

STF: uma abordagem Social para estimar Truck Factor no GitHub

Hercules Sandim¹, Michele A. Brandão¹, Mirella M. Moro¹

¹Universidade Federal de Minas Gerais, Belo Horizonte, Brasil

herculeessandim@ufmg.br, {micheleabrandao,mirella}@dcc.ufmg.br

Abstract. *Truck Factor (TF) is the number of developers who would disrupt a project if they abandoned it. Calculating it is a complex task, and there are few approaches to estimate it. Here, we propose an approach to estimate TF in a GitHub collaboration network based on the ties strength between developers. We also evaluate it against the state of art with promising results.*

Resumo. *Truck Factor (TF) é o número de desenvolvedores que perturbariam um projeto se o abandonassem. Calculá-lo é uma tarefa de alta complexidade, e são poucas iniciativas para estimá-lo. Aqui, propomos um método para estimar o TF em uma rede de colaboração do GitHub com base na força dos relacionamentos entre os desenvolvedores. Sua avaliação produz resultados promissores.*

1. Introdução

Na Engenharia de Software, *Truck Factor* – *TF* (também conhecido como *Bus Factor*, *Lottery Factor*, *Bus/Truck Number* ou *Lorry Factor*) é uma medida de risco que analisa o grau de conhecimento compartilhado em um projeto de software. Apesar da alta complexidade [Hannebauer and Gruhn 2014], o cálculo do TF é importante para identificar a formação de silos de conhecimento (quando apenas uma pessoa ou grupo de pessoas detêm o conhecimento) entre equipes de desenvolvimento, no intuito de antever riscos de descontinuidade em projetos de software [Avelino et al. 2016]. Porém, há poucas iniciativas para estimar o TF de um sistema, e não escalam para grandes equipes. As exceções são os estudos em [Avelino et al. 2016, Ferreira et al. 2017], os quais estimam o TF com boa acurácia de acordo com evidências empíricas, utilizando dados de atividade de manutenção extraídos do sistema de controle de versões de repositórios do GitHub¹.

Os dados coletados do GitHub podem ser utilizados para os mais variados propósitos. Por exemplo, é possível modelar uma rede social de desenvolvedores que interagem através de criação, contribuição e compartilhamento de repositórios e projetos de software [Alves et al. 2016, Batista et al. 2017, Li et al. 2017]. Dessa forma, modelando a rede como um grafo, várias análises podem ser feitas através de métricas topológicas e semânticas sobre os nós e/ou as arestas. A força de colaboração entre os desenvolvedores do GitHub possui ampla aplicabilidade em tarefas de ranqueamento, recomendação e detecção de comunidade [Batista et al. 2017, Easley and Kleinberg 2010, Li et al. 2017].

Aqui, propomos o *STF* - *Social Truck Factor*, uma abordagem social para estimar o TF a partir de uma rede de colaboração sobre o GitHub. STF se baseia no ranqueamento de desenvolvedores a partir da agregação de métricas topológicas e semânticas da

¹Plataforma para hospedagem e controle de versões de códigos-fonte: <https://github.com>

rede de colaboração. Considerar tal aspecto social permite melhor capturar o nível de interação entre os desenvolvedores [Batista et al. 2017]. Ademais, o STF não necessita que sejam realizadas pesquisas com desenvolvedores, como em algoritmos do estado-da-arte [Avelino et al. 2016, Ferreira et al. 2017]. Após apresentá-la, também descrevemos os resultados de uma análise preliminar, comparando seu desempenho ao estado-da-arte.

2. Soluções Atuais para *Truck Factor*

De maneira geral, TF é o número de desenvolvedores que interromperiam o projeto se abandonassem o mesmo. Sistemas com baixo valor de TF apresentam forte dependência de poucas pessoas do time de desenvolvimento, formando silos de conhecimento entre as equipes de desenvolvedores [Avelino et al. 2016]. Apesar de sua importância, soluções ótimas para seu cálculo necessitam de algoritmos de alta complexidade [Hannebauer and Gruhn 2014]. Mesmo assim, considerando o melhor de nosso conhecimento, não existe uma definição precisa e há poucas maneiras para estimá-lo.

