Tie Strength Metrics to Rank Pairs of Developers from GitHub

Natércia A. Batista[1], Guilherme A. Sousa[1], Michele A. Brandão[2],
Ana Paula C. da Silva[1], Mirella M. Moro[1]

[1]Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG – Brazil
[2]Instituto Federal de Minas Gerais (IFMG) – Ribeirão das Neves, MG – Brazil
{natercia,guilhermeaugusto}@dcc.ufmg.br,michele.brandao@ifmg.edu.br,
{ana.coutosilva,mirella}@dcc.ufmg.br

Abstract. The Web provides huge volumes of data, which makes efficient data collecting and processing not easy tasks. An example of such volumes is in *software repositories*, a type of Web storage platform for software and projects, their developers and companies. In this work, we first present a systematic literature review over topics related to such repositories. Then, we extract their data and enrich it by building a development network. Based on such a network, we investigate tie strength metrics on their capability of defining new information through a correlation analysis. We also use the metrics to rank pairs of developers by considering three different aggregate methods. Our experimental analysis shows different results for each ranking method when considering all pairs of developers, which reveals the difficulty of choosing the best way to rank pairs of developers. However, when considering the top 10 best ranked pairs, two methods present similar results. Also, the combination of tie strength metrics with ranking aggregated methods allows to identify important developers in the network and their collaboration strength.

Categories and Subject Descriptors: Information Systems [**World Wide Web**]: Social Networks

Keywords: Metrics, Social Networks, Web Data, Web Software Repositores

1. INTRODUCTION

Data are available everywhere on the Web, from social media and online networks to simple websites. Most of such data are heterogeneous, diverse, and represent different entities from the real world and their interactions. However, extracting relevant information from such volumes of data is not an easy task. Overall, any method to extract and enrich such data in order to discover useful knowledge is welcome and has potentially many real applications.

One specific scenario is *software repositories*, a type of Web storage platform for software and projects, their developers and companies. The data available can be processed in order to not only acquire technical information (about projects and software) but also study social relations (about developers, teams, and their interactions). Furthermore, studying such relations allows to understand social coding as well. Social Coding is a software development approach for collaborative work, which instigates discussion and sharing of ideas and knowledge among developers [Dabbish et al. 2012]. This methodology has changed the way software is developed since geographically distant developers can access collaborative platforms and participate in different projects remotely. Examples of Web platforms that allow Social Coding include Google Code[1] and GitHub[2].

[1] Google Code: `code.google.com`
[2] GitHub: `github.com`

Given the data extracted from developers and their projects, one way to enrich it and produce new knowledge is through building social networks. These networks are usually modeled as graphs with participants as nodes and their relationships as edges. For software collaboration, nodes represent developers who can create/contribute/share software repositories and projects [Alves et al. 2016; Batista et al. 2017b]. Then, such participants can be connected (through graph edges) by the projects and repositories they have in common, as well as through other activities provided in the platform such as adding comments and open/reply issues in the same repositories.

In this work, we analyze the properties of collaborative networks that can represent the strength of the relationship between developers, prioritizing aspects related to the time of such paired contribution. Specifically, our goal is to find semantic properties of the relationship between developers and analyze their usage to determine the strength of developers' relationship. We use the correlation between topological and semantic metrics introduced in our previous works ([Alves et al. 2016] and [Batista et al. 2017c]) to analyze information about tie strength. Such tie strength metrics represent a way to extract relevant information from data, then allowing to acquire knowledge over developers and their collaborations. Unlike Alves et al. [2016] and Batista et al. [2017c], here we analyze how topological metrics change over time for an extended dataset that consider three programming languages classified as more collaborative and three as less collaborative, with updated data till May 2017. Note that we have previously considered a smaller set of languages in [Batista et al. 2017c].

Tie strength metrics may also serve to rank pairs of developers. Then, ranking in this context is useful to, for example: (*i*) analyze team formation by providing evidence about the engagement of individuals on a team [Hahn et al. 2006]; (*ii*) detect communities by grouping individuals with similar strength or evaluate the quality of the community detection algorithm [Brandão and Moro 2017a]; and (*iv*) recommend experts to solve a bug by identifying developers working in similar projects [Yu et al. 2016]. Indeed, we build rankings based on the collaboration strength by using three different methods and present their results. Then, we compare our results with a ranking of developers from GitHub[3], which is based on their popularity in each programming language.

Overall, our contributions are summarized as follows.

—A systematic literature review over metrics, team formation, recommendation and ranking in the context of the data available at software repositories (Section 2).

—A dataset accompanied by a network model, a set of metrics to compute the strength of collaboration in repositories, and a ranking approach for pairs of developers (Section 3).

—Experimental analyses of the correlation between tie strength metrics, and evaluation of the ranking approach (Section 4).

2. SYSTEMATIC LITERATURE REVIEW

Different Web tools allow people and organizations to collaborate towards specific goals. Then, collecting and analyzing the data of such tools allows to understand the collaboration process per se. Going one (or many) step forward, applying specific metrics and enriching such data opens countless possibilities of acquiring useful knowledge. For example, building a collaboration graph (or professional social network) and applying tie strength metrics over it may reveal experts for ranking and recommendation purposes [Brandão and Moro 2017a].

One type of such tools is *software repositories*, in which developers easily contribute to different projects around the world. For example, GitHub[4] is the largest one, with more than 71 million of projects (November 2017)[5] and over 23 million of developers. Another important software repository

[3] Git Awards: `http://git-awards.com/users`

[4] GitHub: `https://github.com`

[5] The world's largest code platform: `http://github.com/features`

is Google Code[6], which has less projects/users than GitHub, but it is very relevant for the free software community. Another example is SourceForge[7], a predecessor of GitHub. There are also many other open source alternatives to version control, a related problem to such repositories [Mockus 2009].

In such a context, there are studies that take distinct views over the data that can be extracted from software repositories. Some of those are in the form of *systematic literature review.* Overall, a systematic literature review aims to identify, evaluate and interpret the available studies that are relevant to a particular research question, topic area or phenomenon of interest [Kitchenham 2004]. Examples of such reviews over the process of software engineering, the practices of software development and the impact of systematic literature reviews include: [Brereton et al. 2007], [Cosentino et al. 2017], and [Kitchenham et al. 2009].

In this article, we complement such efforts by considering metrics, team formation, recommendation and ranking algorithms. Metrics and ranking are also the focus of our work, whereas team formation and recommendation are relevant, useful applications in the broader context of professional social networks [Brandão and Moro 2017a]. Overall, following the systematic review methodology proposed by [Kitchenham 2004], we have identified 2,755 documents and selected 92 publications based on the exclusion/inclusion criteria, as described next.

2.1 Research Questions

The first step in any systematic review is usually to define the research questions; i.e., to specify the goal of such review through objective questions. Here, we consider three questions, as follows.

—*(RQ1)* How do current studies over software repositories (and their data) tackle metrics, team formation, recommendation and ranking? The goal is to understand themes and contexts explored by current studies. Answering this question also allows to identify potential research directions.

—*(RQ2)* How are the statistics regarding affiliation of researchers and publication venues over such topics? This analysis allows to understand the community formed over such topics.

—*(RQ3)* Which repositories such publications evaluate? Given the various possibilities, it is important to verify if there are preferable repositories and in what extent.

