SCIENTIFIC REPORTS
nature research

OPEN

Online division of labour: emergent structures in Open Source Software

María J. Palazzi[1], Jordi Cabot[1,2], Javier Luis Cánovas Izquierdo[1], Albert Solé-Ribalta[1] & Javier Borge-Holthoefer[1]

Received: 15 February 2019
Accepted: 11 September 2019
Published online: 25 September 2019

The development Open Source Software fundamentally depends on the participation and commitment of volunteer developers to progress on a particular task. Several works have presented strategies to increase the on-boarding and engagement of new contributors, but little is known on how these diverse groups of developers self-organise to work together. To understand this, one must consider that, on one hand, platforms like GitHub provide a virtually unlimited development framework: any number of actors can potentially join to contribute in a decentralised, distributed, remote, and asynchronous manner. On the other, however, it seems reasonable that some sort of hierarchy and division of labour must be in place to meet human biological and cognitive limits, and also to achieve some level of efficiency. These latter features (hierarchy and division of labour) should translate into detectable structural arrangements when projects are represented as developer-file bipartite networks. Thus, in this paper we analyse a set of popular open source projects from GitHub, placing the accent on three key properties: nestedness, modularity and in-block nestedness –which typify the emergence of heterogeneities among contributors, the emergence of subgroups of developers working on specific subgroups of files, and a mixture of the two previous, respectively. These analyses show that indeed projects evolve into internally organised blocks. Furthermore, the distribution of sizes of such blocks is bounded, connecting our results to the celebrated Dunbar number both in off- and on-line environments. Our conclusions create a link between bio-cognitive constraints, group formation and online working environments, opening up a rich scenario for future research on (online) work team assembly (e.g. size, composition, and formation). From a complex network perspective, our results pave the way for the study of time-resolved datasets, and the design of suitable models that can mimic the growth and evolution of OSS projects.

Open Source Software (OSS) is a key actor in the current software market, and a major factor in the consistent growth of the software economy. The promise of OSS is better quality, higher reliability, more flexibility, lower cost, and an end to predatory vendor lock-in, according to the Open Source initiative[1]. These goals are achieved thanks to the active participation of the community[2]: indeed, OSS projects depend on contributors to progress[3,4].

The emergence of GitHub and other platforms as prominent public repositories, together with the availability of APIs to access comprehensive datasets on most projects' history, has opened up the opportunities for more systematic and inclusive analyses of how OSS communities operate. In the last years, research on OSS has left behind a rich trace of facts. For example, we now know that the majority of code contributions are highly skewed towards a small subset of projects[5,6], with many projects quickly losing community interest and being abandoned at very early stages[7]. Moreover, most projects have a low *truck factor*, meaning that a small group of developers is responsible for a large set of code contributions[8–10]. This pushes projects to depend more and more on their ability to attract and retain occasional contributors (also known as "drive-by" commits[11]) that can complement the few core developers and help them to move the project forward. Along these lines, several works have focused on strategies to increase the on-boarding and engagement of such contributors (e.g., by using simple contribution processes[12], extensive documentation[13], gamification techniques[14] or *ad hoc* on-boarding portals[15], among others[16]). Other social, economic, and geographical factors affecting the development of OSS have been scrutinised as well, see Cosentino *et al.*[17] for a thorough review.

Parallel to these macroscopic observations and statistical analyses, social scientists and complex network researchers have focused, in relatively much fewer papers, on analysing how a diverse group of (distributed)

[1]Internet Interdisciplinary Institute (IN3), Universitat Oberta de Catalunya, Barcelona, Catalonia, Spain. [2]ICREA, Barcelona, Catalonia, Spain. Correspondence and requests for materials should be addressed to J.B.-H. (email: jborgeh@uoc.edu)

	# Contributors	# Files	# Commits	# Stars	Project age
Largest project	1,061	12,321	75,757	27,500	4 years 11 months
Smallest project	55	27	444	36,900	5 years 6 months
Average	422	3,247	33,936	46,334	4 years 9 months
Most popular project	516	2,833	34,666	293,000	2 years 10 months
Least popular project	117	103	4,057	21,700	5 years 5 months
Oldest project	1,434	10,413	174,452	35,000	11 years 3 months
Youngest project	43	51	210	31,600	0 years 3 months

Table 1. Statistics of our dataset.

contributors work together, i.e. the structural features of projects. Most often, these works pivot on the interactions between developers, building explicit or implicit collaborative networks, e.g. email exchanges[18,19] and unipartite projections from the contributor-file bipartite network[20], respectively. These developer social networks have been analysed to better understand the hierarchies that emerge among contributors, as well as to identify topical clusters, i.e. cohesive subgroups that manifest strongly in technical discussions. However, the behaviour of OSS communities cannot be fully understood only accounting for the relations between project contributors, since their interactions are mostly mediated through the edition of project files (no direct communication is present between group members). To overcome this limitation, here we focus on studying the structural organisation of OSS projects as contributor-file bipartite graphs. On top of technical and methodological adaptations, the consideration of these two elements composing the OSS system allows retaining valuable information (as opposed to collapsing it on a unipartite network) and, above all, recognising both classes as co-evolutionary units that place mutual constraints on each other.