Por exemplo, Ricca et al. [2011] e Zazworka et al. [2010] abordam estratégias para estimar o TF de um sistema, porém sem apresentar evidências empíricas e, portanto, carecendo da confiabilidade dos sistemas reais. Recentemente, Avelino et al. [2016] e Ferreira et al. [2017] propõem e comparam algoritmos para estimar TF usando dados de atividade de manutenção extraídos de sistemas de controle de versão. Por outro lado, apresentamos uma abordagem diferente para estimar o TF de um sistema por meio de métricas topológicas e semânticas extraídas de uma *rede de colaboração* do GitHub. Dessa forma, nossos resultados (ainda em fase inicial) são comparados aos resultados e ao oráculo proposto por Avelino et al. [2016] e Ferreira et al. [2017] na Seção 5.

3. Ranqueamento dos Desenvolvedores

Essa seção apresenta uma modelagem da rede de colaboração do GitHub e a sumarização de métricas topológicas e semânticas apresentadas em [Adamic and Adar 2003, Batista et al. 2017]. Após, introduz uma estratégia para ranqueamento de desenvolvedores com base na agregação das classificações obtidas através das métricas para força de colaboração entre desenvolvedores de um mesmo repositório. Tal ranqueamento é parte crucial para o Algoritmo STF (Seção 4).

Modelagem e Métricas para Força de Colaboração. O GitHub pode ser considerado como uma rede complexa, a qual é modelada como um grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$: \mathcal{V} é o conjunto de nós que representam os desenvolvedores, e \mathcal{E} é o conjunto de arestas não-direcionadas que conectam pares que colaboram em um mesmo repositório. A partir do grafo, é possível identificar propriedades topológicas (referem-se às características da *estrutura* da rede) e semânticas (características particulares da rede que capturam o significado de seus elementos). Considerando o GitHub, a Tabela 1 resume as propriedades topológicas definidas em [Adamic and Adar 2003, Brandão and Moro 2017] e que auxiliam a analisar as colaborações em desenvolvimento de software, bem como as propriedades semânticas de colaboração com as métricas propostas em [Alves et al. 2016, Batista et al. 2017].

Ranqueamento *CombSUM* – CSR. Métodos de classificação podem ser aplicados ao GitHub para identificar especialistas, ou seja, classificar os desenvolvedores de acordo

Tabela 1. Propriedades Topológicas e Semânticas aplicadas ao GitHub.

Para um nó X da rede: $\mathcal{N}(X)$ é o conjunto de vizinhos de X , $w(X)$ é a soma dos pesos das arestas conectadas a X , $w(X, Y)$ é o peso da aresta entre X e Y , e \mathcal{R} é o conjunto de todos os repositórios onde X e Y colaboram

