# Analyzing Rich-Club Behavior in Open Source Projects

**Mattia Gasparini**
Politecnico di Milano
Milano, Italy
mattia.gasparini@polimi.it

**Javier Luis Cánovas Izquierdo**
**Robert Clarisó**
Universitat Oberta de Catalunya
Barcelona, Spain
{jcanovasi,rclariso}@uoc.edu

**Marco Brambilla**
Politecnico di Milano
Milano, Italy
marco.brambilla@polimi.it

**Jordi Cabot**
ICREA - UOC
Barcelona, Spain
jordi.cabot@icrea.cat

## ABSTRACT

The network of collaborations in an open source project can reveal relevant emergent properties that influence its prospects of success. In this work, we analyze open source projects to determine whether they exhibit a *rich-club* behavior, *i.e.*, a phenomenon where contributors with a high number of collaborations (i.e., strongly connected within the collaboration network) are likely to cooperate with other well-connected individuals. The presence or absence of a rich-club has an impact on the sustainability and robustness of the project.

For this analysis, we build and study a dataset with the 100 most popular projects in GitHub, exploiting connectivity patterns in the graph structure of collaborations that arise from commits, issues and pull requests. Results show that rich-club behavior is present in all the projects, but only few of them have an evident club structure. We compute coefficients both for single source graphs and the overall interaction graph, showing that rich-club behavior varies across different layers of software development. We provide possible explanations of our results, as well as implications for further analysis.


*OpenSym'19, August 20–22, 2019, Skövde, Sweden*

ACM ISBN 978-1-4503-6319-8/19/08…$15.00
https://doi.org/10.1145/3306446.3340825

## CCS CONCEPTS

• **Software and its engineering → Open source model**; **Programming teams**.

## KEYWORDS

open source, network analysis, rich-club coefficient, GitHub

**ACM Reference Format:**
Mattia Gasparini, Javier Luis Cánovas Izquierdo, Robert Clarisó, Marco Brambilla, and Jordi Cabot. 2019. Analyzing Rich-Club Behavior in Open Source Projects . In *OpenSym'19: 15th International Symposium on Open Collaboration, August 20–22, 2019, Skövde, Sweden.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3306446.3340825

## 1 INTRODUCTION

GitHub is the most popular service to develop and maintain open source projects: it allows to create public Git repositories, to modify the code through *commits*, to send *pull requests* updating other users' repository or to notify them about *issues* in the code. Each user interacts with many other users in the project development process and these interactions define collaboration networks, that can be exploited to describe software projects. Studying collaboration networks helps in discovering properties that influence the success of a project, possibly giving interesting insights. In particular, we are interested at the *mesoscopic* level of analysis of networks. At this level the main target is to identify relevant connectivity patterns, which may reveal emergent properties that stem from the way in which the nodes in the network interact.

It is in this context that the very well-known property **rich-club behavior** [4] arises as an interesting structural signature to study. This behaviour reflects the tendency of well-connected nodes (*i.e.*, hubs) to interact with other well-connected nodes.

The rich-club behavior has been studied in many diverse domains: scientific collaboration networks [10], migration flows [15], brain connectivity tissue [14], air transportation [4] or the Internet topology [16]. And we believe that a similar behaviour can occur in software development communities too. More precisely, if we consider a network representing collaborations between developers, a rich-club behaviour may apply when developers collaborate mostly with the same fixed subset of other important colleagues, instead of spreading the cooperation to each component of the team.

The availability of GitHub data through its public API [2] and other services (GHArchive [1]) introduces the possibility to study in detail how rich-club is mapped into open source software development.

The aim of this paper is to analyze the interactions networks of the most popular software development projects available on GitHub, in order to verify the presence of the rich-club behaviour and its potential implications for open source projects.

We build a data collection process to integrate the commit data coming from Git and the activity data coming from GitHub into a graph structure suitable for our purpose, which considers users as nodes and collaborations on project's activities as edges. We then computed the rich-club coefficient for three kind of networks: issues-based, pull-requests-based and commits-based. We verify how the rich-club coefficient changes when all the interactions are combined in a collaboration *supergraph*. Our results reveal that the rich-club behavior is present in all the supergraphs, but only few of them have an evident club structure. We compute coefficients also for single source graph, showing that rich-club behavior varies across different layers of software development. Moreover, we manually compare two projects with very different rich-club behavior with respect to external data of the projects as a further step of validation.

The paper is structured as follows: in Section 2 the state of the art is presented; in Section 3 the rich-club coefficient is described in detail; in Section 4 the methodology to build the network and to compute the coefficients are shown, while in Section 5 discussion about results follows; Section 6 and 7 briefly describe treats to validity and replicability package provided and the paper concludes in Section 7 with possible future works.

## 2 STATE OF THE ART

The presence or absence of a rich-club effect in open source projects has not been studied in a systematic way and has not been applied to a large dataset as the one that GitHub can now provide.

