência**: (METEOR, 2024)
22. **Nome**: zx, **Estrelas**: 42517, **Forks**: 1080, **Referência**: (ZX, 2024)
23. **Nome**: Leaflet, **Estrelas**: 40671, **Forks**: 5783, **Referência**: (LEAFLET, 2024)
24. **Nome**: materialize, **Estrelas**: 38862, **Forks**: 4746, **Referência**: (MATERIALIZE, 2024)
25. **Nome**: video.js, **Estrelas**: 37613, **Forks**: 7406, **Referência**: (VIDEO. . . , 2024)
26. **Nome**: htmx, **Estrelas**: 35725, **Forks**: 1205, **Referência**: (HTMX, 2024)
27. **Nome**: fullPage.js, **Estrelas**: 35079, **Forks**: 7294, **Referência**: (FULLPAGE. . . , 2024)
28. **Nome**: koa, **Estrelas**: 35036, **Forks**: 3222, **Referência**: (KOA, 2024)
29. **Nome**: JavaScript, **Estrelas**: 31867, **Forks**: 5464, **Referência**: (JAVASCRIPT. . . , 2024)
30. **Nome**: fastify, **Estrelas**: 31468, **Forks**: 2223, **Referência**: (FASTIFY, 2024)

– Download dos repositórios para ambiente local: Foi criado um script para clonar os repositórios selecionados para uma pasta local.

– Instalação do plugin nos diretórios de cada repositório: Também foi executado um script 7.2 para instalar o plugin, bem como adicionar o arquivo de configuração do ESLint em todos os repositórios.

  - Execução do plugin em larga escala: Shell script 7.3 para iterar pelos diretórios filhos diretos da pasta de repositórios, rodar o comando que executa o plugin e salva os resultados obtidos em um arquivo 'result.json'.
  - Análise dos Resultados: Script 7.4 em python que processa os arquivos 'result.json' em cada um dos repositórios, extrai a contagem de mensagens de erro exibidas pelo plugin e gera um relatório com a contagem de cada mensagem de erro exibida por regra avaliada, bem como um relatório específico para cada repositório.

- Documentação e publicação: A documentação do projeto foi descrita e publicada no gerenciador de pacotes (NPM, 2010), acessível através da URL (NPM-ESLINT-TEST-SMELLS, 2024). Esta documentação inclui informações sobre a instalação, configuração e uso da ferramenta. Também é possível acessar o código-fonte e documentação através do github: (ESLINT-TEST-SMELLS, 2024):

# 5 Resultados Obtidos

Dos 30 repositórios testados, 9 apresentaram ocorrências de possíveis "test smells". Isso indica que os repositórios analisados têm algum grau de problemas relacionados à práticas inadequadas em testes, conforme identificado pelos tipos de erros definidos no plugin. Os resultados dos testes foram analisados e os tipos de erros encontrados foram agrupados e quantificados. Abaixo está o gráfico que representa a contagem de cada tipo de erro encontrado nos repositórios.