2.2 Digital Libraries

The second step in a systematic review is to define the source of the studies to be evaluated. Given the advance of digital libraries, current reviews usually focus on publications available in the Web, and so do we. Therefore, we consider the following widely known digital libraries with vast range of publications in Computer Science: ACM[8], IEEE[9], SPRINGER[10] and ELSEVIER[11]. We do *not* consider DBLP[12], because most of its publications are already available in the aforementioned libraries and its search capabilities are limited (again, when compared to the aforementioned ones).

2.3 Inclusion and Exclusion Criteria

Having defined the goal and digital libraries, the next step is to establish the criteria to include or not a publication in the process. Our inclusion criterion is straightforward: articles and papers that

[6] Google Code: `https://code.google.com`
[7] SourceForge: `https://sourceforge.net`
[8] ACM Digital Library: `https://dl.acm.org`
[9] IEEE Xplore Digital Library: `https://ieeexplore.ieee.org`
[10] Springer Computer Science: `https://www.springer.com/ComputerScience`
[11] Computer Science – Elsevier: `https://www.elsevier.com/physical-sciences/computer-science`
[12] Computer Science Digital Library: `http://dblp.uni-trier.de/search?q=acm`

propose metrics or tackle team formation, recommendation and ranking over software repositories, or using their data. On the other hand, we identify four relevant exclusion criteria: *(i)* duplicate publications that address the same study or results[13]; *(ii)* studies that do not use data collected from software repositories; *(iii)* publications that do not report the name of the used software repository; and *(iv)* lesser known publications on venues with disputable impact or with few citations (as given by Google Scholar at the moment of collecting them).

2.4 Collecting Process

The collecting process was performed in September and October, 2017, and considers the following steps. First, we define two sets of keywords related to software repositories (e.g., software repository and source code) and to the review goal (i.e., metric, rank, recommend, team formation). Note the first set does not consider the name of the software repositories, such as GitHub. This way, the result is not biased and does not affect the response to *RQ3*. We also vary between words in singular and plural, as some of the expressions are considered between quotes.

Second, we search over the digital libraries looking for the Cartesian product of the two sets. Figure 1 presents the amount of publications returned for each pair of keywords in each library. For instance, there are 2,200 publications at the ACM Digital Library for “metric” and “open source software”. Note that the keyword “team formation” does not obtain many results, which may indicate a gap in the literature (or distinct denominations for such a topic). Also, searching for “ranking” retrieves the second smallest number of publications for most situations, except when it is combined with “open source software”. Therefore, there is still room for research on such topics.

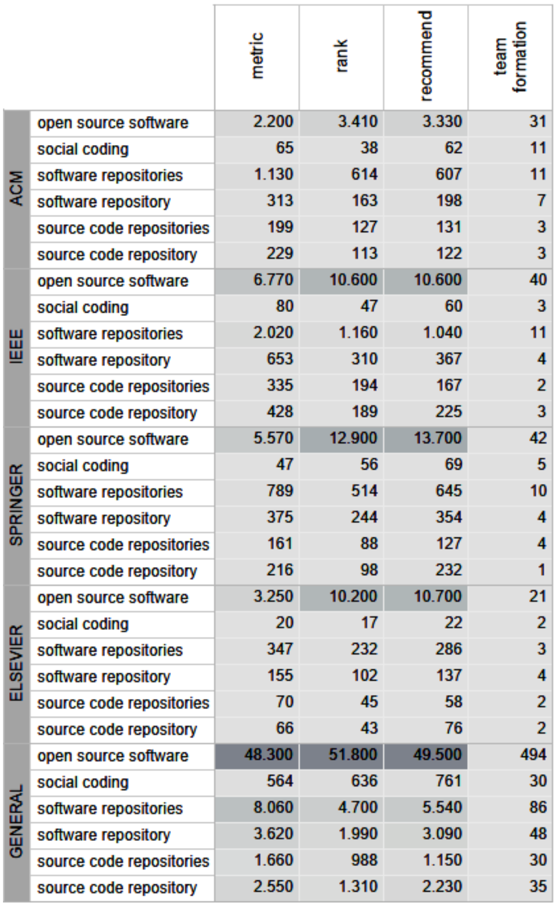
Next, the results are sorted by relevance. Then, we select the publications by analyzing both title and abstract, whether they potentially fulfill our objective. In total, 134 publications are selected in this step. Figure 2 shows a histogram of the number of publications selected per year. The quantity of published articles grew after 2008; which is also the year GitHub was released, and may indicate the importance of this repository in the literature. Finally, the last step of the collecting process is to read and evaluate all publications selected. At the end, we consider 55 publications as relevant to our study. The statistics and bib entries of such publications are available for download at `http://bit.ly/proj-apoena`.

2.5 Data Extraction

After collecting, we extract data (meta-data and content information) from the selected publications, which facilitates their organization and analysis. Specifically, we consider the following data.

—Repository(ies) – the software repositories used as the dataset of the research, either in the construction or in the validation of results;
—Publication authors;
—Author(s) affiliation (institution);
—Publication title;
—Year of publication – as software repositories are recent, we have not defined a time interval as a criterion to start the analysis of the publications;
—Number of citations of the publication (as given by Google Scholar at the moment of collecting);
—Publication venue (journal or conference);
—Research focus (metrics, team formation, recommendation or ranking).

[13] Duplicate publications occur when multiple reports from the same study exist in different venues. In this case, we include only the most *complete* version of the study (for example, the extended journal article from a previously published conference paper).



		metric	rank	recommend	team formation
ACM	open source software	2.200	3.410	3.330	31
	social coding	65	38	62	11
	software repositories	1.130	614	607	11
	software repository	313	163	198	7
	source code repositories	199	127	131	3
	source code repository	229	113	122	3
IEEE	open source software	6.770	10.600	10.600	40
	social coding	80	47	60	3
	software repositories	2.020	1.160	1.040	11
	software repository	653	310	367	4
	source code repositories	335	194	167	2
	source code repository	428	189	225	3
SPRINGER	open source software	5.570	12.900	13.700	42
	social coding	47	56	69	5
	software repositories	789	514	645	10
	software repository	375	244	354	4
	source code repositories	161	88	127	4
	source code repository	216	98	232	1
ELSEVIER	open source software	3.250	10.200	10.700	21
	social coding	20	17	22	2
	software repositories	347	232	286	3
	software repository	155	102	137	4
	source code repositories	70	45	58	2
	source code repository	66	43	76	2
GENERAL	open source software	48.300	51.800	49.500	494
	social coding	564	636	761	30
	software repositories	8.060	4.700	5.540	86
	software repository	3.620	1.990	3.090	48
	source code repositories	1.660	988	1.150	30
	source code repository	2.550	1.310	2.230	35

Fig. 1. Heatmap with the amount of returned publications for each pair of keyword.

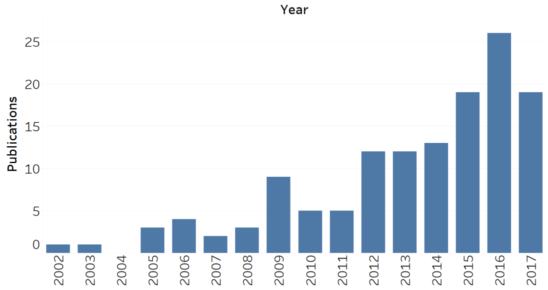


Fig. 2. Histogram with the number of selected publications per year.