Propriedades topológicas	
Métrica	Definição e interpretação
<i>Adamic-Adar Coefficient (AA)</i> [Adamic and Adar 2003]	Estipula maior peso aos vizinhos que não se relacionam com muitos outros, e é definida pela equação: $AA_{(X,Y)} = \sum_{\forall Z \in \mathcal{N}(X) \cap \mathcal{N}(Y)} \frac{1}{\log \mathcal{N}(Z) }$.
<i>Tieness (T)</i> [Brandão and Moro 2017]	Mede a força das relações de coautoria (i.e., em rede formada por autores de trabalhos científicos). No contexto da rede de colaboração do GitHub: $T_{(X,Y)} = \frac{ \mathcal{N}(X) \cap \mathcal{N}(Y) + 1}{1 + \mathcal{N}(X) \cup \mathcal{N}(Y) - \{X, Y\} } w(X, Y) $. A métrica <i>Tieness</i> é um caso particular de propriedade topológica, pois é ponderada por um peso $w(X, Y)$. Entretanto, no contexto do GitHub, tal peso pode ser representado pelas métricas semânticas descritas a seguir*.
Propriedades semânticas [Alves et al. 2016, Batista et al. 2017]	
Métrica	Definição e interpretação
<i>Previous Collaboration (PC)</i>	Seja $ND_{(r_i, t)}$ o número de desenvolvedores no repositório r_i no tempo t , $PC_{(X,Y,t)}$ é a colaboração oferecida por X e Y no tempo t : $PC_{(X,Y,t)} = \frac{\sum_{\forall r_i \in \mathcal{R}} ND_{(r_i, t)}}{ \mathcal{R} }$.
<i>Jointly Developers Contribution to Shared Repositories (JCSR)</i>	Seja $JCSR_{(X,Y,r_i)}$ a razão entre a quantidade de desenvolvedores envolvidos no relacionamento e o total de desenvolvedores em r_i : $JCSR_{(X,Y)} = \frac{\sum_{\forall r_i \in \mathcal{R}} JCSR_{(X,Y,r_i)}}{ \mathcal{R} }$.
<i>Jointly Developers Commits to Shared Repositories (JCOSR)</i>	Sejam $NC_{(X,r_i)}$ o número de <i>commits</i> por um usuário X em um repositório r_i e $NC_{(r_i)}$ o total de <i>commits</i> no repositório r_i , temos que: $JCOSR_{(X,Y)} = \sum_{\forall r_i \in \mathcal{R}} \frac{NC_{(X,r_i)} + NC_{(Y,r_i)}}{NC_{(r_i)}}$.

*As métricas semânticas, se combinadas com a métrica *Tieness*, denotam novas métricas: T_PC, T_JCOSR e T_JCSR.

com algum recurso relacionado à experiência. Aqui, para cada métrica, os desenvolvedores são classificados a partir do somatório da força de seus relacionamentos para identificar os principais desenvolvedores sob o aspecto da métrica utilizada. Por exemplo, o ranqueamento dos desenvolvedores pela métrica JSCR (denotado por R_JSCR) é definido através da ordenação decrescente do conjunto: $\left\{ \sum_{\forall j \in \mathcal{N}(X)} JSCR_{(X,j)} \mid \forall X \in \mathcal{V} \right\}$.

De forma suplementar, denotamos $R_JSCR(r_i)$ como o ranqueamento de desenvolvedores que atuam num mesmo repositório r_i . Assim, os ranqueamentos para as demais métricas são os conjuntos $R_AA(r_i)$, $R_PC(r_i)$, $R_JCOSR(r_i)$, $R_T_JSCR(r_i)$ e $R_T_JCOSR(r_i)$. Após ranquear, combinamos as diferentes classificações para gerar um ranqueamento único através do método *CombSUM* que agrega as classificações pela soma dos valores [Ganjisaffar et al. 2011]. A classificação final (denotada por CSR) é definida por ordem decrescente da soma dos valores obtidos por cada desenvolvedor nas métricas. Há outros métodos para agregação de ranqueamentos, como *Borda Count* [Emerson 2013] e *Condorcet* [Young 1988], mas o método *CombSUM* é o único que mantém o valor agregado, ao invés de simplesmente retornar o novo ranqueamento. Além disso, ressaltamos que todas as métricas apresentam-se normalizadas dentro do intervalo $[0, 1]$ para garantir que não haja viés causado pela distribuição diferente de valores.

