# Análise da Contribuição para Código entre Repositórios do GitHub

**Lais M. A. Rocha, Thiago Henrique P. Silva, Mirella M. Moro**

Universidade Federal de Minas Gerais, Belo Horizonte, MG, Brasil

{laismota,thps,mirella}@dcc.ufmg.br

***Resumo.*** *Considerando uma rede social de desenvolvedores de código e um indicador de contribuição, os objetivos desse artigo são: analisar a contribuição entre repositórios de diferentes linguagens de programação, analisar a correlação entre elas e traçar perfis a partir desse repositório de dados.*

***Abstract.*** *Considering a social network of code developers and a contribution index, the goals of this paper are: to analyze the contribution of repositories of different programming languages, analyze the correlation between languages, and draw interesting profiles from such an interesting data repository.*

## 1. Introdução

A facilidade de extrair dados de várias fontes e conectá-los tem contribuído para o crescimento de Redes Sociais. Uma rede possível é a de desenvolvedores de código a partir de dados de plataformas como o GitHub[1], onde participantes são desenvolvedores que podem criar/contribuir/compartilhar/buscar por repositórios de projetos de acordo com assuntos e Linguagens de Programação (LPs) em que foram desenvolvidos. Uma análise preliminar dos contribuidores por LP em nossa coleta do GitHub (Figura 1) indica que muitos são associados a poucas LPs e pouquíssimos associados a muitas LPs. Além disso, estudos usam tais dados para diversos fins, incluindo medir desempenho ou influência, analisar o desempenho de softwares de código aberto, a importância de programadores e repositórios, todos verificando o comportamento de redes sociais e provendo interessantes visões na Computação [Börner et al 2005, Lima et al 2013, Silva et al. 2015].

Neste contexto de contribuição e colaboração de código, analisamos repositórios do GitHub a partir de Linguagens de Programação pré-selecionadas, onde contribuidores podem ser agrupados como possuidores de habilidades em uma ou mais LPs. Além disso, em muitas aplicações e repositórios, existem mais de uma linguagem operando em diferentes características do projeto (e.g., Python como *back-end* e JavaScript como *front-end*). Então, a transferência de conhecimento entre LPs é oportuna para contribuir com a evolução da qualidade de software e evolução de projetos de código aberto, pois possibilita aplicações de ideias conhecidas e amplamente utilizadas em repositórios de um tipo de LP para resolver problemas de repositórios que utilizam outra LP [Sun et al 2013]. Finalmente, através da metodologia de um indicador de contribuição existente, os objetivos são: analisar a contribuição entre repositórios de diferentes LPs, analisar a correlação entre as linguagens e traçar perfis interessantes a partir desse repositório interessantíssimo de dados. Desse modo, mostramos como a adaptação de um índice de avaliação de *pesquisadores* serve para a avaliação de *contribuidores*, inferindo uma análise qualitativa a partir de dados que individualmente pouco permitem qualquer conclusão profunda.

[1]GitHub, a maior e mais utilizada comunidade de código aberto no mundo, com mais de 15 milhões de usuários e mais de 38 milhões de projetos hospedados: http://github.com

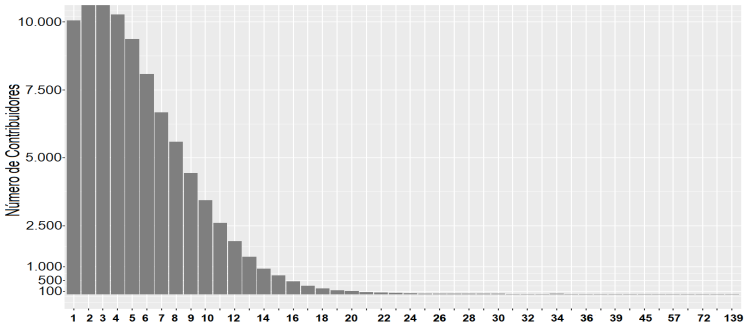



**Figura 1. Distribuição de contribuidores (eixo vertical) por quantidade de linguagens de programação em que programam no GitHub (eixo horizontal), Março de 2016.**

## 2. Trabalhos Relacionados

Neste trabalho focamos na relação social entre os contribuidores e suas comunidades, onde cada comunidade é definida por desenvolvedores que trabalham com uma mesma LP. Estudos que modelam o GitHub como rede social colaborativa (ex. [Thung et al. 2013]) descrevem a estrutura da rede, a formação dos participantes e seus relacionamentos. Já na caracterização da qualidade de código no GitHub, Ray et al. [2014] estudam tipos e usos de LPs e como se relacionam à qualidade de software. Aqui, o objetivo é mais específico, pois pretendemos analisar o GitHub do ponto-de-vista das LPs escolhidas a partir dos repositórios, ou seja, realizando uma análise sobre um *agrupamento* de dados.

