Vinicius Brenner

Uma Avaliação em Larga Escala do Truck Factor em Sistemas Open Source

Belo Horizonte, Minas Gerais
2020

Vinicius Brenner

Uma Avaliação em Larga Escala do Truck Factor em Sistemas Open Source

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

Orientador: André Hora

Belo Horizonte, Minas Gerais
2020

Sumário

1 Introdução

A rotatividade é uma característica muito presente no contexto de desenvolvimento de software e seus efeitos e causas são continuamente estudados, tanto no campo empresarial [1, 2, 3] quanto na comunidade open source [4, 5, 6]. Esse aspecto causa perdas na qualidade do software desenvolvido, especialmente quando se trata de desenvolvedores chaves, que retêm conhecimento sobre o código fonte e funcionamento do sistema e são responsáveis por decisões importantes a respeito da arquitetura e evolução do sistema [7].

Uma forma de moderar as perdas de qualidade relacionadas a rotatividade é garantir que o conhecimento sobre o sistema em questão esteja bem dividido entre os colaboradores, de forma que o número de colaboradores que tenham que deixar o projeto para que este fique defasado seja o maior possível. Esse número é conhecido por Truck Factor, definido como *"the number of people on your team that have to be hit by a truck (or quit) before the project is in serious trouble."* [8], e é o objeto de estudo deste trabalho.

1.1 Objetivos Gerais

Avaliar o comportamento das métricas de software orientadas a tamanho (LOC, quantidade de desenvolvedores, de commits, e de arquivos) e do TF em sistemas open source populares nos quais predominam diferentes linguagens de programação. Para atingir um nível satisfatório de abrangência foram escolhidos os 100 repositórios com mais estrelas no GitHub de cada uma das 5 linguagens de programação mais utilizadas.

1.2 Objetivos Específicos

Entender como as métricas avaliadas se comportam em sistemas de diferentes tamanhos e como o TF é influenciado por essas métricas e pela linguagem predominante em um sistema open source. Avaliar a mudança do TF ao longo da evolução do sistema.

2 Referencial Teórico

2.1 Medição de Software

A medição de software é a área de estudo da Engenharia de Software preocupada em derivar valores numéricos ou traçar perfis para atributos de componentes, sistemas ou processos de software [9]. Utilizando as métricas definidas por essa área de estudo, é possível tirar conclusões sobre a qualidade do software, avaliar a eficácia dos processos, das ferramentas e das metodologias de desenvolvimento. Com a medição de software é possível extrair características de um sistema que permitem realizar uma análise a respeito do seu grau de manutenibilidade e compreensibilidade [10], aspectos importantes quando a rotatividade das equipes de desenvolvimento é levada em consideração. Neste trabalho o conceito de medição de software é muito importante e amplamente utilizado, uma vez que os objetos de análise são métricas de software.

2.2 Truck Factor

Truck Factor é uma métrica de software que mensura a distribuição de conhecimento de um sistema entre seus desenvolvedores, definida por *"the number of people on your team that have to be hit by a truck (or quit) before the project is in serious trouble."* [8]. Essa é uma medida importante do risco gerado pela rotatividade das equipes de desenvolvimento. Baixos valores para o TF indicam que o conhecimento sobre o sistema está pouco distribuído entre os desenvolvedores, o que pode ser preocupante se um desenvolvedor chave deixar a equipe. Em contrapartida, altos valores para o TF indicam uma distribuição homogênea do conhecimento sobre o sistema, em um cenário onde a perda de um membro da equipe tem impactos menores sobre a sobrevivência do sistema.

Existem sistemas bem conhecidos no Github com baixos valores para o TF, como exemplo temos vuejs/vue, lodash/lodash, mockito/mockito, notepad-plusplus/notepad-plus-plus e jupyter/jupyter todos com TF = 1. Também existem sistemas com valores altos para o TF, como são pytorch/pytorch com TF = 28, mongodb/mongo com TF = 27 e elastic/elasticsearch com TF = 17. Um exemplo ainda mais notável é torvalds/linux com um valor de TF=57 [11], mostrando um grande compartilhamento de conhecimento do sistema entre os contribuidores.

