Samuel William Almeida Santos

Análise de código javascript em projetos multifuncionais

Uma abordagem de *bad smells* com SonarQube

Orientador: Eduardo Figueiredo

Belo Horizonte, Minas Gerais

2023

Sumário

1 Introdução

Este estudo tem como objetivo analisar *Bad Smells* em JavaScript, investigando suas características na prática por meio da avaliação de projetos reais com propósitos diversos. Empregaremos uma ferramenta especializada na análise da qualidade do código para identificar indicadores reveladores de problemas no design e na qualidade do código, buscando extrair relações específicas com as características individuais de cada projeto. Uma pesquisa prévia incluiu uma análise detalhada dos principais *Bad Smells*, métricas quantitativas e técnicas de visualização de software, proporcionando uma base sólida para a compreensão do tema e sua aplicação no presente trabalho.

A definição de *Bad Smell* refere-se a sinais que indicam problemas no design e na qualidade do código, podendo prejudicar a manutenção e evolução do sistema. Ao examinar esses sinais, é possível identificar áreas críticas no código e implementar medidas corretivas para aprimorar a qualidade do software.

A linguagem JavaScript é amplamente utilizada no desenvolvimento de aplicações web e móveis devido à sua flexibilidade e capacidade de interação com o usuário. No entanto, o rápido crescimento e a complexidade dessas aplicações acabam introduzindo *Bad Smells*. Esses problemas podem dificultar a manutenção do código, tornando-o mais propenso a erros e dificultando a evolução do software.

Este trabalho busca contribuir para a área de Engenharia de Software, oferecendo compreensão sobre os padrões de ocorrência de *Bad Smells* em JavaScript em projetos reais bem como avaliar a efetividade da ferramenta utilizada.

2 Referencial Teórico

2.1 Bad smell

No âmbito do desenvolvimento de software, a qualidade e a manutenção de um sistema são aspectos cruciais para garantir seu funcionamento eficiente e sustentável ao longo do tempo. Segundo as considerações de Martin Fowler e Kent Beck em sua obra *Refactoring* [5], a identificação de Bad Smells (maus odores) revela indicadores de problemas de design de código que podem comprometer negativamente tais atributos. Esses maus odores englobam estruturas como duplicação de código, complexidade excessiva em classes e métodos, baixa coesão, alto acoplamento e dependências desnecessárias. A presença dessas falhas pode dificultar a compreensão e a manutenção

do código, além de aumentar a probabilidade de ocorrência de erros e desafios relacionados à escalabilidade.

Fowler e Beck estabeleceram uma taxonomia dos principais bad smells encontrados no código, fornecendo uma referência valiosa para a identificação dessas deficiências, alguns deles são:

Tabela 1: Definição de bad smells

Bad smell	**Definição**
Long Method	Um método ou função excessivamente longo
Large Class	Um método ou função excessivamente longo
Primitive Obssesion	Uso excessivo de tipos primitivos em vez de criar classes específicas do domínio
Large Class	Uma classe que se tornou muito grande e contém muitas responsabilidades violando o Princípio da Responsabilidade Única.
Long Parameter List	Um método ou função que recebe muitos parâmetros, tornando-o difícil de entender e potencialmente propenso a erros.
Duplicated Code	Código duplicado ou muito similar que existe em múltiplos lugares, aumentando o esforço de manutenção e o risco de inconsistências.
Switch Statements	A presença de múltiplos switch ou if-else baseados no tipo ou estado de um objeto, indicando falta de polimorfismo e potencial violação do Princípio Aberto-Fechado.
Shotgun Surgery	Uma alteração em uma parte do código que requer múltiplas alterações espalhadas por outras partes não relacionadas, sugerindo falta de coesão e encapsulamento inadequado.
Data Clumps	Vários itens de dados que frequentemente aparecem juntos como parâmetros ou campos, indicando a necessidade de agrupá-los em uma classe separada.
Feature Envy	Um método ou função que utiliza mais recursos de outra classe do que da sua própria, indicando que ele deveria pertencer à outra classe.
Message Chains	Uma sequência longa de chamadas de método em objetos, tornando o código fortemente acoplado e mais difícil de entender.
Lazy Class	Uma classe que faz muito pouco e poderia ser eliminada ou mesclada com outra classe sem afetar o comportamento do sistema.
Comments	Comentários excessivos ou enganosos que indicam falta de clareza no código é possível divergência do comportamento pretendido.