Na perspectiva social, um dos pioneiros é [Granovetter 1973] através da ideia de *weak ties*: as relações que unem diferentes partes da rede através da construção de pontes (posteriormente estendida por muitos). Na área de Computação, Silva et al. [2014] usam o conceito de *comunidade* para ranquear veículos de publicação através do grau das relações externas de seus membros. Concluíram que pesquisadores que publicam em várias comunidades possuem maior probabilidade de levar sua competência para outros contextos, sendo então considerados influentes por conectarem partes da rede (i.e, *weak ties*). Nosso trabalho contribui para esta discussão, pois mostramos que podemos ampliar o conceito da construção de pontes ao explorar a relação contribuidor-comunidade.

Um problema de ranquear pesquisadores é a atuação em diversas áreas que não são comparáveis. Aqui, o problema equivalente é: como comparar o desempenho de desenvolvedores que atuam de forma diferenciada utilizando LPs distintas. Para pesquisadores, a solução encontrada por [Lima et al 2013] (ca-index) e [Silva et al. 2015] (3c-index) foi definir comunidades para as áreas e mapear o desempenho dos pesquisadores considerando *uma* comunidade base. Aqui, seguimos tal estratégia de *comunidade base* para contribuidores de repositórios do GitHub. Porém, nosso objetivo é aplicar as etapas da solução não necessariamente para medir a influência de contribuidores de repositórios, mas para analisar as conexões entre diferentes LPs, a contribuição de desenvolvedores entre diferentes repositórios e analisar os perfis de LPs que surgem após tal análise.

## 3. Contribuição entre Comunidades e Metodologia

Contribuidores podem ser agrupados como possuidores de habilidades em uma ou mais LP. Aqui, aplicamos as etapas de desenvolvimento da métrica *3c-index* que identifica a transferência de conhecimento entre diferentes comunidades. Poderia ser outra, mas essa considera a peculiaridade de avaliar pessoas com múltiplas especialidades de maneira mais justa (conforme longamente explicado no seu artigo original [Silva et al. 2015]).

Adaptando para o nosso contexto, a *comunidade base* de um contribuidor é aquela em que ele possui o melhor desempenho. A escolha da comunidade base é definida em termos de percentis, i.e., de acordo com as posições dos contribuidores no ranqueamento. Formalmente, $p_i^c$ é o percentil do contribuidor *i* na comunidade *c* definido por: $p_i^c = \frac{l_i^c + 0.5e_i^c}{N^c}$, onde $N^c$ é o número de contribuidores na comunidade $c$, $l_i^c$ e $e_i^c$ o número dos contribuidores com valores no ranqueamento inferiores ou iguais ao do contribuidor $i$, respectivamente. Por exemplo, para uma comunidade com 100 contribuidores com scores distintos (sem empates), o contribuidor na posição 10 possui $l_i^c = 89$ e $e_i^c = 1$, resultando no percentil de 89.5%. Após o cálculo de $p_i^c$ para todas as comunidades em que o contribuidor desenvolve código em repositórios, defini-se *comunidade base* $b_i$ como sendo aquela onde o contribuidor possui o maior valor para o percentil, i.e., $b_i = \text{argmax}_{c \in C} p_i^c$. A partir da *comunidade base*, mede-se então o grau de influência $inf d_i$ como a diferença entre os percentis (i.e., $inf d_i = b_i - p_i^c$).

Como metodologia, a avaliação está em duas partes: verificamos a existência da troca de contribuição entre comunidades (Seção 4), e confirmamos a nossa abordagem com uma análise da rede de contribuidores, modelando os contribuidores e os repositórios de cada LP como grafos (Seção 5). A realização dessas avaliações envolve três passos.

Primeiro, define-se o conjunto de linguaguens que serão usadas para coletar dados do GitHub. Escolhemos 12 linguagens (Assembly, C, C++, C#, Java, JavaScript, Pascal, Perl, PHP, Python, Ruby e Visual Basic), as quais foram consideradas as de mais alta popularidade[2] e alvo de contribuidores de maior qualidade. Elas também cobrem as principais classes de (1) aplicação: voltadas para a WEB (PHP, JavaScript, C#), front-end (JavaScript), back-end (C, C++, Python, Java, PHP, Perl); e (2) tipos: mais fortemente tipadas (Java, C# and Pascal), fortemente e dinamicamente tipadas (Perl, Ruby, Python), híbridas (Visual Basic) e não tipadas (Assembly).