Os principais algoritmos para estimar o TF encontrados na literatura: ZWK, proposto por Zazworka et al (2010), CST, proposto por Cosentino et al (2015), AVL, proposto por Avelino et al (2016), e RIG, proposto por Rigby et al (2016), são objetos de estudo de Mívian Ferreira et al (2019) em um artigo que avalia e compara cada um dos algoritmos [12]. As considerações e resultados apresentados por Mívian Ferreira et al (2019) são a motivação para a escolha do algoritmo AVL como ferramenta de estudo deste trabalho. O algoritmo AVL proposto por Avelino et al (2016) para estimar o TF é descrito em detalhes na terceira seção do artigo que o apresenta [11].

3 Metodologia

3.1 Dataset

Foram selecionados os 100 sistemas reais com mais estrelas de cada uma das seguintes linguagens: JavaScript, Python, Java, PhP e C++. Essas linguagens são as 5 mais populares atualmente no Github e aparecem no top 10 do TIOBE Index for March 2020 [13]. A quantidade de estrelas foi considerada por ser uma métrica frequentemente adotada em estudos de mineração de software como um indicador de popularidade e relevância do projeto [14, 15]. Foram escolhidos 500 sistemas com a intenção de alcançar uma amostra que representasse de forma significativa o cenário real do Github e avaliar de forma mais abrangente o algoritmo AVL, que foi validado usando 133 sistemas [11].

3.2 Coleta dos Sistemas

Para cada uma das 5 linguagens escolhidas, 100 sistemas reais foram filtrados dentre os repositórios com mais estrelas, de forma que codebases, guias e coletâneas de algoritmos fossem excluídos do dataset de estudo. Os repositórios freecodecamp/freecodecamp, thealgorithms/python e cyc2018/cs-notes, por exemplo, possuem muitas estrelas mas não cabem no escopo deste trabalho. Agora com os sistemas alvo já listados passamos para o cálculo das métricas escolhidas. Para cada um dos sistemas executamos um algoritmo que ao final retornava o conjunto das métricas daquele sistema. O algoritmo utilizado está detalhado a seguir.

Passo 1 - Clone: Para ter acesso a todos os arquivos e commits localmente, o primeiro passo é realizar o clone do repositório git do sistema.

Passo 2 - Contar commits: Executamos um comando git log, selecionando algumas informações que serão utilizadas posteriormente pelo algoritmo AVL, e contamos a quantidade de commits do repositório.

Passo 3 - Contar arquivos: Para contar o número de arquivos, primeiro precisamos escolher quais tipos de arquivos são de interesse para essa métrica, uma vez que arquivos de texto, documentação e bibliotecas utilizadas pelo sistema não precisam entrar nessa contagem. Faremos isso utilizando o github/linguist, uma biblioteca criada e utilizada pelo Github.com que determina a linguagem utilizada por cada arquivo e permite o descarte dos não desejáveis para uma determinada situação. Com o linguist conseguimos gerar uma lista com todos os arquivos interessantes para essa métrica e assim saber o número de arquivos.

Passo 4 - Contar LOC: A partir da lista de arquivos gerada no passo anterior contamos o número de linhas de código utilizando o AlDanial/cloc, um sistema simples que gera um relatório de LOC de um arquivo, diretório ou repositório git.

Passo 5 - Contar contribuidores: Com o comando git log contamos o número de contribuidores.

Passo 6 - Extração do Truck Factor: Nesse passo a ferramenta disponível em gavelino/Truck-Factor é executada, calculando o TF do sistema. Os arquivos necessários para execução dessa ferramenta foram gerados nos passos 2 e 3.