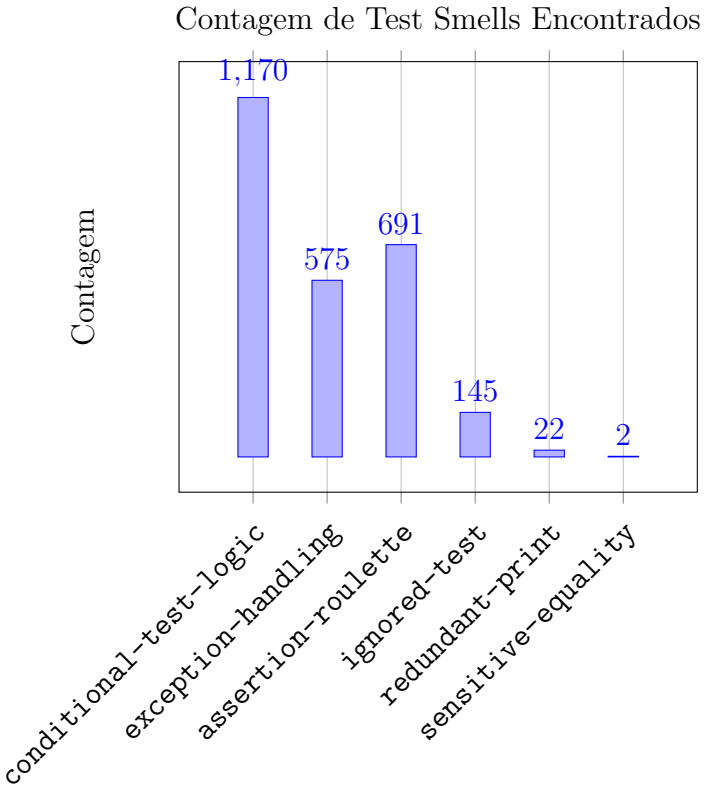


Figura 1 – Contagem Total de Cada Tipo de Test Smell Encontrado

A tabela abaixo apresenta os repositórios onde foram encontrados erros, categorizados pelo tipo de test smell e quantidade total. O índice de problemas é calculado pela soma dos test smells encontrados para cada repositório, ajudando a identificar quais podem ter problemas mais alarmantes nos testes.

| **Repositório** | **Conditional Test Logic** | **Exception Handling** | **Assertion Roulette** |
|---|---|---|---|
| serverless | 43 | 6 | 0 |
| dayjs | 92 | 3 | 332 |
| zx | 1 | 24 | 6 |
| gatsbyjs | 23 | 2 | 8 |
| videojs | 540 | 11 | 0 |
| create-react-app | 27 | 3 | 2 |
| fastify | 564 | 243 | 8 |
| github-readme-stats | 1 | 1 | 38 |
| JavaScript | 6 | 3 | 109 |

Tabela 1 – Contagem de Test Smells por Repositório para Conditional Test Logic, Exception Handling e Assertion Roulette

| **Repositório** | **Ignored Test** | **Redundant Print** | **Sensitive Equality** |
|---|---|---|---|
| serverless | 140 | 0 | 0 |
| dayjs | 0 | 0 | 0 |
| zx | 0 | 5 | 9 |
| gatsbyjs | 0 | 3 | 0 |
| videojs | 0 | 0 | 0 |
| create-react-app | 0 | 0 | 0 |
| fastify | 0 | 8 | 1 |
| github-readme-stats | 0 | 2 | 0 |
| JavaScript | 0 | 0 | 0 |

Tabela 2 – Contagem de Test Smells por Repositório para Ignored Test, Redundant Print e Sensitive Equality

A análise dos resultados sugere que mesmo repositórios open-source grandes e populares podem ser suscetíveis a problemas de qualidade no código de teste. Isso destaca a importância de práticas robustas de revisão e manutenção de testes, bem como a necessidade de ferramentas e abordagens para identificar e corrigir problemas de qualidade nos testes, independentemente do tamanho ou popularidade do projeto.

# 6 Conclusão

Nesta primeira etapa do projeto, o desenvolvimento foi concluído com sucesso e o sistema foi documentado. A documentação detalha as funcionalidades e instruções de uso, assegurando que os usuários possam compreender e utilizar o sistema de maneira eficaz. Além disso, o projeto foi publicado no (NPM, 2010), tornando-o acessível para desenvolvedores e integradores que desejam utilizar ou contribuir com a solução.

Para fomentar a colaboração e o aprimoramento contínuo, o projeto foi disponibilizado como open-source no (GITHUB, 2008), através da URL: (ESLINT-TEST-SMELLS, 2024). Esta abordagem permite que a comunidade de desenvolvedores participe ativamente, forneça feedback e contribua com melhorias. A publicação no (GITHUB, 2008) não só facilita a transparência do desenvolvimento, mas também promove a integração com outras ferramentas e a adoção por uma audiência mais ampla.

Com esses passos, a primeira etapa do projeto está completa, estabelecendo uma base sólida para as próximas fases de desenvolvimento e pesquisa. O acesso aberto ao código e à documentação permitirá que o projeto evolua e se adapte às necessidades da comunidade, garantindo sua relevância e utilidade no longo prazo.