Segundo, é necessário definir medidas utilizadas no GitHub para *qualificação* de repositórios: *stars* é o número de estrelas que um repositório recebeu de outros usuários e classifica a qualidade e o interesse pelo repositório; e *forks* baseia-se no conceito de cópia/bifurcação de projeto, pois criando um fork produz-se uma cópia pessoal do projeto. A bifurcação de repositórios é o cerne da rede colaborativa de código no GitHub. Consideramos como função de score o percentil de acordo com o ordenamento pelo número de stars e de forks de cada repositório, e somente os membros com mais de um repositório em uma LP formam a comunidade da mesma (uma maneira de polir os dados).

Terceiro, a base de dados foi coletada em Março de 2016 por meio da API do GitHub:[3] (1) para cada LP coletou-se uma amostra de 1000 repositórios ordenados pela melhor combinação (*best match* entre stars e forks) em ordem decrescente; e (2) para cada repositório coletou-se uma lista de contribuidores, de quem coletou-se todos os repositórios contendo cada um a LP em que foram desenvolvidos, número de *stars* e de *forks*. Ao final, são mais de 2 milhões de repositórios, onde para cada desenvolvedor os dados são no estilo (genérico): `desenvolvedor X, duas LPs: {{C, 170 stars, 599 forks}, {Java, 200 stars, 430 forks}}` (*stars* e *forks* representam, individualmente, a quantidade total somada de todos os repositórios desenvolvidos em cada LP).

