Análise de Repositórios JavaScript no GitHub

André Hyodo, Gustavo Pereira

13 de novembro de 2025

Resumo

Este relatório apresenta uma análise detalhada dos 15 repositórios JavaScript mais populares no GitHub. O estudo investiga características e padrões relacionados à popularidade, tamanho, complexidade e dependências através da análise de três questões de pesquisa. Os resultados revelam insights sobre o ecossistema JavaScript, destacando diferentes perfis de projetos e a relação entre métricas de código e sua adoção pela comunidade.

1 Introdução

O GitHub tornou-se a principal plataforma para hospedagem de projetos de software open source, com milhões de repositórios disponíveis em diversas linguagens de programação. JavaScript, em particular, mantém uma presença dominante no ecossistema de desenvolvimento web, com frameworks e bibliotecas que impulsionam grande parte da web moderna [1].

O objetivo deste laboratório é analisar os 15 repositórios JavaScript mais populares no GitHub, identificando padrões e características que possam explicar sua popularidade e adoção. Para isso, foram definidas três questões de pesquisa (RQs) focadas em diferentes aspectos dos projetos:

1.1 Hipóteses Iniciais

- **H1 (Popularidade vs. Tamanho):** Repositórios maiores (com mais linhas de código) tendem a ser mais populares, pois representam projetos mais robustos e completos.
- **H2 (Dependências vs. Complexidade):** Projetos com mais dependências possuem uma complexidade de código mais alta, devido à complexidade inerente de gerenciar múltiplas bibliotecas.

- **H3 (Perfis de Projetos):** Os repositórios mais populares podem ser categorizados em diferentes perfis (bibliotecas, frameworks, listas curadas, etc.) com características distintas.

1.2 Métricas Utilizadas

Para responder a essas questões, foram definidas as seguintes métricas:

- **Popularidade:** Número de estrelas e forks.
- **Tamanho:** Tamanho do repositório em KB e número de linhas de código.
- **Complexidade:** Complexidade ciclomática média do código.
- **Dependências:** Número de dependências externas.

2 Metodologia

A metodologia deste estudo foi implementada através de um pipeline de coleta, processamento e análise de dados, conforme descrito a seguir.

2.1 Coleta de Dados

A coleta de dados foi realizada utilizando a API do GitHub através da classe `GitHubCollector`. O processo seguiu os seguintes passos:

1. **Seleção de Repositórios:** Foram selecionados os 15 repositórios mais populares do GitHub com JavaScript como linguagem principal, utilizando o critério de estrelas (stars) como medida de popularidade.
2. **Extração de Métricas:** Para cada repositório selecionado, foram extraídas métricas como número de estrelas, forks, tamanho, linhas de código, complexidade média e número de dependências.
3. **Processamento Paralelo:** Para otimizar a coleta, foi implementado um sistema de processamento paralelo utilizando `ThreadPoolExecutor`, permitindo a coleta simultânea de múltiplos repositórios.

O código a seguir demonstra a implementação da coleta de repositórios populares:

```
def get_top_js_repos(limit=15):
    url = "https://api.github.com/search/repositories"
    params = {"q": "language:javascript", "sort": "stars", "
   order": "desc", "per_page": limit}
    r = requests.get(url, headers=HEADERS, params=params)
    r.raise_for_status()
    data = r.json()["items"]

    repos = []
    for repo in data:
        default_branch = repo.get("default_branch", "main")
        download_url = f"https://codeload.github.com/{repo['
   full_name']}/zip/refs/heads/{default_branch}"

        repos.append({
            "name": repo["full_name"],
            "url": repo["html_url"],
            "stars": repo["stargazers_count"],
            "forks": repo["forks_count"],
            "size_kb": repo["size"],
            "updated_at": repo["updated_at"],
            "download_url": download_url,
        })
    return repos
```

