The Evolving Nature of Developers' Contributions in Open Source Projects

Talita Santana Orfanó
Universidade Federal de Minas Gerais
tali.orfano@gmail.com

Mariza Andrade S. Bigonha
Universidade Federal de Minas Gerais
mariza@dcc.ufmg.br

Kecia A. Marques Ferreira
Centro Federal de Educação Tecnológica de Minas Gerais
kecia@cefetmg.br

ABSTRACT

Code ownership refers to the knowledge and responsibility a developer has about a given software code. Previous studies have shown a significant relationship between software quality and human factors, reporting that one of the main causes of software quality degradation is the lack of developers' knowledge of the source code. This study aims to understand how the knowledge on the code is distributed among developers throughout the life cycle of open source projects. We carried out five case studies. The results show that a large part of the knowledge is concentrated in a restricted set of authors, called heroes. The results also brings the following new insights on code ownership: the main contributors at the beginning of the project did not remain in this position throughout its evolution; there is an alternation of developers as heroes in the projects; the knowledge on the project becomes more distributed among developers as the project evolves.

CCS CONCEPTS

• **Software and its engineering → Open source model**; Software evolution; *Maintaining software*.

KEYWORDS

code ownership, authorship, open source

ACM Reference Format:
Talita Santana Orfanó, Mariza Andrade S. Bigonha, and Kecia A. Marques Ferreira. 2020. The Evolving Nature of Developers' Contributions in Open Source Projects. In *14th Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS '20), October 19–20, 2020, Natal, Brazil.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3425269.3425284

1 INTRODUÇÃO

Existe uma significativa relação entre fatores humanos e a qualidade do software [13, 19]. Um desses fatores humanos é a propriedade de código, que se refere ao conhecimento e a responsabilidade que um desenvolvedor possui sobre um determinado trecho de código [12]. A falta de domínio dos desenvolvedores sobre o código pode reduzir sua qualidade [13].

Estudos prévios em sistemas *open source* [1, 11, 24] definem dois tipos de desenvolvedores: os chamados *heroes*, que são aqueles poucos que contribuem em uma grande parcela de módulos do projeto por um longo período de tempo, e os desenvolvedores *pequenos* ou *periféricos*, que são aqueles que se envolvem pouco com o projeto, por exemplo, desenvolvendo uma pequena funcionalidade ou resolvendo um *bug* simples. Outras nomenclaturas similares a *heroes* são encontradas na literatura: *core developers*, *major* ou *main developer*. A concentração de conhecimento em poucos desenvolvedores pode representar um risco para o projeto. Essa ideia está representada no conceito de *Truck Factor*, (TF)[1] uma métrica de concentração de conhecimento do software proposta pela comunidade Agile. Essa métrica é dada pelo número de desenvolvedores cuja saída do projeto poderia inviabilizá-lo [10, 29]. Avelino et al. [3] concluíram que a maioria dos sistemas *open source* tem *Truck Factor* menor ou igual a dois, o que resulta em uma forte dependência do projeto em poucas pessoas.

Há poucos trabalhos que estudam como a distribuição do conhecimento dos desenvolvedores evolui no decorrer do ciclo de vida dos projetos. Robles et al. [25] avaliaram a evolução e o comportamento dos principais desenvolvedores em sistemas *open source* hospedados no SourceForge[2] e constataram dois padrões de comportamento entre os desenvolvedores principais do projeto: i) grupo de desenvolvedores principais estável, durante todos os períodos analisados; ii) grupo de desenvolvedores principais que sofria alterações no decorrer dos períodos, isto é, alguns autores deixavam de contribuir e outros tornavam-se mais frequentes. Em ambos cenários, esse grupo era composto por um número pequeno de contribuidores. O conhecimento sobre a evolução da atuação dos desenvolvedores ao longo da existência de projetos *open source* ainda não é amplo. Este trabalho visa contribuir para avançar nesse conhecimento, realizando estudos de caso para analisar detalhadamente como ocorre a evolução da distribuição do conhecimento do software ao longo do ciclo de vida dos projetos. O escopo do trabalho se delimita a softwares *open source* presentes nos repositórios GitHub.[3] Especificamente, este estudo visa responder as questões de pesquisa (QPn):

- **QP1:** O conhecimento do software se difunde ao longo de seu ciclo de vida?
- **QP2:** Os desenvolvedores considerados *heroes* nas primeiras versões permanecem *heroes* durante a evolução do projeto?
- **QP3:** A concentração do conhecimento é impactada pela quantidade total de contribuições do projeto?

SBCARS '20, October 19–20, 2020, Natal, Brazil

ACM ISBN 978-1-4503-8754-5/20/09…$15.00
https://doi.org/10.1145/3425269.3425284

[1] *Truck Factor:* http://www.agileadvice.com/2005/05/15/agilemanagement/truck-factor/
[2] SourceForge: https://sourceforge.net/
[3] GitHub: https://github.com

Para realização deste trabalho foram analisados cinco softwares *open source*, desenvolvidos em JavaScript, Python e Java. Foram construídos *scripts* em Python para coleta de dados das contribuições, a partir da API REST GitHub[4]. Os dados coletados foram armazenados no MongoDB Cloud[5] e em arquivos *CSV*, e analisados quantitativamente, a fim de responder as QPs.

Estudar a distribuição de conhecimento entre os autores possibilita compreender as características intrínsecas dos sistemas *open source* e propor ações de incentivo e auxílio aos colaboradores, tanto aos *heroes*, indispensáveis para o projeto, quanto aos desenvolvedores periféricos, para que estes possam aumentar o domínio do código e a frequência de contribuições e adquirirem um conhecimento maduro do sistema. Os resultados obtidos neste trabalho contribuem para aprofundar o conhecimento que se tem até o momento sobre autoria de software, propriedade de código e a distribuição do conhecimento no processo evolutivo do software, temas vinculados ao comportamento humano e que estão se tornando essenciais na área de manutenção e evolução de software. As contribuições incluem: i) análise da evolução da distribuição do conhecimento do software entre os desenvolvedores no decorrer do ciclo de vida do projeto; ii) disponibilização de *scripts* para coleta de dados de autoria de software a partir do GitHub.; iii) conjunto de dados de cada um dos projeto e os arquivos CSV gerados nas coletas possibilitando a replicação e extensão da pesquisa.

Este artigo está organizado da seguinte forma. Seção 2 apresenta os trabalhos relacionados. Seção 3 detalha o conceito da métrica de propriedade de código. Seção 4 descreve o *data set* e a metodologia adotada. Seção 5 mostra uma análise detalhada dos dados coletados e os resultados da métrica de propriedade de código. A discussão dos resultados é apresentada na Seção 6. Seção 7 discute as ameaças à validade do trabalho e Seção 8 apresenta as conclusões e os trabalhos futuros.

2 TRABALHOS RELACIONADOS

Estudos sobre autoria e propriedade de código têm sido realizados na literatura contemporânea, especialmente pela significativa relação encontrada entre fatores humanos e a qualidade do código desenvolvido [13, 19].

Autoria e propriedade de código: Rahman and Devanbu [23] discutem a relação entre a propriedade de código, as falhas, a autoria e a experiência do desenvolvedor no nível do código fonte. Bird et al. [6] apresentaram o primeiro estudo empírico quantificado sobre os efeitos que a propriedade de código tem sobre sua qualidade, analisando dois grandes softwares comerciais: o Windows Vista e o Windows 7. Seus resultados mostraram que, em componentes com um único e claro proprietário são encontradas menos falhas e, consequentemente, maior é a qualidade do código. Foucault et al. [11] analisou a relação entre as métricas de propriedade definidas por Bird et al. [6] e a propensão a falhas em sete projetos *open source*.[6] Os resultados mostraram uma fraca relação entre as métricas de propriedades e as falhas de módulos, indicando que os resultados encontrados por Bird et al. [6] não podem ser generalizados para projetos *open source*. Baseado na autoria de software e nos dados de interação, Fritz et al. [12] monitoraram 19 desenvolvedores por cinco semanas e comprovaram a tese de que quanto mais frequente e recente for a interação de um desenvolvedor com um elemento particular do código fonte, maior o conhecimento dele em relação aquele elemento.