2.2 Sintaxe JavaScript

A linguagem JavaScript é uma linguagem de programação amplamente utilizada para desenvolvimento web e uma das mais populares segundo recentes pesquisas do Stack Overflow [4]. Ela permite que os desenvolvedores criem scripts que são executados em um navegador, permitindo a interação dinâmica com elementos da página, manipulação de dados, validação de formulários e muito mais.

A sintaxe do JavaScript é semelhante a outras linguagens de programação, como C, C++ e Java, mas possui suas próprias peculiaridades. Algumas características importantes da sintaxe do JavaScript incluem:

- **Variáveis:** Para declarar uma variável em JavaScript, utiliza-se a palavra-chave *var*, *let* ou *const*, seguida pelo nome da variável.
- **Tipos de dados:** O JavaScript possui vários tipos de dados, incluindo números, strings, booleanos, objetos, arrays e null/undefined. Não é necessário declarar explicitamente o tipo de uma variável, pois o JavaScript é uma linguagem de tipagem dinâmica.
- **Operadores:** O JavaScript suporta operadores aritméticos (+, -, *, /), operadores de atribuição (=, +=, -=), operadores de comparação (==, ===, !=, !==, <, >, <=, >=) e operadores lógicos (&&, ||, !).
- **Estruturas de controle:** O JavaScript oferece estruturas de controle, como condicionais e loops. As condicionais mais comuns são *if*, *else if* e *else*, enquanto os loops mais comuns são *for*, *while* e *do-while*.
- **Funções**: As funções no JavaScript são declaradas usando a palavra-chave *function* seguida pelo nome da função e seus parâmetros. Elas podem ser chamadas posteriormente usando o nome da função seguido por parênteses. O JavaScript também suporta funções anônimas e funções de seta (*arrow functions*).

3 Metodologia

Para a seleção dos repositórios públicos no GitHub [3], foi adotada a abordagem de classificação decrescente com base no número de estrelas, indicando a preferência dos usuários pelo repositório. A escolha recaiu sobre projetos predominantemente desenvolvidos em JavaScript. Utilizaram-se palavras-chave como *framework*, *library*, *tool*, *game* e *extension* para abranger diversas categorias de software. A Tabela 2 apresenta a lista dos repositórios selecionados.

Tabela 2: Repositórios selecionados

Repositório	Categoria	Descrição
Significant-Gravitas/AutoGPT	tool	AutoGPT is the vision of accessible AI for everyone, to use and to build on. Our mission is to provide the tools, so that you can focus on what matters.
twbs/bootstrap	framework	The most popular HTML, CSS, and JavaScript framework for developing responsive, mobile first projects on the web.
mozilla/BrowserQuest	game	A HTML5/JavaScript multiplayer game experiment
expressjs/express	framework	Fast, unopinionated, minimalist web framework for node.
HabitRPG/habitica	game	A habit tracker app which treats your goals like a Role Playing Game.
bigskysoftware/htmx	tool	</> htmx - high power tools for HTML
jquery/jquery	library	jQuery JavaScript Library
Leaflet/Leaflet	library	🍃 JavaScript library for mobile-friendly interactive maps 🇺🇦
Dogfalo/materialize	framework	Materialize, a CSS Framework based on Material Design
Modernizr/Modernizr	library	Modernizr is a JavaScript library that detects HTML5 and CSS3 features in the user's browser.
parcel-bundler/parcel	tool	The zero configuration build tool for the web. 📦🚀
VincentGarreau/particles.js	library	A lightweight JavaScript library for creating particles
facebook/react	library	The library for web and native user interfaces.
hakimel/reveal.js	framework	The HTML Presentation Framework
tobspr-games/shapez.io	game	shapez is an open source base building game on Steam inspired by factorio!
spicetify/spicetify-cli	tool	Command-line tool to customize Spotify client. Supports Windows, MacOS, and Linux.
mrdoob/three.js	library	JavaScript 3D Library.
AlexNisnevich/untrusted	game	A meta-JavaScript adventure game by Alex Nisnevich and Greg Shuflin.
olistic/warriorjs	game	🏰 An exciting game of programming and Artificial Intelligence
adam-p/markdown-here	extension	Google Chrome, Firefox, and Thunderbird extension that lets you write email in Markdown and render it before sending.
gildas-lormeau/SingleFile	extension	Web Extension and CLI tool for saving a faithful copy of a complete web page in a single HTML file
MetaMask/metamask-extension	extension	:globe_with_meridians: :electric_plug: The MetaMask browser extension enables browsing Ethereum blockchain enabled websites
uBlock-LLC/uBlock	extension	uBlock: a fast, lightweight, and lean blocker for Chrome, Firefox, and Safari.
dessant/buster	extension	Captcha solver extension for humans, available for Chrome, Edge and Firefox
josdejong/jsoneditor	tool	A web-based tool to view, edit, format, and validate JSON