4. Algoritmo Social Truck Factor – STF

O Algoritmo 1 é a nossa solução para estimar o TF de um sistema com base nas propriedades topológicas e semânticas de uma rede social de colaboração do GitHub. Denominado *Social Truck Factor – STF*, tal algoritmo recebe uma rede de colaboração do GitHub, um repositório alvo e um parâmetro ϵ (critério de parada). O STF computa os ranqueamentos de desenvolvedores com base nas métricas (linha 2) e produz um ranqueamento único

Algoritmo 1: SOCIAL_TRUCK_FACTOR

Entrada: RedeDeColaboracaoGitHub, repos, ϵ
Saída: Lista estimada de *Truck Factors* do repositório repos.

```
1 início
2   R[] = ComputarTodosRanks(RedeDeColaboracaoGitHub, repos)
3   CSR = CombSum (R)
4   STF_LISTA=[ CSR[1] ]
5   i = 2
6   enquanto (i ≤ |CSR| e ((CSR[i]-CSR[i-1]) ≤  $\epsilon$ ) faça
7     STF_LISTA[i] = CSR[i]
8     i = i + 1
9   fim
10 fim
11 retorna STF_LISTA
```

(CSR) com base na estratégia apresentada na Seção 3 (linha 3). O primeiro desenvolvedor do ranqueamento CSR inicializa a lista estimada de *Truck Factors* (STF_LISTA, na linha 4). Então, o método iterativo (linhas 6 a 9) seleciona os próximos desenvolvedores do ranqueamento CSR para compor a STF_LISTA, até satisfazer um dos dois critérios de parada (linha 6): percorrer todos os desenvolvedores do ranqueamento CSR; ou atingir um limiar de convergência ϵ , que representa o fator de tolerância máxima para a diferença entre as métricas agregadas dos desenvolvedores das i -ésima e $(i+1)$ -ésima posições de CSR. O STF retorna a lista de desenvolvedores integrantes do TF, em ordem decrescente pelo valor do CSR (linha 11). Consequentemente, o TF é estimado pelo tamanho da lista.

Em suma, o STF seleciona, de maneira gulosa, os desenvolvedores de maior peso agregado no ranqueamento CSR. Contudo, a calibração do parâmetro ϵ é um desafio para alcançar resultados satisfatórios, pois é importante compreender a natureza dos repositórios. Repositórios grandes e relevantes para a comunidade de software livre (por exemplo, torvalds/linux²) tendem a ter um TF maior. Essa afirmação se deve à importância do projeto para a comunidade, onde não pode haver forte dependência de poucos desenvolvedores para mitigar os impactos de possíveis abandonos durante o ciclo de vida do projeto. Por outro lado, a maioria dos demais repositórios tendem a ter TF menor, pois são geralmente instanciados e mantidos por equipes pequenas ou indivíduos. Essas evidências empíricas são relatadas por [Avelino et al. 2016, Ferreira et al. 2017], e a calibração do parâmetro ϵ é discutida a seguir.

5. Análise Experimental

Para apresentar a análise experimental, primeiro descrevemos a metodologia com escolha do conjunto de dados, métricas utilizadas, configuração do parâmetro ϵ e considerações iniciais para a comparação com o estado-da-arte. Em seguida, apresentamos uma análise preliminar do STF em comparação com os resultados obtidos no estado-da-arte. Então, apresentamos considerações que reafirmam a validade do nosso estudo.

Metodologia. Nossa metodologia é composta por uma sequência de passos necessários para a análise preliminar comparativa com o estado-da-arte, como segue. Primeiro, o conjunto de dados utilizado é o GitSED (*GitHub Socially Enhanced Dataset*) [Batista et al. 2017], um conjunto de dados do GitHub curado, expandido e enrique-

²Repositório GitHub com os códigos-fonte do *Kernel* do Linux.

Tabela 2. Número de contribuidores por repositório e linguagem.