[2]TIOBE, Março 2016: `http://www.tiobe.com/tiobe_index`

[3]API GitHub: `https://api.github.com`

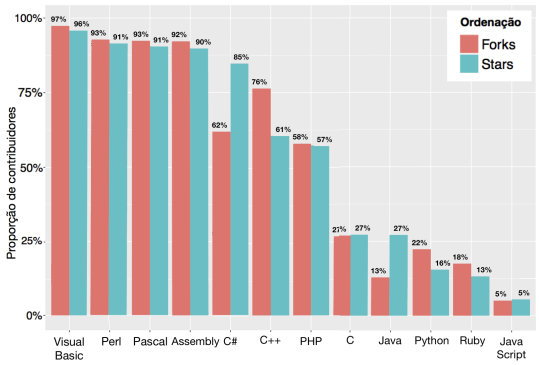



**Figura 2. Porcentagem de contribuidores que possuem a própria LP como sua *comunidade base***

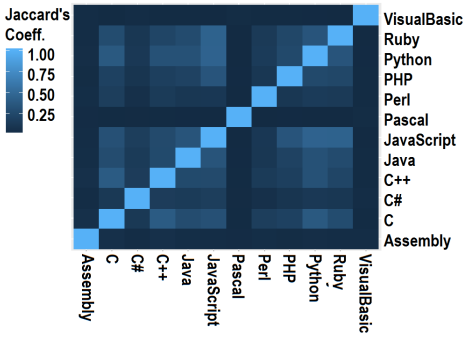


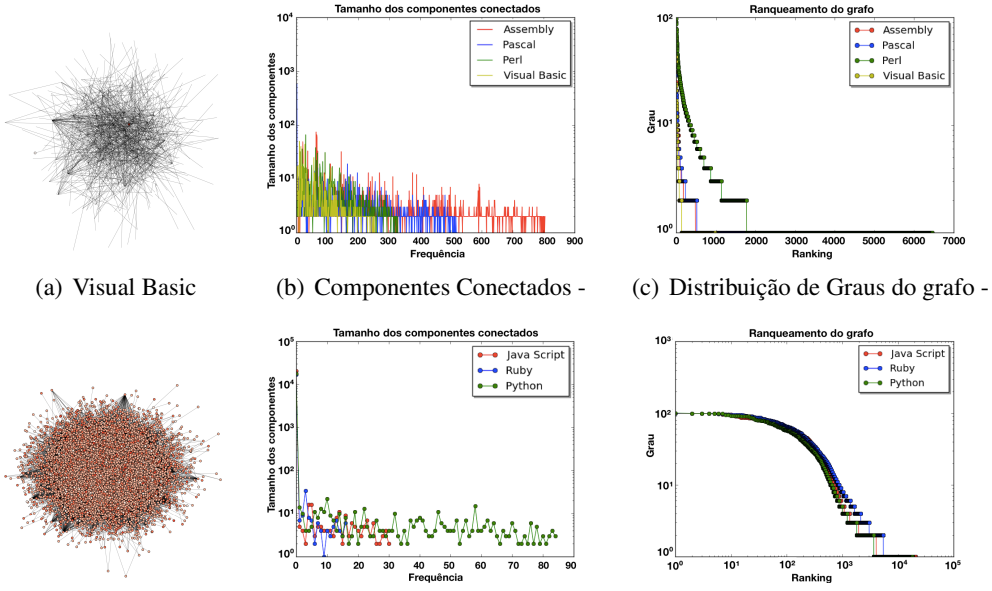

**Figura 3. Distância de Jaccard entre as LPs por meios de uma matriz de dissilaridade**

## 4. Análises de Transferências de Conhecimento e Correlações

Agora investigamos a aplicabilidade da metodologia do 3c-index para o contexto de repositórios de LPs do GitHub. Primeiro, verificamos a dependência das comunidades em relação à formação dos seus membros de acordo com o conceito de *comunidade base*, i.e., estamos interessados em endossar que há evidências suficientes para aplicarmos a métrica em tais contextos. Segundo, verificamos a correlação entre as Linguagens de Programação com objetivo de mostrar a independência ou afinidade entre elas.

**Transferência de Conhecimento.** A questão é: existe transferência de conhecimento entre comunidades? Ou seja, como é composta cada comunidade em relação à participação de membros externos? A Figura 2 mostra a proporção de participantes em cada LP que possui a própria comunidade como base. Por exemplo, contendo mais contribuição externa está a comunidade JavaScript com 5,1% (Forks) e 5,5% (Stars) dos membros que a possuem ela mesma como comunidade base, contabilizando cerca de quase 95% de contribuição externa. Em contrapartida, VisualBasic possui apenas 2,6% (Forks) e 4,2% (Stars) de contribuição externa, indicando que VisualBasic tende a possuir uma comunidade extremamente fechada (de fato, esta LP é produzida pela Microsoft e faz parte do pacote Visual Studio; sua última versão é parte do pacote Visual Studio .NET, focado em aplicações .NET e usado em outras aplicações da Microsoft). Esses resultados reforçam a existência da troca de conhecimento entre comunidades diferentes.

**Correlação e Afinidade entre Comunidades.** Outra questão é: existe correlação e afinidade entre as comunidades comparando-as duas a duas? Aqui, verificamos se há dissimilaridade entre os conjuntos de contribuidores que compõem as comunidades, medida por 1 menos a divisão entre a interseção e união de dois conjuntos (*Jaccard*). Para investigar, a Figura 3 mostra as distâncias de Jaccard entre duas LPs por meio de uma matriz de dissimilaridade, na qual o intervalo de cores de diferentes tons de azul denotam que quanto mais claro o tom de azul apresentado, maior a correlação entre duas LPs. É possível notar: (1) a existência de LPs muito pouco ou quase nada correlacionadas (e.g., Assembly, Pascal, Visual Basic, C#), (2) outras extremamente colaborativas e influenciadas (e.g., JavaScript, Java, Python, Ruby), e (3) a tendência de afinidade e correlação entre LPs de mesmo objetivo e mesmo foco (C e C++, C e Python, C++ e Python, Java e Python, Java e C, Perl e C, PHP e JavaScript, Ruby e Java e JavaScript com quase todas as outras).




(a) Visual Basic (b) Componentes Conectados - (c) Distribuição de Graus do grafo -

(d) JavaScript (e) Componentes Conectados + (f) Distribuição de Graus do grafo +

**Figura 4. Caracterização da rede de contribuidores de repositórios em LPs .**

## 5. Análises das Redes para cada Comunidade

Esta seção apresenta três análises complementares das redes criadas para cada comunidade (LP). A rede social é formada por nós para contribuidores e repositórios, e existe uma aresta entre eles quando o contribuidor colabora no código do repositório (se um contribuidor possui arestas para dois repositórios, ele forma uma ponte entre tais repositórios). Tal modelagem de grafo é feita para cada comunidade (LP). A avaliação considera o comportamento das comunidades por meio de análises de métricas utilizadas no estudo de Redes Sociais, tais como grau do nó, coeficiente de clusterização e tamanho do componente conectado [Easley and Kleinberg 2010]. Aqui, por restrição de espaço, apresentamos uma amostra dos resultados, os quais podem ser conferidos em [Rocha and Moro 2016].