A versão gratuita do SonarQube possibilitou a criação de um servidor local para exposição completa da interação com a ferramenta. O perfil das regras de identificação de *bad smells* foi modificado, excluindo aquelas associadas apenas a versões mais recentes do JavaScript que introduziram novos recursos. Essa adaptação teve como objetivo permitir uma análise atemporal, independente da data de desenvolvimento ou última atualização do código avaliado, o que acabou não se tornando totalmente verdade e será abordado no capítulo de resultados.

Os repositórios escolhidos foram clonados, e a ferramenta realizou uma análise estática dos arquivos de cada projeto. Posteriormente, as informações resultantes, como

bugs, *bad smells* e vulnerabilidades de segurança, foram apresentadas em uma visualização HTML.

No entanto, para uma análise mais aprofundada, as informações coletadas precisaram ser convertidas em um formato mais padronizado como CSV e JSON, o que só foi possível através da API [2] fornecida pela ferramenta e sua abrangente documentação disponível no site oficial do SonarQube. Isso resultou em uma lista dos problemas de qualidade de software (chamadas de *issues*) que foi filtrada para aqueles referentes à linguagem JavaScript e que eram consideradas como *bad smells* pela própria nomenclatura da ferramenta. Essa abordagem possibilitou comparações entre os resultados da análise e as características individuais de cada projeto no GitHub por meio de gráficos utilizando a biblioteca Chart.js [1], conforme evidenciado no próximo capítulo .

5 Resultados

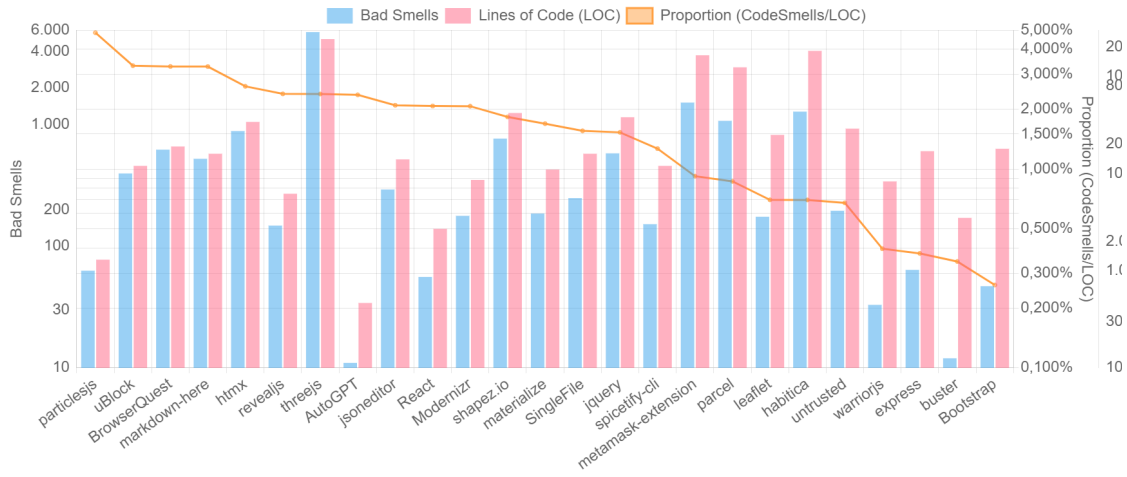
5.1 Análise exploratória

Inicialmente, realizou-se uma análise dos dados coletados para cada um dos projetos, utilizando as métricas identificadas como mais úteis e significativas, conforme detalhado na Tabela 3. Essa abordagem permitiu obter uma visão abrangente da qualidade do código em cada projeto. Notavelmente, ao analisar o Gráfico 1 o projeto *particlejs* destacou-se negativamente ao apresentar a pior proporção de *bad smells* por linha de código (*LOC*), enquanto o *Bootstrap* alcançou os melhores resultados. Vale ressaltar que este gráfico e os demais utilizam uma escala logarítmica para representar os resultados, considerando a ordem de grandeza significativamente menor dos *bad smells* em comparação com o *LOC*.