Meneely and Williams [19] apontam que arquivos que possuem alterações feitas por mais desenvolvedores são mais propensos a ter vulnerabilidade do que os arquivos modificados por menos desenvolvedores, indicando que muitos desenvolvedores alterando o código pode gerar um efeito prejudicial. Greiler et al. [13] também constatou que o número de autores tem relação direta com o número de falhas. Estudos relacionados a *Truck Factor*[7] também aplicam o conceito de autoria e propriedade de código, essa métrica visa detectar como o conhecimento sobre o código está distribuído entre os membros do projeto. A métrica *Truck Factor*(*TF*) indica o número de desenvolvedores que, se forem desligados da equipe, acarretariam sérios problemas para o projeto. Torchiano et al. [29] apresentam um modelo teórico sobre *TF* e investigam o valor máximo alcançável de *TF* em um projeto. Estudos apontam que projetos *open source* não estão preparados para lidar com a rotatividade de desenvolvedores *heroes*, Avelino et al. [3] avaliaram 133 projetos e os resultados indicam que 65% dos sistemas possuem o valor de *TF* menor ou igual a 2.

O presente trabalho também é baseado no conceito de autoria de software. A diferença essencial deste trabalho para os apresentados nesta seção é que ele analisa como o valor da métrica de propriedade de código varia no decorrer do ciclo evolutivo do software, baseado nas definições de Foucault et al. [11] e Bird et al. [6]. Dessa forma, ao considerar intervalos de tempo no cálculo da métrica, torna-se possível avaliar a maneira como a distribuição de conhecimento do software evolui durante o seu ciclo de vida. Assim, poder-se-á afirmar que o conhecimento do software tornou-se mais distribuído quando o valor da métrica nos últimos anos avaliados for menor do que nos primeiros anos de vida desse projeto.

Desenvolvedores *heroes* vs desenvolvedores periféricos: uma característica recorrente entre os projetos *open source* é a concentração do conhecimento em alguns poucos desenvolvedores, chamados *heroes*, e a presença de vários desenvolvedores que realizam poucas e esporádicas contribuições, nomeados por periféricos [8–10, 14, 24]. Ricca and Marchetto [24] buscaram compreender se era comum o surgimento de *heroes* nos projetos e se a presença deles gerava benefícios para o processo de manutenção do software, resultados demonstraram que em 95% dos projetos avaliados existia ao menos um desenvolvedor *hero*. Agrawal et al. [1] apontaram que a presença de *heroes* em projetos médios e grandes é bastante expressiva, tanto nos projetos públicos, quanto naqueles privados. Avelino et al. [2] analisaram 1.932 projetos populares presentes no GitHub com o intuito de compreender como a saída de desenvolvedores *truck factor* impactava na sobrevivência do projeto. Eles concluíram que, dentre os repositórios avaliados, 16% foram abandonados e 41% sobreviveram e que a sobrevivência desses projetos deve-se à atuação de novos desenvolvedores principais. Diversos estudos analisaram [16, 22, 26, 31] o envolvimento e a dificuldade

[4] API REST GitHub: https://developer.github.com/v3/
[5] MongoDB Cloud: https://www.mongodb.com/cloud
[6] FLOSS: https://fsfe.org/freesoftware/comparison.en.html
[7] Truck Factor: https://medium.com/@aserg.ufmg/what-is-the-truck-factor-of-github-projects-bb0d5f019a6f

enfrentada pelos desenvolvedores periféricos. Os resultados sugerem que a maioria desses colaboradores não possuía interesse em tornar-se frequente no projeto. A falta de tempo, a complexidade no processo de envio da contribuição e a complexidade do projeto são as principais barreiras enfrentadas por eles [16].

Robles et al. [25] estudaram a evolução dos desenvolvedores principais (*core developers*) em 19 projetos *open source*. Os contribuidores que efetuaram mais que 10% do total de *commits* do projeto foram classificados como *heroes*. Foram identificados dois padrões dentre os projetos analisados. O primeiro cenário, nomeado de "deuses do código" (*"code gods"*), ocorre quando existe um grupo de desenvolvedores *heroes* altamente estável durante o ciclo de vida do sistema, o que acarretaria um grande risco a sustentabilidade do projeto, caso uma parcela significativa deles saísse. O segundo cenário, chamado "série de gerações", ocorre quando vários grupos de desenvolvedores principais temporários se sucedem no decorrer da vida do software. Avelino et al. [4] analisaram a autoria de código em 66 *releases* do *kernel* do *Linux*. As análises utilizaram como base a métrica *DOA*, definida por Fritz et al. [12] e os resultados indicaram que 75% dos contribuidores do projeto são responsáveis, no máximo, por 10 arquivos, enquanto uma pequena parcela de autores, cerca de 2% do total, é responsável por centenas de arquivos. Posteriormente, o estudo foi replicado em um conjunto de 118 projetos *open source* e apresentou padrões de autoria similares.

Dentre os estudos recentes, há poucos que se dedicam a avaliar a distribuição das contribuições dos *heroes* no decorrer do ciclo de vida do software. Apesar de Robles et al. [25] e Avelino et al. [4] realizarem análises nesse âmbito, tais estudos dedicam-se na compreensão da atuação do grupo de autores, classificados como contribuidores *heroes* analisando de forma global o conjunto de dados de autoria de um grupo de projetos. O presente trabalho realiza estudos de casos de cinco projetos *open source*, com a finalidade de analisar individualmente a atuação de cada *hero* do projeto. Além disso, este trabalho compara a relação da atuação dos *heroes* com o crescimento do número de contribuições totais do projeto e a distribuição do conhecimento. A análise segmentada por períodos de tempo também permite analisar o processo de crescimento e redução do total de contribuições de cada um dos *heroes*. O estudo sobre autoria de software está em discussão na literatura recente e os resultados deste trabalho contribuem com essa discussão, investigando a forma como os *heroes* atuam ao longo da evolução do sistema.

3 PROPRIEDADE DE CÓDIGO

LaToza et al. [15] afirmam que a propriedade do código é uma sensação frequentemente implícita e a prática incidente de correção de *bugs* ou desenvolvimento de novos recursos em um módulo por parte de um mesmo desenvolvedor ou equipe. Rahman and Devanbu [23] definem como autoria o valor resultante da razão entre o número de linhas alteradas para corrigir um *bug* e o número de linhas com as quais o autor contribuiu para corrigir o *bug*. Esse valor resultante equivale à atuação efetiva do autor na correção do *bug*. Assim, para os autores, o termo propriedade de código se refere à autoria do desenvolvedor que mais contribuiu com o fragmento de código analisado. Greiler et al. [13] definem propriedade de código como o número de contribuições totais para o código fonte em relação à proporção de suas contribuições. Assim, esses autores sugerem que o desenvolvedor responsável pelo maior número de mudanças, é considerado o proprietário daquele fragmento. Fritz et al. [12] apresentam um modelo conhecido como DOK (*degree-of-knowledge*), que mede o grau de conhecimento do desenvolvedor acerca do código, baseado na autoria de software e nos dados de interação dos desenvolvedores com o software. O modelo calcula o grau de conhecimento do código de cada desenvolvedor individualmente, utilizando informações tanto da autoria de software quanto de uma simples interação com o código fonte, isto é, consultas realizadas ao código durante o trabalho.