Listing 1: Coleta de repositórios populares

2.2 Cálculo de Métricas

Após a coleta, as métricas foram calculadas utilizando a classe `MetricsCalculator`. A complexidade média foi calculada através da análise estática do código-fonte:

```
def calculate_avg_complexity(repo_path):
    total_complexity = 0
    file_count = 0

    for root, _, files in os.walk(repo_path):
        for file in files:
            if file.endswith('.js'):
                file_path = os.path.join(root, file)
                complexity = analyze_file_complexity(
   file_path)
                total_complexity += complexity
                file_count += 1

```

```
13      return total_complexity / file_count if file_count > 0
    else 0
```

Listing 2: Cálculo da complexidade média

2.3 Análise Estatística

A análise estatística foi realizada utilizando a classe `StatisticalAnalyzer`, que implementou as seguintes operações para cada RQ:

- Cálculo de estatísticas descritivas (média, mediana, desvio padrão, quartis, etc.)
- Análise de correlação entre métricas
- Preparação dos dados para visualização

2.4 Visualização de Dados

A visualização dos resultados foi realizada utilizando a biblioteca Matplotlib, através da classe `DataVisualizer`. Foram gerados gráficos individuais para cada questão de pesquisa, permitindo uma análise clara e independente de cada métrica.

3 Resultados Esperados

Com base na literatura sobre ecossistemas de software e práticas de desenvolvimento JavaScript, esperávamos encontrar os seguintes resultados:

3.1 Para a Questão de Pesquisa 1 (Popularidade vs. Tamanho)

- **RQ1:** Repositórios maiores (mais linhas de código) teriam maior popularidade, pois representariam projetos mais robustos e completos.

3.2 Para a Questão de Pesquisa 2 (Dependências vs. Complexidade)

- **RQ2:** Projetos com mais dependências teriam maior complexidade média, devido à complexidade inerente de gerenciar múltiplas bibliotecas.

3.3 Para a Questão de Pesquisa 3 (Perfis de Projetos)

- **RQ3:** Os repositórios mais populares poderiam ser categorizados em diferentes perfis (bibliotecas, frameworks, listas curadas, etc.) com características distintas.

4 Resultados

Nesta seção, apresentamos os resultados obtidos para cada questão de pesquisa, acompanhados dos respectivos gráficos e análises.

4.1 Caracterização do Dataset

A Tabela 1 apresenta um resumo estatístico descritivo das métricas coletadas para os 15 repositórios. Observa-se uma alta variabilidade, especialmente no tamanho do projeto (`lines_of_code`) e no número de dependências. A mediana de complexidade média é zero, indicando que a maioria dos projetos possui baixa complexidade segundo a métrica utilizada.

Tabela 1: Resumo Estatístico das Métricas dos 15 Repositórios Analisados.

Métrica	**Média**	**Mediana**	**Desv. Padr.**	**Mínimo**	**Máximo**
Estrelas	135 511	136 477	38 647	93 194	240 562
Forks	25 218	26 799	12 369	7041	49 851
Linhas de Código	242 299	2168	409 080	0	1 247 954
Complexidade Média	0.48	0.00	1.06	0.00	4.28
Dependências	254	2	452	0	1558