Our interest on the structural features of OSS projects departs from some obvious, but worth highlighting, observations. First, public collaborative repositories place no limits, in principle, to the number of developers (and files) that a project should host. In this sense, platforms like GitHub resemble online social networks (e.g. Twitter or Facebook), in which the number of allowed connections is virtually unbounded. However, we know that other factors –biological, cognitive– set well-defined limits to the amount of active social connections an individual can have[21], also online[22]. But, do these limits apply in collaborative networks, where contributors work remotely and asynchronously? Does a division of labour arise, even when interaction among developers is mostly indirect (that is, via the files that they edit in common)? And, even if specialised subgroups emerge (as some evidence already suggests, at least in developer social networks[20]), do these exhibit some sort of internal organisation?

To answer these questions, we will look at three structural arrangements which have been identified as signatures of self-organisation in both natural and artificial systems: nestedness[23,24], modularity[25–27], and in-block nestedness[28–30]. The first one, nestedness, is a suitable measure to quantify and visualise how the mentioned low truck factor, and the existence of core/drive-by developers[31], translates into a project's network structure. As for modularity, it provides a natural way to check whether OSS projects split in identifiable compartments, suggesting specialisation, and whether such compartments are subject to size limitations, along the mentioned bio-cognitive limits. Finally, since modularity and nestedness are, to some extent, incompatible in the same network[32,33], in-block nestedness (or the lack of it) can help to determine how projects solve the tension between the emergence of nested (hierarchy, asymmetry) and modular (specialisation, division of labour, bounds to social connections) patterns.

Results

The projects that we analyse in the following were selected according to their popularity (quantified as the number of stars these projects had received on GitHub, at the time of collection in 2016). This criterium mainly responds to two arguments: maturity and success. That is, here we purposefully pay attention to projects which have reached a reasonable degree of evolution, regardless of the absence (or presence) of any given structural organisation at the initial stages.

After pre-processing, formatting and discarding some of the top 100 public OSS projects hosted on GitHub, we ended up retaining 65 of them, see Materials and Methods for details. As can be seen in Table 1, we have a sufficiently broad distribution of project sizes and age. Note also that popularity (number of stars) is not necessarily related to their size (Pearson coefficient $r = -0.03$) nor age ($r = -0.1$).

Each of these projects have been represented as a bipartite unweighted graph, where inter-class links (between contributors c and files f) are allowed, but intra-class links are forbidden. This bipartite network is thus encoded as an $N \times M$ rectangular binary matrix $\mathbf{A}$, where N is the number of contributors, and M is the number of files. An entry $a_{cf} = 1$ if contributor c edited the file f at least once, and 0 otherwise. The total size of a project is $S = N + M$; the smallest project considered here is *resume.github.com*, with $S = 82$, and the largest one is *foundation-sites*, with $S = 13{,}382$. Since we are focusing on the bipartite representation of OSS projects, we first have explored the existence of any relationship between both elements (contributors and files) that compose the projects. We computed the Pearson correlation coefficient between the number of files and contributors, and found a rather weak positive correlation between them ($r = 0.34$). Surprisingly, some projects exhibit strong imbalances between both quantities, see Materials and Methods for details.

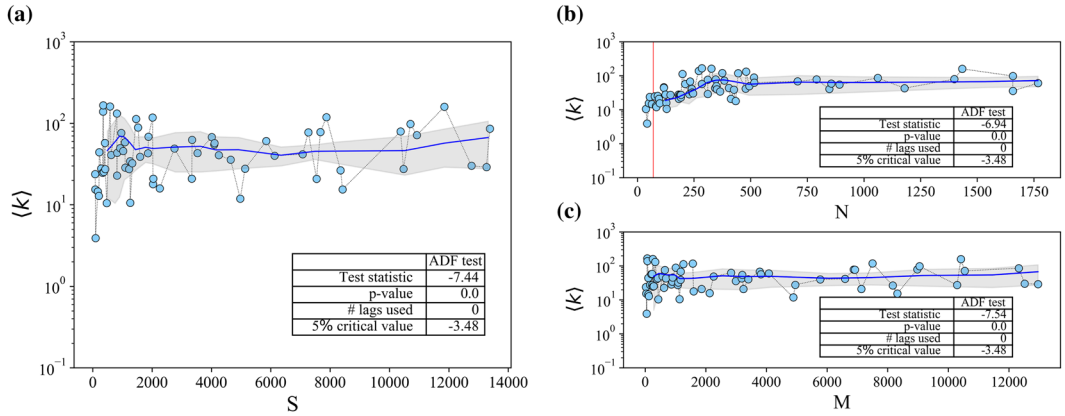

Figure 1. Contributor-contributor network: scatter plots of the developers implicit average degree $\langle k \rangle$ against project size $S = N + M$ (panel (a)), number of contributors (panel (b)) and number of files (panel (c)). The shadowed grey area represents one standard deviation above and below the average, while circles represent each individual project. The red line in panel (b) indicates $N = 70$ contributors. Inset tables in all panels show the results of the ADF stationarity test. All plots are presented in semi-log axes.