Para Bird et al. [6], o termo propriedade de código é usado para descrever a responsabilidade que se tem por um componente de software ou, até mesmo, para indicar que não se tem um claro responsável pelo componente. Eles definem a métrica de propriedade de código de duas formas. A primeira delas define a propriedade pela quantidade de atividades de desenvolvimento de um componente feitas por um determinado desenvolvedor. Outra maneira é a análise da existência de muitos desenvolvedores de baixa especialização trabalhando em um componente. Assim, se muitos desenvolvedores estão realizando algumas poucas mudanças em um componente, isso indica que há muitos não-especialistas trabalhando no componente e, por isso, esse componente possui baixa propriedade. Para Foucault et al. [11], em sistemas *open source*, existem dois tipos de desenvolvedores: os chamados *"heroes"*, aqueles poucos que contribuem em todos os módulos do projeto por um longo período de tempo; e aqueles que se envolvem pouco com o projeto, desenvolvendo uma pequena funcionalidade ou resolvendo um erro simples, chamados de desenvolvedores pequenos ou periféricos.

Neste trabalho, foram adotados os conceitos definidos por Bird et al. [6] e posteriormente utilizados por Foucault et al. [11], uma vez que esses estudos detalham e explicam de forma abrangente a definição dos conceitos e são referências consolidadas em trabalhos prévios. Adaptando as notações desenvolvidas por Foucault et al. [11] para o contexto do presente estudo, define-se que:

- ω (p) é a soma de todas contribuições dos desenvolvedores realizadas em um projeto p;
- ω (p,d) é a soma de todas contribuições realizadas por um desenvolvedor d em um projeto p.

Assim, a propriedade de código mede a razão das contribuições feitas por um desenvolvedor pelo total das contribuições do projeto. Tem-se que o valor da métrica de propriedade de código por desenvolvedor é calculada por:

$$own\_p, d = \frac{\omega(p,d)}{\omega(p)}$$

Bird et al. [6] salienta que o valor mais elevado dessa razão representa a propriedade de código do projeto. E o desenvolvedor responsável por essa contribuição é considerado o proprietário.

A propriedade de código no período de tempo t mede a razão das contribuições feitas por um desenvolvedor pelo total de contribuições dos desenvolvedores no período. Em outras palavras, para um determinado intervalo de tempo t e para cada desenvolvedor d, a métrica de propriedade no tempo é dada por:

$$own\_t, d = \frac{\omega(t,d)}{\omega(t)}$$

O valor mais elevado dessa razão em um espaço de tempo t é o valor da métrica de propriedade de código do projeto naquele intervalo. O desenvolvedor responsável por essa contribuição é considerado o proprietário no período. Desse modo, tem-se que o proprietário é dado por:

$$max(\{own\_t, d | d \in D\})$$

A partir das definições apresentadas sobre propriedade de código, foram adotados neste trabalho os conceitos definidos por Bird et al. [6] e por Foucault et al. [11]. A motivação para essas escolhas justifica-se pelo fato de que esses estudos detalham e explicam de forma mais abrangente a definição dos conceitos de autoria e propriedade de código, além de serem utilizados como principal referência em outros estudos [7, 13, 18, 20, 27, 28, 30].

4 METODOLOGIA

Esta seção apresenta a metodologia usada nesse trabalho.

Coleta de Dados. A quantidade de dados envolvidos ao longo da evolução de um sistema *open source* de grande porte é muito alta. Por essa razão, o escopo deste trabalho delimita-se a estudar cinco projetos *open source* presentes no GitHub. Para seleção foi considerada a relevância dos projetos, dada pela quantidade de estrelas e pela maturidade do projeto, que avaliamos de acordo com o total de autores e *commits*, bem como o número de *releases* publicadas e *forks* efetuados no projeto. Os projetos que compõem o *data set* são: *Bootstrap*, *Django*, *Elasticsearch*, *Pandas* e *Spring Boot*. Eles foram selecionados dentre as três linguagens de programação mais populares no GitHub: JavaScript, Python e Java[8].

Uma vez que os projetos escolhidos estão presentes no GitHub, fez-se necessária a definição de quais dados seriam coletados e qual ferramenta seria adotada para realizar a coleta dos dados de autoria de software. A propriedade de código e autoria podem ser medidas utilizando diferentes granularidades: *commits*, módulos, arquivos ou linhas de código. Ainda não há consenso na literatura sobre a melhor maneira de avaliar autoria e contribuição, especialmente no que se refere a projetos *open source*. Contudo, observando os trabalhos relacionados presentes na literatura, bem como o custo para processar, armazenar e analisar a grande gama de informações geradas por coletas de granularidade menor, optou-se neste estudo, por avaliar o conhecimento de software no nível de contribuições, ou seja, a partir do total de *commits* que cada autor efetuou no projeto. A API REST GitHub[9] foi escolhida para coleta dos dados, devido sua facilidade de utilização, a ampla documentação e a completude das informações coletadas diretamente dos repositórios originais. Dessa forma, o processo de coleta de dados foi realizado por uma aplicação de autoria própria desenvolvida em Python e os dados coletados foram armazenados no MongoDB Cloud[10] e em arquivos do tipo *CSV* para consultas posteriores. O procedimento obteve os seguintes dados dos cinco repositórios em estudo: código identificador do *commit (sha)*; informações essenciais do autor (nome, e-mail e login); data do *commit* e nome do repositório no GitHub. Todos os *commits* recuperados são da *branch default* (*master*) de cada projeto e abrange desde o primeiro *commit* até a data na qual as coletas foram efetuadas.[11]

Janela de Tempo. Uma vez que este trabalho propõe a análise da evolução das contribuições em projetos *open source* durante o ciclo de vida do software, fez-se necessário obter um padrão de comparação em comum entre os diferentes projetos analisados. Para isso, estabeleceu-se que a janela evolutiva deveria ser medida em função de um intervalo *Tn* anual. Por exemplo, seja um projeto *A* que se iniciou em Julho/2009 e um projeto *B* iniciado em Abril/2012. O *T1* do projeto *A* ocorre durante o ano de 2009 e do projeto *B* no ano de 2012; quando ocorre a mudança de ano, ou seja, Jan/2010, o projeto *A* inicia a janela *T2*. O projeto *B* também inicia a janela T2 em Jan/2013. A última janela de tempo considerada em todos os projetos coincide com o ano de 2018, uma vez que as coletas foram finalizadas no primeiro semestre de 2019. O intervalo T é utilizado nas próximas seções para tratamento e análise dos dados.

Tratamento dos Dados. Descrevem-se, aqui, as informações geradas a partir dos dados inseridos no banco MongoDB. Em cada repositório, foram sumarizados: (i) o total de contribuições por autor; (ii) o total de contribuições por autor em uma dada janela de tempo T.

O primeiro tratamento, (i), visa compreender como se dá a distribuição de conhecimento do software entre os desenvolvedores. Considerando que esse conhecimento é medido pelo número de *commits* realizados, é primordial identificar quais autores mais contribuíram no repositório desde sua criação, da mesma forma é fundamental entender como é distribuído o total de *commits* entre os contribuidores daquele projeto. Como resultado desse procedimento, foi concebido um arquivo CSV para cada um dos repositórios do *data set*, contendo os dados do autor (nome e e-mail) e o total de contribuições efetuadas por ele.

O segundo tratamento, (ii), tem como objetivo compreender como ocorre a distribuição de conhecimento entre os autores durante o ciclo de vida do software. Portanto, não basta apenas conhecer a quantidade de contribuições de cada autor, mas também é importante compreender como essas contribuições cresceram ou diminuíram no decorrer do ciclo de vida do projeto. Também interessa saber como foi o processo de evolução daqueles que são considerados *heroes* do projeto. Assim, a segunda etapa da coleta de informações visa agrupar o total de *commits* feitos por cada autor segmentado pela janela de tempo T. O resultado desse programa é a geração de um arquivo CSV para cada intervalo T analisado.

Para diferenciar as três categorias, foram aplicados os seguintes critérios: os autores *heroes* são aqueles que contribuíram mais que 5% do total dos *commits* existentes no projeto, conforme sugerido por Foucault et al. [11]; os autores médios são aqueles cuja soma de contribuição é maior que 1% e menor ou igual a 5% do total; e os periféricos são os que efetivaram menos que 1% da soma total de contribuições do projeto. Esse limares foram definidos com base na análise visual da distribuição de valores de propriedade de código dos desenvolvedores. Observou-se que tal distribuição segue uma cauda pesada, ou seja, poucos desenvolvedores têm uma participação muito grande no projeto enquanto muitos têm pouca participação. Essa análise é descrita em detalhes na Seção 5A.