The work [12] studies the phenomenon for a specific project in SourceForge (the *Azureus*, now *Vuze*, bittorrent client), but it only considers file-level information, ignoring the users collaborating on those files. Similarly, [5] also includes this phenomenon as part of its study of a single FLOSS community.

GitHub has been studied only as a single global community [9]. As part of this study the presence of a rich-club effect is checked, but only across the whole GitHub social network which brings little information to the impact of this effect for individual or closely related projects.

Finally, [13] covers a number of open source communities but restricts the analysis to email exchanges among the participants.

The access to GitHub data allows to extend these analysis both considering new dynamics (issues and pull-requests events) providing information rich enough to perform the analysis at the single project level, providing new insights about a behavior that has only been studied on the surface.

## 3 RICH-CLUB COEFFICIENT

The rich-club coefficient was firstly introduced in [16] as a non-normalized metric dependent of a degree $k$[1]:

$$\phi(k) = \frac{2E_k}{N_k(N_k - 1)} \tag{1}$$

where $N_k$ is the number of nodes with degree greater or equal to $k$ and $E_k$ are the number of edges between these nodes.

Intuitively, $\phi(k)$ measures how far the set of nodes with degree $k$ is from being a complete subgraph, *i.e.*, a *clique*. The value of $\phi(k)$ ranges from 0 (all nodes are disconnected) to 1 (a clique), with higher values showing a stronger rich-club behavior in the network.

However, this coefficient tends to increase as the network grows [4], so a null model is needed to normalize the previous formula. Normalization is given by:

$$\rho(k) = \frac{\phi(k)}{\phi_{random}(k)} \tag{2}$$

where $\phi(k)$ is computed using Equation 1 and $\phi_{random}(k)$ is the coefficient computed for the same $k$ for a random network with the same degree distribution as the original model. This coefficient $\rho(k)$ provides a non-negative number where the value 1 is the baseline: if $\rho(k) > 1$, then the rich-club behavior of the network is above that of a random case.

## 4 RESEARCH METHOD

In this section we discuss how our study has been set up. We first describe how we constructed the dataset for our study, then how we built the graphs for the analysis and finally how we calculated the rich-club coefficient.

[1]The degree of a node is the number of edges incident to that node.

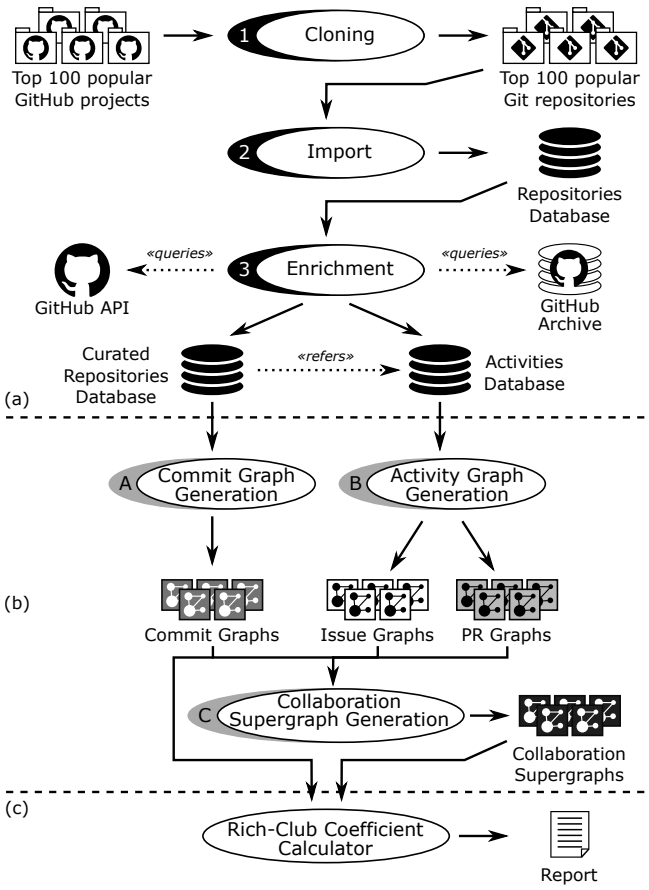


**Figure 1: Process followed for (a) the construction of the dataset, (b) the graph generation and (c) the rich-club coefficient calculation.**

## Data collection

To collect a representative sample of open source projects, we built a dataset comprising the 100 most popular projects in GitHub[2]. In Github, users can star a project to show their interest and follow its progress, thus we choose to measure the popularity of a project in terms of its number of stars (i.e., the more stars the more popular the project is).

The construction of the dataset involved three phases: (1) cloning, (2) import and (3) enrichment. After these steps, we applied a graph generation process to define the main source to calculate the rich-club coefficient. Figure 1a illustrates the construction process. Next we describe each step.