Preliminary observations. Before we focus on the structural arrangements of interest (nestedness, modularity, in-block nestedness), we explore whether a potentially unbounded interaction capability is mirrored in actual OSS projects across 4 orders of magnitude in size. To do so, we work on the projected contributor-contributor network, to measure the developer's implicit average degree $\langle k \rangle$, i.e. the average amount of contributors with whom an individual shares at least one file. Figure 1(a) shows a scatter plot of $\langle k \rangle$ against S (note the semi-log scaling). Panels (b) and (c) in Fig. 1 shows the scatter plots of $\langle k \rangle$ against N and M, respectively. Despite the changes in the $x-$axis scale (which affects the order in which projects are represented), there are no significant differences in the results. Such results indicate that, besides the initial fluctuating pattern, $\langle k \rangle$ presents an almost flat trajectory suggesting that, on average, a contributor indirectly interacts with ~70 peers, regardless of the size of the project. As visual aid, we have added a vertical red line in panel (b) at $N = 70$, to differentiate those networks with $N < 70$ and for which it is not possible to exhibit $\langle k \rangle \approx 70$. The stable behaviour of this average was statistically validated with the Augmented Dickey-Fuller (ADF) test for stationarity of time series (see Materials and Methods for details).

The stationary pattern for the developers implicit average degree in Fig. 1 is interesting in two aspects. First, it points to an inherent limitation to the number of connections (even indirect ones) that a contributor to a project can sustain. Notably, such limitation is below (but not far) from the Dunbar number (somewhere between 100 and 300), which is echoed as well in digital environments[22]. Second, the result is an indication of the existence of some sort of mesoscale organisation in the projects. In Bird *et al.*[19], the authors find that developers in the same community have more files in common than pairs of developers from different communities. Reversing the argument, one may say that relatively small contributor neighbourhoods are indicative, though not a guarantee, of the presence of well-defined subgroups in OSS projects.

Mesoscale patterns. From the previous encouraging result, we move on to the analysis of a comprehensive view of projects. The specificities of the methods to calculate nestedness $\mathcal{N}$, and to optimise modularity Q and in-block nestedness $\mathcal{I}$ are detailed in the Materials and Methods section. For the sake of illustration, Fig. 2 (top row) shows idealised examples of nestedness (left), modular (middle) and in-block nested (right) arrangements. The bottom row of the figure presents actual adjacency matrices of three projects with high values of each structural measure. In this Figure, rows and columns have been rearranged to highlight the different properties.

We start out with a general overview of the results for the three measures of interest. Figure 3 plots the obtained values for $\mathcal{N}$, Q, and $\mathcal{I}$ over all the projects considered in this work. To ease visualisation, and considering that nestedness and modularity are antagonistic organisations[33], projects are sorted to maximise the difference between $\mathcal{N}$ and Q. In general, nestedness is the lowest of the three values at stake, and in-block nestedness is, more often than not, the highest. It can be safely said, thus, that a tendency to self-organise as a block structure is present: 90% of the projects exhibit either Q or $\mathcal{I}$ above 0.4, and values beyond 0.5 are not rare. This evidence is compatible with previous results regarding the division of labour: indeed, be them modular or in-block nested, most projects can be split into communities of developers and files, forming subgroups around product-related activities[19].

Just like there is virtually no technical limit to the overall size of a project, there is not either an explicit bound to the size that a sub-group should have. And yet, previous theory and evidence suggests that larger communities come at an efficiency cost: the dynamics of a group change fundamentally when they exceed the Dunbar number, which is estimated around 150. While most often the number refers to personal acquaintances, it has been (and still is) applied in the industrial sphere[34]. Applied to the OSS environment, exceedingly small working sub-groups might hamper a project's advance; while too many contributors may not allow the group to converge towards a solution[35,36]. We explore whether, indeed, size limitations arise in developers sub-groups, as they emerge from