[8] https://octoverse.github.com/
[9] GitHub API: https://developer.github.com/v3
[10] MongoDB Cloud Database Solution: https://www.mongodb.com/cloud
[11] As coletas foram realizadas no primeiro semestre de 2019.

Figura 1: Total de contribuições efetuadas por autor - Eixo horizontal corresponde ao total de contribuições realizadas e o eixo vertical é a porcentagem de desenvolvedores que efetuou essa quantidade de contribuições

Desambiguação de Autores. Em projetos *open source* é frequente a existência de contribuidores ambíguos, isto é, usuários que no mesmo projeto utilizam nomes ou e-mails diferentes e são identificados erroneamente como sendo duas ou mais pessoas distintas. Essas particularidades, se não tratadas, podem impactar os resultados dos estudos. Assim, inicialmente a ambiguidade foi tratada utilizando a heurística de similaridade definida por Bird et al. [5] e reproduzida por Orfanó et al. [21]. Contudo, ao avaliar manualmente os autores unificados, encontrou-se problemas consideráveis nos resultados. Dessa forma, para garantir maior precisão e confiabilidade, realizou-se a desambiguação manual. Devido ao grande número de autores e ao custo do procedimento manual, optou-se por tratar a ambiguidade dos autores que trariam grande impacto nos resultados: autores *heroes*, médios e pequenos que possuíam grande número de contribuições.

5 RESULTADOS

Essa seção apresenta os resultados do estudo.

A. Distribuição da população de autores. A Figura 1 mostra como o número de contribuições está distribuído entre os autores. O eixo horizontal, que está em escala logarítmica, corresponde ao total de *commits* que um autor fez em todo histórico do projeto analisado. O eixo vertical corresponde à porcentagem de desenvolvedores existentes no projeto que efetuou aquela quantidade de contribuições. Observa-se que há grande concentração de autores que realizaram poucas contribuições no projeto. Os resultados dos cinco repositórios apontam que aproximadamente 65% dos autores realizaram somente uma contribuição no projeto e 15% efetuaram apenas duas contribuições. Portanto, pode-se considerar que ao menos 80% dos autores vinculados a um desses projetos possuem autoria de uma quantidade muito pequena sobre o código daquele software. Esse comportamento foi homogêneo em todos os projetos do *data set*.

Também é possível observar que a distribuição da população dos autores possui uma cauda longa, uma vez que o número de autores diminui exponencialmente e torna-se mais esparso à medida que o número de contribuições no repositório cresce. Os resultados evidenciam que são poucos os desenvolvedores que efetivaram mais de mil contribuições em seus respectivos projetos. Assim, esta análise indica que nos cinco repositórios *open source* estudados o conhecimento é pouco distribuído e que não se difunde dentro do projeto. Essa considerável discrepância que contrasta entre os muitos autores que efetuaram poucos *commits* e os poucos autores que submeteram muitos *commits* reforça a necessidade de estudá-los de forma separada, considerando suas especificidades.

B. Caracterização da população de autores. Os autores pequenos ou periféricos são aqueles que possuem um conhecimento pequeno ou insuficiente em relação ao código do projeto, devido ao número limitado de contribuições. Os autores médios são aqueles que realizaram uma notável contribuição, podendo-se enquadrar nessa categoria, por exemplo, os que permaneceram no projeto em um razoável espaço de tempo e detiveram um significativo conhecimento do código nesse período, mas abandonaram o projeto antes de tornarem-se *heroes*; uma outra possibilidade são aqueles desenvolvedores que ainda estão ativos no projeto e o número de suas contribuições está aumentando a cada iteração, indicando que em breve tornar-se-ão autores *heroes*. Considera-se como *heroes* aqueles desenvolvedores que contribuíram amplamente no projeto durante seu ciclo de vida, possuíram ou ainda possuem uma grande parcela de conhecimento do sistema.

A Tabela 1 apresenta a segmentação das categorias de autores nos cinco projetos analisados. Pode-se constatar um grande número de autores periféricos que efetuaram uma quantidade pequena de *commits* e um pequeno número de autores *heroes* que concentram juntos uma grande parcela das contribuições do projeto. A maior concentração de conhecimento é observada no *Spring Boot*, no qual 83,3% dos *commits* estão concentrados em apenas 0,59% dos autores. No *Elasticsearch* a presença de 21 autores médios sugerem que ele foi o projeto com as contribuições mais balanceadas entre as três categorias, contudo, ao considerar o total de autores *heroes* e médios, constata-se que apenas 2,21% do total de desenvolvedores concentram mais de 75% do conhecimento daquele projeto. O resultado encontrado indica que há uma concentração expressiva do conhecimento do código, apontando a necessidade de caracterizar e estudar com mais detalhes os autores *heroes* e suas contribuições.

C. Distribuição dos *heroes* no ciclo evolutivo. Esta análise busca compreender como os grandes autores se comportam em relação à quantidade de suas contribuições. A Figura 2 exibe a evolução das contribuições de cada um dos *heroes*. O eixo horizontal apresenta o ano de contribuição, variando do primeiro ao último ano analisado em cada projeto. O eixo vertical corresponde à quantidade total de contribuições feitas pelo desenvolvedor naquele tempo *T*.

Na maioria dos casos, os autores não são *heroes* do projeto de modo simultâneo, mas que em cada período de tempo um ou dois deles se destacam. Na Figura 2, a distribuição de autores do *Django*

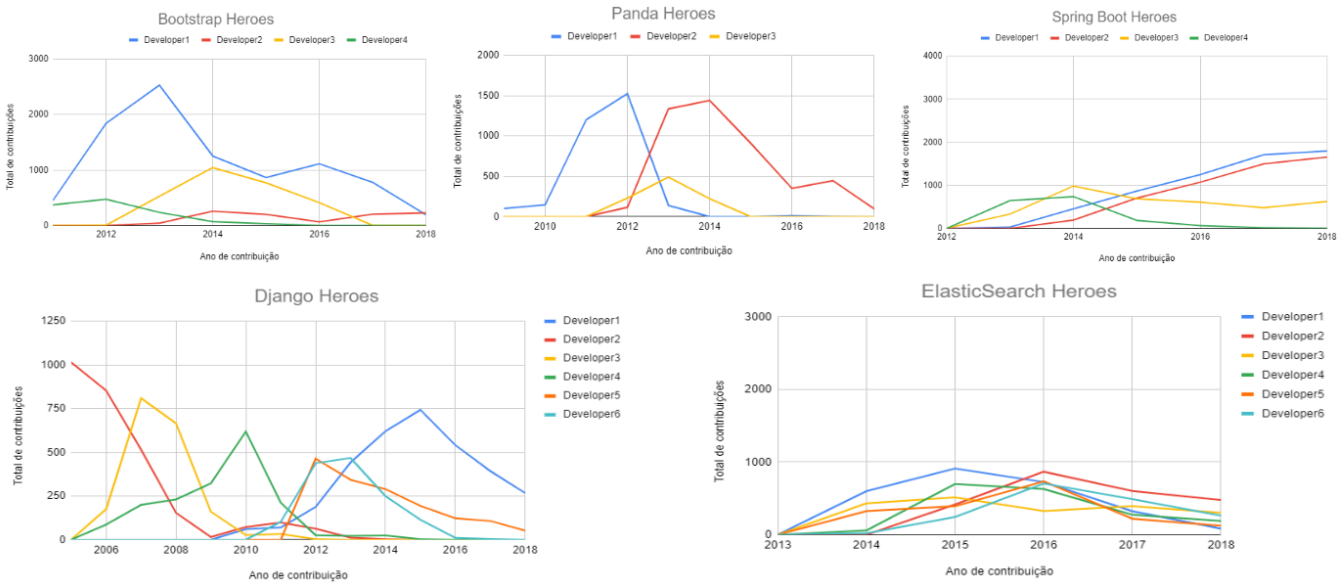

Figura 2: Contribuições dos autores *heroes* durante o ciclo de vida - Eixo horizontal corresponde ao ano de contribuição e o eixo vertical à quantidade total de contribuições feitas pelo desenvolvedor naquele intervalo de tempo

