Software Quality Journal (2023) 31:211–241
https://doi.org/10.1007/s11219-022-09598-x

How do developers collaborate? Investigating GitHub heterogeneous networks

Gabriel P. Oliveira[1] · **Ana Flávia C. Moura**[1] · **Natércia A. Batista**[1] · **Michele A. Brandão**[2] · **Andre Hora**[1] · **Mirella M. Moro**[1]

Accepted: 1 August 2022 / Published online: 7 September 2022

Abstract
Assessing the collaboration among developers is important to understand different aspects of software lifecycle including code smell intensity, bug fixes, and software quality. This kind of collaboration can be obtained from social networks, which represent interactions between individuals in different contexts. In this paper, we model GitHub developers' collaborations in a heterogeneous network by considering three aspects: social collaboration, collaboration time in a repository and technical features. Then, we explore the GitHub network from different perspectives: size, relevance, and potential applications. The results show the considered metrics are not correlated, bringing new information about the collaborations. We also show that such information is useful for social developer ranking, an actual task which is often part of different applications, such as team formation, community detection and pair programming. Finally, as software quality is intrinsic to the people who code it, our methodology and analyses represent initial steps towards people-centered software quality analysis, as further discussed throughout this article.

Keywords Collaborative software development · Social coding · Social network metrics · Software quality · Mining software repositories

✉ Mirella M. Moro
mirella@dcc.ufmg.br

Gabriel P. Oliveira
gabrielpoliveira@dcc.ufmg.br

Ana Flávia C. Moura
anaciriaco@dcc.ufmg.br

Natércia A. Batista
natercia@dcc.ufmg.br

Michele A. Brandão
michele.brandao@ifmg.edu.br

Andre Hora
andrehora@dcc.ufmg.br

[1] Computer Science Department, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

[2] Instituto Federal de Minas Gerais, Ribeirão das Neves, Brazil

1 Introduction

GitHub has quickly become the main collaborative software development online tool. Historically, it was built to control code versioning but it soon evolved to sharing code (and other digital objects) for collaborative development and interaction, through features such as giving a *star* to a project, following *users*, opening *issues* and *pull requesting*. Then, there are many ways to explore GitHub for research and assessment purposes (Rahman & Roy, 2014; Silva & Valente, 2018; Bhasin et al., 2021), and social analyses by considering relationships between users who have collaborated within the same software project or repository (Batista et al., 2017), function (Joblin et al., 2017), or file (Meneely et al., 2008; Meneely & Williams, 2011; Avelino et al., 2017). All aforementioned works focus on *homogeneous* networks, that is, they capture only one type of relationship among the developers at a time. Still, GitHub offers other technical interactions that are software development oriented and could also be mapped to a network edge, e.g., creating a *pull request* and opening an *issue*. Indeed, when separating social features (e.g., following and stars) from such technical interactions (e.g., issues and pull requests), GitHub shows its multidimensional nature of the relationships (Leibzon, 2016). The problem then becomes *how to model such a rich set of social and technical interactions over the actual software engineer collaborations* (i.e., coding within the same context).

Our solution is to map different interactions to a *set* of edges, then forming a *heterogeneous* network, in which nodes represent developers and edges represent *distinct* interactions or collaborations. This type of networks has been largely studied in other contexts, such as blogging and image hosting (Li et al., 2020). Its main difference is to enable a uniform representation of features from both social and technical dimensions at the same time. In other words, considering different types of relationships simultaneously is not possible in a *homogeneous* network. This work builds upon a preliminary version of the heterogeneous modeling from Oliveira et al. (2018) by using its insights to better explore GitHub semantic properties and propose a real-world application using such a model. The solution seems rather simple, but it has specific challenges: (*i*) how to model distinct relationships more accurately within a unique social network; (*ii*) how to measure collaboration between any pair of developers given such new relationships; and (*iii*) how to evaluate such a model and metrics. The third challenge potentially has many solutions over a broad scope of evaluation techniques. Hence, we narrow down the options to three research questions regarding network analyses, metrics correlation, and how practical applications may benefit from our model and metrics. Overall, our contributions are summarized as follows.

- We propose a new heterogeneous network model for GitHub by integrating three aspects: social collaboration, time of collaboration in a repository, and technical features (Sect. 3.1). Specifically, we consider four technical and social features to model our network: issue, pull request, follower and star.
- We then propose five novel metrics for developers' relationships (Sect. 3.2): unidirectional assigned issues, unidirectional pull requests, bidirectional pull requests, bidirectional intensity of followers, and unidirectional intensity of stars. They allow to uncover new information about developers on Github by considering its specific aspects.
- We also generate networks for six programming languages (JavaScript, Python, Ruby, Assembly, Pascal, and Visual Basic — choices justified in Sect. 4.1.1) by mining millions of GitHub repositories. In summary, a previous analysis of net-

work properties of 12 programming languages reveals that the first three networks (JavaScript, Python, and Ruby) are more collaborative, whereas the others (Assembly, Pascal, and Visual Basic) are less collaborative. Thus, our study considers networks with two distinct levels of collaboration, high and low, which may reveal different behavior for our proposed collaboration metrics and better show the distinct values for the metrics on extreme scenarios of collaboration levels. We then extend an existing dataset with information extracted from such networks (Sect. 4.1.2).

– We propose three research questions to assess the new modeling and metrics, as follows.

1. *How do GitHub collaborations look like when considering heterogeneous networks?* By definition, heterogeneous networks represent distinct cooperation among developers, which may increase the size of the generated networks (Oliveira et al., 2018). Thus, as a first research question, we aim to assess the network size in terms of nodes, relationships and components (Sect. 5.1).
2. *How do the new metrics compare to existing ones over the new heterogeneous networks?* As we define metrics for novel aspects of the network, assessing their capability of uncovering new information about GitHub is paramount (Avelino et al., 2016; Batista et al., 2017). Therefore, this second research question is answered by evaluating the presence of collaboration of such new types, and analyzing correlation between existing social metrics and the new ones (Sect. 5.2).
3. *How can software companies and open-source communities benefit from the new heterogeneous modeling when considering an actual application?* Not always is more information better. However, when dealing with complex systems, such as social networks and collaborative software development, more information may be needed to represent users' relationships better (Leibzon, 2016; Li et al., 2020). This is the inspiration behind a heterogeneous model, to bring and consider more, relevant information. To assess the benefits of our model in GitHub, we evaluate the model and metrics for ranking collaborative developers (Sect. 5.3), and discuss potential applications towards software quality (Sect. 6). Regarding the ranking task, the heterogeneous-based ranking presented a high precision when compared to the homogeneous (base) one. Such results indicate that considering the novel types of interaction brings more benefits to the ranking task. Complementary, we discuss potential applications of the heterogeneous model in software quality and we observe that our studies can generate insights that may help developers, software engineering and project managers to make decisions.

2 Background and related work

In this section, we overview concepts from social networks (Sect. 2.1) that are important to understand our research and describe related work on GitHub (Sect. 2.2). Next, we go over studies that consider developer and community-related aspects to identify or assess software quality issues (Sect. 2.3). Then, we briefly discuss the main application considered here, ranking (Sect. 2.4).

2.1 Social network basics

Social Network Analysis has become a widely studied area, mostly due to the advances of technology and online platforms. There is a plethora of published material on the topic, which are impossible to cover in a simple paper section. Therefore, we now briefly define some terms used throughout our work.

Formally, a social network is modeled as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, in which $\mathcal{V}$ is the set of vertices (nodes) that represent individuals (e.g., friends, developers), and $\mathcal{E}$ is the set of non-directed edges that connect vertices of individuals who share a relationship. To qualify such relationships, there are metrics for the edges weight (also known as strength and tie strength), which can be topological (given by the network structure) or semantic (given by the relation meaning), as described next. In a collaborative software development network, considering time is also crucial to avoid false relationships, i.e., two developers working on a project in very distant time windows (Hong et al., 2011).

There are different metrics to qualify a relationship in a social network. The first three lines of Table 1 present some of the existing *topological* metrics that may be applied to different types of social network. For example, Brandão and Moro (2017) used the neighborhood overlap metric to the strength of collaboration in academic social networks.

2.2 GitHub social network

In this work, we focus on GitHub, the largest online platform for collaborative software development. GitHub (or similar proprietary tools) may be used for code review, bug assignment, quality inspection, and other tasks beyond version management (Yu et al., 2016). Moreover, it is used by students (or young developers) to learn how to code, get feedback from others, engage in open sourcing, then building a *developer portfolio* (Bhasin et al., 2021).

Studying GitHub structure and interactions reveals knowledge about the development process itself (Dalla Palma et al., 2020; Jiang et al., 2019; Nguyen et al., 2020), from conception to maintenance. As a platform, it allows different connections and interactions. Specifically, repositories can be forked and then joined by a pull request, which needs to be approved by an integrator user (Rahman & Roy, 2014). Over time, the number of such requests may increase, making it difficult for integrator users to approve. Hence, there are methods designed to filter and select more relevant requests (Jiang et al., 2019), then filtering out possible connections but increasing the effectiveness of the requests.

GitHub may be modeled as a graph: developers or team members (nodes) are connected (edges) with whom they collaborate in a repository, file or project. Such modeling is a *homogeneous* network, as it has only one type of nodes and edges. Then, different studies explore GitHub data for a variety of purposes, such as classifying relationships over time (Batista et al., 2017), understanding the repositories stars growth (Silva & Valente, 2018), investigating infrastructure-as-code properties by cataloging 46 metrics (Dalla Palma et al., 2020), and identifying core members of a project and their relation to its status (Leibzon, 2016).