2.6 Discussion on the Literature Review

We summarize our review findings by presenting the three best-evaluated publications on each topic and then answering the research questions.

Metrics. Overall, current metrics evaluate certain aspects of different areas. For instance, Biazzini and Baudry [2014] propose metrics to analyze projects whose code-base is distributed among several forks on GitHub. The goal is to quantify the dispersion degree of contributions in a project with many repositories. In the context of measuring the success or performance of projects, McDonald et al. [2013] conduct a qualitative and exploratory research with leading developers of three successful projects on GitHub and identify that employee growth, community engagement and visible activity are more important metrics for success than the quality of the code itself. In turn, Bissyandé et al. [2013] investigate different correlations of software development metrics in 100,000 open source software projects from GitHub to answer research questions about "popularity", "interoperability" and "impact" of various programming languages.

Team formation. Such a concept appears in various scenarios, from training teams for new product development to organizing social events. For example, Han et al. [2017] develop a solution that considers communication cost and geographical proximity to perform team formation on GitHub. Majunder et al. [2012] introduce and analyze algorithms that form teams socially close in GitHub. Finally, Vasilescu et al. [2015] conduct an user research to understand how developers who collaborate

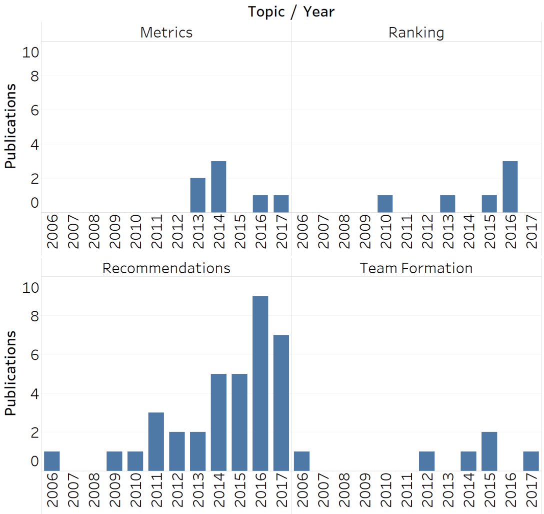


Fig. 3. Amount of papers per topic.

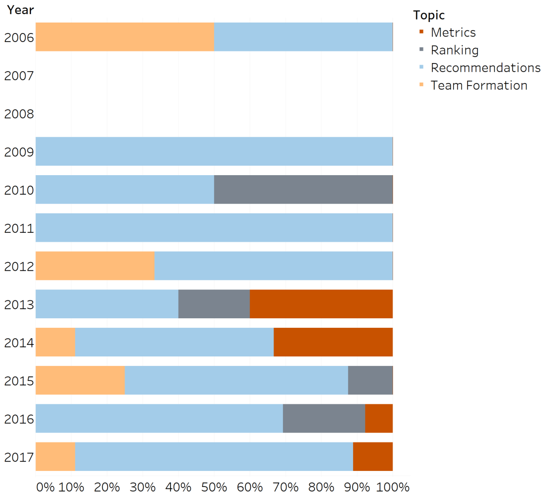


Fig. 4. Proportion of papers per topic.

on GitHub perceive teamwork and individual attributes (e.g. programming and social skills, overall GitHub experience, reputation as programmer and educational level).

Recommendation. Likewise, there are different applications for recommendation systems, such as e-commerce, social networks and Web search. For example, Guendouz et al. [2018] use collaborative-filtering to recommend open source projects on GitHub. Specifically, the authors develop a prototype that presents a list of repositories recommended for developers. Another common problem is the recommendation of developers to fix bugs in collaborative networks. For instance, Anvik et al. [2006] present a semi-automated approach to make easier to assign a bug to a developer with the appropriate experience to solve it. Lastly, Wu et al. [2011] propose an approach to recommend developers for bug-solving based on the K-Nearest-Neighbor search with bug similarity and skill ranking.

Ranking. The ranking systems use performance indicators associated with quality in order to classify entities from different domains. For instance, Wan et al. [2018] propose a generic probabilistic expertise ranking model to effectively locate developers with specific coding skills. Also, Grechanik et al. [2010] introduce an approach called "Exemplar" to identify and rank highly relevant software projects by combining program analysis techniques with information retrieval. The Issue outperformed Sourceforge in terms of reporting higher levels of confidence and accuracy for retrieved Java applications. Finally, Thung et al. [2013] identify influential developers and projects in the GitHub network through the PageRank algorithm.

Research Questions. Next, we discuss the following aspects: most common topics (*RQ1*); institutions with most authors and common venues (*RQ2*); and software repositories (*RQ3*).

Overall, for *RQ1*, Figures 3 and 4 show the amount and proportion of papers per topic between 2006 and 2017. Overall, publications addressing recommendations are more common, with studies covering recommendations of repositories, developers, reviewers and source code. Regarding metrics, rankings and team formation, there is still a lot to explore and potentially define new research problems.

For *RQ2*, Table I presents the top 10 institutions with more publications. This table is particularly interesting due to its variety of countries. Specifically, three institutions are from United States, three from China, one from Japan, two from Brazil, and one from Poland. Regarding the venues in which the papers are published, Table II shows most are from the Software Engineering area (due to lack of space, we do not show those venues with only one paper published). The top venues are the *Journal of Information and Software Technology* (Info. Sw. Tech), the *International Conference on Software*

Table I. Top 10 institutions.

Institution	Pub.
SMU, USA	6
UC, USA	6
Zheda U., China	6
Microsoft, USA	3
BUAA, China	3
NUDT, China	3
PJATK, Japan	3
UFAC, Brazil	3
UFF, Brazil	3
Wroclaw UT, Poland	3

Table II. Most common venues.

Source	Cit.	Pub.
ICSE	958	4
MSR	243	4
Info.Sw.Tech	41	4
ICSME	132	2
WCRE	107	2
CHASE	48	2
COMPSAC	44	2
CHI	41	2
CSI-SE	7	2
FSE	6	2
ASE	3	2
WI	1	2

Table III. Most used repositories.

Repository	Cit.	Pub.
GitHub	490	38
Bugzilla	1321	10
SourceForge	234	6
Gerrit	34	4
Savannah	36	1
Maven	4	1
CodeFlow	13	1
OpenHub	8	1

Engineering (ICSE), and the *Mining Software Repositories* workshop (MSR). Also, the most cited publications are from the ICSE conference.

Table III presents the software repositories considered by the publications in our systematic review. There are publications that consider more than one software repository. GitHub is the most studied one, which could be because GitHub is the most used source code hosting platform. The second one is Bugzilla[14], which appears more on studies over the problem of bug assignment, i.e. the recommendation of developers to solve problems in software. Moreover, the systematic review reveals that the most cited publications are those that address GitHub and Bugzilla repositories.

Overall Discussion. One important contribution of this systematic review is the identification of gaps in the research regarding metrics, team formation, recommendation and ranking over software repositories and their data. Also, this review shows institutions and venues with more publications in these four topics, which enable the easy identification of potential collaborations and publication venues. Next, our methodology also considers GitHub, because it is the most popular social coding platform, as presented in this review and confirmed by others [Cosentino et al. 2017].