Tabela 1: Categorização de autores e como as contribuições totais do projeto estão distribuídas entre eles

Projeto	Categoria	∑ autores	∑ *commits*
Bootstrap	Autor periférico	1.278	3.579
	Autor médio	3	811
	Autor *hero*	4	13.969
Django	Autor periférico	2.065	7.869
	Autor médio	13	5.928
	Autor *hero*	6	12.694
Elasticsearch	Autor periférico	1.196	8.963
	Autor médio	21	14.870
	Autor *hero*	6	12.352
Pandas	Autor periférico	1.690	5.624
	Autor médio	9	4.187
	Autor *hero*	3	8.770
Spring Boot	Autor periférico	664	1.970
	Autor médio	3	1.365
	Autor *hero*	4	16.618

torna mais evidente essa realidade, pois os seis autores alternam no protagonismo de contribuições. Os projetos *Bootstrap* e *Pandas* seguem um padrão similar ao observado no *Django*. A distribuição de autores *Spring Boot* mostra que no período inicial dois desenvolvedores dividiram o protagonismo do projeto e nos últimos anos eles reduziram suas contribuições. Concomitantemente a isso, outros dois desenvolvedores tornaram-se os principais *heroes* do projeto nos anos finais da análise. Esse comportamento reforça o resultado observado em relação à intercalação da responsabilidade principal do projeto. Ao analisar a atuação dos grandes autores do *Elasticsearch*, percebe-se que o comportamento foi distinto dos anteriores. Os seis autores tiveram a curva de crescimento, seus picos de contribuições e redução de modo quase concomitante. Esse fato revela que os *heroes* somaram esforços conjuntamente para o crescimento do projeto, especialmente entre os anos de 2015 a 2017. Como o número de contribuições teve um aumento significativo nesse período, acredita-se que houve algum incentivo externo para promover o aumento da colaboração no referido projeto. Esta análise possibilitou compreender como se dá a distribuição das contribuições dos *heroes* durante o ciclo de vida do software. Os resultados indicam que há uma alternância do principal proprietário na maioria dos casos e que o tempo gasto para um autor pequeno se tornar um *hero* é, em geral, de um a dois anos.

D. Propriedade de código no ciclo evolutivo do software. Ao observar o comportamento dos autores *heroes* apresentado anteriormente, nota-se que existe uma alternância entre eles em relação ao protagonismo das contribuições em cada período, com exceção do projeto *Elasticsearch*. Isso sugere duas hipóteses: (i) a propriedade de código permanece durante todos os períodos do projeto com elevados valores, sempre vinculados ao *hero* dominante naquele período; (ii) o aumento e a diminuição no valor da métrica acompanha o crescimento e a queda das contribuições dos *heroes*, seguindo o comportamento dos gráficos da Seção 5C.

As contribuições de cada desenvolvedor foram agrupadas em um período de tempo T correspondente e a propriedade de código do projeto em T é o valor de $\max(own_{t,d})$. O proprietário é aquele que efetuou essas contribuições. Quanto maior for esse valor, mais o conhecimento do sistema naquele período ficou concentrado. É importante dizer que essa análise considera todos os desenvolvedores do período, não apenas os *heroes*. A Figura 3 mostra como o valor da métrica oscila durante o ciclo de vida dos projetos. É possível observar uma semelhança entre o comportamento nos sistemas *Django*, *Pandas* e *Spring Boot*, apresentados nas Figura 3. Nesses

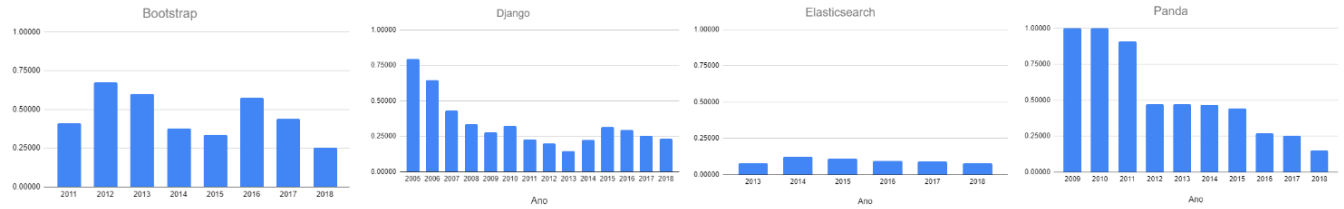

Figura 3: Propriedade de código no decorrer do ciclo de vida - Eixo horizontal corresponde ao período analisado e o eixo vertical ao valor da max($own_{t,d}$)

três projetos, a propriedade de código na primeira versão analisada é muito alta. Contudo, no decorrer do ciclo evolutivo do software, a propriedade de código reduziu gradativamente atingindo a marca de 0,3 no *Spring Boot* e 0,15 no *Pandas* e no *Django*. O resultado mostra que os três sistemas tem o conhecimento altamente concentrado em um desenvolvedor, porém, no decorrer da evolução do projeto o conhecimento foi se tornando mais distribuído. No caso do *Elasticsearch*, o comportamento da métrica apresentou, desde o primeiro intervalo, um valor menor que 0,15 e permaneceu assim até o último intervalo T avaliado. Esse resultado mostra um sistema que em todos as etapas de seu ciclo de vida houve distribuição regular entre os autores. Por fim, o resultado do *Bootstrap* foi distinto dos outros quatro sistemas, pois apresentou comportamento variável no decorrer dos anos. Observa-se, no entanto, que o maior valor da métrica de propriedade é de 0,67 ocorrido no segundo período do ciclo evolutivo, enquanto o menor valor encontrado no último período analisado apresenta o total de 0,25. O comportamento desse gráfico assemelha-se a uma onda, na qual há momentos de pico e outros de leve queda.

Pode-se concluir a partir dos resultados apresentados que a propriedade de código não permanece com elevados valores durante todos os períodos do projeto em nenhum dos sistemas analisados. O comportamento é, ao contrário disso, a medida que aumentava o tempo de vida do projeto, a propriedade de código do período tende a diminuir. A segunda hipótese também não pode ser considerada verdadeira, pois, na maioria dos casos, o aumento e a diminuição no valor da métrica não acompanha os resultados encontrados nos gráficos das contribuições dos *heroes*. O estudo realizado permitiu avaliar o comportamento da métrica de propriedade de código, bem como a forma como ela se desenvolve durante os períodos do ciclo de vida dos projetos *open source* analisados. Constatou-se que os sistemas tendem a iniciar com o conhecimento mais concentrado, porém, no decorrer da evolução do software, existe uma forte tendência do conhecimento se disseminar e tornar mais distribuído.

E. Frequência de Contribuição no Ciclo de Vida do Software. A Figura 4 retrata a variação no total de *commits* efetuados nos repositórios, durante seu ciclo evolutivo. O eixo horizontal mostra o ano da contribuição, enquanto o eixo vertical corresponde à quantidade total de contribuições feitas no projeto naquele período de tempo. Os projetos *Bootstrap*, *Django*, *Elasticsearch* e Panda apresentaram um comportamento similar. Nesses projetos, o pico de contribuições coincidiu com o pico de contribuição dos autores *heroes*. Além disso, o primeiro e último período analisados possuem uma taxa de contribuições reduzida. Por outro lado, o projeto *Spring Boot* está em constante crescimento desde sua criação no ano de 2012. O pico de contribuições ocorreu no último período avaliado. Esse estudo possibilita a compreensão da forma de atuação temporal dos desenvolvedores nos cinco projetos. Assim, é possível identificar em quais períodos de tempo T houve uma maior dedicação no repositório ou uma redução no investimento, bem como identificar que ocorreu uma considerável relação entre a variação da atuação dos desenvolvedores do projeto e dos *heroes*.