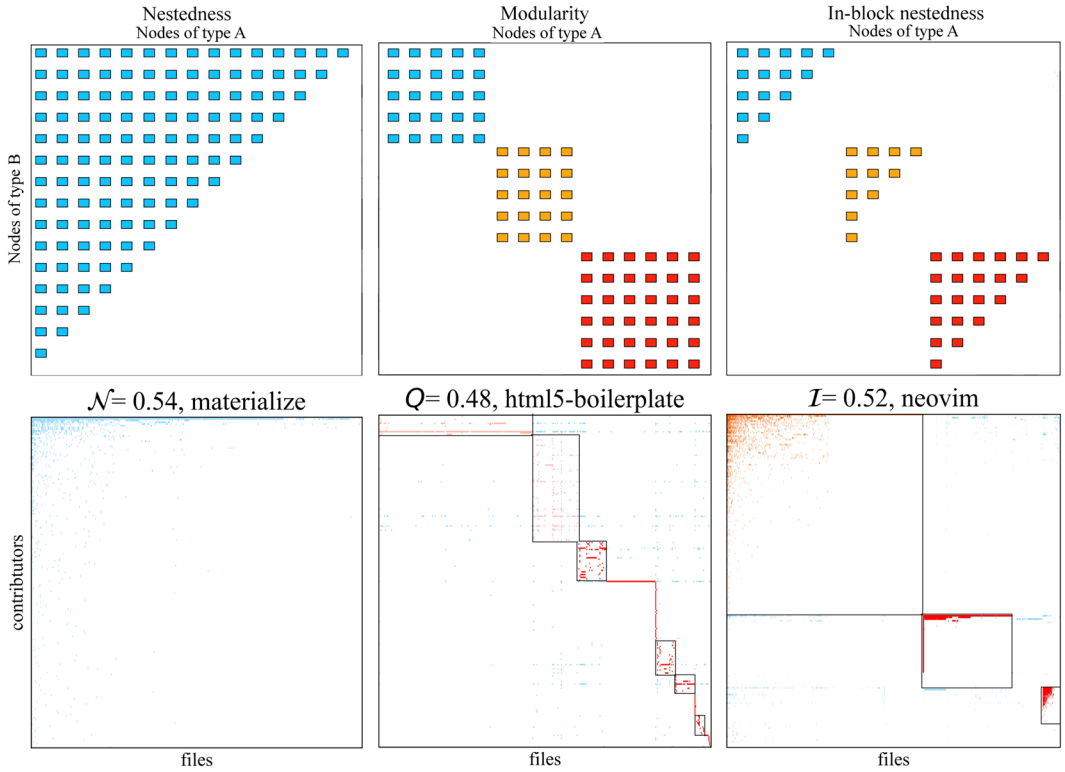

Figure 2. Top row: left: Nestedness $\mathcal{N}$, middle: Modularity Q, bottom: In-block nestedness $\mathcal{I}$. Bottom row: Interaction matrices for three projects with high values for each one the structural patterns of interest.

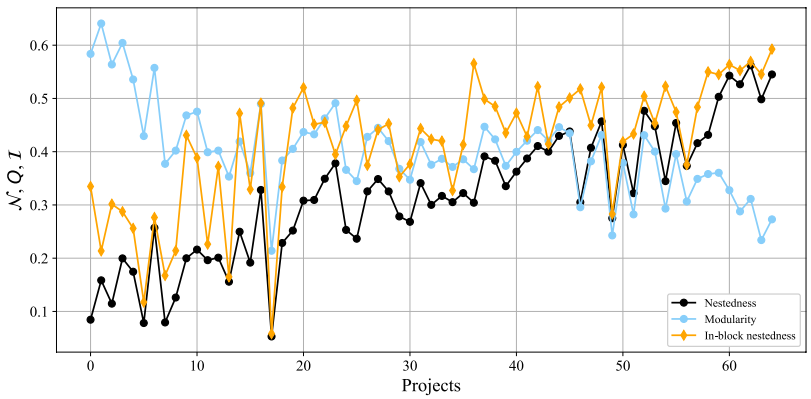

Figure 3. $\mathcal{N}$, Q, and $\mathcal{I}$ obtained values, for each project of our dataset. The projects were sorted to maximise the difference between $\mathcal{N}$ and Q.

either Q or $\mathcal{I}$ optimisation procedures. Although partitions are hybrid, i.e. a community has both developers and files, in the following, we will report the community sizes in terms of developers.

Figure 4 provides a global overview of the 65 projects studied here, with the distribution of their largest subgroup sizes as they are identified via Q (panel (a)) or $\mathcal{I}$ (panel (b)). In both cases the average (dashed orange vertical line) is below 200, and the histogram is evenly distributed around 100: most communities belong in the range from 80 to 200. Given the obvious similarity between both distributions, we perform a Mann-Whitney U test, so as to find out whether these two distributions are actually compatible (the null hypothesis cannot be rejected, p-value = 0.3). In other words, block sizes are independent from the optimisation strategy adopted. Indeed, the test indicates that both size distributions can be regarded as drawn from populations having the same distribution, and the combined distribution is shown in panel (c). The solid red line represents a log-normal fit (notice the logarithmic scale in the x-axis), and the insets in all panels show the QQ plots, to compare both theoretical and empirical distributions revealing that the fit is accurate.