3. METHODOLOGY

This section presents the dataset and the network model used for the analysis, and the definition of the topological and semantic metrics to measure the strength of the relationships.

Data. The dataset is an *expanded* and *updated* version of GitSED (*GitHub Socially Enhanced Dataset*) [Batista et al. 2017b]. Specifically, this new version compares to the original one in the following aspects. First, the dataset is updated with data until May 2017. Second, it includes GitHub network information for languages with higher and lower levels of collaboration [Rocha et al. 2016]: JavaScript, Ruby and Python (high collaboration), and Assembly, Visual Basic and Pascal (low collaboration). Considering these two different groups of collaboration allows to understand the difference between them, specifically, about the tie strength and its ranking. Third, the original version considers the start date of the collaboration between two developers as the most recent date between the first commits of both developers in a repository, and the end date of collaboration as the last commit in the repository by *any* developer. Now, the end of the collaboration has been *changed* to the earliest date between the last commits of a *pair* of developers in the repository, which allows to limit the relationship within the joint contribution time interval.

Network Model. The developer network is represented by a weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$: $\mathcal{V}$ is the set of nodes (vertices) representing developers, and $\mathcal{E}$ is the set of non-directed edges that connect two developers when they both have contributions to the same repository within the same time interval.

[14] Bug Repository: `https://www.bugzilla.org/`

Table IV. Statistics for each language in the dataset

Language	Number of Repositories	Number of Nodes (Developers)	Number of Edges (Connections)	Density (10^{-3})
JavaScript	6,767,297	854,255	2,571,154	0.007
Python	3,074,827	519,771	3,699,096	0.027
Ruby	2,536,133	279,281	33,979,590	0.871
Assembly	35,073	7,516	14,906	0.528
Visual Basic	33,275	5,602	7,205	0.459
Pascal	20,330	3,520	9,377	1.514

The initial and final developer collaboration dates in a repository are defined by the first and last commit date in such a repository. Therefore, the relationship between developers exists only when their time intervals overlap – following our original proposition on [Batista et al. 2017c]. The weight of the edges corresponds to the values that can be calculated from the metrics proposed by Alves et al. [2016] and Batista et al. [2017a] (presented later in this section).

Network Basic Statistics. Overall, Table IV shows the number of repositories and developers present in the network for each of the programming languages considered, and the number of connections (edges) in each network. Although the network for JavaScript has the largest number of repositories and developers, such a network is not as dense as the other three most collaborative languages. On the other hand, the network for Ruby has the smallest number of repositories and developers between the first three languages, but it has the largest number of connections as well.

Topological and Semantic Properties. Table V summarizes the topological and semantic properties considered by metrics to evaluate the developers relationships (or ties, links). Such metrics represent social aspects that can be used to analyze and interpret software engineering aspects, for example, the test of software development hypothesis and the development of tools to improve practices [Begel et al. 2010]. Specifically, the topological and semantic properties capture information about the collaboration between developers which helps to understand the process of creating and developing software (part of software engineering concerns).

Semantic Properties with Temporal Aspect. Table V also summarizes three semantic properties that consider the temporal aspect of the relationships proposed by Batista et al. [2017b]. Local Potential Contribution (LPC) and Global Potencial Contribution (GPC) calculate the contribution time between a pair of developers in a given repository with local and global variations, respectively.

Our Proposal. Here, we propose an update on the definition of the end date of collaboration between developers (from Oct 2007[15] to May 2017, as already discussed); then, the results for the respective metrics change as well. Such an update is necessary because LPC calculation can be biased, especially in short-lived repositories as exemplified in Table VI: the pair of developers A, B contributed for less time than C, D, but has a higher value for the metric. To understand the time-based collaboration force for all developer pairs, a global view of the whole network is necessary. Then, GPC considers the maximum contribution time between all developer pairs as the denominator in order to normalize the value of the calculated metric. For the previous example in Table VI, considering that the longest contribution time of the network is 12 months, the results are divided by such a value. At the end, the GPC metric allows to classify the potential force of contribution in relation to all possible contributions within the network and, therefore, it should be especially used when analyzing the network evolution.

Topological Property with Semantic One. The combination of a topological property with a semantic one allows to analyze a different perspective of tie strength, since it provides one value that represents two distinct characteristics of the relationship. For example, combining a topological property with the number of shared repositories or previous collaboration aggregate information that may define new knowledge about the developers. Here, we use Tieness [Brandão and Moro 2017b],

[15] The first commit on GitHub was made in October 2007: `https://github.com/about`

Table V. Given two nodes A and B, let $\mathcal{N}(A)$ and $\mathcal{N}(B)$ be the set of neighbors of A and B, $w(A)$ and $w(B)$ be the weighted degree (a sum of the weight of each edge connected to the node), $w(A, B)$ be the weight of the edge between A and B, and $\mathcal{R}$ the set of shared repositories between A e B.