*Cloning* In the first phase, we obtained the list of the 100 most popular projects in GitHub (at the moment of collecting the data[3] via its API[4]) and clone them to collect the corresponding Git repositories.

*Import* To facilitate the query and exploration of the projects we imported the Git repositories into a relational database using Gitana [6]. In the Gitana database, Git repositories are represented in terms of users (i.e., contributors with a name and an email), files, commits (i.e., changes performed in one or more files), references (i.e., branches and tags), and file modifications (i.e., they link commits with files). For two projects, the import process failed to complete due to missing or corrupted information in the GitHub repository.

*Enrichment* Our study needs a clear identification of the author of each commit so that we can properly link contributors and files they have modified (as commits are related to files). Unfortunately, Git does not control contributors' name and email when pushing commits, thus resulting in plenty of clashing and duplication problems in the data.

Clashing appears when two or more different contributors have set the same value for their names (again, note that in Git the contributor name is manually configured), thus resulting in commits coming from different contributors appearing with the same commit name (e.g., this often happens when using common names such as "Mike" or "John Smith"). On the other hand, duplicity appears when a contributor has indicated several emails, thus there are commits linked to different emails suggesting different contributors while in fact they come from the same person.

After evaluating the impact of this issue, we assessed that on average around 60% of the commits in each project were modified by contributors that involved a clashing/duplicity problem (and affecting a similar ratio of files). Therefore, for our analysis to be meaningful we needed a corrective action to uniquely identify contributors.

To address this problem, we relied on available GitHub data for the project, as GitHub uses unique usernames. By linking commits to that unique username we could disambiguate the contributors behind the commits. Therefore, in this phase we enriched our repository data with GitHub usernames. We linked the commits with the GitHub usernames via the commit SHA values to uncover the GitHub usernames responsible for the commit with that SHA and relying on that info instead on the info provided as part of the Git commit metadata.

This method only failed for commits without a GitHub username associated, which happened when the user that made that commit was no longer existing in GitHub (e.g.,

[2]https://github.com

[3]List obtained in September, 2016.

[4]Using the request: https://api.github.com/search/repositories?q=stars:>1&sort=stars&order=desc&per_page=100

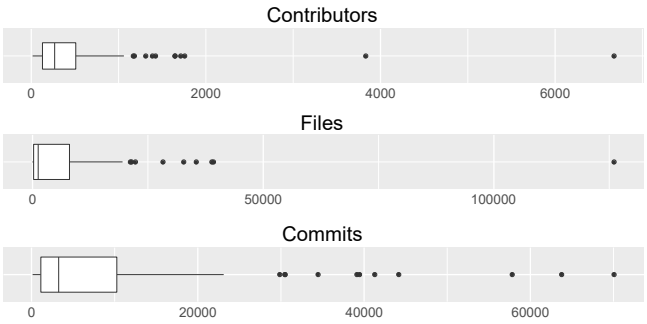


**Figure 2: Boxplots for the main dimensions of commits dataset.**

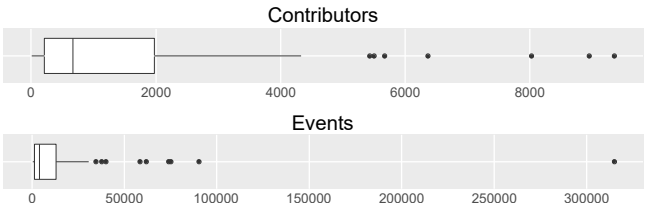


**Figure 3: Boxplots for the main dimensions involved in the issue data.**

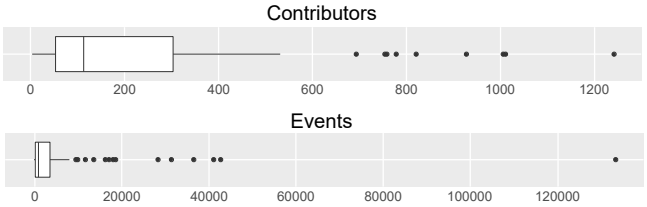


**Figure 4: Boxplots for the main dimensions involved in the pull requests data.**

because s/he deleted the account). In these cases we stick to the email in Git commit as contributor identifier.

Thanks to this method, we reduced considerably the both clashing and duplicity problems in our dataset. The percentage of commits modified by contributors that may involve either clashing or duplicity problems was reduced to 0.007% on average ($\sigma = 0.039$). We also calculated the percentage of files affected by this problem: 0.025% ($\sigma = 0.089$).

Additionally, for collecting data regarding activities on GitHub, we followed a parallel pipeline. Given that GitHub API rate limits do not allow to access in a reasonable time all the data we needed, we decided to rely on the data from GHArchive [1], a project which records every public event triggered by GitHub since February 2011 and make the data easily accessible for further analysis. We downloaded the entire GHArchive database of 2016 and filter only the events referred to the projects of our dataset, based on the unique project identifier of GitHub. We therefore created a dataset for the activity events relevant to our study: all the activities that involve multiple users' interaction are considered (i.e., issues, pull requests and relative comments).