A key factor in any of such studies is how to quantify the collaborations or relationships among developers. The solution usually includes proposing *semantic* metrics that are strict to a specific context, as presented in the bottom half of Table 1. For example, in the context of GitHub, a metric may consider the number of shared repositories, which is a

Table 1 Existing metrics for topological and semantic properties. For two distinct nodes X and Y, consider $\mathcal{N}(X)$ as the set of neighbors of X, $w(X)$ as the sum of the weights of all edges connected to X, and $w(X, Y)$ as the weight of the edge between X and Y. Also, let $\mathcal{R}$ be the set of all repositories in which both users X and Y collaborate

Topological properties	
Neighborhood Overlap (NO)	According to Easley and Kleinberg (2010), it is a measure of tie strength between nodes by considering their neighbor similarity, thus being defined as $NO_{(X,Y)} = \frac{\lvert\mathcal{N}(X)\cap\mathcal{N}(Y)\rvert}{\lvert\mathcal{N}(X)\cup\mathcal{N}(Y)-\{X,Y\}\rvert}$.
Preferencial Attachment (PA)	There is a linear relation between the number of neighbors of a node and the probability of connecting itself to another node (Barabási & Albert, 1999): $PA_{(X,Y)} = \lvert\mathcal{N}(X)\rvert\lvert\mathcal{N}(Y)\rvert$.
Adamic-Adar Coefficient (AA)	Defined by Adamic and Adar (2003), it gives more importance to neighbors that do not connect to many others: $AA_{(X,Y)} = \sum_{\forall Z\in\mathcal{N}(X)\cap\mathcal{N}(Y)} \frac{1}{log\lvert\mathcal{N}(Z)\rvert}$.
Weighted topological properties (Brandão & Moro, 2017)	
Tieness (T)*	It measures tie strength in co-authorship networks. In the GitHub collaboration network: $T_{(X,Y)} = \frac{\frac{\lvert\mathcal{N}(X)\cap\mathcal{N}(Y)\rvert+1}{1+\lvert\mathcal{N}(X)\cup\mathcal{N}(Y)-\{X,Y\}\rvert}\lvert\lvert w(X,Y)\rvert\rvert}{2}$, where $w(X, Y)$ is the normalized edge weight.
Semantic properties (Batista et al., 2017)	
Number of Shared Repositories (SR)	Defined as the number of repositories shared between two developers: $SR_{(X,Y)} = \lvert\mathcal{R}\rvert$.
Jointly Developers Contribution to Shared Repositories (JCSR)	Let $JCSR_{(X,Y,r_i)}$ be the ratio between the number of developers in the relationship and the total of developers in the repository r_i. Thus, $JCSR_{(X,Y)} = \frac{\sum_{\forall r_i\in\mathcal{R}} JCSR_{(X,Y,r_i)}}{\lvert\mathcal{R}\rvert}$.
Jointly Developers Commits to Shared Repositories (JCOSR)	Let $NC_{(X,r_i)}$ be the number of commits made by a developer X in a repository r_i and $NC_{(r_i)}$ the total of commits in r_i: $JCOSR_{(X,Y)} = \sum_{\forall r_i\in\mathcal{R}} \frac{NC_{(X,r_i)}+NC_{(Y,r_i)}}{NC_{(r_i)}}$.
Pevious Collaboration (PC)	Let $ND_{(r_i,t)}$ be the number of developers in a repository r_i at a time t. Hence, $PC_{(X,Y,t)}$ is defined as the amount of collaboration offered by X and Y at the time t: $PC_{(X,Y,t)} = \frac{\sum_{\forall r_i\in\mathcal{R}} \frac{1}{ND_{(r_i,t)}}}{\lvert\mathcal{R}\rvert}$. High values of the fraction $\frac{1}{ND_{(r_i,t)}}$ means that developer X is more likely to collaborate with Y and vice versa. Therefore, the value of PC is the average probability of X collaborate with Y for all repositories.
Global Potential Contribution (GPC)	Let $T_{(X,Y,r_i)}$ be the time interval in which the developers X and Y contribute to the repository r_i and $\mathcal{D}$ the set of all developers in the network: $GPC_{(X,Y)} = \frac{\sum_{\forall r_i\in\mathcal{R}} T_{(X,Y,r_i)}}{max_{\forall\{D_i,D_j\}\in\mathcal{D},r_i\in\mathcal{R}} T_{(D_i,D_j,r_i)}}$. In other words, GPC corresponds to the ratio between the sum of how many times X and Y have collaborated and the maximum lifetime over all project repositories

*Tieness can be used with each semantic metric of this table and the ones proposed in this work, whose weight is $w(X, Y)$

feature provided by GitHub platform; however, there is no use for such a metric in a *friendship* social network that does not provide any software-related feature (e.g., Instagram). Hence, this table includes existing metrics proposed in the context of GitHub-based social networks. Although limited to specific scenarios, semantic metrics are important to deeply assess properties and acquire information from the network. Then, proposing new metrics is relevant to better understand the patterns and behaviors in a network. Furthermore, the existing semantic metrics summarized here consider the amount of commits only, then leaving room to consider other aspects, further introduced in Sect. 3.2.

The aforementioned works have two common types of information by extracting features from: the nodes (i.e., individuals) and the edges that characterize their relationships. Such relationships can be analyzed in different levels, and GitHub allows to create specific contexts or ecosystems based on programming languages (Blincoe et al., 2015). Current studies focus on one coding language or a small group of languages, and analyzing such contexts uncover social and technical changes that affect the ecosystem or motivate developers to migrate to other ecosystems (Constantinou & Mens, 2017).

We build up over related work by introducing a solution to consider both social and technical interactions at the same time through a heterogeneous modeling of GitHub features. We also propose a set of new metrics for GitHub collaboration strength considering four technical and social features: *follow*, *issues*, *pull requests* and *stars*. Although such features have been studied *individually* (Anvik et al., 2006; Baysal et al., 2009; Borges et al., 2016; Lima et al., 2014; Yu et al., 2014a, b; Silva & Valente, 2018), they were not used to measure relationship (tie) strength on a collaboration-based developer network.

2.3 Developer and community-related quality issues

Research on software quality usually tackles either product or process quality by proposing or analyzing features assessed through one or more metrics. Simply put, product quality metrics focus on specification, design and code related features, such as size, complexity, cohesion, coupling, and encapsulation (Colakoglu et al., 2021), whereas process quality focuses on construction specification and development-related features, such as number of changes, developers, commits, active developers and their neighbors (Rahman & Devanbu, 2013). There is also a handful of works that tackle both product and process quality in the context of defect prediction (Majumder et al., 2020), and others that favor process metrics in such a context, e.g., Rahman and Devanbu (2013). For product quality, Colakoglu et al. (2021) present a recent systematic mapping that covers 70 studies published from 2008 on. There is also a plethora of work that focuses on distinct issues that jeopardize software quality, how to locate, detect and predict them, including smells (code, community, design), low truck factors, bugs, defects and faults, to name a few.

For example, Palomba et al. (2018) introduce community- and non-community-related properties to investigate and predict code smell intensity on GitHub — a perspective closely related to software quality. The non-community-related properties are more kin to our work because they represent the source code structure (e.g., lines of code) and the development process (e.g., total commits, class change process, developer-related factors and maintainability measures). Likewise, Tamburri et al. (2019) propose a tool called YOSHI (Yielding Open-Source Health Information) that considers properties such as commits, pull requests, comments, and new watchers to discover community patterns.

Then, Almarimi et al. (2020) evaluate machine learning algorithms to detect community smells in open-source software projects. Their experimental analyses consider more

than 70 projects from GitHub. Among different conclusions, we highlight: number of commits and developer per timezone, number of developers per community, and betweenness and closeness centrality are the most influential community features for detecting community smells. Likewise, while studying critical success factors, Garousi et al. (2019) also emphasize the influence of teams (or communities in a more abstract level). Specifically, beside project monitoring and controlling (i.e., management and organizational features), the expertise of a team with the task at hand and its experience with software development methodologies are central to software project success.

Overall, the aforementioned works are some examples of current research that shows the importance of social factors and the potential roles of developers while evaluating software quality. We note that most product related solutions are language dependent (Malhotra & Chug, 2013; Colakoglu et al., 2021), and not necessarily translate from one language or coding paradigm to another (Majumder et al., 2020; Rahman & Devanbu, 2013). In this sense, our work also collaborates with process-related features, as we focus on developers interactions and collaborations within code repositories. Moreover, being language-agnostic, our metrics are calculated for repositories coded in 12 of the most popular languages on GitHub, instead of just one — as done by Lenhard et al. (2019) and Madeyski and Jureczko (2015) for finding architectural inconsistencies and improving defect prediction models, for example.

2.4 Collaboration and its applications

Collaboration is inherent to human behavior, being essential to software engineering. Indeed, Bagley and Chou (2007) establish that collaboration helps inexperienced programmers to learn computer programming, and Singh (2010) reveals a correlation between small-world properties of a community and the success of the software developed by its collaborators. Regarding GitHub developer information and software quality, Çaglayan and Bener (2016) study the effect of collaborative coding on the quality of the final product, specially the impact on defect software proneness. The authors study how developer collaboration relates to software quality on source code level, and its possible reasons (developer's experience, developer's activities, and number of collaborations). Nonetheless, the authors consider the Git versions of only two software projects.

Going further, Wang et al. (2018) propose a system to recommend tags that help developers to find questions to easily solve their doubts, and Zhang et al. (2017) present a new strategy to detect similar repositories on GitHub. Nonetheless, understanding developers' relationships can help improve both approaches, as those with strong relationships tend to have similar contributing behavior. Then, studying the strength of collaboration can also help to improve project or pull request recommendation (Palomba et al., 2018; Yu et al., 2014b) by revealing the behavior of developers interactions, and indirectly, developers' behavior.

Furthermore, the social aspect is present in the software development process, as a developer's main priority at work is to solve problems, which is done by collaborating on projects (Batista et al., 2017) and gathering with other people as part of a software development team (Costa et al., 2020). Other types of relationships between developers are considered in a preliminary heterogeneous model to measure tie strength (Oliveira et al., 2018). In fact, a developer's social network is also relevant for the developer, as it may lead to referrals to new jobs and promotions (Singer et al., 2013; Sarma et al., 2016). These *social skills* are becoming increasingly important in the industry overall (Torres, 2015), which then impacts any recruiting process by software companies and

open-source communities. For example, a study over LinkedIn network shows the importance of being at the center of focus groups during a hiring process Jere et al. (2017) — such analysis could be easily adapted for working over a GitHub collaboration network. Finding the right developers may also be part of a solution to untangle or avoid quality issues such as community smells.

Overall, studying collaboration properties is relevant for other reasons as well. First, it can help in pair programming by identifying related individuals, given the benefits of such as strategy varies depending on their prior relationship and the nature of their exchanges. Second, such a study may reveal soft skills to recommend or rank developers, which are important to different job opportunities (Zhou et al., 2018; Montandon et al., 2021). Third, following such applications, recommendation and ranking algorithms may use collaborative filtering and then collect information from many users as well. At the end, our research may potentially help any of such studies and applications, as we advance many steps towards understanding and qualifying developers' interactions.

Specially, we use the new semantic metrics as descriptors of social skills to rank developers on GitHub. Finally, the proposed model and metrics enable to apply the new acquired knowledge into more interesting and complex applications, such as recommendation, team formation, software quality analysis, and community detection — which are part of our current and future research efforts (further connections between our work and software quality are discussed in Sect. 6).

3 GitHub heterogeneous network

We now propose a new heterogeneous model for collaborations in GitHub (Sect. 3.1) and semantic metrics for collaboration strength over such a network (Sect. 3.2).

3.1 Network modeling

We model the heterogeneous collaboration network for GitHub developers by starting from the homogeneous network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of Sect. 2.1. We call it the *base network*, in which developers (nodes) are linked (through an edge) when they contribute (i.e., make commits) to the same repository within a time interval. Note the relationships are defined at language level (a key feature of any repository), as it allows to understand the collaboration patterns in different coding languages. Figure 1(a) and (b) show the base network defined for five people who work on two repositories.