F. Relação entre Concentração de Conhecimento e Quantidade de Contribuições. Ao longo da evolução do software, a quantidade de contribuições varia e a concentração de conhecimento pode variar também. Esta análise visa identificar se existe correlação entre a quantidade total de contribuições realizadas em um período e a propriedade de código. Observou-se que os dados de propriedade de código e total de contribuições não seguem uma distribuição normal e a amostra é pequena (o tamanho da amostra é menor do que 30). Por isso, para calcular o coeficiente foi utilizada a correlação de *Spearman*.[12] Sabe-se que um coeficiente de correlação de *Spearman* maior que 0,9 (positivo ou negativo) indica uma correlação muito forte; entre 0,7 a 0,9 indica uma correlação forte; entre 0,5 a 0,7 uma correlação moderada; entre 0,3 a 0,5 uma correlação fraca e de 0 a 0,3 resulta em uma correlação desprezível. O valor do coeficiente varia entre -1 (correlação inversa) e +1 (correlação direta).[13] Os resultados da correlação de *Spearman* para os projetos *Bootstrap*, *Django*, *Elasticsearch*, *Pandas* e *Spring Boot*, foram respectivamente, 0,5;-0,376; 0,257; -0,339; -0,571. É possível observar que não há uniformidade entre os coeficientes calculados. Portanto, constata-se que para os projetos analisados não existe uma correlação entre o valor da propriedade de código e a variação da quantidade de contribuições efetuadas no repositório.

Contudo, os resultados da Seção 5E mostram indícios de que há uma correlação entre o aumento de contribuições nos repositórios e o crescimento da atuação dos *heroes*. Por isso, a partir desses dois fatores o coeficiente de correlação de *Spearman* foi calculado. Para cada intervalo de tempo T, foi considerada a quantidade total de contribuições e o valor máximo dentre as contribuições dos *heroes*, isto é, a quantidade de *commits* efetuada pelo proprietário do período. Os valores comprovam que há uma forte correlação entre o crescimento das contribuições no projeto e o aumento das contribuições dos *heroes*: 0,952 (*Bootstrap*); 0,841 (*Elasticsearch*); 0,672 (*Pandas*) e 0,964 (*Spring Boot*). Apenas o sistema *Django* manifesta

[12] Spearman: http://www.leg.ufpr.br/~silvia/CE701/node80.html
[13] Correlation coefficient: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3576830

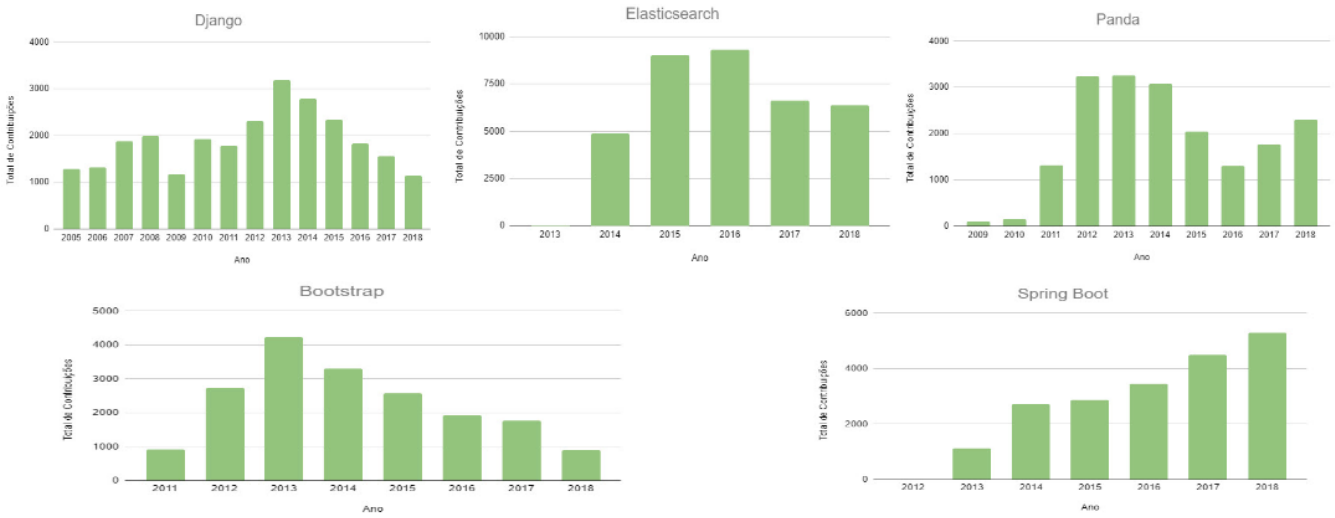

Figura 4: Total de contribuições no decorrer do ciclo de vida - Eixo horizontal corresponde ao período analisado e o eixo vertical ao número de *commits*

uma correlação fraca (0,2), possivelmente ocasionada pela troca constante de proprietários *heroes* durante seu ciclo evolutivo. Os resultados encontrados mostram correlação direta forte entre as contribuições do proprietário do período e o total de contribuições do período nos projetos.

6 DISCUSSÃO DOS RESULTADOS

Esta seção discute os resultados do estudo, respondendo as questões de pesquisa.

QP1: O conhecimento do software se difunde ao longo de seu ciclo de vida?

O intuito dessa questão de pesquisa é compreender se o conhecimento do software permanece desbalanceado entre os contribuidores, como sugerido por Foucault et al. [11], ou se, em uma determinada etapa da maturidade do software, as contribuições tornam-se uniformes. Os resultados da Seção 5 sugerem que o conhecimento do software não se difunde ao longo do seu ciclo de vida, uma vez que, em média, 80% dos autores efetuaram no máximo duas contribuições nos projetos analisados. Assim, pode-se considerar que o conhecimento permanece restrito a uma pequena parcela de autores, nomeados *heroes*. Esses dados estão em consonância com outros trabalhos que se dedicaram a estudar projetos *open source* [1, 3, 8, 17, 22, 24, 25].

Contudo, os resultados encontrados no estudo da propriedade de código na Seção 5D, mostram que existe uma tendência de disseminação do conhecimento a medida que o software evolui. Essa constatação só é possível quando se analisa o valor da métrica de propriedade de código, de forma periódica. Ao comparar os primeiros e o últimos períodos de tempo T dos cinco projetos, tem-se que a métrica de propriedade é sempre menor nos últimos períodos, com exceção do projeto *Elasticsearch*, que teve baixos valores para a métrica no decorrer de todos os anos, mas que, apesar disso, também atingiu o menor valor no último período analisado. Assim, os resultados sugerem que o conhecimento torna-se mais distribuído entre os desenvolvedores no decorrer de seu processo evolutivo. Isso pode facilitar a entrada de novos contribuidores e possibilitar que eles venham a se tornar contribuidores centrais para o projeto. Conforme apontam Avelino et al. [2], a sobrevivência de uma quantidade relevante de projetos *open source* (41%) ocorre devido a ajuda ativa de novos desenvolvedores.

QP2: Os desenvolvedores considerados *heroes* nas primeiras versões do projeto permanecem *heroes* até as suas últimas versões?

O objetivo desta questão de pesquisa é estudar e caracterizar o comportamento dos desenvolvedores responsáveis pela maior parte das contribuições efetuadas nos projetos, os chamados *heroes*. Procura-se entender se os principais contribuidores no início do projeto permaneceram contribuindo até os períodos analisados. Se o pressuposto for positivo, deseja-se compreender se essas contribuições permanecem numerosas ou se houve uma diminuição em relação ao número de *commits* desses autores. Caso contrário, pretende-se entender em quais condições outros contribuidores tornam-se *heroes* e como os antigos diminuíram suas contribuições, se foi gradativamente ou se houve uma queda súbita no número de contribuições indicando um abandono do projeto.