Tabela 3: Métricas utilizadas

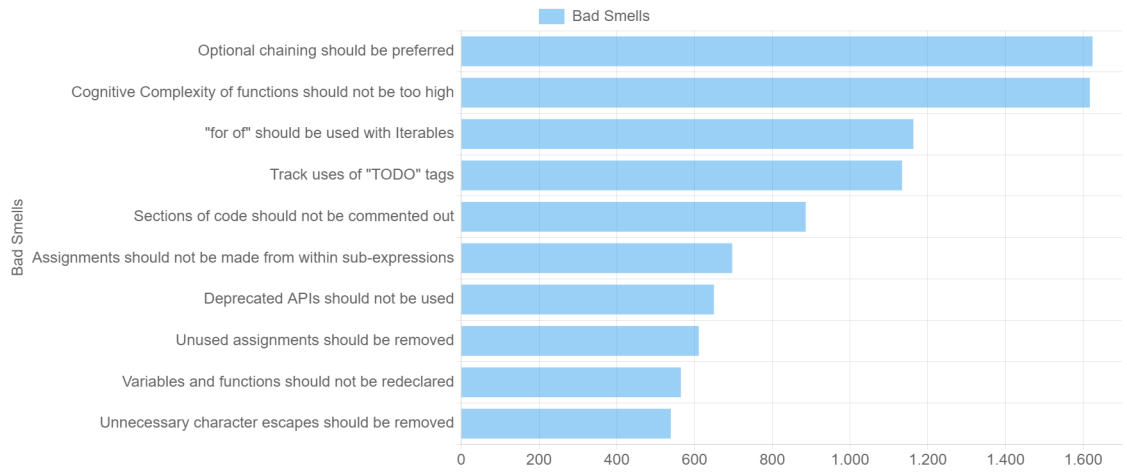
Métrica	**Definição**
Code smell	Número de bad smells
Lines of Code (LOC)	Quantidade de linhas Javascript
Proportion (Code Smells/LOC)	Proporção entre as duas métricas Code Smell e LOC de forma percentual

Gráfico 1: Análise geral dos repositórios



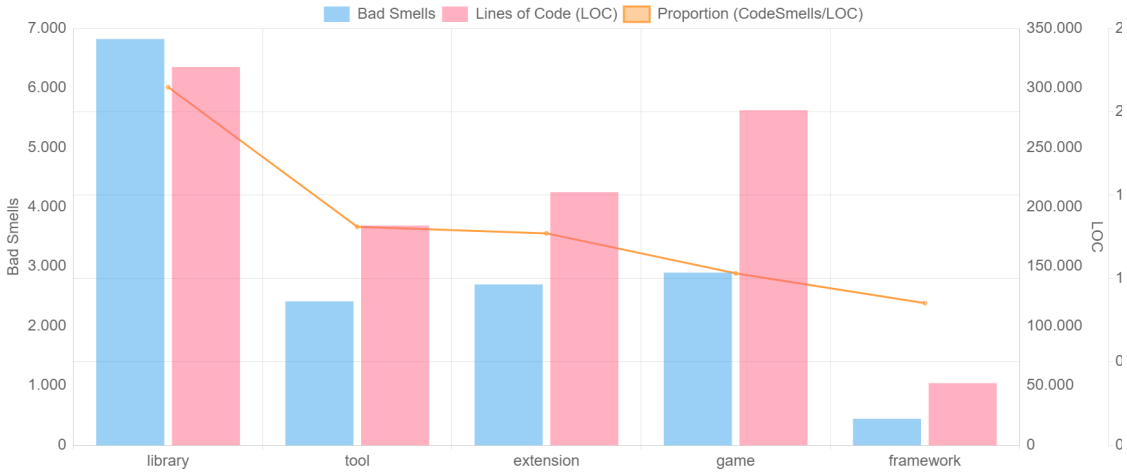
Dentre os *bad smells* identificados, alguns mostraram-se mais comuns do que outros, conforme evidenciado no Gráfico 2. Mesmo após a aplicação de filtros nas regras, ainda foi possível detectar bad smells relacionados a recursos mais recentes do JavaScript, como *"Optional Chaining Should be Preferred"* e *"'for of' should be used with iterables"*. Destaca-se, no entanto, a presença notável de *"Cognitive Complexity of function should not be too high"*, que ocupou a segunda posição e é considerado crítico pela própria ferramenta. Essa classificação sugere que lidar com a complexidade cognitiva das funções representa um dos maiores desafios na criação de código de alta qualidade.