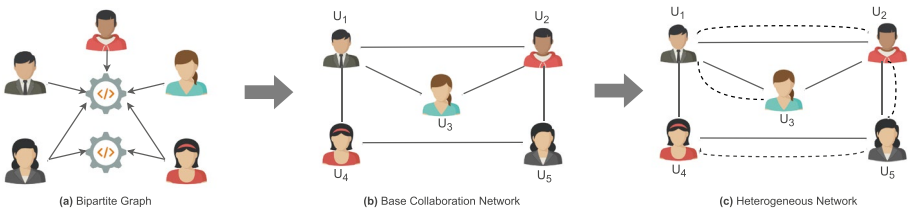


Fig. 1 Network construction from the bipartite graph (**a**) of developers and repositories. The base network (**b**) connects developers who collaborated in the same repository within a period of time (solid lines), while our heterogeneous model (**c**) incorporates the new technical interactions (dashed lines). There is no connection between {U3,U4} and {U3, U5}, as they have not collaborated within the same period of time

From the base network, our new *heterogeneous* modeling regards distinct types of edges for all potential interactions. The network is then represented by a multigraph $\mathcal{G}' = (\mathcal{V}, \mathcal{E}_1 \cup \mathcal{E}_2)$, where the set $\mathcal{V}$ still contains vertices for developers who commit to the same repository within a time interval. The novelty relies on the two edge sets $\mathcal{E}_1$ and $\mathcal{E}_2$, which represent (*i*) social collaboration in the same repository within a time period (from the base network, and it is mandatory because they are *collaborating* developers); and (*ii*) GitHub *technical* features for representing the creation of pull requests in the same repository, creation of pull requests in repositories that belong to another user, creation of issues assigned to another user, and GitHub *social* features for representing followers and favorite repositories (*stars*). Such GitHub features were chosen because they are relevant to software development and are available for most users and repositories (Anvik et al., 2006; Baysal et al., 2009; Borges et al., 2016; Lima et al., 2014; Silva & Valente, 2018; Yu et al., 2014b, a).

Also, each edge in the graph (pair of developers) may be qualified by a weight value. The weight of each set $\mathcal{E}_k$ of edges corresponds to the values of existing metrics for the base network and the new metrics on collaboration strength (Sect. 3.2) for the new interaction types. An example of heterogeneous modeling is presented in Fig. 1(c), which shows the same five users of GitHub and their relationships through repositories and other features.

Overall, the following list summarizes the new model main features and benefits:

- Each developer within the GitHub social network has at least one edge, the one from the base network; i.e., this person has collaborated on a repository with another person within the same time window.
- Each pair of developers may have other connections (besides the base one) regarding two GitHub technical features (*issues* and *pull requests*) and two social features (*followers* and *stars*).
- Having both set of edges allows to assess how collaborative each developer is; which, in turn, enables to assess and even rank developers or pairs of developers according to their level of collaboration, or collaboration strength.
- Having new types of relationships in the model requires new properties or metrics to quantify them, as introduced in the following section.

Figure 2 illustrates a toy example of all types of connections allowed in the new heterogeneous model. Each type of new edge adds information to the heterogeneous network, and we may call them *Semantic Properties* as well. When assessing this network collaborations, it is clear that the

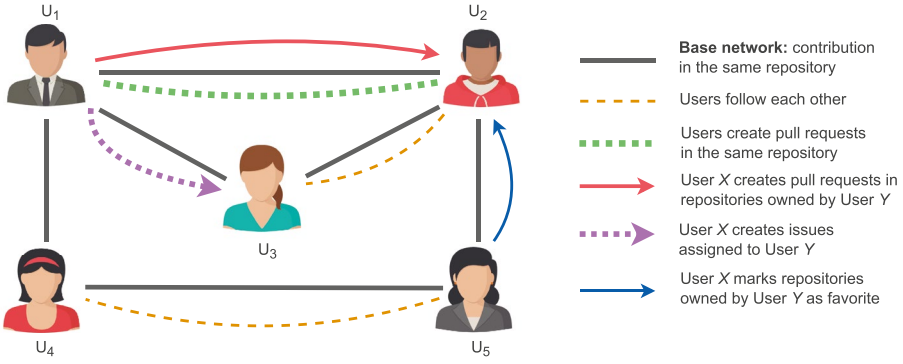


Fig. 2 Types of interactions considered in our proposed heterogeneous model

pair (U_1, U_4) is less collaborative than the pair (U_1, U_2), for example. Similar analyses may be performed at the node level. For instance, user U_2 is more collaborative than U_4, as the former has more types of collaborations with more people than the latter. Note the homogeneous network (as the base network) limits its analyses to the single type of edges (relationships) it contains. For example, it allows to assess that users U_1 and U_2 have more connections than the others.

3.2 New semantic properties

This section presents the new semantic properties to measure collaboration (tie) strength in the heterogeneous network by considering specific factors derived from the semantics of connections that GitHub offers. Note that most existing metrics either are generic to any kind of graph (e.g., Topological Properties from Table 1), i.e., they do not capture any software development-related behavior, or consider only the collaboration of users in the same repository (e.g., Semantic Properties from Table 1). Now, we improve the properties with the new types of relationships (previous section). Such new properties are separated according to the type of interaction considered, either technical (i.e., related to the code itself) or social (directly between two users). The inspiration for these metrics come from the social and technical features provided by GitHub (follow, star, issues, pull request) and their potential influence within the strength of collaboration between developers.

For all metrics, consider two distinct developers X and Y. We start by introducing the new semantic metrics that connect developers through two GitHub technical factors of software development: issues and pull requests.

Unidirectional assigned issues (UAI) Issues allow to keep track of tasks, enhancements and bugs,[1] all essential to quality assessment. Finding the right developer to handle an issue is challenging but may reduce the time to implement the change request (Anvik et al., 2006; Baysal et al., 2009). Therefore, this metric considers issues assigned from one user to another (i.e., unidirectional), as follows. Let $NI_{(X,Y)}$ be the number of issues created by user X that are assigned to user Y, and $TNI_{(Y)}$ the total number of issues designated to user Y. The metric for unidirectional assigned issues is:

$$UAI_{(X,Y)} = \frac{NI_{(X,Y)}}{TNI_{(Y)}}$$

This metric can be interpreted as the level of trust from one developer to another. Its value is in the range [0, 1], and the higher the UAI value, the greater the trust of X (or X's team) in Y to fix an issue. For example, in Fig. 1, if four out of 10 issues assigned to U_3 were created by U_1, then $UAI_{(U_1,U_3)} = \frac{4}{10} = 0.4$.

Unidirectional pull requests (UPR) Pull request is the primary method for coding contributions of developers in GitHub (Yu et al., 2014b). We understand that each pull request is a form of one person contributing with (usually) code to the repository of others. Hence, we measure such collaboration as follows. Let $PR_{(X,Y)}$ be the number of pull requests created by user X in repositories owned[2] by user Y, and $TPR_{(Y)}$ the total number

[1] GitHub Mastering Issues: https://guides.github.com/features/issues

[2] Following GitHub documentation, repositories owned by user accounts have one *owner*, and ownership permissions cannot be shared with another user account. Owners may also invite users on GitHub to their repositories as *collaborators*.

of pull requests of the repositories owned by user Y. The unidirectional metric for pull requests is:

$$UPR_{(X,Y)} = \frac{PR_{(X,Y)}}{TPR_{(Y)}}$$

In other words, this metric represents the level of cooperation from one user to another via pull requests. Its value is in the range [0, 1], and the more user X contributes to repositories owned by Y via pull requests, the higher the UPR value; i.e., higher values suggest more cooperative work of user X towards helping user Y. For example, if U_1 creates three pull requests in repositories belonging to U_2, and the total of pull requests in U_2's repositories is 10, then $UPR_{(U_1,U_2)} = \frac{3}{10} = 0.3$.

Bidirectional pull requests (BPR) UPR measures collaboration from one direction only. Still, having a pair of developers who *mutually* uses pull requests on each others' code makes for a stronger collaboration. In other words, UPR measures the cooperation via pull requests from user X to user Y unidirectionally, whereas BPR assesses the cooperative work between X and Y as a *reciprocal* work via a bidirectional relationship, as follows. Let $NPR_{(X,r)}$ be the number of pull requests created by user X in a repository r, $TNPR_{(r)}$ the total number of pull requests in r, and $\mathcal{U}$ the universe set of the existing repositories in the dataset. For each pair (X, Y) in a repository $r \in \mathcal{U}$, if $NPR_{(X,r)} \neq 0$ and $NPR_{(Y,r)} \neq 0$, the BPR metric is defined by:

$$BPR_{(X,Y)} = \sum_{\forall r_i \in \mathcal{U}} \frac{NPR_{(X,r_i)} + NPR_{(Y,r_i)}}{TNPR_{(r_i)}}$$

Considering the presented metrics so far, BPR is the only one that does not have values in the range [0, 1]. These values are not normalized because doing so causes loss of information on the strength of the relationships. The higher the BPR value, the more cooperative work exists between users X and Y. For example, for a repository r_1 in Fig. 1, two out of the 10 existing pull requests were created by U_1 and one by U_2. Then, in the repository r_2, there are 20 pull requests, five of them created by U_1 and 10 by U_2. Thus, $BPR_{(U_1,U_2)} = \frac{2+1}{10} + \frac{5+10}{20} = 0.3 + 0.75 = 1.05$.

Besides its many technical features for collaborative software development, GitHub offers two functions that are inspired by online social networks: followers and stars. As we aim to propose a heterogeneous network that explores different aspects within the social dimension, both followers and stars are evaluated in the next metrics.

Bidirectional intensity of followers (BIF) Developers follow other users on GitHub to receive notifications about their activity and to discover projects in their communities.[3] Thus, the social feature *follow* represents the interest of user X in the work of user Y (Lima et al., 2014; Yu et al., 2014a). Hence, we propose the following values to measure such intensity:

[3] Following People: https://help.github.com/en/articles/following-people

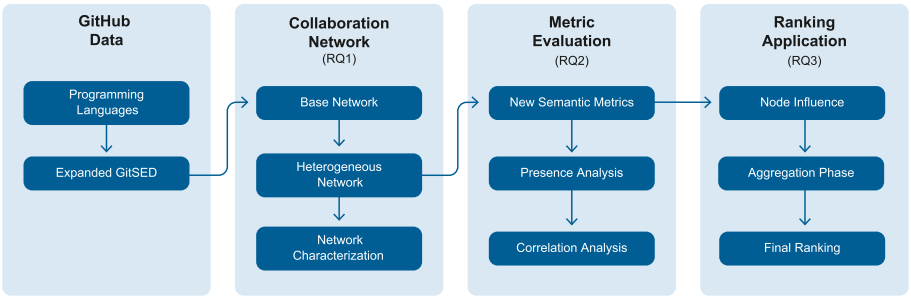


Fig. 3 Proposed study design with the main steps of our methodology

$$BIF_{(X,Y)} = \begin{cases} 1 & \text{if } X \text{ follows } Y \textbf{ AND } Y \text{ follows } X \\ 0.5 & \text{if } X \text{ follows } Y \textbf{ XOR } Y \text{ follows } X \\ 0 & \text{otherwise} \end{cases}$$

In other words, the intensity of followers is defined based on the relationship in which a user X follows a user Y in GitHub. In Fig. 1, if U_1 follows U_2 but U_2 does not follow U_1 back, the value of $BIF_{(U_1,U_2)}$ is 0.5. If U_4 follows and is followed by U_5, $BIF_{(U_4,U_5)} = 1$.