Topological Properties	
Metric	**Definition and Interpretation**
Neighborhood Overlap (NO)	According to Easley and Kleinberg [2010], NO can be used to compute the strength of the links. The higher the value of NO, the stronger the relationship. It measures the neighborhood similarity for any two pair of nodes and is computed as: $NO_{(A,B)} = \frac{\lvert\mathcal{N}(A)\rvert\cap\lvert\mathcal{N}(B)\rvert}{\lvert\mathcal{N}(A)\rvert\cup\lvert\mathcal{N}(B)\rvert}$
Adamic-Adar Coefficient (AA)	Neighbors that are not shared with many others receive more weight. This metric in network context is customized as: $AA_{(A,B)} = \frac{\sum \forall z \in \lvert\mathcal{N}(A)\rvert\cap\lvert\mathcal{N}(B)\rvert}{log\lvert\mathcal{N}(z)\rvert}$.
Preferential Attachment (PA)	According to Barabási and Albert [1999], there is a linear relationship between the number of neighbors of a node and the probability of attachment (i.e.,"the rich get richer"). The greater the number of neighbors of a node, the higher the value of preferential attachment, defined as $PA_{(A,B)} = \lvert\mathcal{N}(A)\rvert\lvert\mathcal{N}(B)\rvert$.
Semantic Properties – [Alves et al. 2016]	
Metric	**Definition and Interpretation**
Number of shared repositories (SR)	This metric refers to the number of shared repositories between developers and is defined by the cardinality of the set $\mathcal{R}$ (i.e., $SR_{(A,B)} = \lvert\mathcal{R}\rvert$)
Jointly developers contribution to shared repositories (JCSR)	Defined by: $JCSR_{(A,B)} = \frac{\sum_{\forall r_i \in \mathcal{R}} JCSR_{(A,B,r_i)}}{\lvert\mathcal{R}\rvert}$.Example: Let two repositories r_1 and r_2, r_1 is shared only by developers A and B, the joint contribution of the pair AB in r_1 ($JCSR_{(A,B,r_1)}$) is 1. Then, r_2 is shared by developers A, B and C, and the joint contribution of A and B in r_2 ($JCSR_{(A,B,r_2)}$) is 0.66. Assuming A and B share only r_1 e r_2, the joint contribution to them is the mean of the values for each repository: $(JCSR_{(A,B)}) = 0.83$.
Jointly developers commits to shared repositories (JCOSR)	Given that $NC_{(A,r)}$ and $NC_{(B,r)}$ are the total numbers of commits made by developers A and B, respectively, in the repository r, and $NC_{(r)}$ is the total number of commits made in the repository r independent of the developer, $JCOSR_{(A,B)} = \sum_{\forall r_i \in \mathcal{R}} \frac{(NC_{(A,r_i)}+NC_{(B,r_i)})}{NC_{(r_i)}}$.
Semantic Properties with Temporal Aspect – [Batista et al. 2017b]	
Metric	**Definition and Interpretation**
Local Potential Contribution (LPC)	Considering $T_{(A,B,r)}$ as the time interval of contribution between developers in the repository r, defined by the collaboration start and end dates (represented by the first and last commit in a repository) and $T(r)$ the whole time of repository r, this metric can be computed by $LPC_{(A,B)} = \frac{\sum_{\forall r_i \in \mathcal{R}} \frac{T_{(A,B,r_i)}}{T_{(r_i)}}}{\lvert\mathcal{R}\rvert}$.
Global Potential Contribution (GPC)	Batista et al. [2017b] extended the LPC metric to a global version to compensate for the bias in its computation, defined as $GPC_{(A,B)} = \frac{\sum_{\forall r_i \in \mathcal{R}} T_{(A,B,r_i)}}{\max_{\forall (A',B') \in \mathcal{D},\ r_i \in \mathcal{R}} T_{(A',B',r_i)}}$, where D is the set of developers on the network.
Previous Collaboration (PC)	It counts the previous collaborations between a pair of developers. Thus, at time t the metric is defined by $PC_{(A,B,t)} = \frac{\sum_{\forall r_i \in \mathcal{R}} \frac{1}{ND_{(r_i,t)}}}{\lvert\mathcal{R}\rvert}$, where $ND_{(r,t)}$ is the number of developers that contributed in the repository r at time t before the developer B start collaboration in r. High values of the fraction $1/ND_{(r,t)}$ indicate that A is more likely to work with B and vice versa. For example, if there are only two people in a given repository, there is only one possibility of collaboration; otherwise, the more people in one repository, more choice and less possibility of collaboration with a particular developer. Thus, there is a greater possibility of establishing a connection with a developer if the attention of the developer is not divided into many options.
Topological Property with Semantic One – [Brandão and Moro 2017b]	
Metric	**Definition and Interpretation**
Tieness (T)	It combines a topological property with a given weight that represents semantic properties, defined as $T(A,B) = \frac{\lvert(\mathcal{N}(A)\cap\mathcal{N}(B))+1\rvert}{\lvert\mathcal{N}(A)\cup\mathcal{N}(B)\rvert-(A\ or\ B\ themselves)} *\ w(A,B)$. For any calculation of Tieness, the new name is given by the prefix T_ and the name of the metric used as weight. For example, considering SR as weight, Tieness is represented by T_SR.

a metric to tie strength that combines a modification in neighborhood overlap (topological property) with a weight (semantic property), as defined in Table V.

Ranking Pairs of Developers. Usually, ranking methods are applied to GitHub to identify experts [Wan et al. 2018], i.e. developers are ranked by some feature that relates to expertise. Here, we rank pairs of developers to identify the strongest relationships, which can be applied to team formation analysis, community detection and recommendation. We do so by combining different tie strength metrics in order to generate one single ranking.

Table VI. LPC and GPC examples for two pairs of developers.

Developer Pair	**Repository**	**Repository Duration**	**Contribution Time**	**LPC**
Dev A - Dev B	R1	3 months	3 months	3/3 = 1,00
Dev C - Dev D	R2	12 months	6 months	6/12 = 0,50

Developer Pair	**Contribution Time**	**Longer Network Contribution Time**	**GPC**
Dev A - Dev B	3 months	12 months	3/12 = 0,25
Dev C - Dev D	6 months		6/12 = 0,50

Overall, we consider the top 1,000 users at Git Awards[16], excluding organizations. Then, we consider three commonly used methods to aggregate the values of different metrics: CombSUM [Ganjisaffar et al. 2011], Borda count [Emerson 2013] and Condorcet procedure [Young 1988]. Generally, these aggregation methods help to determine the order of social preference in elections. None of them is clearly optimal; thus, we use these three different aggregation methods and compare their results.

Specifically, the CombSUM aggregates the rankings by the summing values. The final ranking is defined by descending order of the sum of the values obtained by each of the pairs in the metrics. On the other hand, Borda Count aggregates the rankings by adding the positions of each pair in each ranking, resulting in an added value. Then, this value is ordered incrementally, generating a unique classification that considers all the metrics. In the last ordering, the pair that obtains the lowest added value will be preferred in relation to the others. Finally, Condorcet Procedure consists of comparing pairs from each ranking. If pair {A,B} has a better result than pair {B,C} in most rankings, pair {A,B} gets 1 point. Note that there is no weight determined to the rankings, i.e. they all have the same importance. In case of a tie, each pair receives 0.5 points. Then, pairs are sorted based on their scores, resulting in a unique ranking. In order to apply these methods, we normalize the metrics within the interval [0,1] to ensure there is no bias caused by the different distribution of values.

4. ANALYSIS AND RESULTS

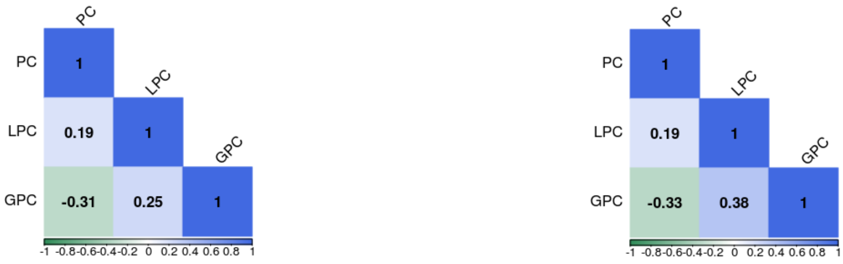
In this section, we discuss the correlation analysis between the properties with temporal aspects and the others. We present results only for the Spearman correlation coefficient, since we obtain similar results for the Pearson correlation coefficient. Then, we apply the three ranking aggregation methods to combine metrics to get a single value for the developers collaboration strength. Such an aggregated value allows to rank and compare pairs of developers.

4.1 Metrics Correlation Analyses

Our analyses are separate into three topics: semantic properties with temporal aspects, combined topological and semantic properties, and correlation with existing properties. For all analyses, we chose two languages to present results, JavaScript and Assembly, because they are the languages with the largest number of repositories in the most collaborative and least collaborative groups in the dataset, respectively. The results are similar to other languages in the same group.