Linguagem Javascript		Linguagem Ruby		Linguagem Java	
Repositório	# Contribuidores	Repositório	# Contribuidores	Repositório	# Contribuidores
r.js	10	Google-Maps-for-Rails	10	code_swarm	10
code-standards	10	practicing-ruby-web	10	join-plugin	10
html5shiv	10	orm_adapter	10	jogl-demos	10
grunt	30	parallel_tests	30	kernel	29
node_redis	30	sprockets	30	playn	30
Font-Awesome	30	janky	30	thredds	30
node	703	gitlabhq	521	elasticsearch	435
angular.js	1440	rails	2296	frameworks_base	506

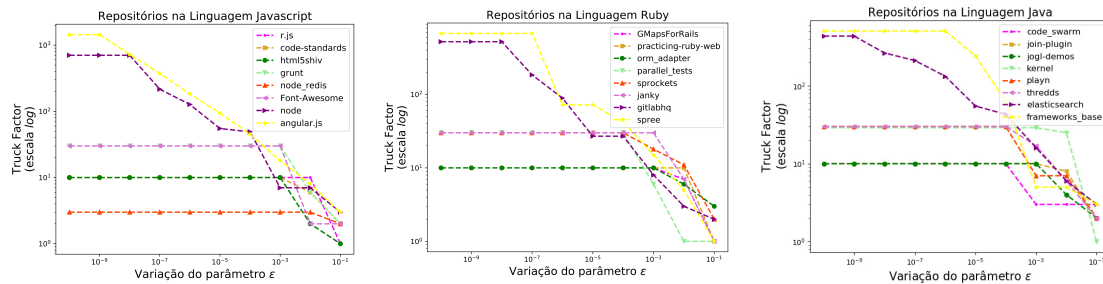


Figura 1. Resultados do STF variando o ϵ no intervalo $[10^{-10}, 10^{-1}]$. Os eixos das ordenadas (ϵ) e das abscissas (Truck Factor) estão na escala \log .

cido a partir do GHTorrent [Gousios 2013]³. Este conjunto contém dados até setembro de 2015 e apresenta informações sobre desenvolvedores e repositórios, de acordo com a modelagem e o cálculo das métricas discutidas na Seção 3, o que poupa relevante esforço computacional. Tal versão do GitSED considera repositórios desenvolvidos em apenas três linguagens de programação (Javascript, Ruby e Java). Apesar disso, o conjunto de dados é adequado para uma comparação preliminar com os resultados em [Avelino et al. 2016, Ferreira et al. 2017], que contém dados até fevereiro de 2015.

Segundo, sobre métricas para força de colaboração no GitHub, Batista et al. [2017] propõem um modelo (com base no estudo de correlações entre as métricas) para selecionar as métricas mais adequadas para mensurar a força de colaboração no mesmo contexto. Assim, nós selecionamos as métricas AA, PC, JCSR, JCSR, T_JCSR e T_JCSR, para a construção do ranqueamento de desenvolvedores.

Terceiro, para a calibração do parâmetro ϵ , realizamos experimentos iniciais em 24 repositórios do GitHub (oito repositórios para cada linguagem de programação do GitSED). A seleção desses repositórios segue a variação na linguagem de programação e no número de contribuidores ativos, conforme a Tabela 2. A Figura 1 apresenta os resultados dessa avaliação preliminar sobre o comportamento do STF, variando o ϵ no intervalo $[10^{-10}, 10^{-1}]$. Corroborando os resultados obtidos por [Avelino et al. 2016, Ferreira et al. 2017], o valor de ϵ deve ser calibrado para o intervalo $[10^{-3}, 10^{-1}]$, pois assim os repositórios com maior número de contribuidores apresentam TF menores, mas ainda maiores em comparação aos repositórios com menor número de contribuidores (onde o TF é próximo de 1). Além disso, não identificamos diferença significativa entre as três linguagens analisadas.

³Vários estudos sobre produção de software consideram dados extraídos do GHTorrent em vez de extrair diretamente do GitHub, como por exemplo em [Jarczyk et al. 2018].