Passo 7 - Guardando dados: Neste último passo as métricas calculadas são escritas no arquivo de saída e o repositório local é apagado deixando o algoritmo pronto para clonar o próximo repositório git.

Algoritmo 1: Extração das Métricas

Entrada: Arquivo de texto com URLs de todos os sistemas selecionados

Saída: Conjunto: [Nome do repositório, Número de commits, Número de arquivos, Número de contribuidores, Truck Factor] para cada URL da entrada

início

```
abrirArquivo(entrada);
repita
    URL = linha da entrada;
    nome = getNome(URL);
    git clone(URL);
    commits = commitLogScript(URL);
    arquivos = linguistScript(URL);
    LOC = cloc(URL);
    contribuidores = devsLogScript(URL);
    TF = gitTruckFactor(URL);
    Saida = escreve(nome, commits, arquivos, LOC, contribuidores, TF);
até eof;
retorna Saida
```

fim

4 Resultados

4.1 LOC, Commits, Arquivos e Contribuidores

As cinco linguagens analizadas tem distribuições semelhantes para cada uma das métricas, com médias e limites baixos e apresentando alguns outliers. Se levando em consideração número de commits e linhas de código, C++ aparece mais acima das demais linguagens, enquanto para linhas de código e número de arquivos as linguagens Python e PHP apresentam valores abaixo das demais.

4.2 Truck Factor

A maior parte dos repositórios análisados tem um Truck Factor baixo, sendo um total de 404 com valor abaixo de 4, divididos em: 232 com TF = 1, 114 com TF = 2 e 58 com TF= 3. Existem também repositórios com valores significativamente altos, como se pode observar na Figura 1.

Repositories	Commits	Files	LOC	TF	Devs
pytorch/pytorch	29570	6548	1129496	28	2370
mongodb/mongo	53747	8939	1137647	27	857
PaddlePaddle/Paddle	28257	4150	590985	27	645
odoo/odoo	134241	4701	675469	27	1876
v8/v8	64248	10393	2081251	21	932
apache/incubator-mxnet	11289	1819	335605	18	1049
elastic/elasticsearch	54518	14120	1754569	17	1817
apache/hadoop	24173	12434	1908489	16	591
python/cpython	108048	2644	988581	16	1582
nodejs/node	31551	4526	423490	15	3236

Figura 1 – Sistemas com maior Truck Factor

As linguagens apresentam distribuições semelhantes para os valores de TF, com médias e limites baixos e apresentando alguns outliers que podem ser observados na Figura 2. As linguagens Java e PHP são as que mostram as menores médias, próximas de 1.

Sabemos que na média um repositório da base de dados utilizada tem cerca de 148 mil linhas de código, 1138 arquivos, 7968 commits e 381 contribuidores e 2.81 de TF. Contudo a maior parte das amostras se concentra abaixos dessas médias e por issod precisamos levar em consideração que esses valores são deslocados pelos outliers.