**Análise Visual do Grafo das Redes.** Primeiro, analisamos o grafo das redes formadas entre contribuidores e repositórios – Figura 4(a) e (d). Através da caracterização completa das 12 linguagens (em [Rocha and Moro 2016]), verificamos similaridades de comportamento permitindo a definição de quatro grupos: (1) menos colaborativas - Assembly, Pascal, Visual Basic e Perl, (2) médio colaborativas - C++ e C, (3) muito colaborativas - Java, PHP e C, e (4) mais colaborativas - JavaScript, Ruby e Python. A seguir, continuamos as análises considerando apenas os dois grupos extremos (menos e mais colaborativos).

**Componente Conectado.** O tamanho do grafo de cada LP (com relacionamentos entre repositórios e contribuidores) pode ser calculado com base no componente conectado.[4] O tamanho de um componente é dado pela fração de nós da rede que fazem parte desse componente. A análise desses componentes pode ajudar a entender como a rede foi formada. Foram encontrados componentes conectados de vários tamanhos, conforme ilustrado nas

[4]Um componente conectado de uma rede é um subconjunto de nós em que existe um caminho entre quaisquer dois nós desse subconjunto.

Figura 4(b) para as menos conectadas e Figura 4(e) para as mais conectadas. Mesmo com a variação em torno dos valores, o comportamento global é de *power law*, indicando muitos repositórios com poucos contribuidores e poucos repositórios com muitos. Além disso nota-se uma variação dos tamanhos dos componentes conectados, com o tamanho máximo até 50 em Visual Basic e 25.000 em JavaScript. Para se ter uma ideia das outras linguagens, os tamanhos máximos incluem até 14.000 em C++ e 16.000 em PHP.

**Grau do Nó.** O grau de um nó é obtido calculando-se o número de arestas incidentes sobre ele. As Figura 4(c) e Figura 4(f) ilustram a distribuição dos nós do grafo para cada uma das LPs. É possível observar que repositórios com grau mais alto (i.e., com muitos contribuidores) estão em menor quantidade, enquanto os com grau mais baixo estão em maior quantidade. Novamente, o mesmo comportamento apresentado anteriormente em relação aos componentes conectados aparece para o grau dos nós.

## 6. Conclusão

Apresentamos a aplicação de um índice de influência para mostrar a contribuição de desenvolvedores através de comunidades formadas por LPs específicas. Discutimos também a correlação e a similaridade entre as comunidades. Com as análises, foi possível perceber a presença de perfis de comportamentos em sub-conjuntos das LPs estudadas, algumas LPs com maior afinidade que outras, possibilitando uma caracterização mais aprofundada de tais perfis. Como possíveis trabalhos futuros estão a aplicação de outras métricas de redes sociais nas comunidades avaliadas, bem como a análise das contribuições existentes quando usuários não dão fork nos repositórios e também quando apesar de darem fork em um repositório não contribuem com o projeto.

**Agradecimentos.** Este trabalho foi parcialmente financiado pelo CNPq, Brasil.

## Referências

Börner et al, K. (2005). Studying the Emerging Global Brain: Analyzing and Visualizing the Impact of Co-Authorship Teams. *Complexity*, 10(4):57–67.

Easley, D. and Kleinberg, J. (2010). *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press.

Granovetter, M. S. (1973). The Strength of Weak Ties. *American Journal of Sociology*, 78(6):1360–1380.

Lima et al, H. (2013). Aggregating Productivity Indices for Ranking Researchers Across Multiple Areas. In *JCDL*, pages 97–106.

Ray et al, B. (2014). A large scale study of programming languages and code quality in github. In *SIGSOFT FSE*, pages 155–165.

Rocha, L. M. A. and Moro, M. M. (2016). Contribuição de Código entre Repositórios do GitHub. Technical report, UFMG. Disponível em `http://www.dcc.ufmg.br/~mirella/projs/apoena`.

Silva, T. H. P., Rocha, L. M. A., da Silva, A. P. C., and Moro, M. M. (2015). 3c-index: Research contribution across communities as an influence indicator. *JIDM*, 6(3).

Silva et al, T. H. P. (2014). Community-based Endogamy as an Influence Indicator. In *JCDL*, pages 67–76.

Sun et al, X. (2013). Social dynamics of science. *Sci. Rep.*, 3(1069).

Thung, F., Bissyandé, T. F., Lo, D., and Jiang, L. (2013). Network structure of social coding in github. In *CSMR*, pages 323–326.