Unidirectional intensity of stars (UIS) Developers star repositories to keep track of projects they find interesting and to discover related content in their news feed.[4] In addition, the star amount also represents appreciation to a project and is an important measure that the community considers before using or contributing to a project (Silva & Valente, 2018) — note: *forks* are another social feature on GitHub that is strongly correlated with *stars* (Borges et al., 2016); hence, there is no new metric for it. Given its importance, intensity of stars is defined as follows. Let $NS_{(X,Y)}$ be the number of repositories owned by user Y in which X is interested (by adding a *star*), and $TNS_{(X)}$ the total number of repositories in which X is interested. The unidirectional intensity of stars is:

$$UIS_{(X,Y)} = \frac{NS_{(X,Y)}}{TNS_{(X)}}$$

In Fig. 1, consider that U_5 starred 50 repositories, 20 of which belong to U_2. Then, $UIS_{(U_5,U_2)} = \frac{20}{50} = 0.4$.

4 Evaluation design

In order to assess how the proposed heterogeneous network works, we use the evaluation design illustrated in Fig. 3.

The first step is to acquire data from GitHub. According to current statistics from its webpage, GitHub has more than 200 million repositories, 65 million developers, and 3

[4] GitHub stars: https://help.github.com/en/articles/saving-repositories-with-stars

million businesses and organizations worldwide (as of July 2021). As it is unfeasible to build one sole network for such huge volume of data, we study the networks of the most popular programming languages (Sect. 4.1.1). By collecting data from the repositories and users of such languages, we have extended an existing dataset to include the heterogeneous features as well, which enables to build the heterogeneous network (Sect. 4.1.2).

Having the data, the next steps of the methodology contain the design to answer each research question. For the first research question, we build the heterogeneous networks over the base network and characterize them in terms of size, relationships and components (Sect. 4.2). For the second question, we assess the presence of collaborations that belong to the new types, then analyze the correlation among the new types, and between them and the existing ones (Sect. 4.3). Finally, we apply the new heterogeneous model and metrics in a ranking of most collaborative developers (Sect. 4.4).

4.1 GitHub data

This section describes the considered programming languages to build the developer's collaboration network (Sect. 4.1.1) and presents the expanded version of GitSED (Sect. 4.1.2) with its properties.

4.1.1 Programming languages

Our research started by considering six selected programming languages[5] to perform proof of concept for initial metrics. Note that individually analyzing each programming language is also important for better characterizing the developers' collaborations and communities as well. Moreover, popular languages include not only coding languages (such as PHP and C) but also environment-oriented languages (such as R, Jupyter Notebook and VIM), Web design languages (such as CSS, HTML) and database languages (SQL and XML). With such broad purposes, we decided to focus on *coding* languages only.

Then, we decided to expand the initial dataset to the 12 most popular programming languages (communities) in the TIOBE Index[6] in March 2016[7] (month of our initial collecting processes): Assembly, C, C++, C#, Java, JavaScript, Pascal, Perl, PHP, Python, Ruby and Visual Basic. We do not consider the list of most used languages on GitHub because it changes pretty fast and the first official list was released only in the end of 2016.[8]

For each of them, the data collection process selected a sample of 1,000 repositories (sorted by their number of stars and forks) and all their developers. As initial characterization, we considered a bipartite graph with nodes for developers and repositories. Then, edges connect the developers to the repositories in which they have contributed for the source code. Hence, if a developer is connected to more than one repository, a bridge is formed between each pair of such repositories.

[5] The initial version of our research is published in Portuguese within a local venue (Rocha et al., 2016).

[6] TIOBE Index: https://www.tiobe.com/tiobe-index

[7] As of July 2021, most of such languages continue at the top 12, with exceptions of Ruby (17th), Perl (18th) and Pascal (20th). Their places in the top 12 are now filled with SQL (database language at 10th), Classic Visual Basic (at 11th), and R (environment-oriented at 12th). Hence, our analyses are still relevant for considering the most used coding languages.

[8] GitHub's state of Octoverse: https://octoverse.github.com/2016/

Table 2 Comparison of GitSED versions

	Original Version	Expanded Version
Data collection	Sep/2015	Jun/2019
Languages	3	6
Repositories	149,665	8,556,778
Developers	3,435,423	32,411,674
Metrics	9	13

We then performed a complete characterization of each language network based on classic metrics, including node degree and giant component size (i.e., the largest connected component of a network). From such analyses, we classified the 12 languages into four different groups, according to their metric value patterns: (*i*) *Top* — JavaScript, Python, Ruby; (*ii*) *High* — Java, PHP and C; (*iii*) *Medium* — C# and C++; and (*iv*) *Low* — Assembly, Pascal, Visual Basic and Perl. For instance, languages with a *Top* level of collaboration present a larger giant component and higher node degrees when compared to those with a Low level of collaboration. In other words, the *Top*-language communities (formed by developers who code in such language) are much more interconnected and diverse, which characterizes their collaborative nature. Therefore, we consider such levels when creating our dataset (next), as they may influence in our experimental results.

4.1.2 Expanded GitSED

This section has two parts: a summary of the dataset used to build the *base* network, and the new version obtained by adding novel information from the heterogeneous network proposed in this work.

First, the **original data** is extracted from GHTorrent (Gousios, 2013; Gousios et al., 2014),[9] which is then curated and enriched — called GitSED (Oliveira et al., 2021). It merges GHTorrent with new information obtained from the (homogeneous) collaboration-based network (i.e., only developers and their connections over the same repositories during the same time interval — Sect. 2.2). For the homogeneous network modeling, GitSED includes only real people as developers (i.e., changes in source codes are all made by personal accounts).

The original GitSED contains the main information about developers and repositories up to September 2015 and presents new features not available in GHTorrent, such as commit-related statistics for each developer and the time interval in which they contributed to a repository. It also provides the existing topological and semantic metrics calculated for each pair of developers (Sect. 2.1).

Second, the **new dataset** is an *expanded* version of GitSED that contains more features of six languages divided in two groups according to their level of collaboration (Sect. 4.1.1): *top* with **JavaScript**, **Ruby** and **Python**; and *low* with **Assembly**, **Pascal** and **Visual Basic**. The other languages (high and medium levels) present in-between performance and, hence, are not further presented here. Thus, the behavior of the networks of the two extreme groups of languages can be compared using the network metrics.

[9] GHTorrent is an offline repository of data collected through the GitHub REST API.

Using the same methodology for collecting and processing, GitSED was updated and considerably expanded with data obtained from GHTorrent until June 2019[10], as summarized in Table 2. Overall, the dataset has more recent data and is enhanced with the new semantic metrics proposed in this work, being publicly available as well (Oliveira et al., 2021).

4.2 Collaboration network (RQ1)

To answer our first research question (*How do GitHub collaborations look like when considering heterogeneous networks?*), we build GitHub collaboration networks for each programming language in GitSED (both top and low level of collaboration). We follow the model of Sect. 3.1, starting from the base networks, we add the new types of connection to evolve such networks into the proposed heterogeneous model. To assess the collaborations in each language, we perform a characterization analysis using the following classical network science concepts (Barabási, 2016).

- *Clustering Coefficient (CC)*: a measure of the degree to which nodes tend to cluster together;
- *Density*: percentage of the number of edges in a complete graph with the same number of vertices;
- *Giant Component (GC)*: the largest connected component of a given network that contains a finite fraction of the entire network's vertices; and
- *Node Degree*: number of edges connected to a given node.

4.3 Metric evaluation (RQ2)

The second research question (*How do the new metrics compare to existing ones over the new heterogeneous networks?*) aims to verify if the new semantic metrics proposed for the heterogeneous networks (Sect. 3.2) bring relevant information about developers' relationships when compared to the existing ones. To answer such a question, we divide our analyses into two main parts: presence in the network and correlation analysis.

Presence analysis The five proposed metrics quantify the new relationships added to the heterogeneous networks. Following their definition, we may assume that if a metric value is zero for two developers, the relationship it captures does not exist between them. For example, if $BIF(A, B) = 0$, A and B do not follow each other. Therefore, to verify the presence of the new relationships in our networks, we count the numbers of developers (node) and pairs (edges) that remain in the network when we remove edges that do not have a specific relationship (i.e., its metric value is zero).

Correlation analysis In statistics, correlation is a metric that reports how two or more variables are related. It can be represented by a numeric coefficient whose value varies between -1 and 1 (negative and positive correlation, respectively). Here, we use such coefficients to verify whether our new metrics provide new information on the relationships

[10] June 2019 is the most recent available dump at the time of defining our model. As of July 2021, there are only two versions newer than it: July 2020 and March 2021.

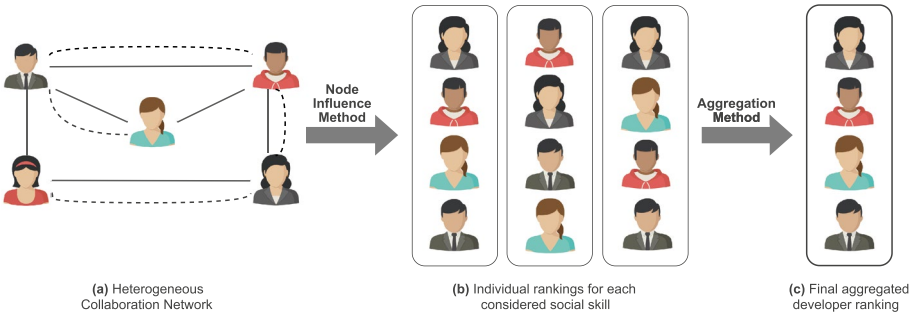


Fig. 4 Developer ranking workflow. From the heterogeneous network (**a**), we rank developers for each considered metric/social skill (**b**), and then aggregate such rankings into one complete version (**c**)

when compared to the existing ones (Table 1). We calculate the two most used correlation coefficients: Pearson (r), which measures the linear relationship between the metric values; and Spearman (ρ), which measures the monotonic correlation, i.e., uses the order of the data (rank) instead of the values themselves. We do the same analysis considering the set of new metrics itself in order to show their independence.

4.4 Ranking application (RQ3)

Regarding RQ3 (*How can anyone benefit from the new heterogeneous modeling when considering an actual application?*), we present a ranking of the most collaborative developers as a real-world application of our model and metrics. As collaboration is the core of our methodology, we can easily map the semantic properties (existent and proposed ones) from the GitHub context into technical and social skills desired by the industry and open-source communities. Examples of such skills include: (*i*) collaborating in many repositories (metrics SR/JCSR/JCOSR); (*ii*) collaborating for a considered amount of time (metrics GPC/PC); (*iii*) suggesting improvements to other people's code (metrics UPR/BPR); (*iv*) reporting bugs and problems (metric UAI); (*v*) being aware of current work in their community (metric UIS); and (*vi*) following other people to track their activity (metric BIF).

Here, we propose a simple yet powerful approach to build a ranking of most collaborative developers in GitHub. Such a model considers collaboration as the key aspect for ranking and the semantic properties to measure the aforementioned social skills. Hence, a collaborative developer means a person who has a mature (above average) level for all skills according to the metrics that represent them. We perform such ranking by following a predefined workflow illustrated in Fig. 4 and explained as follows.