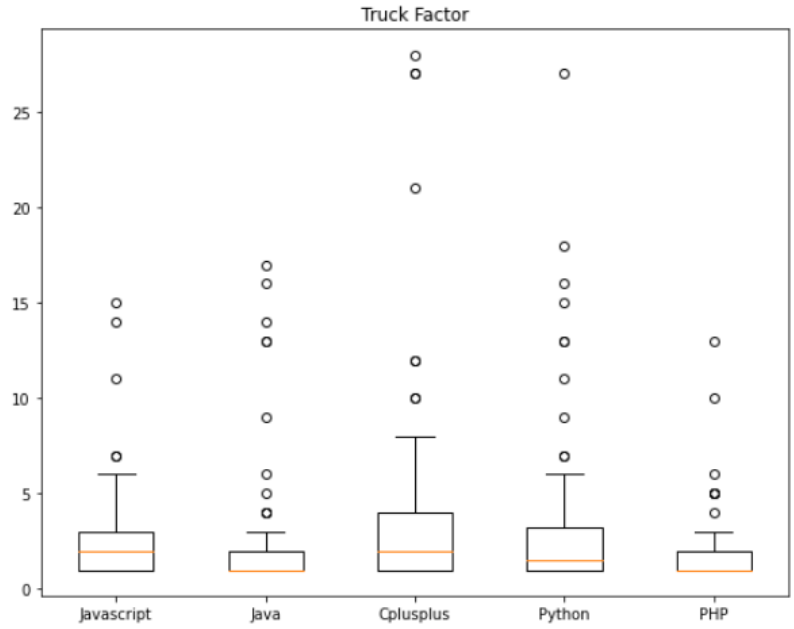
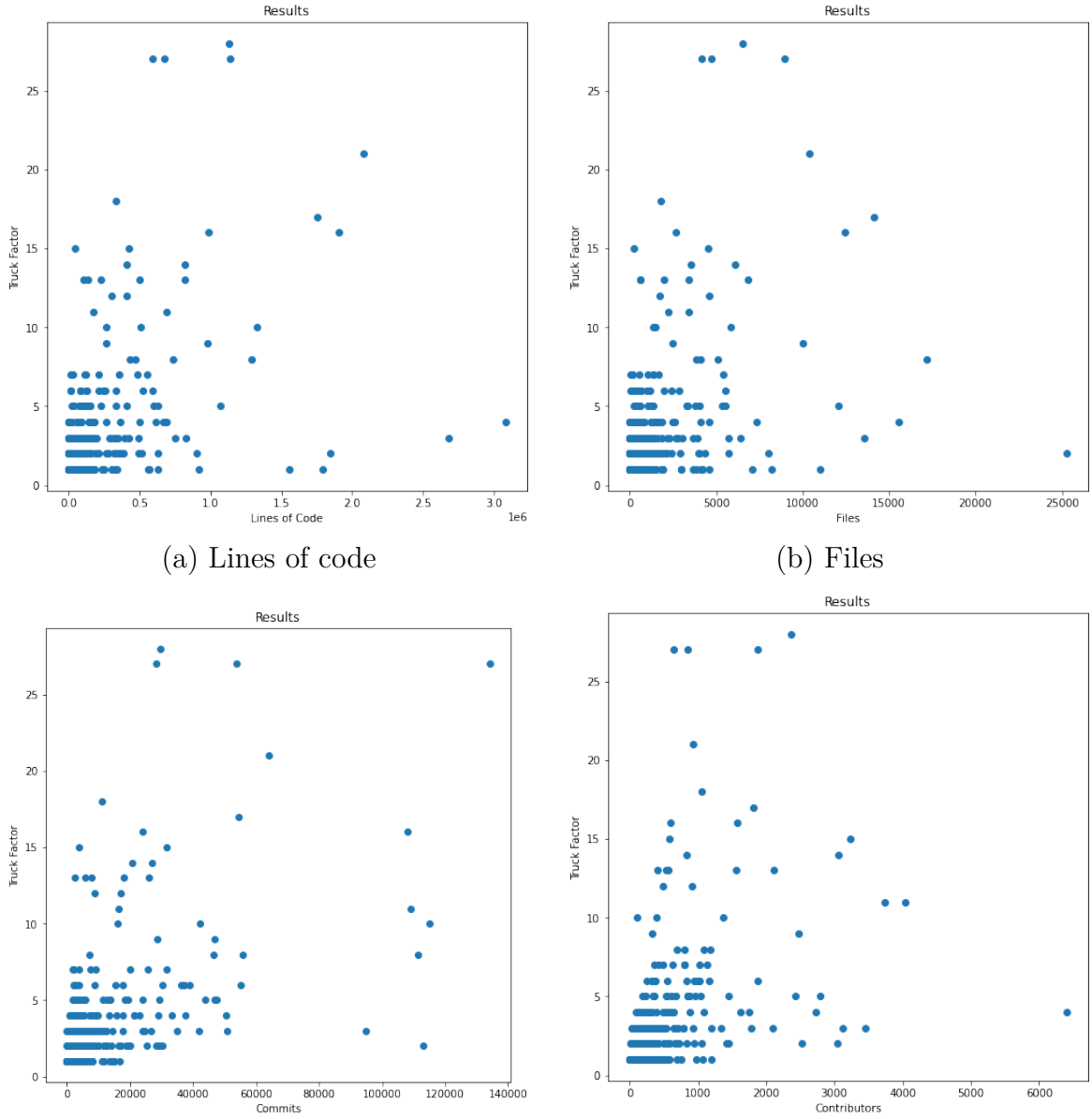