Although Fig. 4(c) evidences, on average, a well-defined maximum community size (at 169.7 users, and 95% confidence interval [139.4,206.7] as measured for log-normal distributions[37]), we must ensure that the size of the

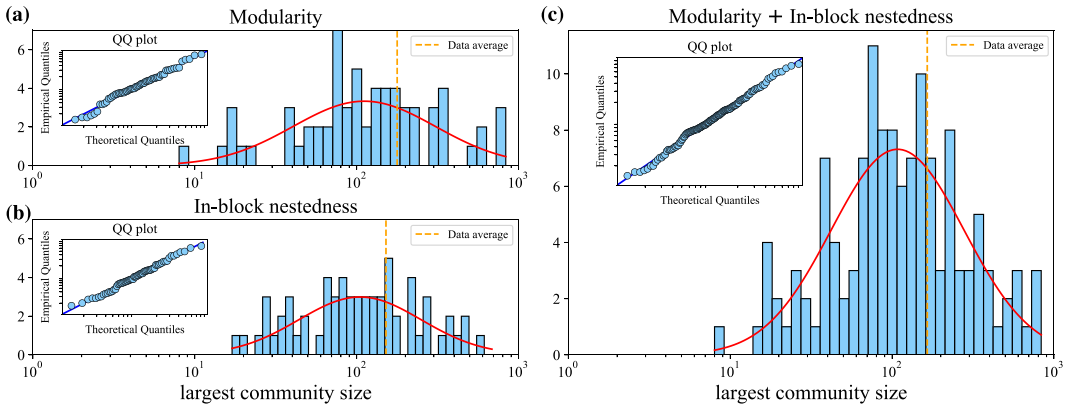

Figure 4. Frequency distribution of the largest community size for each project obtained after optimisation of modularity (panel a) and in-block nestedness (panel b). The distribution of largest community size when combining both opmtimization strategies is shown in panel c. In the three panels, the solid red line corresponds to the log-normal fit performed to each distribution, which are centred around 100. The dashed orange line indicates the average values of our dataset, and inset panels show the QQ plots of the empirical versus theoretical quantiles from the log-normal distribution fit.

largest communities detected for each project is independent of the size of the project, in order to validate such organisational limit. That is, we need to test that the largest blocks (far right in panel (c)) do not necessarily correspond to the largest projects. To do so, we go down to the project level. Figure 5 reports average (panels (a) and (b)) and maximum (panels (c) and (d)) subgroup sizes for both community identification strategies, as a function of the project size S. In general, results point at the existence of upper bounds to community size. This impression is confirmed statistically, as the ADF test for stationarity indicates (see p-values in insets) that subgroup sizes, after a fluctuating behaviour when $S < 2000$, remain stable across S in panels (b) to (d). This is not so in panel (a): average Q-communities exhibit a subtle growth with respect to project size, and the ADF test signals such non-stationarity. Nonetheless, it is apparent that in all cases –even in panel (a), despite its increasing trend– the size of communities is compatible with the limits described by Dunbar's number: in panel (c), largest community size is slightly above 200. In panel (d), even the largest projects reflect that the maximum size of a community is between 100 and 200.

These results are surprising, since such trend towards the compartmentalisation of the workload is not only decentralised, in the sense that it does not emerge from a predefined plan, but also implicit, because the interaction between developers is most often indirect.

Co-existing architectures and project maturity. As it has been suggested[33], empirical evidence indicates that more than one structural pattern may concur within a network, each evincing different properties of the system. We take the same stance here: a network is not regarded, for example, as *completely* modular or *completely* nested; rather, it may combine structural features that reflect the evolutionary history of the system, or the fact that the system evolves under different dynamical pressures that favour competing arrangements.

A convenient way to grasp this mixture is a ternary plot (or simplex), see Fig. 6. In the ternary plot, each project is located with three coordinates $f_{\mathcal{N}}, f_{Q}$ and $f_{\mathcal{I}}$, which are simply calculated from the original scores, e.g. $f_{\mathcal{N}} = \mathcal{N}/(\mathcal{N} + Q + \mathcal{I})$ (note that the three quantities are, by definition, in the [0, 1) range). The simplex can be partitioned according to "dominance regions", bounded by the three angle bisectors. These regions intuitively tell us which of the three patterns is more prominent for any given project. Note that certain areas of the simplex (in grey in Fig. 6) are necessarily empty, see Materials and Methods, and Palazzi *et al.*[33] for further details. Figure 6 reveals that most projects lie in the nested regions, while the predominantly modular region is relatively empty.

Together with their dominant architecture, points in Fig. 6 are colour-coded according to the total number of commits that each project has received. We take this number as a proxy to the level of development or maturity of the project (note that a project's age may be misleading due to periods of inactivity). The distribution of colour on the simplex suggests that more mature projects tend to exhibit nested or in-block nested structures, whereas predominantly modular projects appear to be relatively immature (with exceptions, admittedly). Such result is resonant to the fact that topical conversations in online social networks ("information ecosystems") evolve through different stages –modular when the discussion is still brewing in a scattered way; nested when the discussion becomes mainstream to the group of interest[32]. More relevant to OSS development, Fig. 6 reconciles the idea of workload compartmentalisation (sub-communities forming around product-related activities)[19], and the emergence of hierarchies[8] or a rich club[18] of developers, at least in well-developed projects. This partial picture is however complemented by the fact that hierarchies emerge as well on the code class: the presence of generalists and specialists applies to both developers and files in a nested or in-block nested scenario.