At the end of this process, the curated repositories dataset contains a total number of 98 projects, 50,697 contributors, 670,182 files and 926,204 commits. On average, each project has 258 contributors, 6,839 files and 9,451 commits. Figure 2 shows the boxplots for each variable. On the other hand, the activities dataset comprises 124,260 contributors (118,919 interacting with issues, 20,140 interacting with pull requests), 1,359,823 issues-related events and 280,807 pull-request-related events. Figures 3 and 4 show the distributions over the involved projects. Note that the activities dataset includes a higher number of contributors, as there are GitHub users that can create issues, pull requests or comments and may not contribute code (and therefore it is not tracked in the Git repositories, i.e., the curated repositories dataset).

## Graph Generation

We rely on the so-called collaboration supergraphs to calculate the rich-club coefficient in our study, which are obtained from two set of graphs: (a) commit graphs using Git data and (b) activity graphs using GitHub data. They are both weighted, undirected graphs that include users based on different kind of interactions. We have defined three generation steps to create these graphs, as illustrated in Figure 1b. Next we describe the graphs generated in each step.

*Commit Graph Generation* Commit graphs are composed of nodes, which represent contributors; and edges, which joins two authors who have edited the same file in the repository. Nodes are weighted according to the number of contributors, while edge weight represents the number of files edited by the two authors.

*Activity Graph Generation* Activity graphs are of two types: issues-based and pull-request-based. In the first group, edges connect contributors that interacted on the same issues, either both commenting the issue or performing other actions such as opening, closing and assigning the issue. Similarly, pull-request graphs aggregate contributors that worked on the same pull requests, commenting or reviewing others' work. Even activities graph are weighted based on the number of common issues or pull requests each pair of users has interacted with.

*Collaboration Supergraph Generation* The previous three graphs (i.e., commit, issue and pull request graphs) are merged together into a **collaboration supergraph**: each pair of users has an edge if either they have committed the same file,

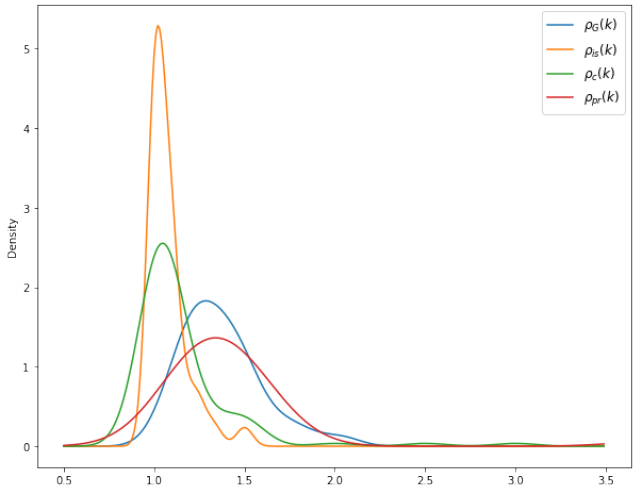


**Figure 5: KDE distribution of maximum values of $\rho$ for the types of graphs considered. The rich-club effect obtained in the supergraph (blue line) is higher than the single graph one, because the average values are further from 1. It is noticeable that a quite strong behavior is present in the pull-requests graphs, probably symptom that this critical step is concentrated on few core developers.**

or they worked on the same issue or pull request. Matching of the nodes among the three source graphs is made via the GitHub username, which uniquely identifies the contributor. Note that this is possible thanks to having addressed the duplicity and clashing problem in Git, as described before.

The three interactions graphs and the collaboration supergraph are used as input to compute the rich-club coefficient, as we explain in next section.

### Computing the Rich-Club Coefficient

The coefficient is computed using the implementation[5] in the NetworkX [7] Python package for network analysis, that provides both the non-normalized and normalized version of the coefficient. Figure 1c illustrates this last step in our method.

Note that the computation of the rich-club coefficient is run for each project graph, but results are not always available: if the graph is too simple, the randomization process used for normalization fails. We consider only the projects with a defined normalized value for the supergraph, indicated as $\rho_G(k)$, otherwise results could not be validated.For this set of projects, rich-club coefficient is calculated as well for the issues, commits and pull-requests graphs and it is indicated as $\rho_{is}(k)$, $\rho_{pr}(k)$ and $\rho_c(k)$, respectively.