Figura 2 – Truck Factor

4.3 Associação das métricas

A Figura 3 mostra como as métricas se relacionam com o Truck Factor. A maior parte dos sistemas tem valores baixos para o TF e para as demais métricas, como pode ser facilmente observado na figura. Contudo se observarmos com mais atenção os sistemas com TF maior que 5, perceberemos que essa parte da amostra se concentra na primeira metade do gráfico, indicando que ter menos linhas de código, arquivos, commits e contribuidores pode ser um fator para alcançar valores satisfatórios de TF. Uma interpretação que pode ser feita é que um sistema com essas características pode ser entendido com mais facilidade, e por consequência também é mais fácil distribuir conhecimento do código entre os contribuidores.

Alguns sistemas fogem ao comportamento da amostra, como é o caso do odoo/odoo, que tem um valor alto para o TF e uma grande quantidade de commits, o que pode ser justificado pelo propósito deste sistema de ser um conjunto de *"business apps"*, tendo contribuidores trabalhando em diversas partes do sistema. Outro sistema que se destaca bastante da amostra é o magento/magento2, por ser o primeiro em número de arquivos e o quinto em linhas de código e ter um TF = 2, mostrando uma grande concentração de conhecimento em um sistema que é bem grande. Esse exemplo é seguido por outros sistemas que tem muitas linhas de código e baixo TF, indicando que o tamanho do sistema influencia na forma como o conhecimento deste é distribuído.



(a) Lines of code

(b) Files

(c) Commits

(d) Contributors

Figura 3 – Truck Factor x demais métricas

4.4 Evolução

Outro resultado interessante que alcançamos foi a evolução do TF ao longo do tempo. Selecionamos os 5 repositórios com maior TF de cada linguagem e calculamos essa métrica para 10 versões diferentes, escolhendo commits igualmente distribuídos ao longo da idade do repositório. Então, a cada ponto no eixo das abscissas considera-se que passou 10% da idade de cada repositório.

A Figura 4 descreve um crescimento aparentemente linear do TF da amostra ao longo do tempo, indicando que essa métrica acompanha a evolução dos sistemas e que o conhecimento sobre um projeto precisa de tempo para ser bem distribuído. Entretanto em alguns casos houve um esforço para que esse TF aumentasse, mas não se sabe ao certo se por

causa ou consequência de algum outro fator. Por exemplo, o sistema odoo/odoo aumenta de TF = 3 para 10, do ponto dois para o ponto três, assim como o elastic/elasticsearch que aumenta de TF = 3 no ponto seis para TF = 12 no ponto oito, após ter se tornado obsoleto no ponto sete atingindo um TF = 0.