In the GitHub context, we build developer rankings for every considered programming language, as each language has its own network. Such granularity is very useful for companies and open-source communities looking for developers with high expertise in a given language. Therefore, for the ranking based on the heterogeneous network, we select six GitHub semantic metrics that represent the considered types of interaction in our modeling (i.e., relevant social skills in the development process): (*i*) collaboration itself — SR; (*ii*) period of collaboration — GPC; (*iii*) following users — BIF; (*iv*) issue assignment — UAI; (*v*) pull request creation — UPR and BPR (each uncovers a distinct facet of this process); and (*vi*) starring repositories — UIS. For the homogeneous-based ranking, we only consider (*i*) and (*ii*) since the other interaction types are not possible in the base network.

From the heterogeneous network, we first build a separate ranking for each metric using PageRank, an influence-based node ranking algorithm (Brin & Page, 1998). Then, we combine all rankings into a final one by using two distinct aggregation strategies: Borda (Emerson, 2013) and Condorcet (Young, 1988). Borda count is based on the general idea of a winner by consensus, i.e., a developer is well ranked when well-positioned in all rankings. The ranking is calculated through the distribution of points for each individual. In contrast, the Condorcet procedure compares the developers' positions in each ranking one by one. Therefore, the best developer is the one who beats everyone else in all rankings.

Evaluating our rankings requires to select a target ranking, which stands for a real developer ordering. To the best of our knowledge, there are no existent approaches for ranking most collaborative developers per language considering the social aspects. Thus, we collect the Git Awards ranking[11], a project that ranks GitHub developers through the number of stars of their own repositories, i.e., repositories that must have software with recognized quality. With that target set, we are now able to evaluate our model using two well-known ranking metrics: Average Precision (AP) and Normalized Discounted Cumulative Gain (NDCG) (Aggarwal, 2016). In our analyses, we do not present the Mean Average Precision (MAP) because we want to evaluate each language separately for comparison purposes. However, it is easily obtained from the AP values for all languages.

- *Average Precision (AP@k)*: This metric informs if the produced ranking recommended relevant (relevance here is binary) items at a cut-off k, also considering their order. The higher its value, the more accurate the recommendation (i.e., the ranked developers are relevant and are placed in the correct order).
- *Normalized Discounted Cumulative Gain (NDCG@k)*: The goal of this metric is similar to AP, but NDCG is able to deal with a relevance scale, which means that it is now possible to measure the relevance with numbers, not only with a binary variable, as AP does. Therefore, higher NDCG values represent better developer recommendations.

5 Exploratory analysis and evaluation

Next, to answer RQ1 and RQ2, we present evaluations over two perspectives: characterization of the collaboration networks of each language (Sect. 5.1) and analyses of the new semantic properties (Sect. 5.2). We then answer RQ3 by presenting a ranking of most collaborative developers as a real-world application of the new heterogeneous modeling and metrics (Sect. 5.3).

5.1 Characterization of the heterogeneous networks (RQ1)

Analyzing different network properties allows to better understand the behavior of collaborations between developers in each programming language. Hence, we now discuss the properties of the heterogeneous networks for six languages: the three with top level of collaboration, and the three with low level of collaboration. The other languages (high and medium levels) do not present any outstanding performance that would justify further analyses here. Table 3 presents such a characterization with: number of repositories

[11] Git Awards: https://github.com/vdaubry/github-awards

Table 3 General characterization and statistics on the networks of each language

	Top Level of Collaboration			Low Level of Collaboration		
	JavaScript	**Python**	**Ruby**	**Assembly**	**Pascal**	**Visual Basic**
# of repositories	4,649,111	2,343,276	1,376,993	31,860	16,867	38,971
# of nodes	1,152,467	711,232	305,980	9353	4241	7419
# of pairs (edges)	5,262,491	7,633,287	103,637,391	18,843	8894	11,359
Density (10^{-3})	0.008	0.030	2.214	0.431	0.989	0.413
Average Degree	9.13	21.46	677.41	4.03	4.19	3.06
Average CC	0.37	0.39	0.48	0.35	0.37	0.31
Nodes on GC	43.35%	46.21%	62.80%	5.74%	16.95%	2.41%
Pairs on GC	88.06%	90.58%	99.88%	21.89%	29.62%	17.82%

collected for each language, number of nodes within its heterogeneous network, number of pairs of developers who are connected (through a number of edges that is potentially greater than one), density, average node degree,[12] average clustering coefficient (CC),[13] number of nodes in the giant component (GC), and number of pairs of developers in the giant component.

The more collaborative networks (left side of Table 3) have many nodes and edges in their giant components. Also, all such GCs have more than 80% of the network edges. In other words, these networks are well connected. Moreover, although the JavaScript network has the largest number of nodes and repositories, it has the smallest number of edges, lowest density and average node degree. This result indicates its developers do not cooperate as much with each other as the developers of the other two collaborative languages (Python and Ruby). On the other hand, Ruby is the language with the lowest number of nodes and repositories in the network, but it has almost 20 times more edges than JavaScript. Such behavior is justified by the high average degree of nodes and the highest density of the networks. Therefore, Ruby developers collaborate a lot with each other. Finally, almost all edges of the Ruby network belong to its GC; i.e., it is *the most collaborative language* studied here.

The less collaborative languages (right side of Table 3) have similar behavior in their networks: all have a low average node degree and a low percentage of edges in the GC. Thus, the repositories of such languages are mostly composed by few developers who collaborate little among themselves. Also, higher density for such languages when compared to JavaScript and Python means their networks are more complete.[14] As such languages have much fewer developers, their connections may represent higher density values. Besides, as the order of magnitude of such metric for all languages (except Ruby) is the same, we cannot affirm that the behavior of such languages is different based only on the density value.

[12] The average node degree is calculated over all nodes in the network.

[13] The average CC is calculated over all nodes in the network.

[14] A graph is complete when all its nodes are connected by a unique edge.

Table 4 Participation in the network from selected metrics (absolute and percentage values)

Metric	Developers				Pairs			
	JavaScript		Assembly		JavaScript		Assembly	
UAI	51,907	4.5%	132	1.41%	56,418	0.54%	98	0.26%
UPR	114,260	9.91%	462	4.94%	96,548	0.92%	331	0.88%
BPR	208,469	18.09%	1286	13.75%	828,157	7.87%	2640	7.01%
BIF	178,957	15.53%	2486	26.58%	194,769	1.85%	1736	4.61%
UIS	84,180	7.3%	472	5.05%	82,284	0.78%	314	0.83%
Size	**1,152,467**		**9,353**		**10,524,982**		**37,686**	

UAI Unidirectional Assigned Issues, *UPR* Unidirectional Pull Requests, *BPR* Bidirectional Pull Requests, *BIF* Bidirectional Intensity of Followers, *UIS* Unidirectional Intensity of Stars

Summary of RQ1 *How do GitHub collaborations look like when considering heterogeneous networks?* (1) Ruby is the most collaborative language, (2) JavaScript developers cooperate less than Python and Ruby ones, and (3) Assembly, Pascal, and Visual Basic developers collaborate little among themselves. This analysis may bring to light good practices that should be fomented by those communities (e.g., high collaboration) and harmful ones that should be avoided (e.g., high isolation).

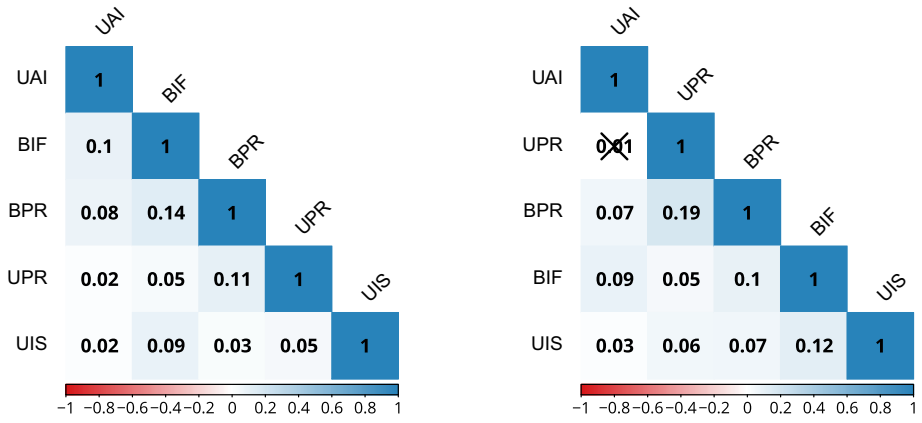
5.2 Analyses of the new semantic properties (RQ2)

To answer RQ2, we now assess the new metrics on two different perspectives: a filtered analysis focusing on the presence of the new relationships in the heterogeneous networks (Sect. 5.2.1), and two correlation analyses among the proposed metrics (Sect. 5.2.2) and against existing metrics (Sect. 5.2.3).

5.2.1 Presence of collaboration types in the heterogeneous networks

To assess the proportion of developers and pairs for each of the four features and their metrics in the whole network, we perform a filter per metric (related to a relationship). Table 4 shows the number of remaining developers and pairs when we remove edges that do not have a specific metric. For example, after filtering the JavaScript network by the UAI metric, 4.5% of the developers and 0.54% of the pairs remain in relation to the complete network.

Overall, the results show few nodes and edges remain in the network for metrics with social factors (BIF and UIS); i.e., most GitHub developers do not consider such functionalities when collaborating in a repository. Regarding technical metrics (UAI, UPR and BPR), there are fewer nodes and edges, because a large part of issues and pull requests in the repositories comes from external users, i.e., non-collaborators. Such behavior may occur because, most probably, they are made by the software (or application) users or developers who forked the base repository. As forks are not considered in our modeling, they are not accounted for in the relationships based on technical features.



(a) JavaScript **(b)** Assembly

Fig. 5 Pearson correlation analysis among the new metrics. Values are in the range $[-1, 1]$, in which -1 indicates a complete negative correlation, 1 stands for a total positive correlation, and zero means no linear correlation. Values marked with a cross are not statistically significant (p-values ≥ 0.05). Obs: UAI, Unidirectional Assigned Issues; UPR, Unidirectional Pull Requests; BPR, Bidirectional Pull Requests; BIF, Bidirectional Intensity of Followers; UIS, Unidirectional Intensity of Stars. **a** JavaScript. **b** Assembly

5.2.2 Correlation analysis of new metrics

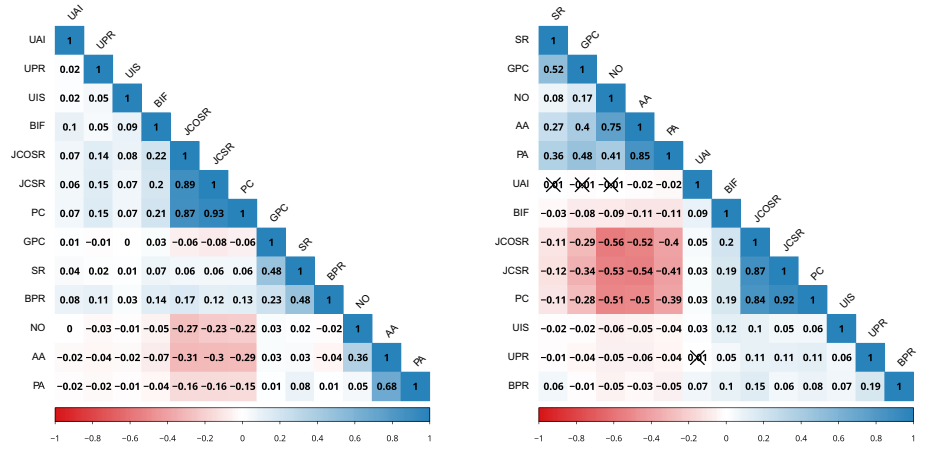
Now, we verify the correlation among the new metrics to assess their independence, or ability to provide new information. Figure 5 presents the correlation matrix between the five new metrics for JavaScript and Assembly networks.