[5]https://networkx.github.io/documentation/stable/reference/algorithms/rich_club.html

## 5 DISCUSSION

### Presence of rich-club behavior on the overall contributions' graph

A total of 60 projects have a defined $\rho_G$ value. The distribution of the maximum coefficient for each project is shown in Figure 5. Focusing on the supergraph performance (blue line), it is possible to notice that each project has a maximum coefficient slightly higher than 1: the rich-club behavior is present in all the inspected projects, but it has more relevance only for a few of them, i.e. those that are in the right tail of the distribution and so that are more distant from a random network. In Table 1, the top-10 projects with respect to $\rho_G$ are listed, as well as the maximum rich-club coefficient for all the other source graphs. Intuitively, the higher the coefficient the more prominent is the effect of the club on the network, but quantitatively describing this effect is hard, because the distribution of coefficients is not Gaussian and confidence values cannot be directly applied.

We believe this diversity of results reflects the different maturity of the projects and the alternative ways open source projects can grow and evolve. For instance, many open-source projects start as a collaboration effort from a small team of developers. As the project grows and gains popularity, it attracts new contributors. At this stage, in some projects the original team retains "ownership" of the codebase, with external developers submitting small contributions. Meanwhile, in other cases, the project matures and is able to attract developers that become core contributors, diluting the presence of the team of founders.

Other projects, typically high-profile ones, reach GitHub already in a mature state, after the initial development is performed privately, *e.g. React*, in a company or a closed community. In this scenario, once it becomes public, the project *evolves* rather than *grows*, and the role of a core team of developers has a lower impact.

Therefore, the project history must be taken into account when internally reacting to a rich-club measure. For instance, let us focus on the two projects depicted in Figure 6, one with a high value for the normalized rich-club coefficient (`materialize`[6], a CSS framework based on Material Design) and another with a low value (`swift`[7], the Swift programming language). Regarding `materialize`, this project was established in 2014 by a team of 4 developers, with 3,853 commits and 252 contributors. Nevertheless, the project only has two top contributors (more than 1,000 commits), which belong to the original team, and no frequent contributors (between 100 and 1,000 commits). Thus, it is clear that this project is still "owned" by its founders. On the other hand,

[6]https://materializecss.com
[7]https://swift.org/

**Table 1: Top-10 projects with highest normalized rich-club coefficient $\rho_G$. Coefficients of the other graphs are also presented when available.**

| id | name | $\rho_G$ | $\rho_{is}$ | $\rho_c$ | $\rho_{pr}$ |
|---|---|---|---|---|---|
| 23974149 | Dogfalo/materialize | 2.068966 | 1.500000 | 1.500000 | 1.605793 |
| 3100121 | nwjs/nw.js | 1.981061 | 1.092742 | 1.100000 | — |
| 557980 | socketio/socket.io | 1.836576 | 1.000000 | 1.052632 | — |
| 1200050 | ariya/phantomjs | 1.808190 | 1.250000 | 1.062500 | 1.639279 |
| 3228505 | atom/atom | 1.739953 | 1.000000 | 1.080000 | 1.524229 |
| 25315643 | nylas/N1 | 1.705238 | — | 1.000000 | — |
| 12256376 | driftyco/ionic | 1.658773 | 1.076923 | 1.066667 | 1.379662 |
| 3402537 | h5bp/Front_end_Developer_Interview_Questions | 1.619048 | — | 1.000000 | — |
| 10270250 | facebook/react | 1.567072 | 1.000000 | 1.087719 | 1.526232 |
| 15204860 | papers_we_love/papers_we_love | 1.561404 | 1.000000 | 1.071429 | 1.545455 |