Analysis of semantic properties with temporal aspect. The GPC and LPC metrics have similar calculation forms, but the normalization present in the GPC formula differentiates their results. Figure 5 presents the Spearman correlation for the three metrics with temporal aspects. There is a weak correlation between GPC and LPC. Nonetheless, GPC better represents the connection of the developers pair over time when relating to all other pairs in the network, as presented in Section 3. In addition, the PC metric does not strongly relate to any of the others, regardless of the correlation coefficient. This behavior can be justified by the collaboration time interval. For LPC and GPC metrics, two developers have the metric value related to the time of joint collaboration in the same

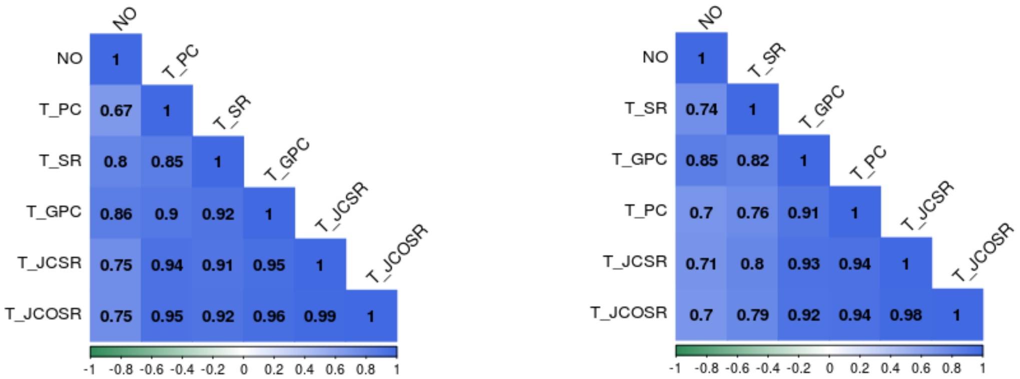
[16] Git Awards: https://git-awards.com/



(a) JavaScript

(b) Assembly

Fig. 5. Spearman correlation coefficient between new properties with temporal aspects: GPC, LPC, and PC.



(a) JavaScript

(b) Assembly

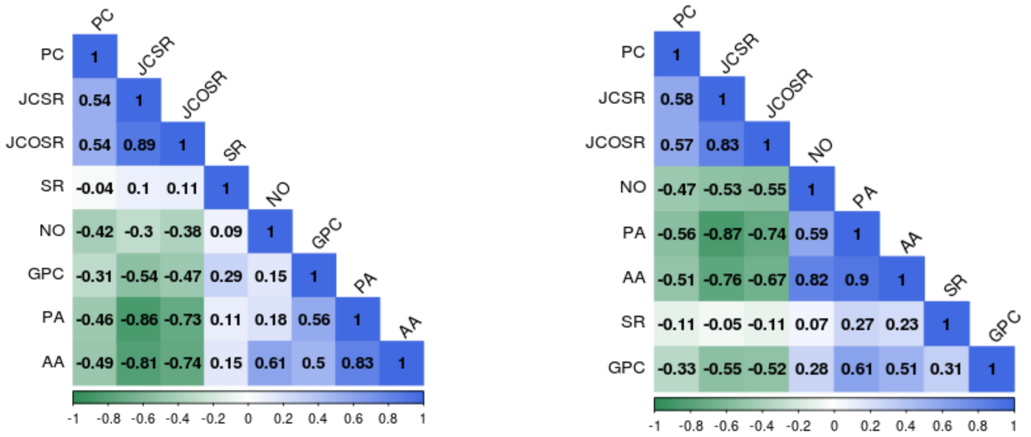
Fig. 6. Spearman correlation coefficient between NO and Tieness (T_) weighted by all other proposed metrics.

repository. Meanwhile, PC gives importance to the number of repositories they already contributed and the number of collaborators in such repositories. Therefore, the non-correlation between such metrics can be justified by the values of the metrics associated to the edges that have a little time intersection of contribution, but lots of shared repositories with a small number of contributors in the past. The correlation results by Spearman and Pearson coefficients for those metrics are similar and reflect a non correlation between PC and the other ones. Thus, PC generates distinct knowledge about the pairs, making important to calculate it and consider it to measure the collaboration strength.

Combined Topological and Semantical Properties. Tieness is a metric that allows combining topological and semantic metrics as weight. It was calculated for all proposed metrics in this study, and their correlation is in Figure 6. For Spearman (and Pearson) correlation coefficients, all combinations are highly correlated with each other and with NO. Thus, we can choose a value based on the meaning of the metric on the network or on its computational cost.

Correlation Between All Properties. Considering all topological and semantic properties from Table V and semantic properties with temporal aspects, Figure 7 presents the Spearman correlations. There is no significant variation among the results for different programming languages. Nonetheless, in all of them, topological metrics AA and PA are strongly correlated. Thus, we can consider only one of them to measure the strength of the collaboration. Meanwhile, GPC and SR are weakly correlated with others, because they bring different information about contribution time and the number of shared repositories. Therefore, it is important to consider these two values in any study. Finally, PC, JCSR and JCOS metrics are strongly correlated in all scenarios. Thus, it is possible to choose just one to calculate and consider in a final analysis.

Summary of Results. Finally, considering all the metrics and their correlation, it is possible to



	PC	JCSR	JCOSR	SR	NO	GPC	PA	AA
PC	1							
JCSR	0.54	1						
JCOSR	0.54	0.89	1					
SR	-0.04	0.1	0.11	1				
NO	-0.42	-0.3	-0.38	0.09	1			
GPC	-0.31	-0.54	-0.47	0.29	0.15	1		
PA	-0.46	-0.86	-0.73	0.11	0.18	0.56	1	
AA	-0.49	-0.81	-0.74	0.15	0.61	0.5	0.83	1

(a) JavaScript

	PC	JCSR	JCOSR	NO	PA	AA	SR	GPC
PC	1							
JCSR	0.58	1						
JCOSR	0.57	0.83	1					
NO	-0.47	-0.53	-0.55	1				
PA	-0.56	-0.87	-0.74	0.59	1			
AA	-0.51	-0.76	-0.67	0.82	0.9	1		
SR	-0.11	-0.05	-0.11	0.07	0.27	0.23	1	
GPC	-0.33	-0.55	-0.52	0.28	0.61	0.51	0.31	1

(b) Assembly

Fig. 7. Spearman correlation coefficient between all metrics presented

combine them to measure tie strength. Note that is not necessary to use all the metrics together, they can be applied individually in specific scenarios. Next, we indicate what metrics should be chosen in specific comparisons: between GPC and LPC, the former should be chosen; between AA and PA, any could be chosen; SR should be chosen; among PC, JCSR and JCOSR, anye could be chosen; and Tieness combined with metrics as weight, any could be chosen.

4.2 Ranking Pairs of Developers

The strength of social coding collaboration can be measured in different ways with distinct goals [Bartusiak et al 2016; Casalnuovo et al. 2015; Tsay et al. 2014]. Thus, it is important to identify which aspects better represent such strength for a specific application. Here, we have different metrics to compute collaboration among developers and they can be aggregated in order to become a single value. Thus, pair of developers can be classified and ranked.

Specifically, the ranking considers all pairs of the 1,000 best-placed users from Git Awards in the networks of each language. It also considers the following metrics from the previous analyses: PA (Preferred Attachment), PC (Previous Collaboration), SR (Number of Shared Repositories), GPC (Global Potential Contribution), and T_JCOSR (Jointly Developers Commits to Shared Repositories).

Now, considering the ranking aggregation methods from Section 3, we note they are not correlated with each other (now shown due space constraints). In other words, each aggregates the metric values differently, giving importance to distinct factors. Their main difference is the "requirement level" in relation to the set of metric values. Hence, for each aggregation, getting a good placement depends on the developers pair having a high or medium value in all metrics, or having a great classification on specific metrics regardless of others.