## 6.1 Trabalhos futuros

Para a continuação do trabalho, a proposta é identificar e definir novos tipos de test smells que possam impactar a qualidade dos testes. Exemplos incluem verificações de dependências não explicitadas, padrões de design inadequados em testes, e testes que não seguem boas práticas de isolamento. Também desenvolver um mecanismo para fornecer sugestões de correção aos desenvolvedores quando um test smell é identificado. Isso pode incluir recomendações sobre como refatorar o código do teste ou melhorar a estrutura dos testes. Além disso, é possível integrar as sugestões de correção em ambientes de desenvolvimento e sistemas de integração contínua para fornecer feedback em tempo real aos desenvolvedores, bem como a implementação de correçõesa automáticas quando possível, ou seja, implementar funcionalidades que permitam a correção automática de problemas identificados pelos lint rules. Também é possível garantir que as correções automáticas sejam testadas e validadas para evitar introdução de novos problemas ou regressões no código. A expansão do projeto para incluir novas regras de teste, sugestões de correção e correções automáticas permitirá uma abordagem mais robusta e prática para melhorar a qualidade dos testes automatizados. Esse trabalho contribuirá significativamente para a eficácia dos testes em projetos de software, promovendo melhores práticas e facilitando a manutefnção e evolução dos testes.

# Referências

ANGULAR.JS. 2024. Disponível em: <https://github.com/angular/angular.js>. 10

CHART.JS. 2024. Disponível em: <https://github.com/chartjs/Chart.js>. 9

CREATE-REACT-APP. 2024. Disponível em: <https://github.com/facebook/create-react-app>. 9

CYPRESS. 2024. Disponível em: <https://github.com/cypress-io/cypress>. 10

DAMASCENO, H., Bezerra, C., Campos, D., Machado, I., Coutinho, E. (2023). Test smell refactoring revisited: What can internal quality attributes and developers' experience tell us?. Journal of Software Engineering Research and Development, 11(1), 13:1 – 13:16. https://doi.org/10.5753/jserd.2023.3195. 2018. 5

DAYJS. 2024. Disponível em: <https://github.com/iamkun/dayjs>. 10

ESLINT. 2013. Https://eslint.org/. 3, 7, 8

ESLINT-TEST-SMELLS. 2024. Disponível em: <https://github.com/luissmonteiro/eslint-plugin-test-smells>. 11, 14

EXPRESS. 2024. Disponível em: <https://github.com/expressjs/express>. 9

FASTIFY. 2024. Disponível em: <https://github.com/fastify/fastify>. 10

FULLPAGE.JS. 2024. Disponível em: <https://github.com/alvarotrigo/fullPage.js>. 10

GATSBY. 2024. Disponível em: <https://github.com/gatsbyjs/gatsby>. 10

GITHUB. 2008. Disponível em: <https://github.com/about>. 14

GITHUB-README-STATS. 2024. Disponível em: <https://github.com/anuraghazra/github-readme-stats>. 9

HTMX. 2024. Disponível em: <https://github.com/bigskysoftware/htmx>. 10

IPTV. 2024. Disponível em: <https://github.com/iptv-org/iptv>. 9

JAVASCRIPT Algorithms. 2024. Disponível em: <https://github.com/TheAlgorithms/JavaScript>. 10

JQUERY. 2024. Disponível em: <https://github.com/jquery/jquery>. 9

JSNOSE. 2013. <https://people.ece.ubc.ca/aminmf/SCAM2013.pdf>. Acesso em 25 de março de 2024. 6

KOA. 2024. Disponível em: <https://github.com/koajs/koa>. 10

LEAFLET. 2024. Disponível em: <https://github.com/Leaflet/Leaflet>. 10

MARKDOWN-HERE. 2024. Disponível em: <https://github.com/adam-p/markdown-here>. 9

MARKTEXT. 2024. Disponível em: <https://github.com/marktext/marktext>. 10

MATERIALIZE. 2024. Disponível em: <https://github.com/Dogfalo/materialize>. 10

MERMAID. 2024. Disponível em: <https://github.com/mermaid-js/mermaid>. 9

METEOR. 2024. Disponível em: <https://github.com/meteor/meteor>. 10

MOCHA. Https://github.com/mochajs/mocha. 8

NEXT.JS. 2024. Disponível em: <https://github.com/vercel/next.js>. 9

NGUYEN, Hung Nguyen, Hoan Nguyen, Tung Nguyen, Anh Tien, Nguyen. (2012). Detection of embedded code smells in dynamic web applications. 10.1145/2351676.2351724. 2012. 5