Quarto, para comparar com o estado-da-arte, os estudos em [Avelino et al. 2016, Ferreira et al. 2017] se destacam por se apoiar em evidências empíricas obtidas pela construção de um oráculo para aferir uma boa acurácia (medida pelo erro absoluto: $TF_{\text{algoritmo}} - TF_{\text{oráculo}}$) de seus resultados. Inicialmente, Avelino et al. [2016] propõem uma nova abordagem para estimar o TF de um sistema, executando-a em 133 repositórios do GitHub e avaliando-a por meio de uma pesquisa realizada com desenvolvedores de 67 destes repositórios. Neste caso, a acurácia em descobrir os principais desenvolvedores alcançou 84%, enquanto que a acurácia em estimar corretamente o TF atingiu 53%. Similarmente, Ferreira et al. [2017] realizam uma comparação entre três algoritmos para estimar o TF, verificando a acurácia por meio de um oráculo construído através de pesquisas realizadas com desenvolvedores em 35 repositórios de software-livre no GitHub. Neste caso, a acurácia chega a 100% em 20 repositórios que obtiveram TF=1, porém a acurácia cai para cerca de 50% nos demais repositórios. A disponibilidade pública dos resultados obtidos pelo estado-da-arte⁴ e do conjunto de dados do GitSED⁵ possibilitam que nossos resultados sejam reproduzíveis em estudos correlatos. Como aqui apresentamos uma investigação inicial para estimar o TF de um sistema, as análises preliminares foram realizadas apenas com nove repositórios que estão descritos na Tabela 3.

Comparação com o Estado-da-Arte. A Tabela 3 sumariza nossos resultados quantitativos. O Algoritmo STF estima corretamente o TF em 55% dos casos analisados (celluloid/celluloid, gruntjs/grunt, Leaflet/Leaflet, excilys/androidannotations, netty/netty), e o resultado foi muito próximo nos demais casos (erro médio de 1,25). Apesar disso, percebemos que tais resultados são obtidos com variação do parâmetro ϵ , afirmando a necessidade de maior discussão a respeito da adequação de tal parâmetro. Enquanto o algoritmo do estado-da-arte apresenta maior acurácia para repositórios com TF=1, o STF apresenta maior acurácia para repositórios com TF no intervalo [1,4].

Além da análise quantitativa, o oráculo possibilita uma análise qualitativa no sentido de verificar quais são os desenvolvedores melhor ranqueados para compor o TF. Por exemplo, no repositório emberjs/ember.js, o ranqueamento dos desenvolvedores pelas métricas topológicas e semânticas está descrito na Tabela 4, destacando os ranqueamentos R_JCOSR, R_JCSR e R_AA por identificar o maior número de desenvolvedores (até sete) dentre os elencados pelo oráculo. Tais métricas ainda se destacam por identificar desenvolvedores listados pelo oráculo e não identificados pelo estado-da-arte (MatthewBeale, ErikBryn, AlexMatchneer e TrekGlowacki). Além disso, o STF não retorna o desenvolvedor CharlesJoolley, o qual foi identificado pelo estado-da-arte mas não confirmado pelo oráculo. Note que, por considerar os *commits*, R_JCOSR possui maior proximidade ao conceito de *Truck Factor*. Por outro lado, as demais métricas (R_PC, R_T_JCSR e R_T_JCOSR) não apresentam resultados satisfatórios e poderiam ser descartadas para a composição do ranqueamento agregado.

Ameaças à Validade. A nossa avaliação com dados reais foi possível (e realizada) devido à disponibilidade pública dos mesmos. O STF é original, com implementação inspecionada por nosso grupo de pesquisa. A diferença temporal entre o conjunto de dados utilizado e o conjunto de dados do estado-da-arte é pequena diante do esforço computa-

⁴Disponível em <http://aserg.labsoft.dcc.ufmg.br/truckfactor>.

⁵Disponível em <http://bit.ly/proj-apoena>.

Tabela 3. Repositórios utilizados na análise preliminar do Algoritmo STF.