For example, the **CombSUM** sums the metric values to each pair and uses such result to rank them. Thus, this method gives the same importance to each metric used in the model. Getting a good placement in the final ranking requires to have high values for most metrics. This is the most restrictive method because just one low value can negatively impact on the collaboration aggregated value of the developer pair, decreasing their classification.

Meanwhile, **Borda Count** is not so restrictive as the CombSUM, but low metric values may also decrease the pair's classification. In this method, pairs with median values generally are better classified. For example, when Borda Count is applied to the voting context, if a candidate is the favorite of most electoral colleges (high metric values), but ignored by others (low metric values), such

Table VII. Centrality metrics for the top-10 best ranked pairs of developers from JavaScript.

CombSUM					**Borda Count**					**Condorcet**				
D1	D2	De	CC	PR	D1	D2	De	CC	PR	D1	D2	De	CC	PR
23	33	1.5	0.000	0.000	1	11	461.5	0.502	0.002	1	11	461.5	0.502	0.002
24	34	1.5	0.000	0.000	2	12	32.5	0.000	0.000	2	12	32.5	0.000	0.000
25	35	5.5	0.000	0.000	3	13	1148.5	0.004	0.005	3	13	1148.5	0.004	0.005
26	36	1433.5	0.007	0.005	4	14	91.0	0.509	0.000	21	22	114.0	0.024	0.000
27	37	2.5	0.000	0.000	5	15	43.5	0.220	0.000	4	14	91.0	0.509	0.000
28	38	1.5	0.000	0.000	6	16	31.0	0.175	0.000	5	15	43.5	0.220	0.000
29	39	2.0	0.000	0.000	7	17	473	0.253	0.002	7	17	473.0	0.253	0.002
30	40	1.5	0.000	0.000	8	18	245.5	0.264	0.001	6	16	31.0	0.175	0.000
31	41	1.5	0.000	0.000	9	19	161.5	0.506	0.001	9	19	161.5	0.506	0.001
32	42	1.5	0.000	0.000	10	20	910.5	0.336	0.003	8	18	245.5	0.264	0.001

Table VIII. Centrality metrics for the top-10 best ranked pairs of developers from Assembly.

CombSUM					**Borda Count**					**Condorcet**				
D1	D2	De	CC	PR	D1	D2	De	CC	PR	D1	D2	De	CC	PR
73	83	1.5	0.000	0.001	51	58	11.0	0.587	0.002	68	70	1.0	0.000	0.001
74	84	2.0	1.000	0.001	52	59	50.0	0.000	0.026	51	58	11.0	0.587	0.002
75	85	1.5	0.000	0.001	53	60	13.0	0.000	0.026	52	59	50.0	0.000	0.026
76	86	1.5	0.000	0.001	54	61	16.0	0.006	0.007	52	71	50.0	0.000	0.026
77	87	1.5	0.000	0.001	53	62	13.0	0.000	0.007	51	63	11.0	0.587	0.002
78	88	1.5	0.000	0.001	51	63	11.0	0.587	0.002	53	60	13.0	0.000	0.007
79	89	1.5	0.000	0.001	55	64	4.5	0.000	0.003	54	61	16.0	0.006	0.007
80	90	1.5	0.000	0.001	56	65	3.0	0.833	0.001	53	62	13.0	0.000	0.007
81	91	2.0	0.000	0.002	57	66	1.5	0.000	0.001	69	72	3.0	0.000	0.002
82	92	1.5	0.000	0.001	51	67	13.5	0.515	0.003	55	64	4.5	0.000	0.003

candidate's final placement is adversely affected [Morais and de Almeida 2012; Nurmi 1983]. Thus, pairs poorly evaluated by some factor receive a lower rating than the others. Then, to get a high aggregate value and a good placement, the pair cannot have very low values for any metrics used.

Regarding the ranking generated by **Condorcet Procedure** is more flexible than the others, despite being more complex computationally. Here, pairs with very high metric values (highlights in the evaluation by some of the metrics) can be well graded regardless of having low or zero values at some other evaluation point. In this way, we want to characterize pairs with strong relationships without necessarily requiring them to interact well with all the evaluated factors, since a pair with a long time interval of joint collaboration (metric GPC) may have few shared repositories (metric SR), but still have a strong relationship.

Indeed, we compare the results of each method with the ranking of the GitHub Award. However, there is no consensus about the rankings generated by the three methods when considering the top 1,000 developers and no one is significantly similar to the GitHub Award. This result can be explained because the ranking of GitHub Award is exclusively based on the number of stars from the users' repositories. Such stars are attributed by users who want to track the progress of the repository or discover similar projects in their news feed. Thus, the results show that the strength of collaboration is not related to this classification criterion. However, considering the top 10 best-ranked pairs of developers, there is a large similarity between the rankings of Borda count and Condorcet, as presented in Tables VII and VIII for JavaScript and Assembly, respectively. This indicates that the aggregation of tie strength metrics can be used to rank pairs of developers. Note the ranking of CombSUM is the different one, which may indicate that only adding up the values of the metrics is not the best way to rank pairs of developers.

Then, we analyze how developers involved in the best-ranked relationships are central in the network. To do so, we compute the metrics: nodes degree (De), clustering coefficient (CC) and PageRank

(PR). The results in Tables VII and VIII for JavaScript and Assembly, respectively, show that the Borda Count and Condorcet Procedure methods rank better pairs of developers who are more central, whereas CombSUM does not. Therefore, this reveals that developers who are hubs (an important node with many connections, i.e. central node) in the network of GitHub tend to have a strong relationship with their collaborators. It could be such developers are not central because they collaborate sporadically with different developers and lots of repositories; however, they are central because they intensively contribute to distinct repositories and thus collaborate with various developers.

5. CONCLUSION AND FUTURE WORK

The main contributions of this article are a systematic review over four distinct topics in the context of software repositories, an enriched dataset, a correlation analysis between tie strength metrics, and an analysis of ranking aggregation methods. The systematic review revealed that the topics team formation and ranking need to be more investigated over software repositories. Also, the correlation analysis allowed to reduce the number of tie strength metrics to be considered in an application from thirteen to five; i.e., there are eight metrics that do not bring new information about the strength of collaboration between developers. These five metrics were then considered in the three ranking aggregation methods: CombSUM, Borda Count and Condorcet Procedure. In general, Borda Count and Condorcet Procedure presented the best results to rank pairs of developers. All these results showed that tie strength can be used in the task of ranking.

As future work, we plan to move forward on the study of metrics to build a computational model for collaboration strength and to explore applications in specific contexts, such as team formation and recommendation. Also, we plan to continue expanding the dataset with more programming languages and collaboration metrics to support studies with different goals.

Acknowledgments. Work partially funded by CAPES, CNPq and FAPEMIG.

REFERENCES

Alves, G. B., Brandão, M. A., Marques, D., Silva, A. P. C., and Moro., M. M. The strength of social coding collaboration on github. In *Brazilian Symposium on Databases - Short Papers*. Salvador, Brazil, pp. 247–252, 2016.

Anvik, J., Hiew, L., and Murphy, G. C. Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering*. ICSE '06. ACM, New York, NY, USA, pp. 361–370, 2006.