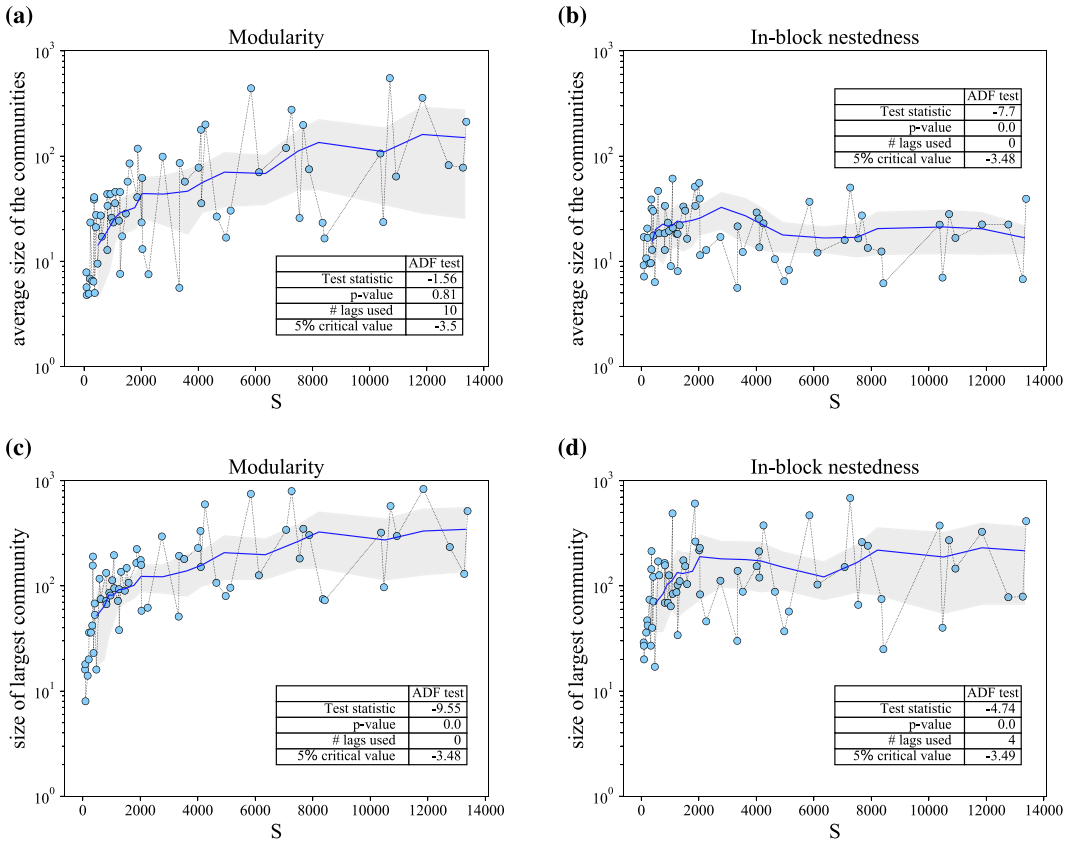

Figure 5. The evolution of the average community size as a function of S presents differences for Q- and $\mathcal{I}$-optimised partitions (panels (a) and (b), respectively). Regarding the size, average Q-communities are in general larger than $\mathcal{I}$-communities. Furthermore, the scaling behaviour is also different: an average community size for Q-optimised partitions moderately grows with S, while it remains almost constant for $\mathcal{I}$ beyond $S > 2000$. Turning from average to maximum community size, Q- and $\mathcal{I}$-optimised partitions (panels (c,d), respectively) present very similar bounds, from 30 to 300 contributors. Again, the largest Q-community slightly tends to grow with S, while this size stabilises around 100 for the case of $\mathcal{I}$. Inset tables in all panels show the results of the ADF stationarity test, confirming the presence of bounded values for the maximum subgroup sizes (panels (c) and (d), respectively). Note semi-log scaling.

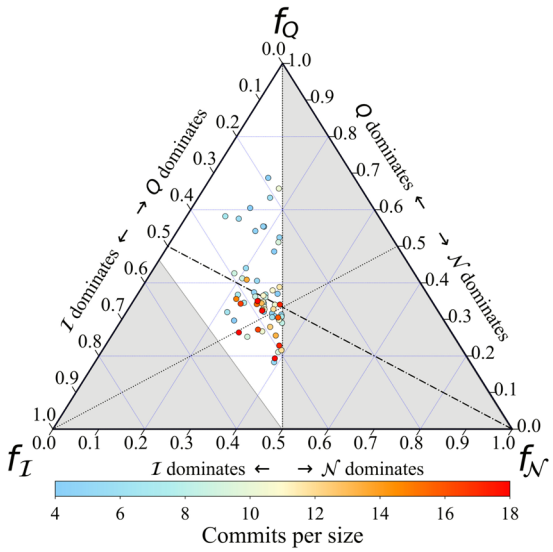

Figure 6. Distribution of the three architectural patterns for each the projects across a ternary plot. The colourbar indicates the number of commits received by each project, normalised by the size of it.