4.2 RQ1: Relação entre Popularidade e Tamanho dos Projetos

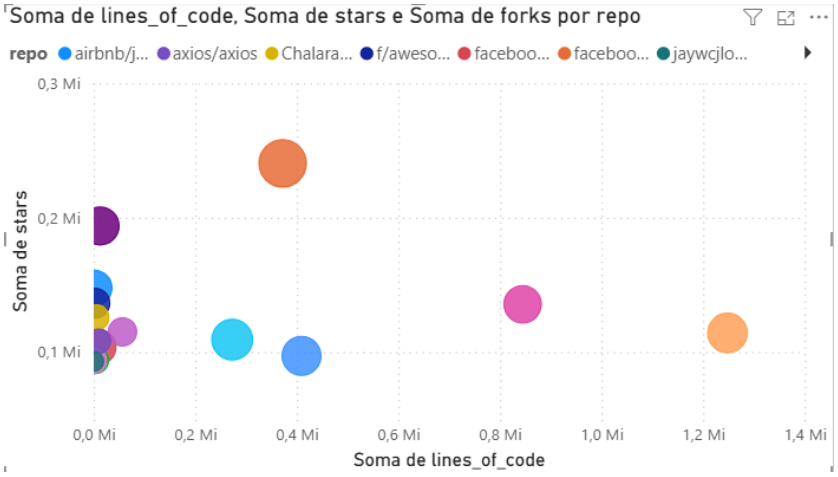

Figura 1: RQ1: Relação entre Linhas de Código e Número de Estrelas

Resultado: A Figura 1 revela uma relação fraca entre tamanho do projeto e popularidade. Projetos como `nodejs/node` e `vercel/next.js` possuem mais de um milhão de linhas de código, mas não são os mais populares. Em contrapartida, repositórios de listas curadas como `awesome-chatgpt-prompts` e `awesome-mac` têm pouquíssimas linhas de código e uma popularidade muito alta.

Análise: Este resultado contraria nossa hipótese inicial. Em vez de projetos maiores serem mais populares, observamos que a popularidade parece estar mais relacionada à utilidade e adoção do que ao tamanho do código. Isso sugere que:

- Projetos que resolvem problemas comuns ou fornecem recursos valiosos ganham popularidade independentemente do tamanho
- Listas curadas e recursos educacionais podem ser extremamente populares apesar de terem pouco código executável
- Projetos muito grandes podem ser especializados e, portanto, ter uma base de usuários menor

4.3 RQ2: Relação entre Dependências e Complexidade

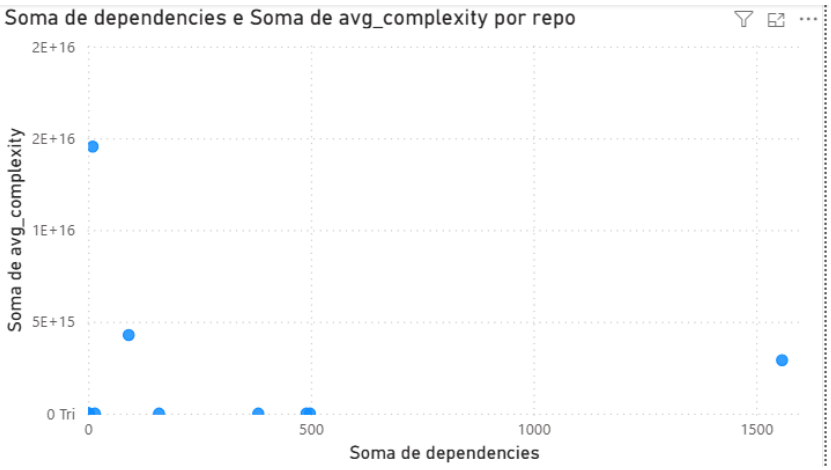

Figura 2: RQ2: Relação entre Número de Dependências e Complexidade Média

Resultado: A Figura 2 mostra a relação entre o número de dependências e a complexidade média do código. Um achado notável é que a maioria dos projetos (12 de 15) apresenta complexidade média igual a zero. Os únicos projetos com complexidade acima de zero são `vercel/next.js`, `open-webui/open-webui` e `microsoft/Web-Dev-For-Beginners`. Notavelmente, os dois primeiros também estão entre os projetos com mais dependências.