A Seção 5 descreveu a análise do comportamento dos autores *heroes* durante o ciclo de vida do software. Resultados apontaram que, apesar de permanecerem como grandes contribuidores do projeto pelo total de contribuições, ao comparar as contribuições do primeiro e do último período de tempo, observa-se que os *heroes* iniciais já não estão presentes entre os *heroes* finais. O autor que mais contribuiu no primeiro período apresentou nenhuma ou uma quantidade pequena de contribuições no último período analisado. Esse comportamento foi unânime nos cinco projetos. O estudo também permite concluir que os *heroes* de um projeto *open source*,

em geral, não ocupam essa posição de modo simultâneo. Em cada período de tempo T um ou dois deles concentram as contribuições do projeto. Além disso, a transformação de um novo autor, inicialmente considerado periférico até tornar-se um dos *heroes*, ocorre de forma gradual e leva em média dois anos. Os resultados mostram que, na maioria dos casos, houve alternância entre os *heroes*. Esse cenário no qual grupos de desenvolvedores principais temporários se sucedem no decorrer do ciclo de vida do software é denominado "série de gerações"(*"series of generations"*) [25]. Contudo, é importante a realização de outros estudos como o apresentado para se obter maior aprofundamento do assunto e compreender se é um comportamento padrão entre os projetos *open source*.

A análise por períodos foi importante para compreender o comportamento dos projetos e de seus contribuidores. Os *heroes* do repositório, como categorizado na Seção 5, apenas são considerados como tal devido à sua expressividade no total de contribuições efetuadas. Contudo, apesar de ser considerado *hero*, ele não possui domínio e conhecimento daquele software no momento corrente. Um exemplo desse fato ocorre com um dos autores principais do *Django*: ele iniciou o projeto em 2005, mas após 2008 teve uma grande queda na atuação do projeto e, no ano de 2014, ele encerrou suas contribuições no desenvolvimento. Dessa forma, apesar de existirem oito *heroes* no *Django*, ao considerar o último período do projeto, apenas dois deles detêm o conhecimento e o domínio do software no momento.

Os resultados indicam que em um projeto *open source*, os desenvolvedores *heroes* não abandonam ou reduzem a contribuição repentinamente no projeto; ele permanece por algum tempo colaborando e dando suporte ao sistema. Essa diminuição gradual é uma diferença essencial em relação a projetos comerciais de código fechado, uma vez que ao ser desligado da empresa, as contribuições do desenvolvedor diminuem de forma abrupta, fato que pode dificultar a difusão do conhecimento do software que estava concentrado nesse desenvolvedor. O surgimento de *heroes* no decorrer da evolução do software observada em todos os casos também reforça o resultado encontrado na QP1. Esses achados ressaltam a importância da gestão de projetos *open source* e sua atuação, de modo a identificar os desenvolvedores *heroes* e incentivá-los a permanecer no projeto. A criação de mecanismos de disseminação do conhecimento, que esses autores detêm, também é importante, para que o tempo de aprendizado dos futuros *heroes* seja otimizado.

QP3: A concentração do conhecimento é impactada pela quantidade total de contribuições do projeto?

O intuito dessa questão de pesquisa é compreender se existe uma relação entre a quantidade total de contribuições no projeto e a disseminação do conhecimento entre os desenvolvedores, isto é, se a medida que as contribuições no repositório aumentam, o conhecimento torna-se mais distribuído entre os autores.

A Seção 5.F calculou por meio da correlação de *Spearman*, a relação entre o total de contribuições no repositório e os valores encontrados para a métrica de propriedade de código, considerando para isso todos períodos de tempo do projeto. Além disso, a relação entre a quantidade de *commits* efetuados pelos *heroes* em cada período e a quantidade de contribuições totais no mesmo período, também foi calculada utilizando o método de *Spearman*. Os resultados indicam que não se pode inferir nenhuma relação, direta ou inversa, entre o valor da métrica de propriedade de código e a variação no número de contribuições totais do projeto. Não houve uniformidade entre os valores dos coeficientes de correlação calculados para os cinco repositórios analisados. Esse resultado sugere que a concentração do conhecimento não é impactada exclusivamente pela quantidade de contribuições. Assim, podem ser encontrados projetos *open source* com distintos cenários: i) a concentração do conhecimento, calculada pela métrica propriedade de código, permanece constante, independente da variação no total de contribuições; ii) concentração do conhecimento diminui, quando há um aumento no número de contribuições, possibilitando maior disseminação do conhecimento entre os desenvolvedores; e iii) concentração do conhecimento aumenta quando há um crescimento no total de *commits*. Os achados dessa questão de pesquisa mostram que apenas a avaliação da variação da quantidade de contribuições não é o suficiente para explicar a variação no valor da métrica de propriedade de código no decorrer do ciclo de vida do projeto.

Por outro lado, os resultados indicaram que há uma relação direta forte entre o aumento dos *commits* efetuados pelos desenvolvedores *heroes* com o aumento no total de contribuições realizadas no projeto no mesmo período. Esse resultado expressa a importância dos *heroes* para o crescimento dos projetos *open source*, corroborando com o trabalho de [25] que indica que a permanência dos *heroes* é fundamental para a evolução do projeto.

7 AMEÇAS À VALIDADE

Foram utilizados, neste estudo de caso, cinco projetos *open source*, portanto não se pode generalizar os resultados. No entanto, foram analisados repositórios relevantes e maduros no contexto GitHub, com um grande número de estrelas e *commits*. Apesar do minucioso cuidado manual no tratamento de ambiguidades dos autores, ela pode conter erros e influenciar o resultado. Além disso, neste trabalho, foi considerado o método proposto por Foucault et al.(2014) para cálculo da propriedade de código. Porém, a análise de propriedade de código poderia ser realizada considerando o total de arquivos alterados pelo autor ao invés do número de *commits*. A melhor forma de avaliar a propriedade de código de um software ainda é um campo aberto para pesquisas. Para o método de cálculo da propriedade aplicado neste trabalho, entretanto, há resultados prévios de avaliação.

8 CONCLUSÃO

Fatores humanos influenciam a qualidade do software desenvolvido. No campo de software *open source*, um dos fatores humanos é a propriedade do código, que se refere ao grau de contribuição de um desenvolvedor em relação a um projeto. Compreender as características da distribuição de conhecimento de software entre desenvolvedores é importante para se propor ações, por exemplo, para a incentivo à contribuições de desenvolvedores e conhecer os desenvolvedores chave nas diferentes fases do projeto. Este trabalho foi conduzido nesse contexto, tendo como objetivo investigar a distribuição do conhecimento dos contribuidores em projetos *open source* durante seu ciclo evolutivo. Foram estudados cinco projetos *open source* presentes no *GitHub*, sendo eles *Bootstrap*, *Django*, *Elasticsearch*, *Pandas* e *Spring Boot*. Os dados dos projetos foram

estudados e analisados para se explorar como o conhecimento se distribuiu entre os desenvolvedores ao longo do tempo. Essas análises envolveram a distribuição e caracterização da população de autores, a distribuição dos *heroes* no ciclo evolutivo, a frequência de contribuições no projeto e o cálculo da métrica de propriedade de código. Os *scripts* criados neste trabalho para coleta de dados foram disponibilizados no GitHub, o que garante a replicabilidade do estudo e a realização de trabalhos futuros na linha deste trabalho[14].