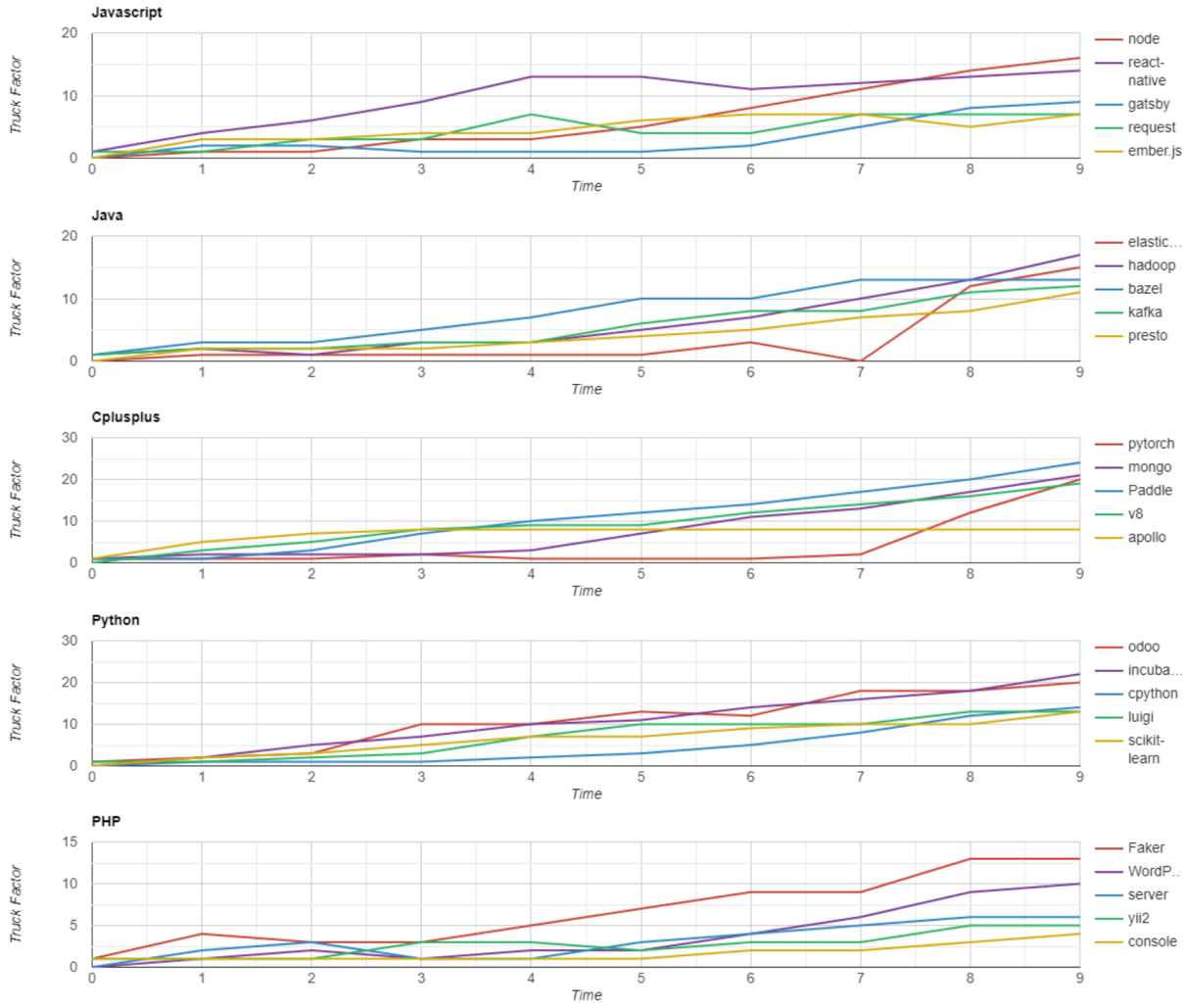


Figura 4 – Truck Factor ao longo do tempo

5 Conclusão

Nesse trabalho foi feita uma avaliação do Truck Factor em 500 sistemas open source hospedados no Github e sua relação com outras métricas de software. Descobrimos que a maior parte dos sistemas é mantido por poucos contribuidores, mesmo que o número total desses seja relativamente grande, e que a média de TF da amostra é bem baixa se levado em consideração o conceito do Truck Factor, explicado na seção 2.2. Sabemos que uma parte pequena mas importante da amostra apresenta valores baixos para o conjunto de métricas estabelecidas e um TF alto. Os sistemas com TF alto apresentam um crescimento linear desse valor ao longo de sua idade, o que pode ocorrer tanto por esforço dos mantenedores do sistema como por interesse dos contribuidores de aprender e contribuir mais com o projeto. Isso pode ser uma tendência de todos os sistemas, mas não foi abordado por esse trabalho.