Repositório	Linguagem	NC*	TF_O*	TF_EA*	STF	
					STF*	ϵ
alexreisner/geocoder	Ruby	233	1	1	2	10^{-1}
celluloid/celluloid	Ruby	90	1	1	1	10^{-1}
rails/rails	Ruby	2296	12	9	13	10^{-3}
gruntjs/grunt	Javascript	64	1	1	1	10^{-2}
Leaflet/Leaflet	Javascript	446	1	1	1	10^{-1}
emberjs/ember.js	Javascript	364	11	6	8	10^{-3}
nicolasgramlich/AndEngine	Java	21	1	1	2	10^{-2}
excilys/androidannotations	Java	58	4	2	4	10^{-1}
netty/netty	Java	238	2	2	2	10^{-1}

*NC: Número de contibuidores, TF_O: TF do oráculo do Estado-da-Arte, TF_EA: TF do Estado-da-Arte, STF: TF estimado pelo Algoritmo STF.

Tabela 4. Os 10 principais desenvolvedores no repositório emberjs/ember.js através das métricas apresentadas na Seção ??.

Pos.	R_AA	R_PC	R_JCSR	R_T_JCSR	R_JCOSR	R_T_JCOSR
1º	Ryunosuke SATO	Rob Monie	Yehuda Katz	Roy Daniels	Stefan Penner	Kristofor Selden
2º	Peter Wagenet	Roy Daniels	Peter Wagenet	Kristofor Selden	Peter Wagenet	Eric Schank
3º	Yehuda Katz	Chris Conley	Stefan Penner	Falk Pauser	Robert Jackson	Mitch Lloyd
4º	Trek Glowacki	Kristofor Selden	Erik Bryn	Christoph	Yehuda Katz	Roy Daniels
5º	Tom Dale	Gustavo Beathyate	Clemens Müller	darkbaby123	Matthew Beale	Yoran Brondsema
6º	Robert Jackson	Brandon Turner	Tom Dale	HipsterBrown	Erik Bryn	Godfrey Chan
7º	Stefan Penner	Doug Mayer	Tomhuda Katzdale	Godfrey Chan	Alex Matchneer	Selva Ganesh
8º	Matthew Beale	Falk Pauser	Francesco Rodríguez	Nook Orphan	Stanley Stuart	Travis Hoover
9º	Tomhuda Katzdale	Alex Matchneer	Matthew Beale	Richard Lopes	Martin Muñoz	Adam Luikart
10º	Stanley Stuart	Michael Latta	James Rosen	Craig Teegarden	Tom Dale	Rob Monie

Estado-da-arte: Robert Jackson, Peter Wagenet, Charles Jolley, Tomhuda Katzdale, Stefan Penner e Martin Munoz.

Oráculo: Robert Jackson, Peter Wagenet, Tomhuda Katzdale, Stefan Penner, Martin Munoz, Kristofor Selden, Erik Bryn, Alex Matchneer, Matthew Beale, Trek Glowacki e Edward Faulkner.

Observação: Nomes em cor azul são nossos acertos em relação ao oráculo.

O nome em cor vermelha representa um erro do estado-da-arte comparado ao oráculo e às nossas métricas.

cional para composição de um novo conjunto de dados, com a modelagem e o cálculo das métricas. A análise comparativa foi realizada com um pequeno sub-conjunto dos repositórios analisados pelo estado-da-arte, porém houve o cuidado em selecionar repositórios com variação em linguagem de programação, número de contribuidores e TF_O (respeitadas as limitações do conjunto de dados de entrada). Todas essas considerações são importantes e apontam para a continuidade deste estudo.