Por meio dos resultados encontrados, pode-se salientar que a distribuição de conhecimento do software aumenta no decorrer de seu ciclo de vida, possibilitando a entrada de novos contribuidores e que eles venham a tornar-se futuros *heroes* do projeto (QP1). No entanto, devido às características intrínsecas da modalidade *open source*, todos os softwares analisados possuíam um grande número de autores periféricos que efetuaram apenas uma ou duas contribuições, enquanto um grupo pequeno de desenvolvedores são responsáveis pela maior parte dos *commits* existentes no repositório. O estudo também permitiu concluir que os desenvolvedores *heroes* no início do projeto não permaneceram nessa posição até os últimos períodos analisados. Contudo, esses autores tiveram uma diminuição gradual das suas atividades no repositório, diferentemente do que ocorreria em um projeto privado (QP2). Foi constatado também que não existe relação direta entre a quantidade de contribuições do projeto e a métrica de propriedade de código (QP3). Contudo, há uma forte correlação entre o crescimento das contribuições e o aumento da atuação dos desenvolvedores *heroes* no projeto.

Os achados deste estudo de caso contribuem, principalmente, na compreensão do processo evolutivo na atuação dos desenvolvedores *heroes*, bem como a forma como a propriedade de código, conforme definida por [11] varia no decorrer do tempo, sugerindo que o conhecimento torna-se mais distribuído ao longo do ciclo vida do projeto *open source*. Como trabalho futuros, é importante é replicar o presente trabalho utilizando uma quantidade maior de softwares no *data set*. Também sugere-se realizar um estudo calculando a propriedade de código utilizando outros métodos e granularidades e efetuar uma análise comparativa com os resultados deste trabalho. Além de estudos qualitativos para investigar: os motivos da saída dos *heroes* do projeto; se os desenvolvedores periféricos têm o desejo de efetuar mais contribuições no projeto; e como a disseminação do conhecimento é experimentada pelos desenvolvedores no cotidiano.

REFERÊNCIAS

[1] A. Agrawal, A. Rahman, R. Krishna, A. Sobran, and T. Menzies. 2018. We Don't Need Another Hero. In *40th Int. Conference on Soft. Engineering Software Engineering in Practice-IPSE-*, Vol. 18.

[2] G. Avelino, E. Constantinou, M. T. Valente, and A. Serebrenik. 2019. On the abandonment and survival of open source projects: An empirical investigation. In *Int. Symp. on Empirical Soft. Eng. and Measurement*. IEEE, 1–12.

[3] G. Avelino, L. Passos, A. Hora, and M. T. Valente. 2016. A novel approach for estimating truck factors. In *24th Int. Conf. on Program Comprehension*. IEEE, 1–10.

[4] G. Avelino, L. Passos, A. Hora, and M. T. Valente. 2019. Measuring and analyzing code authorship in 1+ 118 open source projects. *Science of Computer Programming* 176 (2019), 14–32.

[5] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. 2006. Mining email social networks. In *Proceedings of the international workshop on Mining software repositories*. 137–143.

[6] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu. 2011. Don't touch my code!: examining the effects of ownership on software quality. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 4–14.

[7] G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella. 2012. Who is going to mentor newcomers in open source projects?. In *Proc. of the 20th Int. Symposium on the Foundations of Soft. Engineering*. 1–11.

[8] J. Coelho, M. T. Valente, L. L. Silva, and booktitle=Proceedings of the 11th Int. Workshop on Cooperative and Human Aspects of Software Engineering pages=114–121 year=2018 Hora, A. [n.d.]. Why we engage in FLOSS: Answers from core developers.

[9] K. Crowston, K. Wei, Q. Li, and J. Howison. 2006. Core and periphery in free/libre and open source software team communications. In *Proc. of the 39th Annual Hawaii Int. Conference on System Sciences*, Vol. 6. IEEE, 118a–118a.

[10] M. Ferreira, T. Mombach, M. T. Valente, and K. Ferreira. 2019. Algorithms for estimating truck factors: a comparative study. *Software Quality Journal* 27, 4 (2019), 1583–1617.

[11] M. Foucault, J. Falleri, and X. Blanc. 2014. Code ownership in open-source software. In *Proceedings of the 18th Int. Conf. on Evaluation and Assessment in Software Engineering*. ACM, 39.

[12] T. Fritz, G. Murphy, E. Murphy-Hill, J. Ou, and E. Hill. 2014. Degree-of-knowledge: Modeling a developer's knowledge of code. *Transactions on Software Engineering and Methodology* 23, 2 (2014), 14.

[13] M. Greiler, K. Herzig, and J. Czerwonka. 2015. Code ownership and software quality: a replication study. In *Proceedings of the 12th Working Conference on Mining Software Repositories*. IEEE Press, 2–12.

[14] M. Joblin, S. Apel, C. Hunsen, and W Mauerer. 2017. Classifying developers into core and peripheral: An empirical study on count and network metrics. In *39th Int. Conference on Software Engineering*. IEEE, 164–174.

[15] T. LaToza, G. Venolia, and R. DeLine. 2006. Maintaining mental models: a study of developer work habits. In *Proceedings of the 28th international conference on Software engineering*. ACM, 492–501.

[16] A. Lee, J. Carver, and A. Bosu. 2017. Understanding the impressions, motivations, and barriers of one time code contributors to FLOSS projects: a survey. In *39th Int. Conference on Software Engineering*. IEEE, 187–197.

[17] S. Majumder, J. Chakraborty, A. Agrawal, and T. Menzies. 2019. Why Software Projects need Heroes (Lessons Learned from 1000+ Projects). *arXiv preprint arXiv:1904.09954* (2019).

[18] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan. 2016. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering* 21, 5 (2016), 2146–2189.

[19] A. Meneely and L. Williams. 2009. Secure open source collaboration: an empirical study of linus' law. In *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 453–462.

[20] C. Müller, G. Reina, and T. Ertl. 2015. In-Situ Visualisation of Fractional Code Ownership over Time. In *Proceedings of the 8th International Symposium on Visual Information Communication and Interaction*. ACM, 13–20.

[21] T. S. Orfanó, K. A. M. Ferreira, and M. A. S. Bigonha. 2018. Heurísticas para Identificação de Ambiguidade de Autores em Projetos Open Source. (2018).

[22] G. Pinto, I. Steinmacher, and M. A Gerosa. 2016. More common than you think: An in-depth study of casual contributors. In *23rd Int. Conference on Software Analysis, Evolution, and Reengineering*, Vol. 1. IEEE, 112–123.

[23] F. Rahman and P. Devanbu. 2011. Ownership, experience and defects: a fine-grained study of authorship. In *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 491–500.

[24] F. Ricca and A. Marchetto. 2010. Are heroes common in FLOSS projects?. In *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. 1–4.

[25] G. Robles, J. M. Gonzalez-Barahona, and I. Herraiz. 2009. Evolution of the core team of developers in libre software projects. In *6th International working conference on mining software repositories*. IEEE, 167–170.

[26] I. Steinmacher, T. Conte, M. A. Gerosa, and D. Redmiles. 2015. Social barriers faced by newcomers placing their first contribution in open source software projects. In *18th ACM conference on Computer supported cooperative work & social computing*. 1379–1392.

[27] P. Thongtanunam, S. McIntosh, A. E Hassan, and H. Iida. 2016. Revisiting code ownership and its relationship with software quality in the scope of modern code review. In *Proc. of the 38th int. conference on software engineering*. 1039–1050.

[28] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, and K. Matsumoto. 2015. Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. In *22nd Int. Conference on Soft. Analysis, Evolution, and Reengineering*. IEEE, 141–150.

[29] M. Torchiano, F. Ricca, and A. Marchetto. 2011. Is my project's truck factor low?: theoretical and empirical considerations about the truck factor threshold. In *2nd Int. Workshop on Emerging Trends in Soft, Metrics*. ACM, 12–18.

[30] M. Tufano, F. Palomba, G. Bavota, R. Oliveto, M. Di Penta, A. De Lucia, and D. Poshyvanyk. 2015. When and why your code starts to smell bad. In *37th Int. Conf. on Soft. Engineering*, Vol. 1. IEEE, 403–414.

[31] M. Zhou and A. Mockus. 2014. Who will stay in the floss community? modeling participant's initial behavior. *Trans. on Soft. Engineering* 41, 1 (2014), 82–99.

[14] Dados para replicação: github.com/taliorfano/knowledgedistribution_master