For both languages, the correlation between the new metrics is small or insubstantial, with values close to zero. Such result is somehow expected, as each metric considers different factors to calculate the interaction between pairs of developers. The small correlation between UPR and BPR is justified as: UPR considers only a fraction of the total set of pull requests (those created in repositories of a given user); whereas BPR considers all the repositories in the dataset and analyzes the number of pull requests made by the pairs of developers. The low correlation between the metrics is a strong indicator that the proposed semantic properties add new information to the GitHub collaboration network. Therefore, they can all be considered for measuring the strength of such relationships.

5.2.3 Correlation with existing metrics

This section analyzes the correlation between the set of new metrics and properties from the GitHub state-of-the-art; i.e., we show that our proposed metrics bring new information about developers' collaboration. The results are similar for both correlation coefficients and for all programming languages, with some exceptions. Figure 6 presents the Pearson correlation matrices for JavaScript and Assembly.

There is no significant linear or monotonic correlation between most properties. A significant difference between the more and less collaborative languages is the existence of negative correlation between JCOSR with GPC, PA, AA and NO for Assembly, Pascal and Visual Basic. Such a result indicates that in less collaborative languages, the relationships

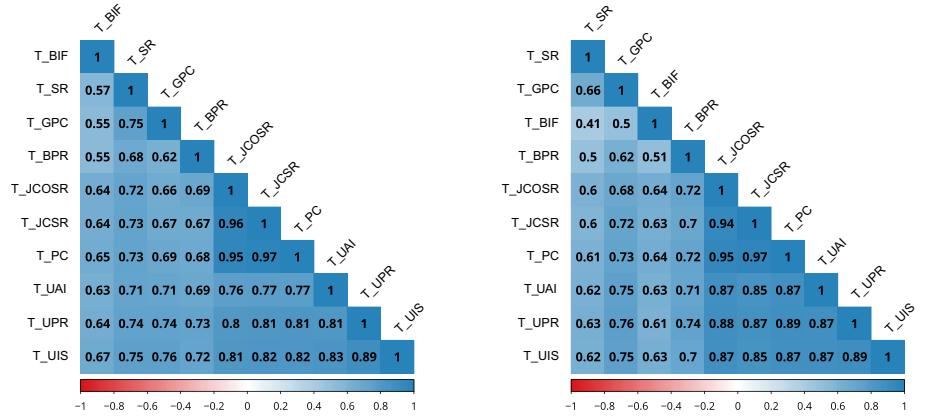


(a) JavaScript **(b)** Assembly

Fig. 6 Pearson correlation analysis among the existing metrics and the proposed ones. Values marked with a cross are not statistically significant (p-values ≥ 0.05). Obs: JCOSR, jointly developers commits to shared repositories; JCSR, jointly developers contribution to shared repositories; PC, previous collaboration; GPC, global potential contribution; SR, number of shared repositories; NO, neighborhood overlap; AA, Adamic-Adar coefficient; PA, preferencial attachment. **a** JavaScript. **b** Assembly

of a pair of developers with their neighbors negatively influence the joint contribution by commits. On the other hand, the negative correlation between JCOSR and GPC indicates that pairs of developers tend not to contribute to the same repository for long period. In turn, this result is an insight on a possible high turnover of developers, which may jeopardize the quality of those products with low truck factor or any community smell.

Additionally, Fig. 7 shows a strong correlation between *Tieness* weighted by all the studied metrics. *Tieness* allows to combine a topological property with a semantic one by considering it as the weight in its formula. The high correlation indicates that *Tieness* with



(a) JavaScript **(b)** Assembly

Fig. 7 Pearson correlation analysis among Tieness weighted with all semantic metrics. Values are statistically significant (p-values < 0.05). **a** JavaScript. **b** Assembly

different weights brings the same information to the analysis of tie strength, and thus, we may choose one to measure such force. In this case, it is better to use metrics with low computational cost, such as T_BIF or T_SR (combination of T with BIF and T with SR, respectively). These results reinforce that the location of a pair relative to their neighbors strongly influences their relationships.

Summary of RQ2 Finally, as an example of application, the new metrics may be used to represent technical and social skills desired by the industry and open-source communities, such as collaborating in repositories, suggesting improvements to code, reporting bugs and problems, and so on. Evaluating such skills is fundamental for both developers and companies. For the first, they may lead to referrals to new jobs and promotions, whereas for the latter, they help building a highly collaborative team.

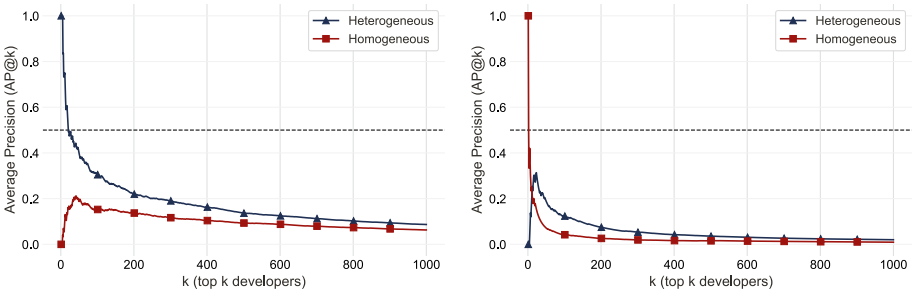
How do the new metrics compare to existing ones over the new heterogeneous networks? The proposed metrics represent new information about developers' collaboration. Our new metrics are then potentially useful to applications that must represent such collaborations in a more realistic way, e.g., by aggregating more information.

5.3 Using model and metrics within an application (RQ3)

Improving only technical skills does not guarantee success in the software development industry. Thus, in this section, we rank developers through social skills as a real-world application of the novel heterogeneous modeling and metrics (*RQ3*). We build and evaluate such a ranking by following the workflow of Sect. 4.4 and compare the results obtained from the heterogeneous with the homogeneous networks.

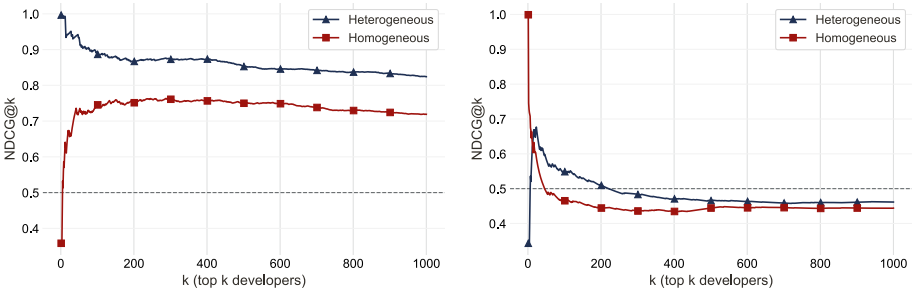
In all considered languages, both rankings (Borda and Condorcet) are highly correlated (using both Pearson and Spearman coefficients), with $\rho > 0.9$. Therefore, we are able to select only one method without losing meaningful information. We then choose Borda count, as it is much more simple and computationally less expensive.

The ranking evaluation for JavaScript and Assembly using AP@k and NDCG@k is presented in Figs. 8 and 9, respectively. We choose $k = 1,000$ and set as relevant all developers who appear on the top k ranking of Git Awards. The relevance metric for



(a) JavaScript **(b)** Assembly

Fig. 8 Results of Average Precision (AP@k) for the top 1000 recommended developers for **a** JavaScript and **b** Assembly



(a) JavaScript **(b)** Assembly

Fig. 9 Normalized Discounted Cumulative Gain (NDCG@k) for the top 1000 recommended developers for **a** JavaScript and **b** Assembly

NDCG is set as the *rank_score* for each user. The *rank_score* of a user i is obtained as $rank\_score(i) = max\_rank - rank(i) + 1$, where *max_rank* is the lowest rank of the list and *rank(i)* is the rank of the user.

The evaluation for the JavaScript ranking (Figs. 8a and 9a) shows the ability to identify well the top 20 developers ($AP@k \geq 0.5$ and $NDCG@k \geq 0.9$), as in this case, collaborative developers (in terms of social skills) are also between the most popular on Git Awards. However, when increasing the value of k, precision decreases, probably because social skills (the core of our method) are not always related to popularity, which is the Git Awards methodology.

Regarding less collaborative languages, such as Assembly, the gap between our ranking and Git Awards is higher. Despite having a lower average precision, the *NDCG* close to 0.7 when $k = 20$ reveals that the developers who are recommended by our ranking are indeed popular. Overall, when comparing more versus less collaborative languages, we note the more collaborative the language, the higher the relation between social skills and popularity. In fact, the less collaborative ones have fewer developers and repositories (i.e., a smaller community), then affecting their amount of stars.

In general, precision for the heterogeneous-based ranking is higher when compared to the homogeneous (base) one for both languages. Such results emphasize the proposed network model brings information to the ranking task, since they reflect the importance of social skills in classifying developers. The exception is the Top 10 developers for Assembly (Figs. 8b and 9b), in which the homogeneous-based ranking performs best. As aforementioned, smaller communities do not present a strong relationship between social skills and popularity. Moreover, the low precision values show that our ranking is *not* similar to Git Awards for high values of k. This is expected, since both rankings have different methodologies. As future work, we plan to assess such ranking with actual teams; i.e., to verify if team members agree with the rankings produced.

Finally, we compare the two rankings in terms of the social skills considered in our heterogeneous modeling: followers, repositories, issues, pull requests, and stars. Table 5 shows that our model succeeds in capturing developers with higher average issues and pull requests and stars. The number of followers and repositories are more related to the developer's popularity, which explains the higher averages on Git Awards. Moreover, considering that social skills are crucial to make a good developer, our ranking approach manages to combine all such factors in a simple and easily understandable model. Therefore, such a novelty

Table 5 Comparison between our proposed ranking method and Git Awards considering the main social features within GitHub. Values are the average number of followers, repositories, issues, pull request and stars calculated from all ranked users

	Top 50 Developers				**Top 1,000 Developers**			
	JavaScript		**Assembly**		**JavaScript**		**Assembly**	
	Our Ranking	Git Awards	Our Ranking	Git Awards	Our Ranking	Git Awards	Our Ranking	Git Awards
Followers	4999.5	**6764.8**	191.4	**284.1**	1102.4	**1483.5**	79.4	**201.5**
Repositories	80.6	**91.8**	**9.9**	9.5	**46.4**	37.4	6.3	**7.9**
Issues	**1047.8**	451.9	**156.6**	100.0	**529.7**	245.5	**86.6**	66.0
Pull requests	**2378.9**	1584.4	**281.7**	127.7	**984.0**	522.6	**161.5**	102.6
Stars	**782.3**	539.4	**228.5**	105.8	**492.9**	482.6	104.3	**206.3**

is shown to be relevant and necessary to find and recommend the best people for software projects, bringing benefits for both software companies and open-source communities.