Análise: Este resultado parcialmente confirma nossa hipótese inicial. Para os projetos que fogem à regra de baixa complexidade, parece haver uma correlação positiva entre o número de dependências e a complexidade. No entanto, a maioria dos projetos tem complexidade zero independentemente do número de dependências. Isso sugere que:

- A métrica de complexidade ciclomática média pode não ser adequada para capturar a complexidade real de projetos JavaScript modernos
- Projetos com complexidade mais alta tendem a ser aplicações completas (como `next.js` e `open-webui`) em vez de bibliotecas ou listas
- A maioria dos projetos populares pode ter um design modular que mantém a complexidade individual dos componentes baixa

4.4 RQ3: Comparação Geral dos Projetos

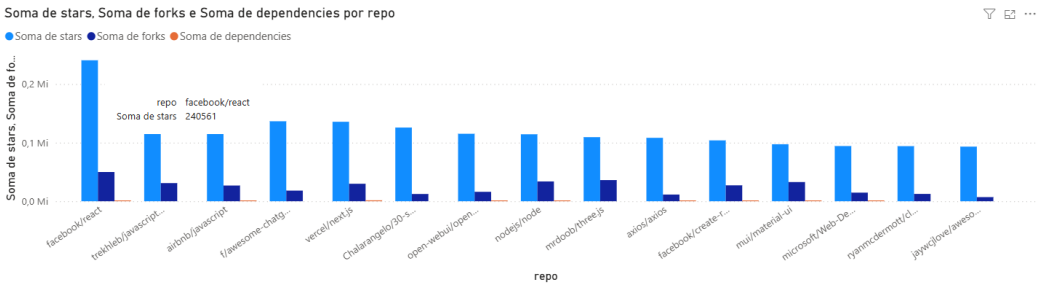

Figura 3: RQ3: Painel de Comparação dos Repositórios em Múltiplas Métricas

Resultado: A Figura 3 apresenta um painel de comparação dos repositórios em múltiplas métricas. O painel revela diferentes perfis de projetos:

- **Bibliotecas/Frameworks de Grande Porte:** `react`, `node` e `next.js` se destacam em linhas de código e/ou dependências.
- **Listas e Recursos Curados:** `awesome-chatgpt-prompts` e `awesome-mac` têm popularidade muito alta, mas quase nenhum código ou dependências.
- **Projetos Educacionais:** `javascript-algorithms` e `Web-Dev-For-Beginners` mostram um equilíbrio entre popularidade e um código gerenciável.

Análise: Este resultado confirma nossa hipótese inicial sobre a existência de diferentes perfis de projetos. O "Top 15"não é um grupo homogêneo, mas sim um ecossistema diversificado de tipos de projetos. Isso sugere que:

- O ecossistema JavaScript é diverso, com diferentes tipos de projetos ganhando popularidade por diferentes razões
- Não há um único caminho para a popularidade; tanto bibliotecas complexas quanto listas simples podem ser amplamente adotadas
- A categorização de projetos pode ser útil para entender as tendências e necessidades da comunidade JavaScript

5 Discussão

Nesta seção, discutimos os principais resultados, comparando-os com as expectativas iniciais e a literatura existente, além de explorar as implicações dos padrões identificados.

5.1 Comparação com Resultados Esperados

Tabela 2: Comparação entre Resultados Esperados e Observados

RQ	Resultado Esperado	Resultado Observado
RQ1	Repositórios maiores teriam maior popularidade	Relação fraca entre tamanho e popularidade; listas curadas são populares apesar do pequeno tamanho
RQ2	Projetos com mais dependências teriam maior complexidade	Apenas alguns projetos com mais dependências têm maior complexidade; maioria tem complexidade zero
RQ3	Existem diferentes perfis de projetos	Confirmado: identificados bibliotecas, listas curadas e projetos educacionais

Como mostrado na Tabela 2, apenas uma das três questões de pesquisa foi confirmada como esperávamos (RQ3), enquanto as outras duas apresentaram resultados mais complexos do que o previsto inicialmente.

5.2 Análise dos Padrões Identificados

Os resultados revelam padrões interessantes sobre o ecossistema JavaScript:

5.2.1 Popularidade vs. Tamanho

A ausência de correlação forte entre tamanho e popularidade (RQ1) sugere que a comunidade JavaScript valoriza mais a utilidade e a qualidade do que a quantidade de código. Isso está alinhado com estudos que mostram que desenvolvedores preferem bibliotecas e ferramentas que resolvem problemas específicos de forma eficiente [2].