Gráfico 2: Top 10 Bad smells mais encontrados



Em seguida, procedeu-se a uma análise das diversas categorias de software escolhidas, com a expectativa de identificar possíveis relações entre a ocorrência de *bad smells* e a variada natureza e especificidades de cada categoria. A única disparidade notada surgiu ao compararmos as bibliotecas, que exibiram uma proporção inferior de *bad smells* por linha de código (*LOC*) em relação às demais. Isso sugere que esse tipo específico de software pode apresentar desafios adicionais na busca por uma maior qualidade, conforme evidenciado nos dados do Gráfico 3.

Gráfico 3: Análise das categorias de software



5.2 Métricas do GitHub

Com o intuito de estabelecer uma comparação com as métricas dos projetos no GitHub, foram escolhidos como atributos indicativos relevantes as estrelas, *forks*, contribuintes e número de *commits* descritos na Tabela 4.

Tabela 4: Métricas do GitHub utilizadas

Atributo	**Descrição**
Estrelas	Indica a popularidade ou quantidade de usuários que marcaram o repositório como favorito
Forks	Representa o número de cópias independentes do repositório, indicando colaborações ou derivações.
Contribuintes	Refere-se ao número de indivíduos que contribuíram para o projeto, sugerindo diversidade e apoio.
Commit	Representa uma alteração ou conjunto de alterações feitas em um repositório. Cada commit é uma marcação única que registra as mudanças no código-fonte, incluindo adições, modificações ou exclusões de arquivos.

Esses elementos visam avaliar a robustez, tamanho, colaboração e evolução do projeto, permitindo a análise de como essas características podem influenciar a qualidade esperada do software em cada repositório. Nos Gráficos 4 e 5, observamos que não parece haver uma relação clara entre o número de estrelas e a presença de bad smells; apenas projetos com 6000 ou mais *forks* apresentaram um desempenho inferior, embora essa seja uma amostra muito pequena para ser considerada relevante.