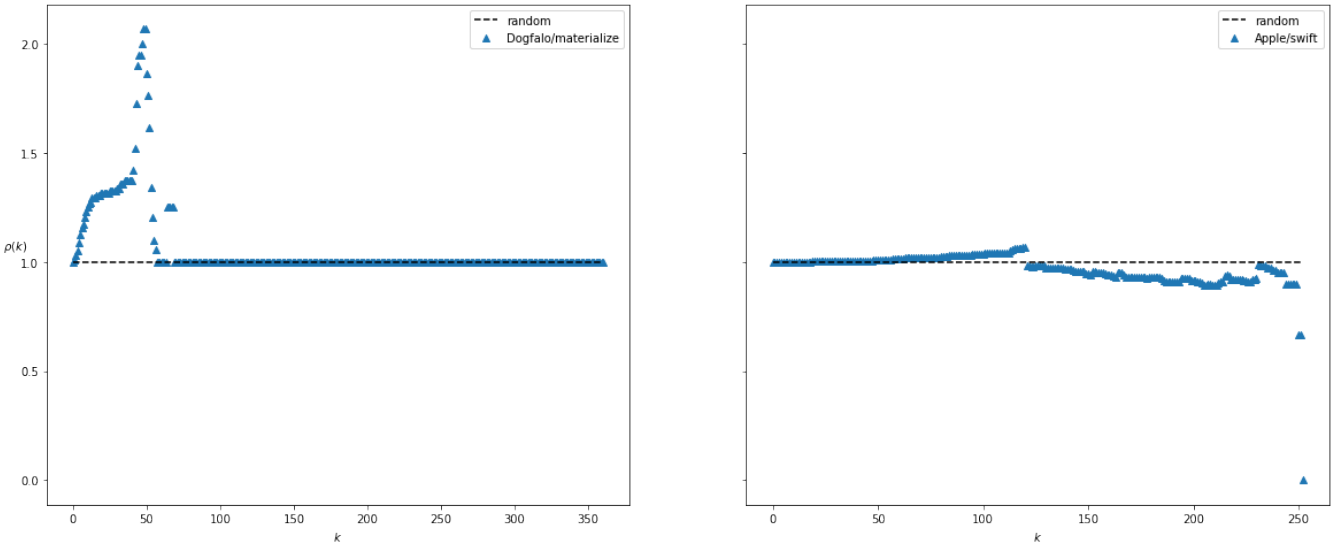


**Figure 6: Normalized rich-club coefficient for the `materialize` project (left) and the `swift` project (right). The presence of the club is evident in the first project, given the high value of $\rho$; the latter is not showing the behavior.**

`swift` started in 2010. It was publicly announced by Apple in 2014 and was later open sourced in December 2015. Currently, the project has more than 84k commits and 674 contributors, with 14 top contributors (more than 1.000 commits) and 44 frequent contributors (between 100 and 1.000 commits). Remarkably, 4 of the top contributors and 21 of the frequent contributors do not belong to Apple according to their GitHub profile. This is a sign that the project has successfully attracted and retained external talent. As further validation, the collaborations graphs used for comparison are presented in Figures 7 and 8, respectively for `materialize` and `swift`. The color represents the degree of each node: rich-club effect is quite evident in the first graph, where the 2 core developers have the highest degree (in red), followed by two clusters of well-connected users (in orange). On the other graph, connections are more homogeneous and most users are connected to each other, building a single wider core cluster.

### Rich-club behaviour on issues, pull and commit graphs

Computing single graphs' coefficients has the objective of revealing possible clubs inside a particular dimension of the project collaboration. In Figure 5, maximum coefficients distribution for issues (orange), pull request (red) and commits (green) graphs are presented too: mean of distribution is near to 1 for issues and commits, while it has higher values for pull requests. It is important to notice that the tail of distribution

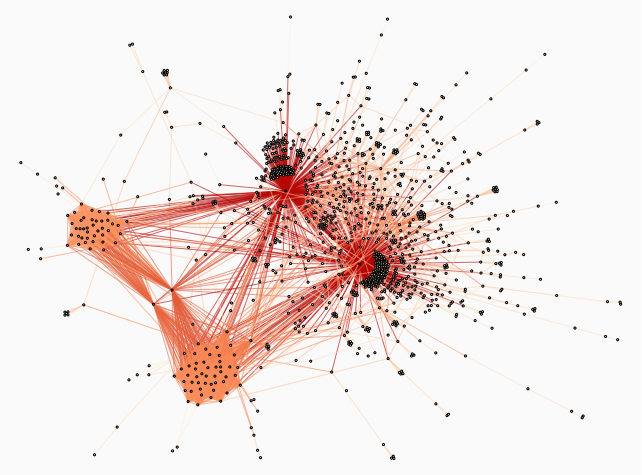

**Figure 7: Interactions graph for `materialize` project. The color of the edges is proportional to the correspondent node degree: the *club* is formed by two main users and two well-connected subgroups.**

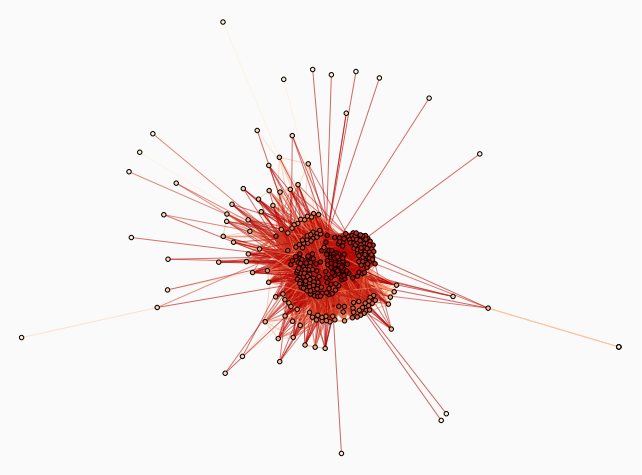

**Figure 8: Interactions graph for `swift` project. Most of the users are connected to each other, giving the idea of a wider collaboration across all the developers involved.**

is not zero for both commits and pull requests, indicating that for some projects the club effect is more relevant in the single interaction network with respect to the supergraph.

A possible explanation to the strong difference between pull requests coefficients and the others comes from the fact that pull requests are critical activities in the development process and the most time-consuming one (it is much easier for an occasional contributor to open an issue than preparing a pull request submission). This reduces the potential number of people that could appear as node in this graph. Moreover, given the criticality of this task, it is also usual that projects keep a more tight control here to ensure the "consistent" evolution of the project.