5.2.2 Complexidade e Dependências

O fato de a maioria dos projetos ter complexidade média zero (RQ2) é surpreendente e pode indicar limitações na métrica utilizada. No entanto, para os projetos com complexidade acima de zero, a correlação com dependências sugere que aplicações completas tendem a ser mais complexas, o que é consistente com a literatura sobre arquitetura de software [3].

5.2.3 Diversidade de Perfis

A identificação de diferentes perfis de projetos (RQ3) destaca a maturidade do ecossistema JavaScript, que suporta desde bibliotecas de baixo nível até recursos educacionais e listas curadas. Essa diversidade é um sinal de um ecossistema saudável e maduro [4].

5.3 Implicações dos Resultados

Os resultados obtidos têm implicações tanto para desenvolvedores quanto para pesquisadores:

5.3.1 Para Desenvolvedores

- Foco na utilidade e resolução de problemas em vez de simplesmente aumentar o tamanho do projeto
- Considerar a complexidade ao gerenciar dependências, especialmente para aplicações completas
- Explorar diferentes nichos dentro do ecossistema JavaScript, desde bibliotecas especializadas até recursos educacionais

5.3.2 Para Pesquisadores

- Necessidade de métricas mais adequadas para capturar a complexidade de projetos JavaScript modernos
- Importância de considerar diferentes perfis de projetos ao analisar o ecossistema
- Oportunidade de investigar mais a fundo os fatores que influenciam a popularidade de projetos de diferentes tipos

6 Conclusão

Este estudo analisou os 15 repositórios JavaScript mais populares no GitHub, com o objetivo de identificar padrões relacionados à popularidade, tamanho, complexidade e dependências. Os resultados revelaram insights importantes sobre o ecossistema JavaScript, destacando sua diversidade e complexidade.

6.1 Principais Descobertas

- **Popularidade e Tamanho:** Não há correlação forte entre tamanho do projeto e popularidade; listas curadas e recursos educacionais podem ser tão populares quanto bibliotecas complexas.
- **Complexidade e Dependências:** A maioria dos projetos tem complexidade média zero, mas projetos com mais dependências tendem a ter maior complexidade.
- **Diversidade de Perfis:** Os repositórios mais populares podem ser categorizados em diferentes perfis, incluindo bibliotecas, frameworks, listas curadas e projetos educacionais.

6.2 Limitações

O estudo apresenta várias limitações importantes:

- **Métrica de Complexidade:** A métrica de complexidade ciclomática média pode não ser adequada para capturar a complexidade real de projetos JavaScript modernos.
- **Amostra:** A limitação a apenas 15 repositórios pode não ser representativa de todo o ecossistema JavaScript.
- **Momento da Coleta:** Os dados representam um momento específico no tempo e podem não refletir a evolução dos projetos.

6.3 Trabalhos Futuros

Com base nos resultados e limitações identificadas, sugerimos os seguintes trabalhos futuros:

- **Métricas Alternativas:** Explorar outras métricas de complexidade mais adequadas para projetos JavaScript.
- **Análise Temporal:** Investigar como as métricas evoluem ao longo do tempo para os projetos populares.
- **Expansão da Amostra:** Incluir mais repositórios e de diferentes linguagens para comparação.
- **Análise Qualitativa:** Complementar a análise quantitativa com investigações qualitativas sobre os fatores que influenciam a popularidade.

Apesar das limitações, este estudo contribui para a compreensão do ecossistema JavaScript, destacando a importância da utilidade sobre o tamanho e a diversidade de perfis de projetos que ganham popularidade na comunidade.

Referências

[1] GitHub. *The Octoverse 2023*. Disponível em: https://octoverse.github.com/, 2023.

[2] Taeuber, F., et al. *When do changes trigger updates? a study of the linux kernel*. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, 2017.

[3] Fowler, M. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.

[4] Munaiah, N., et al. *Curating GitHub for engineered software projects*. Empirical Software Engineering, 22(6), 3219-3255, 2017.