Discussion

In summary, our analyses have unveiled that OSS projects evolve into a relatively narrow set of structural arrangements. At the mesoscale, we observe that projects tend to form blocks, a fact that can be related to the need of contributors to distribute coding efforts, allowing a project to develop steadily and in a balanced way. Focusing on the file class, the emergence of blocks is interesting as well, since a modular architecture (understood now as a software design principle) is a desired feature in any complex software project. Furthermore, those blocks or subgroups have a relatively stable size no matter how large a project is. Remarkably, such size is compatible with the Dunbar number.

Previous research reported that OSS projects are largely heterogeneous, in the sense that developers self-organise into hierarchical structures. This conclusion is reinforced here, as we find evidence of nested arrangements in OSS bipartite networks. And yet, such statement may seem to clash with a modular arrangement, to the extent that modularity Q does not make any assumption regarding the internal organisation of the subgroups. Our findings, however, point at the fact that more mature projects tend to present a nested organisation inside modules. Thus, the presence of workload compartmentalisation is compatible with the emergence of hierarchies, with generalists and specialists throughout a project. Paradoxically, a more evolved and structured architecture does not imply better overall performance here: the nested arrangement inside blocks can hamper a project's progress, since the occasional and least committed contributors (those acting upon a small part of the code) tend to edit precisely the most generalist files, neglecting the least developed ones –a fact that has been observed from very different methodologies[8–10,17].

These findings open up a rich scenario, with many questions lying ahead. On the OSS environment side, our results contribute to an understanding of how successful projects self-organise towards a modular architecture: large and complex tasks, involving hundreds (and even thousands) of files appear to be broken down, presumably for the sake of efficiency and task specialisation (division of labour). Within this compartmentalisation, mature projects exhibit even further organisation, arranging the internal structure of subgroups in a nested way –something that is not grasped by modularity optimisation only. More broadly, our results demand further investigation, to understand their connection with the general topic of work team assembly (size, composition, and formation), and to the (urgent) issue of software sustainability[38]. OSS is a prominent example of the "tragedy of the commons": companies and services benefit from the software, but there is a grossly disproportionate imbalance between those consuming the software and those building and maintaining it. Indeed, by being more aware of the internal self-organisation of their projects, owners and administrators may design strategies to optimise the collaborative efforts of the limited number (and availability) of project contributors. For instance, they can place efforts to drive the actual project's block decomposition towards a pre-defined software architectural pattern; or ensure that, despite the nested organisation within blocks, all files in a block receive some minimal attention. More research on the derivation of effective project management leadership strategies from the current division of labour in a project is clearly needed and impactful.

Closer to the complex networks and data analysis tradition, our results leave room to widen the scope of this research. First, the present analysis could be complemented with weighted information. On first thought, this is within reach –one should just adapt the techniques and measurements to a weighted scenario. However, the problem is not so much a methodological one, but semantic: the number of times that a contributor interacts with a file (*commits*, in Git jargon) is not necessarily an accurate measure of the amount of information allocated in the file. Second, future research should tackle a larger and more heterogeneous set of projects, and even across different platforms such as Bitbucket. Admittedly, this work has focused on successful projects, inasmuch we only consider a few dozens among the most popular. Other sampling criteria could be discussed and considered in the future, to ensure richer and more diverse project collection. Beyond the richness of the analysed dataset, the relationship between maturity and structural arrangement (specially in regard to the internal organisation of subgroups) clearly demands further work. Two obvious –and intimately related– lines of research are related to time-resolved datasets, and the design of a suitable model that can mimic the growth and evolution of OSS projects. Regarding a temporal account of OSS projects, some challenges emerge due to the bursty development of projects in git-like environments. For example, a fixed sliding-window scheme would probably harm, rather than improve, possible insights into software development. On the modelling side, further empirical knowledge is needed to better grasp the cooperative-competitive interactions within these type of projects, which in turn determine the dynamical rules for both contributors and files which, presumably, differ largely.

Material and Methods

Data. Our open source projects dataset was collected from GitHub[39], a social coding platform which provides source code management and collaboration features such as bug tracking, feature requests, tasks management and wiki for every project. Given that GitHub users can star a project (to show interest in its development and follow its advances), we chose to measure the popularity of a GitHub project in terms of its number of stars (i.e. the more stars the more popular the project is considered) and selected the 100 most popular projects. Other possible criteria –number of forks, open issues, watchers, commits and branches– are positively correlated with stars[17], and so our proxy to mature, successful and active projects probably overlaps with other sampling procedures. The construction of the dataset involved three phases, namely: (1) cloning, (2) import, and (3) enrichment.