Summary of RQ3 *How can software companies and open-source communities benefit from the new heterogeneous modeling when considering an actual application?* Ranking developers from collaboration metrics sheds light on the importance of the social aspect in the software development process, as the ability to solve problems does not rely on technical skills only. Besides, finding the most collaborative developers benefit both software companies and open-source communities, as this process assures the recommended developers are also socially collaborative people. Moreover, having a highly qualified and collaborative team improves the development process, resulting in software products with higher quality.

6 Potential connections to software quality

In practice, there are many ways to assess and improve software quality. Most approaches include identifying, assessing, predicting and solving issues related to software that jeopardize quality. For example, Meneely et al. (2008) argue that few failure-predicting models consider a key cause of failures, i.e., *people*, and the reliability of final products could be explained by understanding the structure of *people collaboration*, i.e., how developers collaborate within common files. Later, Meneely and Williams (2011) corroborate metrics based on social network analysis do represent socio-technical collaborations in open-source projects. This work builds up on such literature and introduces a model that allows to measure different aspects of developers' collaboration. The next step is to combine our model to existing software quality metrics, which will then allow to properly compare ours to the aforementioned work. In other words, *we focus on developers and the way they collaborate in common repositories — technical, and interact in a collaborative software development platform — social.*

How may software quality-oriented approach that considers technical and social features from the developers collaboration network benefit from the new network model and respective metrics? For example, Meneely et al. (2008) propose a simple approach based on topological metrics over developer networks for predicting failures. Such an approach could be tailored to work over our heterogeneous model. The main challenge would be to adapt the model to work at code level, instead of repository. Likewise, approaches that focus on the developers' communities and their interactions as a team may also benefit. For example, Almarimi et al. (2020) define eight issues as community smells; finding and qualifying some of them (specially organizational silo effect and truck factor) could also benefit from our work, as our metrics add information relevant to such issues.

First, the organizational silo effect is defined as the presence of highly decoupled subgroups of developers within the same repository (Almarimi et al., 2020). In our modeling, they can be detected by comparing a combination of metrics over the base network and the new metrics. An example is finding a group of 10 developers in the base network which is clearly divided in two decoupled groups when considering the new technical and social metrics. Second, the *truck factor* (TF) issue is a metric for the number of people who concentrate knowledge in software development environments (Almarimi et al., 2020; Avelino et al., 2016). Our novel metrics Unidirectional Assigned Issues (UAI) and Unidirectional

Pull Requests (UPR) both point to developers who are potentially central do the project: those who are able to solve issues (UAI) and to those who have solved issues or contributed in anyway with the project (UPR). Such information may be used to tailor or improve any TF metric, and even to help identify who those developers captured by TF are.

Furthermore, we evaluate our model and metrics to developer rankings. Both applications are relevant to current software quality techniques, mostly those that predict issues. Specially, temporal classification may be adapted to work for detecting community smells, and developer rankings for assessing truck factors — both are relevant issues that jeopardize quality, as explained by Almarimi et al. (2020).

7 Threats to validity

Generalization of the results We analyze millions of projects hosted on GitHub, which is currently the most popular social coding platform. We cover six programming languages as JavaScript and Assembly, and therefore our results only apply to such language communities within GitHub. Nonetheless, as usual in empirical software engineering, our findings cannot be directly generalized to other social coding platforms (e.g., GitLab) nor to projects written in other programming languages.

Base network In a collaboration network, relationships may be created when developers modify the same software entity, for example, project or repository (Batista et al., 2017) and function (Joblin et al., 2017). In this study, the base network represents developers who make commits to the same repository. Thus, the generated network models the interactions of developer at repository level. Such a design choice may include developers who work in distinct modules in a large project; but, they may not be considered collaborators as they do not interact *directly*. Therefore, further analysis can be performed to cover collaboration at lower levels of granularity, such as when developers modify the same function or file.

Collaboration dependence We focus on how other technical and social properties impact on the strength of relationships between GitHub developers. The relationships here are based on collaboration (through commits), meaning that a pair of developers must have collaborated for them to be connected (base network). Only after that, the other types of interactions are analyzed. Therefore, a limitation is the model does not consider developers who have not coded in GitHub but rather interacted in other ways (e.g., following each other *only*). Further adaptations could be developed to consider such relationships that do not depend on collaboration.

Aggregation method The proposed model for ranking the best developers is simple and efficient, but some limitations on the aggregation method can deeply influence its results. For instance, the fact that Borda Count penalizes an individual who is not very well ranked in any of the rankings may strongly affect their final position. For the human resources office of a software company, the popularity factor (measured by BIF and UIS) may not be so relevant, which means that even if someone is not well ranked in this regard, they can still be a good candidate for a job position. Thus, further improvements must consider weighting the rankings at aggregation.

Ranking assessment To evaluate the proposed model, we use Git Awards as a target ranking. However, the difference between the date of such ranking (collected in 2020) and ours

(obtained from GitSED 2019) may impact our results. For example, developers who were not relevant in 2019 may become popular in 2020 or the opposite. Nonetheless, as our goal is only to present a real application of the new network modeling and metrics, we believe that our evaluation is still worthwhile.

8 Conclusion and future work

This paper introduced a heterogeneous model for the developers network on GitHub and five new semantic metrics for collaboration strength, which allow to evaluate networks built from more and less collaborative programming languages. The analysis of such metrics revealed a lack of linear and monotonic correlation between them, i.e., they all represent new information about the relationships. Other conclusions include: pairs of developers with intense joint contribution in repositories may not connect with many other developers; and the location of a pair of developers relative to their neighbors influences their relationships.

Further, we used the new metrics in a social model to rank the most collaborative developers, and the findings revealed that the new interaction types can capture developers with a high collaboration strength and well-developed social skills in the network. Finally, we also found that social interactions on GitHub may contribute to more intense collaborations and also affect the number of repository stars. Such findings emphasize the relevance of the social aspect in the software development process, benefiting not only the software companies but also the Software Engineering community. For example, developers may invest in certain skills to improve their status within their communities (i.e., programming languages), and companies may use the social developer ranking to build highly qualified and socially collaborative team.

As future work, we plan to investigate the relation between the semantic properties and the influence of developers in GitHub, as well as to explore the relation between semantic properties and process-related quality metrics. Furthermore, we plan to assess our ranking approach with actual teams; i.e., to verify if the rankings produced are in accordance with the real team members.

Author contribution Gabriel P. Oliveira: conceptualization, methodology, software, formal analysis, investigation, writing — original draft, visualization. Ana Flávia C. Moura: software, investigation, data curation. Natércia A. Batista: conceptualization, methodology, investigation, writing — original draft. Michele A. Brandão: conceptualization, methodology, formal analysis, investigation, writing — original draft. Andre Hora: conceptualization, validation, writing — review and editing supervision. Mirella M. Moro: conceptualization, resources, writing — review and editing, supervision, project administration, funding acquisition.

Funding This work was supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) and Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Brazil.

Availability of data and material Our dataset, called GitHub Socially Enhanced Dataset (GitSED), is publicly available in Zenodo (Oliveira et al., 2021).

Code availability All relevant information related to this work is available in Project Apoena homepage: https://bit.ly/proj-apoena

Declarations

Conflict of interest The authors declare no competing interests.

References

Adamic, L. A., & Adar, E. (2003). Friends and neighbors on the Web. *Social Networks, 25*(3), 211–230. https://doi.org/10.1016/S0378-8733(03)00009-1

Aggarwal, C. C. (2016). Recommender Systems - The Textbook. *Springer*. https://doi.org/10.1007/978-3-319-29659-3

Almarimi, N., Ouni, A., & Mkaouer, M. W. (2020). Learning to detect community smells in open source software projects. *Knowledge-Based Systems, 204*, 106201. https://doi.org/10.1016/j.knosys.2020.106201

Anvik, J., Hiew, L., & Murphy, G. C. (2006). Who should fix this bug? In *International Conference on Software Engineering* (pp. 361–370). Shanghai, China. https://doi.org/10.1145/1134285.1134336

Avelino, G., Passos, L., Hora, A., & Valente, M. T. (2016). A novel approach for estimating truck factors. In *Int'l Conf. on Program Comprehension* (pp. 1–10). IEEE Computer Society. https://doi.org/10.1109/ICPC.2016.7503718

Avelino, G., Passos, L., Hora, A., & Valente, M. T. (2017). Assessing code authorship: The case of the Linux kernel. In *International Conference on Open Source Systems (OSS)* (pp. 151–163). Buenos Aires, Argentina. https://doi.org/10.1007/978-3-319-57735-7_15

Bagley, C. A., & Chou, C. C. (2007). Collaboration and the importance for novices in learning java computer programming. *SIGCSE Bulletin, 39*(3), 211–215.

Barabási, A. L. (2016). Network science. Cambridge University Press.

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

Batista, N. A., Brandão, M. A., Alves, G. B., da Silva, A. P. C., & Moro, M. M. (2017). Collaboration strength metrics and analyses on GitHub. In *Proceedings of the International Conference on Web Intelligence* (pp. 170–178). Leipzig, Germany.

Baysal, O., Godfrey, M. W., & Cohen, R. (2009). A bug you like: A framework for automated assignment of bugs. In *International Conference on Program Comprehension* (pp. 297–298). Vancouver, Canada. https://doi.org/10.1109/ICPC.2009.5090066

Bhasin, T., Murray, A., & Storey, M. D. (2021). Student experiences with github and stack overflow: An exploratory study. In *IEEE/ACM Int'l Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)* (pp. 81–90). IEEE, Madrid, Spain. https://doi.org/10.1109/CHASE52884.2021.00017

Blincoe, K., Harrison, F., & Damian, D. (2015). Ecosystems in github and a method for ecosystem identification using reference coupling. In *IEEE/ACM 12th Working Conference on Mining Software Repositories* (pp. 202–211). https://doi.org/10.1109/MSR.2015.26

Borges, H., Hora, A., & Valente, M. T. (2016). Understanding the factors that impact the popularity of GitHub repositories. In *IEEE International Conference on Software Maintenance and Evolution* (pp. 334–344). https://doi.org/10.1109/ICSME.2016.31

Brandão, M. A., & Moro, M. M. (2017). The strength of co-authorship ties through different topological properties. *Journal of the Brazilian Computer Society, 23*(1). https://doi.org/10.1186/s13173-017-0055-x

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer Networks, 30*(1–7), 107–117. https://doi.org/10.1016/S0169-7552(98)00110-X

Çaglayan, B., & Bener, A. B. (2016). Effect of developer collaboration activity on software quality in two large scale projects. *Journal of Systems and Software, 118*, 288–296.