As for the clubs dimension (i.e. the *number* of nodes that are part of the club, defined as the set of users with a degree higher or equal to $k_{max}$, where $k_{max}$ corresponds to maximum value of $\rho(k)$) for each graph: one remarkable point is that 25 over 60 projects have a set of common users across all their clubs, which means that a core set of users is incident to all the steps of the software development and form what we could even call a *superclub* since they have influence on all community levels of the project. In this sense, this superclub could be regarded as more relevant than the club of the supergraph, which are the users that dominate the supergraph but may not have such a dominant impact on specific graphs if they are not part of the sueprclub as well. Distribution of the overlapping users for each project is presented in Figure 9.

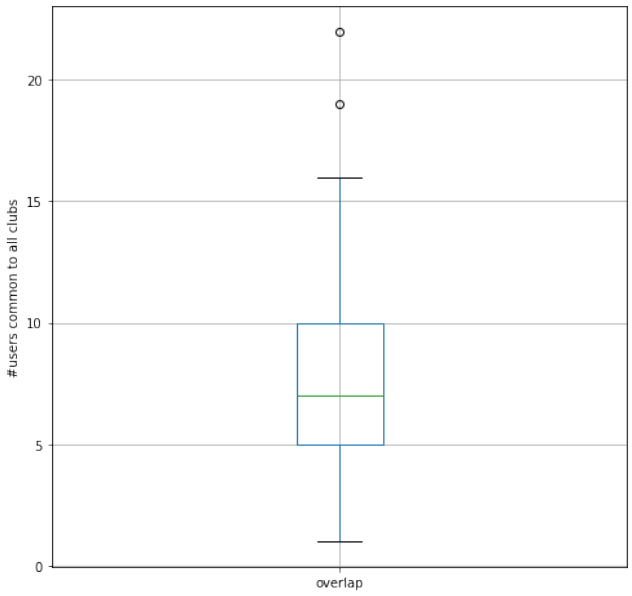


**Figure 9: Distribution of common users across all the clubs of the project.**

### Rich-club implications at the individual project level

We have assessed the presence of rich-clubs inside some projects among the most popular on GitHub.

In the literature, presence of rich-club behaviour has shown both positive and negative effects on the network. On the one hand, the rich club is thought to be critical for global communication given that these nodes have high betweenness centrality, in that, if the shortest paths between all pairs

of nodes is found, many of these shortest paths involve rich club members [3]. On the other hand, the presence of this behavior relates with the tendency of the most active community members to control the network.

Therefore, project owners should evaluate their rich-club coefficient, understand where its coming from (e.g. see some of the potential interpretations pointed above) and then react in consequence. If there is a low rich-club, they should make sure that the project information still flows across all nodes even if the hubs are not connected and therefore do not play this central role. If there is a high rich-club, governance policies [8] should be put in place to guarantee that all important decisions require a broad community participation and cannot be just dominated by the few hub nodes colluding together.

## 6 THREATS TO VALIDITY

Our work is subjected to a number of threats to validity which we classify into: (1) internal validity, which is related to the inferences made based on the application of our research methodology; and (2) external validity, which discusses the generalization of our findings.

Regarding the internal validity, our dataset construction process relied on the GitHub repository information, which failed to retrieve two of the initial top 100 repositories due to missing or corrupted information in the repository. Moreover, all GitHub projects revealed a high ratio of clashing and duplicity problems, which leads to collaboration graphs that may have duplicated nodes (i.e., two nodes which actually represent the same contributor). We took measures to alleviate and correct these issues.

As for the external validity, note that our sample is based on the 100 most popular projects in GitHub and therefore our results should not be generalized to other software development platforms or to projects hosted in private repositories.

## 7 REPLICABILITY PACKAGE

To facilitate the replication of our study, we have prepared a complete replication repository for interested researchers. The repository includes the list of all considered projects and their corresponding prepared *graph* files to calculate the rich-club coefficient. More specifically, each source graph (issues, pull-requests and commits) is already provided, while the supergraph can be computed using the correspondent script. A second script is used to calculate the rich-club coefficient for each type of graph. Results are stored in JSON format. Notice that the computation of coefficients is time and resource consuming for many graphs. The package is available at this link[8].

[8]https://drive.google.com/open?id=1P25fQIYMERxGAhLMyzkMGLuoPfKtpA3R

## 8 CONCLUSIONS AND FURTHER WORK

We have presented the first systematic evaluation of the rich-club behaviour on open source projects. We defined a pipeline to collect data both from Git and Github and then used this data to build three kind of graphs, representing three different types of collaboration inside open-source software development (commits, issues and pull requests), and a global-view supergraph that combines all the previous. We show that the rich-club behavior is present in all the supergraphs, but only few of them have an evident club structure. We compute coefficients both for single source graphs and overall interaction graphs, showing that rich-club behavior varies across different layers of software development. Furthermore, we inspected two projects to validate our observations about their club structure, showing that rich-club behavior is a complex phenomenon that can provide plenty of useful information but that must be studied in detail in order to properly interpret its impact on specific projects.