Referências

[1] Sahu, Anamika, and Meenakshi Gupta. “An Empirical Analysis of Employee Turnover in a Software Organization.” Indian Journal of Industrial Relations, vol. 35, no. 1, 1999, pp. 55–73. JSTOR, www.jstor.org/stable/27767634. Accessed 14 Aug. 2020.

[2] Nagadevara, V., Srinivasan, V. & Valk, R. (2008). Establishing a Link between Employee Turnover and Withdrawal Behaviours: Application of Data Mining Techniques, Research and Practice in Human Resource Management, 16(2), 81-99.

[3] L. Bao, Z. Xing, X. Xia, D. Lo and S. Li, "Who Will Leave the Company?: A Large-Scale Industry Study of Developer Turnover by Mining Monthly Work Report,"2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), Buenos Aires, 2017, pp. 170-181, doi: 10.1109/MSR.2017.58.

[4] Robles G., Gonzalez-Barahona J.M. (2006) Contributor Turnover in Libre Software Projects. In: Damiani E., Fitzgerald B., Scacchi W., Scotto M., Succi G. (eds) Open Source Systems. OSS 2006. IFIP International Federation for Information Processing, vol 203. Springer, Boston, MA . https://doi.org/10.1007/0-387-34226-5_28

[5] Matthieu Foucault, Marc Palyart, Xavier Blanc, Gail C. Murphy, and Jean-Rémy Falleri. 2015. Impact of developer turnover on quality in open-source software. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015). Association for Computing Machinery, New York, NY, USA, 829–841. DOI:https://doi.org/10.1145/2786805.2786870

[6] B. Lin, G. Robles and A. Serebrenik, "Developer Turnover in Global, Industrial Open Source Projects: Insights from Applying Survival Analysis,"2017 IEEE 12th International Conference on Global Software Engineering (ICGSE), Buenos Aires, 2017, pp. 66-75, doi: 10.1109/ICGSE.2017.11.

[7] Enrico di Bella, Alberto Sillitti, Giancarlo Succi, A multivariate classification of open source developers, Information Sciences, Volume 221, 2013, Pages 72-83, ISSN 0020-0255, https://doi.org/10.1016/j.ins.2012.09.031.

[8] L. Williams and R. Kessler, Pair Programming Illuminated. Addison Wesley, 2003.

[9] Ian Sommerville. Software Engineering, 9ª Edition. Pearson Education, 2011. Cap. 24 Quality Management

[10] Zuse, Horst. "History of software measurement."online, last update 9 (1995).

[11] Guilherme Avelino, Leonardo Passos, Andre Hora, Marco Tulio Valente. A Novel Approach for Estimating Truck Factors. In 24th International Conference on Program Comprehension (ICPC), pages 1-10, 2016.

[12] Mivian Ferreira, Thais Mombach, Marco Tulio Valente, Kecia Ferreira. Algorithms for Estimating Truck Factors: A Comparative Study. Software Quality Journal, vol. 1,

pages 1-37, 2019.

[13] TIOBE Index for March 2020, arquivado em
web.archive.org/web/20200313171922/https://www.tiobe.com/tiobe-index/

[14] Hudson Borges, Andre Hora, Marco Tulio Valente. Understanding the Factors that Impact the Popularity of GitHub Repositories. In 32nd IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 334-344, 2016.

[15] Hudson Silva, Marco Tulio Valente. What's in a GitHub Star? Understanding Repository Starring Practices in a Social Coding Platform. Journal of Systems and Software, vol. 146, pages 112-129, 2018.