Gráfico 4: Comparativo com número de estrelas no GitHub

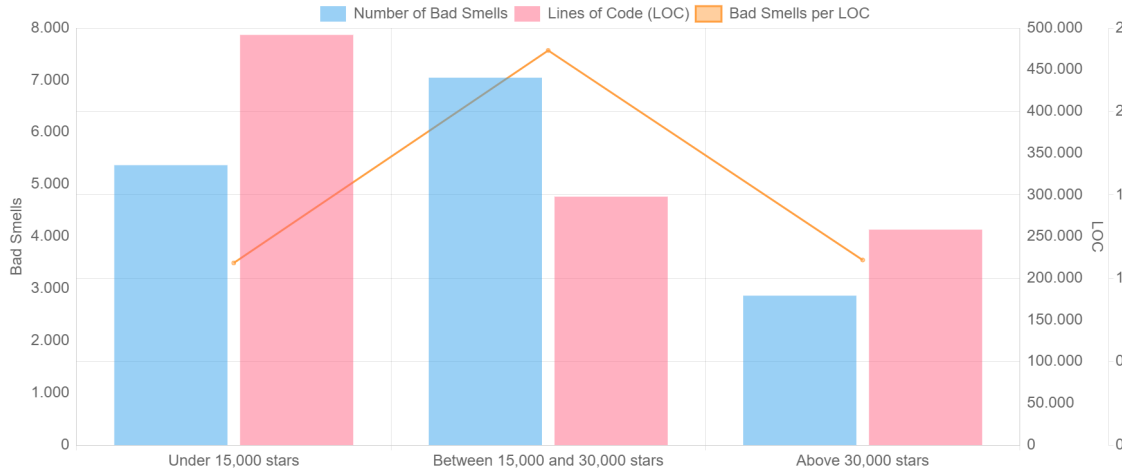


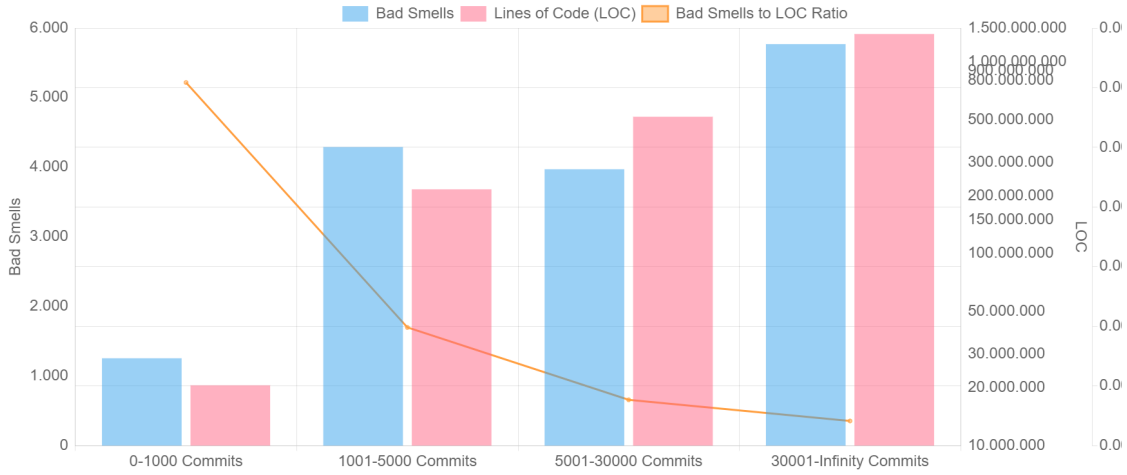
Gráfico 5: Comparativo com número de forks no GitHub

No entanto, ao examinarmos os Gráficos 6 e 7, que comparam o número de *commits* e contribuintes, percebemos uma tendência clara de redução dos *bad smells* por linha de código (*LOC*). Isso indica que equipes maiores e um maior número de *commits*, seja devido a um tempo de projeto mais extenso ou a uma maior ênfase na realização de

commits atômicos, estão associados a uma menor incidência de problemas de qualidade no código.

Gráfico 6: Comparativo com número de contribuintes no GitHub

Gráfico 7: Comparativo com número de commits no Github



6 Conclusão

Em resumo, a investigação dos repositórios GitHub proporcionou *insights* valiosos sobre a presença de *bad smells* em diferentes categorias de software. Destaca-se que os repositórios do tipo biblioteca apresentaram uma incidência significativamente maior desses indicadores de problemas de código, em contraste com as demais categorias, que demonstraram uma presença mais contida e consistente.

Entre os bad smells, a *"Cognitive Complexity of functions should not be too high"* emergiu como um elemento comum e crítico, apontando para a importância de evitar complexidades excessivas que possam comprometer a compreensão do código, especialmente em termos de aninhamento de estruturas condicionais e expressões booleanas.

Surpreendentemente, a relação entre a quantidade de *bad smells* e a popularidade de um repositório, medida por *forks* e estrelas, não se revelou evidente. Contudo, ao observarmos projetos mais amplos, caracterizados por um maior número de colaboradores e *commits*, identificamos uma tendência clara de redução nos *bad smells*. Essa correlação sugere que projetos mais robustos, com um ciclo de desenvolvimento prolongado e um compromisso contínuo com a refatoração, tendem a apresentar uma menor incidência de problemas de qualidade no código. Essa descoberta é crucial para orientar equipes de desenvolvimento na busca por práticas eficazes para aprimorar a qualidade do código em seus repositórios, destacando a relevância da atenção contínua à estrutura e manutenção do código-fonte.

Para futuras pesquisas, é recomendável realizar uma revisão manual das configurações de regras do SonarQube, a fim de validar aquelas que são amplamente aplicáveis. Este estudo em andamento fundamentou-se nas etiquetas fornecidas pela ferramenta para conduzir essa filtragem, uma abordagem menos confiável. Isso se deve ao fato de *bad smells* como *"Optional Chaining Should be Preferred"* e *"for of' should be used with iterables"* serem resultantes de recursos mais recentes do JavaScript e não terem sido filtrados, ressaltando a necessidade de uma análise mais minuciosa para assegurar a relevância das regras selecionadas. Apesar de o SonarQube ser uma das ferramentas mais populares, apresentando bons resultados e sendo de fácil utilização, uma futura análise de forma similar e com outra ferramenta deve proporcionar informações complementares a este estudo.

7 Referências Bibliográficas

[1] https://www.chartjs.org/docs/latest/

[2] https://next.sonarqube.com/sonarqube/web_api/api/alm_integrations

[3] https://github.com/

[4] https://survey.stackoverflow.co/2023/

[5] FOWLER, M. Refactoring : improving the design of existing code. Boston: Addison-Wesley, 2019.

[6] Lanza, M. and Marinescu, R. (2007). Object-Oriented Metrics in Practice. Springer Science & Business Media.

[7] Barros, A.X. and Adachi, E. (2021). Bad Smells in Javascript - A Mapping Study. [online] sol.sbc.org.br. doi:https://doi.org/10.5753/vem.2021.17208.