Indeed, we believe this analysis opens a number of interesting discussions and interpretations on the health of the open source communities, useful also for project owners that want to optimize the collaborations and contributions within their project network.

As further work, we plan to go deeper in this rich-club analysis by exploring as well weighted rich-club calculations [11] and the rich-club effect at, both, the module and ecosystem level. The former would aim to detect "local" rich-club behaviours that may be hidden when looking the project from a global view but still be important for the success and evolution of specific project components. The latter looks for potential coordinated rich-clubs behaviours that span multiple projects in the same domain. Moreover, time dimension could be included to highlight temporal clubs (e.g.: clubs present only in specific moments of software development, such as before a release).

Additionally, further sources of information (*e.g.*, email exchanges in public project mailing lists) will be also considered to enrich our graphs to check whether rich-club structures do exist but manifest beyond the project data tracked by GitHub.

## ACKNOWLEDGMENTS

This work is partially funded by the H2020 ECSEL Joint Undertaking Project "MegaM@Rt2: MegaModelling at Runtime" (737494) and the Spanish Ministry of Economy and Competitivity through the project "Open Data for All: an API-based infrastructure for exploiting online data sources" (TIN2016-75944-R).

## REFERENCES

[1] [n. d.]. GH Archive. https://www.gharchive.org/
[2] [n. d.]. GitHub Rest API v3. https://developer.github.com/v3/

[3] Max A. Bertolero, B. T. Thomas Yeo, and Mark D'Esposito. 2017. The diverse club. *Nature communications* 8, 1 (2017), 1277.

[4] Vittoria Colizza, Alessandro Flammini, M. Ángeles Serrano, and Alessandro Vespignani. 2006. Detecting rich-club ordering in complex networks. *Nature Physics* 2, 2 (2006), 110–115.

[5] Guido Conaldi. 2010. Flat for the few, steep for the many: Structural cohesion and Rich-Club effect as measures of hierarchy and control in FLOSS communities. *International Journal of Open Source Software and Processes (IJOSSP)* 2, 2 (2010), 14–28.

[6] Valerio Cosentino, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2018. Gitana: A software project inspector. *Sci. Comput. Program.* 153 (2018), 30–33.

[7] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring Network Structure, Dynamics, and Function using NetworkX. In *Proceedings of the 7th Python in Science Conference*. 11 – 15.

[8] Javier Luis Cánovas Izquierdo and Jordi Cabot. 2015. Enabling the Definition and Enforcement of Governance Rules in Open Source Systems. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Volume 2*. 505–514.

[9] Antonio Lima, Luca Rossi, and Mirco Musolesi. 2014. Coding Together at Scale: GitHub as a Collaborative Social Network. In *Proceedings of the 8th International Conference on Weblogs and Social Media, (ICWSM'2014)*. The AAAI Press. http://www.aaai.org/ocs/index.php/ICWSM/ICWSM14/paper/view/8112

[10] Julian J. McAuley, Luciano da Fontoura Costa, and Tibério S. Caetano. 2007. Rich-club phenomenon across complex network hierarchies. *Applied Physics Letters* 91, 8 (2007), 084103.

[11] Tore Opsahl, Vittoria Colizza, Pietro Panzarasa, and Jose J Ramasco. 2008. Prominence and control: the weighted rich-club effect. *Physical Review Letters* 101, 16 (2008), 168702.

[12] Weifeng Pan, Bing Li, Yutao Ma, and Jing Liu. 2011. Multi-granularity evolution analysis of software using complex network theory. *Journal of Systems Science and Complexity* 24, 6 (2011), 1068–1082.

[13] Sergi Valverde and Ricard V. Solé. 2007. Self-organization versus hierarchy in open-source social networks. *Physical Review E* 76, 4 (2007), 046118.

[14] Martijn P. van den Heuvel and Olaf Sporns. 2011. Rich-Club Organization of the Human Connectome. *Journal of Neuroscience* 31, 44 (2011), 15775–15786. https://doi.org/10.1523/JNEUROSCI.3539-11.2011 arXiv:http://www.jneurosci.org/content/31/44/15775.full.pdf

[15] Ye Wei, Wei Song, Chunliang Xiu, and Ziyu Zhao. 2018. The rich-club phenomenon of China's population flow network during the country's spring festival. *Applied Geography* 96 (2018), 77–85.

[16] Shi Zhou and Raúl J Mondragón. 2004. The rich-club phenomenon in the Internet topology. *IEEE Communications Letters* 8, 3 (2004), 180–182.