NPM. 2010. Disponível em: <https://docs.npmjs.com/about-npm>. 11, 14

NPM-ESLINT-TEST-SMELLS. 2024. Disponível em: <https://www.npmjs.com/package/eslint-plugin-test-smells>. 11

PRETTIER. 2024. Disponível em: <https://github.com/prettier/prettier>. 10

REVEAL.JS. 2024. Disponível em: <https://github.com/hakimel/reveal.js>. 9

SERVERLESS. 2024. Disponível em: <https://github.com/serverless/serverless>. 10

TEST Smells. 2018. <https://testsmells.org/pages/testsmells.html#ConditionalTestLogic>. Acesso em 27 de de 2024. 5

THREE.JS. 2024. Disponível em: <https://github.com/mrdoob/three.js>. 9

UPTIME-KUMA. 2024. Disponível em: <https://github.com/louislam/uptime-kuma>. 10

VIDEO.JS. 2024. Disponível em: <https://github.com/videojs/video.js>. 10

VSCODE spell checker. 2021. <https://github.com/streetsidesoftware/vscode-spell-checker/>. Acesso em 27 de março de 2024. 5

WEBPACK. 2024. Disponível em: <https://github.com/webpack/webpack>. 9

ZX. 2024. Disponível em: <https://github.com/google/zx>. 10

ÁRVORE de sintaxe abstrata. Https://github.com/estree/estree. 8

# 7 Apêndice

## 7.1 Script que lista seleciona repositórios passíveis de teste no Github

```
import requests
from github import Github

# Fun  o para buscar projetos do GitHub com base em crit rios espec ficos
def buscar_projetos_github(linguagem, estrelas_minimas, forks_minimos, per_page=100):
    url = f"https://api.github.com/search/repositories?q=language:{linguagem}+stars:>={estrelas_minimas}+forks:>={forks_minimos}&sort=stars&order=desc&per_page={per_page}"
    headers = {'Accept': 'application/vnd.github.v3+json'}
    response = requests.get(url, headers=headers)
    if response.status_code == 200:
        return response.json()['items']
    else:
        return []

# Fun  o para verificar se um reposit rio cont m testes unit rios
def verificar_testes_unitarios(repo):
    try:
        # Listar arquivos e diret rios na raiz do reposit rio
        contents = repo.get_contents("")
        test_dirs = ['test', 'tests', '__tests__']
        test_files = ['jest.config.js', 'mocha.opts', 'karma.conf.js']

        for content_file in contents:
            if content_file.type == 'dir' and content_file.name.lower() in test_dirs:
                return True
            if content_file.type == 'file' and content_file.name.lower() in test_files:
                return True

        # Se a verifica  o inicial falhar, verificar recursivamente subdiret rios
        for content_file in contents:
            if content_file.type == 'dir':
                sub_contents = repo.get_contents(content_file.path)
```

```
                for sub_content_file in sub_contents:
                    if sub_content_file.type == 'dir' and
                       sub_content_file.name.lower() in test_dirs:
                        return True
                    if sub_content_file.type == 'file' and
                       sub_content_file.name.lower() in test_files:
                        return True
    except Exception as e:
        print(f"Erro ao verificar o reposit rio {repo.full_name}: {e}")

    return False

# Configura  o inicial
linguagem = "JavaScript"
estrelas_minimas = 500
forks_minimos = 100
token = 'Token removido por quest es de seguran a'

g = Github(token)

# Buscar projetos
projetos = buscar_projetos_github(linguagem, estrelas_minimas,
   forks_minimos)

# Filtrar projetos que possuem testes unit rios
projetos_com_testes = []
for projeto_info in projetos:
    repo = g.get_repo(projeto_info['full_name'])
    if verificar_testes_unitarios(repo):
        projetos_com_testes.append(projeto_info)

# Exibir os projetos selecionados
for projeto in projetos_com_testes:
    print(f"Nome: {projeto['name']}, URL: {projeto['html_url']}, Stars:
       {projeto['stargazers_count']}, Forks: {projeto['forks_count']}")
```