Barabási, A.-L. and Albert, R. Emergence of scaling in random networks. *Science* 286 (5439): 509–512, 1999.

Bartusiak et al, R. Cooperation prediction in github developers network with restricted boltzmann machine. In *ACIIDS*. Vietnam, pp. 96–107, 2016.

Batista, N. A., Alves, G. B., Gonzaga, A. L., and Brandão, M. A. GitSED: Um Conjunto de Dados com Informações Sociais Baseado no GitHub. In *Braz. Symp. on Databases, Dataset Showcase Work.* pp. 224–233, 2017a.

Batista, N. A., Brandão, M. A., Alves, G. B., da Silva, A. P. C., and Moro, M. M. Collaboration strength metrics and analyses on github. In *Procs. Int'l Conf. on Web Intelligence*. Leipzig, Germany, 2017b.

Batista, N. A., Brandão, M. A., da Silva, A. P. C., and Moro, M. M. Aspectos Temporais para Medir a Força da Colaboração no GitHub. In *Brazilian Symposium on Databases - Short Papers*. pp. 234–239, 2017c.

Begel, A., DeLine, R., and Zimmermann, T. Social media for software engineering. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*. Santa Fe, New Mexico, USA, pp. 33–38, 2010.

Biazzini, M. and Baudry, B. "may the fork be with you": Novel metrics to analyze collaboration on github. In *Proceedings of the 5th International Workshop on Emerging Trends in Software Metrics*. WETSoM 2014. ACM, New York, NY, USA, pp. 37–43, 2014.

Bissyandé, T. F., Thung, F., Lo, D., Jiang, L., and Réveillère, L. Popularity, interoperability, and impact of programming languages in 100,000 open source projects. In *Proceedings of the 2013 IEEE 37th Annual Computer Software and Applications Conference*. COMPSAC '13. IEEE Computer Society, Washington, DC, USA, pp. 303–312, 2013.

Brandão, M. A. and Moro, M. M. Social professional networks: A survey and taxonomy. *Computer Communications* vol. 100, pp. 20 – 31, 2017.

Brandão, M. A. and Moro, M. M. Strength of co-authorship ties in clusters: a comparative analysis. In *Procs. of Alberto Mendelzon International Workshop on Foundation of Databases and the Web*. Montevideo, Uruguai, pp. 1–10, 2017a.

Brandão, M. A. and Moro, M. M. The strength of co-authorship ties through different topological properties. *Journal of the Brazilian Computer Society* 23 (1): 5, 2017b.

Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M., and Khalil, M. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of systems and software* 80 (4): 571–583, 2007.

Casalnuovo, C. et al. Developer onboarding in github: The role of prior social links and language experience. In *Procs. Joint Meeting on Foundations of Software Engineering.* Bergamo, Italy, pp. 817–828, 2015.

Cosentino, V., Izquierdo, J. L. C., and Cabot, J. A systematic mapping study of software development with github. *IEEE Access* vol. 5, pp. 7173–7192, 2017.

Dabbish, L. et al. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. In *Computer Supported Cooperative Work.* Seattle, USA, pp. 1277–1286, 2012.

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

Emerson, P. The original borda count and partial voting. *Social Choice and Welfare* 40 (2): 353–358, 2013.

Ganjisaffar, Y., Caruana, R., and Lopes, C. V. Bagging gradient-boosted trees for high precision, low variance ranking models. In *Procs. International ACM SIGIR Conference on Research and Development in Information Retrieval.* Beijing, China, pp. 85–94, 2011.

Grechanik, M., Fu, C., Xie, Q., McMillan, C., Poshyvanyk, D., and Cumby, C. A search engine for finding highly relevant applications. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1.* ICSE '10. ACM, New York, NY, USA, pp. 475–484, 2010.

Guendouz, M., Amine, A., and Hamou, R. M. Open source projects recommendation on github. In *Optimizing Contemporary Application and Processes in Open Source Software.* IGI Global, pp. 86–101, 2018.

Hahn, J., Moon, J. Y., and Zhang, C. Impact of social ties on open source project team formation. In *IFIP international conference on open source systems.* Springer, pp. 307–317, 2006.

Han, Y., Wan, Y., Chen, L., Xu, G., and Wu, J. Exploiting geographical location for team formation in social coding sites. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining.* Springer, pp. 499–510, 2017.

Kitchenham, B. Procedures for performing systematic reviews. *Keele, UK, Keele University* 33 (2004): 1–26, 2004.

Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., and Linkman, S. Systematic literature reviews in software engineering–a systematic literature review. *Info. and Software Technology* 51 (1): 7–15, 2009.

Majumder, A., Datta, S., and Naidu, K. Capacitated team formation problem on social networks. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* KDD '12. ACM, New York, NY, USA, pp. 1005–1013, 2012.

McDonald, N. and Goggins, S. Performance and participation in open source software on github. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems.* CHI EA '13. ACM, New York, NY, USA, pp. 139–144, 2013.

Mockus, A. Amassing and indexing a large sample of version control systems: Towards the census of public source code history. In *2009 6th IEEE International Working Conference on Mining Software Repositories(MSR)*. Vol. 00. pp. 11–20, 2009.

Morais, D. C. and de Almeida, A. T. Group decision making on water resources based on analysis of individual rankings. *Omega* 40 (1): 42 – 52, 2012.

Nurmi, H. Voting procedures: A summary analysis. *British Journal of Political Science* 13 (2): 181–208, 1983.

Rocha, L. M. A., Silva, T. H. P., and Moro, M. M. Análise da Contribuição para Código entre Repositórios do GitHub. In *Brazilian Symposium on Databases - Short Papers*. pp. 103–108, 2016.

Thung, F., Bissyande, T. F., Lo, D., and Jiang, L. Network structure of social coding in github. In *Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering.* CSMR '13. IEEE Computer Society, Washington, DC, USA, pp. 323–326, 2013.

Tsay, J., Dabbish, L., and Herbsleb, J. Influence of Social and Technical Factors for Evaluating Contribution in GitHub. In *International Conference on Software Engineering.* Hyderabad, India, pp. 356–366, 2014.

Vasilescu, B., Filkov, V., and Serebrenik, A. Perceptions of diversity on github: A user survey. In *Proceedings of the Eighth International Workshop on Cooperative and Human Aspects of Software Engineering.* CHASE '15. IEEE Press, Piscataway, NJ, USA, pp. 50–56, 2015.

Wan, Y., Chen, L., Xu, G., Zhao, Z., Tang, J., and Wu, J. Scsminer: mining social coding sites for software developer recommendation with relevance propagation. *World Wide Web*, 2018.

Wu, W., Zhang, W., Yang, Y., and Wang, Q. Drex: Developer recommendation with k-nearest-neighbor search and expertise ranking. In *Proceedings of the 2011 18th Asia-Pacific Software Engineering Conference.* APSEC '11. IEEE Computer Society, Washington, DC, USA, pp. 389–396, 2011.

Young, H. P. Condorcet's theory of voting. *American Political science review* 82 (4): 1231–1244, 1988.

Yu, Y., Wang, H., Yin, G., and Wang, T. Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment? *Information and Software Technology* vol. 74, pp. 204 – 218, 2016.