Colakoglu, F. N., Yazici, A., & Mishra, A. (2021). Software product quality metrics: A systematic mapping study. *IEEE Access, 9*, 44647–44670. https://doi.org/10.1109/ACCESS.2021.3054730

Constantinou, E., & Mens, T. (2017). Socio-technical evolution of the ruby ecosystem in github. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering* (pp. 34–44). Klagenfurt, Austria. https://doi.org/10.1109/SANER.2017.7884607

Costa, A., et al. (2020). Team formation in software engineering: A systematic mapping study. *IEEE Access, 8*, 145687–145712. https://doi.org/10.1109/ACCESS.2020.3015017

Dalla Palma, S., et al. (2020). Towards a catalogue of software quality metrics for infrastructure code. *Journal of Systems and Software*, p 110726.

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

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

Garousi, V., Tarhan, A., Pfahl, D., Coşkunçay, A., & Demirörs, O. (2019). Correlation of critical success factors with success of software projects: an empirical investigation. *Software Quality Journal, 27,* 429–493. https://doi.org/10.1007/s11219-018-9419-5.

Gousios, G. (2013). The GHTorrent Dataset and Tool Suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories* (pp. 233–236).

Gousios, G., et al. (2014). Lean GHTorrent: GitHub data on demand. In *11th Working Conference on Mining Software Repositories* (pp. 384–387). Hyderabad, India. https://doi.org/10.1145/2597073.2597126

Hong, Q., et al. (2011). Understanding a developer social network and its evolution. In *IEEE 27th International Conference on Software Maintenance, ICSM* (pp. 323–332). IEEE Computer Society. https://doi.org/10.1109/ICSM.2011.6080799

Jere, S., Jayannavar, L., Ali, A., & Kulkarni, C. (2017). Recruitment graph model for hiring unique competencies using social media mining. In *Proceedings of the International Conference on Machine Learning and Computing* (pp. 461–466). Singapore. https://doi.org/10.1145/3055635.3056575

Jiang, J., et al. (2019). Who should make decision on this pull request? analyzing time-decaying relationships and file similarities for integrator prediction. *Journal of Systems and Software, 154*, 196–210. https://doi.org/10.1016/j.jss.2019.04.055

Joblin, M., et al. (2017). Classifying developers into core and peripheral: An empirical study on count and network metrics. In *Proceedings of the 39th International Conference on Software Engineering* (pp. 164–174). Buenos Aires, Argentina. https://doi.org/10.1109/ICSE.2017.23

Leibzon, W. (2016). Social network of software development at GitHub. In *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining* (pp. 1374–1376). San Francisco, USA. https://doi.org/10.1109/ASONAM.2016.7752419

Lenhard, J., Blom, M., & Herold, S. (2019). Exploring the suitability of source code metrics for indicating architectural inconsistencies. *Software Quality Journal, 27*, 241–274. https://doi.org/10.1007/s11219-018-9404-z

Li, H., et al. (2020). Privacy leakage via de-anonymization and aggregation in heterogeneous social networks. *IEEE IEEE Transactions on Dependable and Secure Computing, 17*(2), 350–362. https://doi.org/10.1109/TDSC.2017.2754249

Lima, A., Rossi, L., & Musolesi, M. (2014). Coding together at scale: Github as a collaborative social network. In *Proceedings of the Eighth International Conference on Weblogs and Social Media* (pp. 295–304). Ann Arbor, USA.

Madeyski, L., & Jureczko, M. (2015). Which process metrics can significantly improve defect prediction models? an empirical study. *Software Quality Journal, 23,* 393–422. https://doi.org/10.1007/s11219-014-9241-7.

Majumder, S., Mody, P., & Menzies, T. (2020). Revisiting process versus product metrics: a large scale analysis. CoRR abs/2008.09569. https://arxiv.org/abs/2008.09569

Malhotra, R., & Chug, A. (2013). An empirical study to redefine the relationship between software design metrics and maintainability in high data intensive applications. In *Proceedings of the World Congress on Engineering and Computer Science* (vol. 1).

Meneely, A., & Williams, L. (2011). Socio-technical developer networks: Should we trust our measurements? In *Proceedings of the International Conference on Software Engineering* (pp. 281–290). Honolulu, USA. https://doi.org/10.1145/1985793.1985832

Meneely, A., et al. (2008). Predicting failures with developer networks and social network analysis. In *ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 13–23). Atlanta, USA. https://doi.org/10.1145/1453101.1453106

Montandon, J. E., et al. (2021). What skills do IT companies look for in new developers? A study with stack overflow jobs. *Information and Software Technology, 129*, 106429. https://doi.org/10.1016/j.infsof.2020.106429

Nguyen, P. T., Rocco, J. D., Rubei, R., & Ruscio, D. D. (2020). An automated approach to assess the similarity of github repositories. *Software Quality Journal, 28*(2), 595–631. https://doi.org/10.1007/s11219-019-09483-0

Oliveira, G. P., Batista, N. A., Brandão, M. A., & Moro, M. M. (2018). Tie strength in gitHub heterogeneous networks. In *Brazilian Symposium on Multimedia and the Web* (pp. 363–370). https://doi.org/10.1145/3243082.3243101

Oliveira, G. P., Moura, A. F. C., Batista, N. A., Brandão, M. A., & Moro, M. M. (2021). GitSED: GitHub Socially Enhanced Dataset. https://doi.org/10.5281/zenodo.5021329

Palomba, F., et al. (2018). Beyond technical aspects: How do community smells influence the intensity of code smells? *IEEE Transactions on Software Engineering*.

Rahman, F., & Devanbu, P. T. (2013). How, and why, process metrics are better. In D. Notkin, B. H. C. Cheng, & K. Pohl (Eds.). *International Conference on Software Engineering, IEEE Computer Society* (pp. 432–441). https://doi.org/10.1109/ICSE.2013.6606589

Rahman, M. M., & Roy, C. K. (2014). An insight into the pull requests of github. In *ACM 11th Working Conference on Mining Software Repositories* (pp. 364–367).

Rocha, L. M. A., et al. (2016). Análise da Contribuição para Código entre Repositórios do GitHub. In *Brazilian Symposium on Databases - Short Papers* (pp 103–108).

Sarma, A., et al. (2016). Hiring in the global stage: Profiles of online contributions. In *11th IEEE International Conference on Global Software Engineering* (pp. 1–10). Orange County, CA, USA. https://doi.org/10.1109/ICGSE.2016.35

Silva, H., & Valente, M. T. (2018). What's in a GitHub star? understanding repository starring practices in a social coding platform. *Journal of Systems and Software, 146,* 112–129. https://doi.org/10.1016/j.jss.2018.09.016.

Singer, L., et al. (2013). Mutual assessment in the social programmer ecosystem: an empirical investigation of developer profile aggregators. In *Computer Supported Cooperative Work* (pp. 103–116) San Antonio, TX, USA. https://doi.org/10.1145/2441776.2441791

Singh, P. V. (2010). The small-world effect: The influence of macro-level properties of developer collaboration networks on open-source project success. *ACM Transactions on Software Engineering and Methodology*, *20*(2), 6:1–6:27.

Tamburri, D. A., et al. (2019). Discovering community patterns in open-source: a systematic approach and its evaluation. *Empirical Software Engineering, 24*(3), 1369–1417.

Torres, N. (2015). Technology is only making social skills more important. *Harvard Business Review*, pp August 26, 2015.

Wang, S., et al. (2018). Entagrec ++: An enhanced tag recommendation system for software information sites. *Empirical Software Engineering, 23*(2), 800–832. https://doi.org/10.1007/s10664-017-9533-1

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

Yu, Y., Wang, H., Yin, G., & Wang, T. (2016). Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment? *Information and Software Technology, 74*, 204–218. https://doi.org/10.1016/j.infsof.2016.01.004

Yu, Y., et al. (2014a). Exploring the patterns of social behavior in github. In *International Workshop on Crowd-based Software Development Methods and Technologies* (pp. 31–36). https://doi.org/10.1145/2666539.2666571

Yu, Y., et al. (2014b). Reviewer recommender of pull-requests in github. In *International Conference on Software Maintenance and Evolution* (pp. 609–612). Victoria, Canada. https://doi.org/10.1109/ICSME.2014.107

Zhang, Y., et al. (2017). Detecting similar repositories on github. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering* (pp. 13–23). Klagenfurt, Austria. https://doi.org/10.1109/SANER.2017.7884605

Zhou, C., Kuttal, S. K., & Ahmed, I. (2018). What makes a good developer? an empirical study of developers' technical and social competencies. In *IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC* (pp. 319–321). Lisbon, Portugal. https://doi.org/10.1109/VLHCC.2018.8506577

Gabriel P. Oliveira received the M.Sc. degree from the Computer Science Graduate Program at Universidade Federal de Minas Gerais (UFMG). He received the B.Sc. degree in computer science from UFMG in 2018. He is currently a member of the Laboratory of Interdisciplinary Computer Science (CS+X) and his research interests include data science and social network analysis, with a strong emphasis on data analysis in collaborative domains.

Ana Flávia C. Moura is a BSc student in the Computer Science Department at the Federal University of Minas Gerais (UFMG), Brazil. Her research interest is focused on social network analysis, such as data analysis in collaborative domains.

Natércia A. Batista is a M.Sc. student in the Computer Science Graduate Program at Universidade Federal de Minas Gerais (UFMG). She received bachelor's degree in Information Systems from UFMG in 2017. Currently, she is a member of the Laboratory of Interdisciplinary Computer Science (CS+X) and her main research interests include data analysis, social networks and collaborative metrics.

Michele A. Brandão is a Professor at IFMG. She finished her postdoctoral in the Graduate Program in Computer Science at the Federal University of Minas Gerais (UFMG) in 2018, and has received PhD and MS in Computer Science at UFMG, Bachelor in Computer Science at the Universidade Estadual de Santa Cruz (UESC, Bahia). Her main research interests include data science, social networks and digital forensics.

Andre Hora is a Professor in the Computer Science Department at the Federal University of Minas Gerais (UFMG), Brazil. His research interests include software evolution, software repository mining, and empirical software engineering. He earned his PhD in Computer Science from the University of Lille, France. Webpage: www.dcc.ufmg.br/~andrehora.

Mirella M. Moro is Professor at the Computer Science department at UFMG (Belo Horizonte, Brazil). She holds a Ph.D. in Computer Science (University of California Riverside - UCR, 2007), and MSc and BSc in Computer Science as well (UFRGS, Brazil). She was a member of the ACM Education Council and the Education Director of the Brazilian Computer Society (SBC). She is currently a Council Member of the SBC and part of the SBC Meninas Digitais (Digital Girls) Steering Committee. Her research interests include data-driven research, social analysis, gender diversity, and education in Computer Science. She is also an advocate for increasing women participation in Computer Science, coordinating projects such as BitGirls.