## 7.2 Instala ESLint test-smells em todos os diretórios

```
#!/bin/bash
ARQUIVO_CONFIG="home/luis/Documents/eslint.config.mjs"

PASTA_REPOSITORIOS="/home/luis/Documents/test_repos"

# Itera sobre cada diret rio filho direto em PASTA_REPOSITORIOS
for DIR in "$PASTA_REPOSITORIOS"/*; do
    if [ -d "$DIR" ]; then
```

```
        echo "Instalando␣ESLint␣no␣reposit rio:␣$DIR"
        cd "$DIR"
        cp $ARQUIVO_CONFIG .
        npm install ~/Documents/eslint-test-smells
    fi
done

echo "Processo␣conclu do."
```

## 7.3 Executa plugin em cada diretório de testes

```
#!/bin/bash

# Caminho para a pasta contendo os reposit rios
PASTA_REPOSITORIOS="/home/luis/Documents/test_repos"

# Caminho para o arquivo de sa da
ARQUIVO_SAIDA="/home/luis/Documents/test_repos/eslint_output.txt"

# Limpar o conte do do arquivo de sa da se ele j  existir
> $ARQUIVO_SAIDA

# Iterar sobre cada diret rio filho direto em PASTA_REPOSITORIOS
for DIR in "$PASTA_REPOSITORIOS"/*; do
    if [ -d "$DIR" ]; then
        echo "Executando␣ESLint␣no␣reposit rio:␣$DIR" >> $ARQUIVO_SAIDA
        cd "$DIR"
        # Executar o comando ESLint e salvar o output no arquivo de
           sa da
        npx eslint . --format json --output-file result.json
    fi
done

echo "Processo␣conclu do.␣Os␣resultados␣foram␣salvos␣em␣$ARQUIVO_SAIDA"
```

## 7.4 Processa resultados obtidos após execução dos testes em larga escala

```
import os
import json
from collections import Counter, defaultdict

pasta_repositorios = '/home/luis/Documents/test_repos'
```

```
arquivo_saida_global = '/home/luis/Documents/scripts-eslint/novos-resultados.txt'
arquivo_saida_por_diretorio = '/home/luis/Documents/scripts-eslint/novos-resultados-por-dir.txt'

# Dicion rios para armazenar contagens de ruleIds
contador_ruleIds_global = Counter()

contador_ruleIds_por_diretorio = defaultdict(Counter)

# Obter a lista de diret rios diretamente dentro da pasta_repositorios
repositorios = [d for d in os.listdir(pasta_repositorios) if os.path.isdir(os.path.join(pasta_repositorios, d))]

# Percorrer cada reposit rio
for repo in repositorios:
    caminho_repo = os.path.join(pasta_repositorios, repo)
    caminho_result_json = os.path.join(caminho_repo, 'result.json')
    if os.path.exists(caminho_result_json):
        try:
            with open(caminho_result_json, 'r') as f:
                dados = json.load(f)
                for objeto in dados:
                    for message in objeto.get('messages', []):
                        ruleId = message.get('ruleId')
                        if ruleId:
                            contador_ruleIds_global[ruleId] += 1
                            contador_ruleIds_por_diretorio[repo][ruleId] += 1
        except json.JSONDecodeError:
            print(f"Erro ao ler o arquivo JSON em {caminho_result_json}")
        except Exception as e:
            print(f"Erro inesperado no reposit rio {caminho_repo}: {e}")

# Escrever os resultados globais nos arquivos de sa da
with open(arquivo_saida_global, 'w') as f_global:
    f_global.write("Contagens globais de ruleId:\n")
    for ruleId, contagem in contador_ruleIds_global.items():
        f_global.write(f"{ruleId}: {contagem}\n")

with open(arquivo_saida_por_diretorio, 'w') as f_diretorio:
    f_diretorio.write("Contagens de ruleId por diret rio:\n")
    for repo, contador in contador_ruleIds_por_diretorio.items():
        f_diretorio.write(f"\nReposit rio: {repo}\n")
        for ruleId, contagem in contador.items():
```

```
47                 f_diretorio.write(f"␣␣{ruleId}:␣{contagem}\n")
48
49 print(f"Processo␣conclu do.␣Os␣resultados␣globais␣foram␣salvos␣em␣{
       arquivo_saida_global}␣e␣os␣resultados␣por␣diret rio␣foram␣salvos␣em␣
       {arquivo_saida_por_diretorio}")
```