6. Conclusão

Computar o *Truck Factor* - TF de um sistema é uma tarefa pouco escalável e de alta complexidade. Por exemplo, a maioria das soluções existentes apresentam resultados estimados para pequenas equipes de desenvolvimento (com menos de 40 desenvolvedores) e sem apoio de evidências empíricas. Assim, propostas para estimar o TF são pertinentes para identificar riscos de descontinuidade em projetos de software. Neste artigo, foi descrita uma solução através do ranqueamento de desenvolvedores a partir da agregação de métricas topológicas e semânticas de uma rede de colaboração do GitHub. Realizamos ainda uma análise comparativa preliminar com o estado-da-arte para a validação do nosso modelo. Em resumo, o STF apresentou resultados muito próximos (ou iguais) aos obtidos no estado-da-arte, principalmente para repositórios com TF menor (entre 1 e 4). Além disso, em um repositório específico, os ranqueamentos de desenvolvedores (pelas métricas AA, R_JCSR e R_JCOSR) foram capazes de elencar melhor os desenvolvedores em comparação com o estado-da-arte. Assim, nossa proposta configura-se como uma alternativa viável para estimar o TF, principalmente pela simplicidade de cálculo

ao considerar o uso das métricas pré-processadas pelo GitSED. Como trabalhos futuros, planeja-se considerar repositórios de mais linguagens de programação, calibrar melhor o ϵ , identificar métricas que considerem outras granularidades de relacionamento (número de linhas adicionadas ou removidas nos *commits*, por exemplo), bem como utilizar outros métodos de agregação.

Agradecimentos. Trabalho parcialmente financiado por CAPES, CNPq e FAPEMIG.

Referências

- [Adamic and Adar 2003] Adamic, L. A. and Adar, E. (2003). Friends and neighbors on the web. *Social Networks*, 25(3):211–230.
- [Alves et al. 2016] Alves, G. B. et al. (2016). The strength of social coding collaboration on github. In *SSBD*, pages 247–252.
- [Avelino et al. 2016] Avelino, G., Passos, L. T., Hora, A. C., and Valente, M. T. (2016). A novel approach for estimating truck factors. In *ICPC*, pages 1–10.
- [Batista et al. 2017] Batista, N. A. et al. (2017). Collaboration Strength Metrics and Analyses on GitHub. In *WI*, pages 170–178.
- [Brandão and Moro 2017] Brandão, M. A. and Moro, M. M. (2017). The strength of co-authorship ties through different topological properties. *JBCS*, 23(1):5.
- [Easley and Kleinberg 2010] Easley, D. A. and Kleinberg, J. M. (2010). *Networks, Crowds, and Markets - Reasoning About a Highly Connected World*. Cambridge Un. Press.
- [Emerson 2013] Emerson, P. (2013). The original borda count and partial voting. *Social Choice and Welfare*, 40(2):353–358.
- [Ferreira et al. 2017] Ferreira, M. M. et al. (2017). A comparison of three algorithms for computing truck factors. In *ICPC*, pages 207–217.
- [Ganjisaffar et al. 2011] Ganjisaffar, Y. et al. (2011). Bagging gradient-boosted trees for high precision, low variance ranking models. In *SIGIR*, pages 85–94, Beijing, China.
- [Gousios 2013] Gousios, G. (2013). The GHTorrent Dataset and Tool Suite. In *MSR*, pages 233–236.
- [Hannebauer and Gruhn 2014] Hannebauer, C. and Gruhn, V. (2014). Algorithmic complexity of the truck factor calculation. In *PROFES*, pages 119–133.
- [Jarczyk et al. 2018] Jarczyk, O. et al. (2018). Surgical teams on GitHub: Modeling performance of GitHub project development processes. *Information and Software Technology*, 100:32–46.
- [Li et al. 2017] Li, L. et al. (2017). Predicting software revision outcomes on GitHub using structural holes theory. *Computer Networks*, 114:114–124.
- [Ricca et al. 2011] Ricca, F. et al. (2011). On the difficulty of computing the truck factor. In *PROFES*, pages 337–351.
- [Young 1988] Young, H. P. (1988). Condorcet’s theory of voting. *American Political science review*, 82(4):1231–1244.
- [Zazworka et al. 2010] Zazworka, N. et al. (2010). Are developers complying with the process: an XP study. In *ESEM*, pages 14:1–14:10.