Cloning and import. After collecting the list of 100 most popular projects in GitHub (at the moment of collecting the data) via its API[40], we cloned them to collect 100 Git repositories. We analysed the cloned repositories and discarded those ones not involving the development of a software artifact (e.g. collection of links or questions), rejecting 15 projects out of the initial 100. We then imported the remaining Git repositories into a relational database using the Gitana[41] tool to facilitate the query and exploration of the projects for further analysis. 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 to the files); references (i.e. branches and tags); and file modifications. For two projects, the import process failed to complete due missing or corrupted information in the source GitHub repository.

Enrichment. Our analysis needs a clear identification of the author of each commit so that we can properly link contributors and files they have modified. Unfortunately, Git does not control the name and email contributors indicate when pushing commits resulting on clashing and duplicate problems in the data. Clashing appears when two or more contributors have set the same name value (in Git the contributor name is manually configured), resulting in commits actually coming from different contributors appearing with the same commit name (e.g., often when using common names such as "mike"). In addition, duplicity appears when a contributor has several emails, thus there are commits that come from the same person, but are linked to different emails suggesting different contributors. We found 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 number of files). To address this problem, we relied on data provided by GitHub for each project (in particular, GitHub usernames, which are unique). By linking commits to unique usernames, we could disambiguate the contributors behind the commits. Thus, we enriched our repository data by querying GitHub API to discover the actual username for each commit in our repository, and relied on those instead on the information provided as part of the Git commit metadata. This method only failed for commits without a GitHub username associated (e.g. when the user that made that commit was no longer existing in GitHub). In those cases we stick to the email in Git commit as contributor identifier. We reduced considerably the clashing/duplicity problem in our dataset. The percentage of commits modified by contributors that may involve a clashing/duplicity problem was reduced to 0.004% on average ($\sigma = 0.011$), and the percentage of files affected was reduced to 0.020% ($\sigma = 0.042$).

At the end of this process, we had successfully collected a total number of 83 projects, adding up to 48,015 contributors, 668,283 files and 912,766 commits. 18 more projects (to the total of 65 reported in this work) were rejected due to other limitations. On one hand, we discarded some projects that presented very strong divergence between the number of nodes of the two sets, e.g. projects with very large number of files but very few contributors. In these cases, although $\mathcal{N}$, Q and $\mathcal{I}$ can be quantified, the outcome is hardly interpretable. An example of this is the project *material-designs-icons*, with 15 contributors involved in the development of 12,651 files. As mentioned above, we also discarded projects that are not devoted to software development, but are rather collections of useful resources (free programming books, coding courses, etc.). Finally, we considered only projects with a bipartite network size within the range $10^1 \leq S \leq 10^4$, as the computational costs to optimise in-block nestedness and modularity for larger sizes were too severe. The complete dataset with the final 65 projects is available at http://cosin3.rdi.uoc.edu, under the Resources section.

Matrix generation. We build a bipartite unweighted network as a rectangular $N \times M$ matrix, where rows and columns refer to contributors and source files of an OSS project, respectively. Cells therefore represent links in the bipartite network, i.e. if the cell a_{ij} has a value of 1, it represents that the contributor i has modified the file j at least once, otherwise a_{ij} is set to 0.

We are aware that an unweighted scheme may be discarding important information, i.e. the heterogeneity of time and effort that developers devote to files. We stress that including weights in our analysis can introduce ambiguities in our results. In the Github environment, the size of a contribution could be regarded either as the number of times a developer commits to a file, or as the number of lines of code (LOC) that a developer modified when updating the file. Indeed, both could represent additional dimensions to our study. Furthermore, at least for the first (number of commits), it is readily available from the data collection methods. However, weighting the links of the network by the number of commits is risky. Consider for example a contributor who, after hours or days of coding and testing, performs a commit that substantially changes a file in a project. On the other side, consider a contributor who is simply documenting some code, thus committing many times small comments to an existing software –without changing the internal logic of it. There is no simple way to distinguish these cases. The consideration of the second item (number of LOC modified) could be a proxy to such distinction, but this is information is not realistically accessible given the current limitations to data collection. Getting a precise number of LOCs requires a deeper analysis of the Git repository associated to the GitHub project, parsing the commit change information one by one –an unfeasible task if we aim at analysing a large set of projects. The same scalability issue would appear if we rely on the GitHub API to get this information, which additionally would involve quota problems with such API. One might consider even a third argument: not every programming language "weighs" contributions in the same way. Many lines of HTML code may have a small effect on the actual advancement of a project, while two brief lines in C may completely change a whole algorithm. In conclusion, we believe there is no generic solution that allows to assess the importance of a LOC variation in a contribution. This will depend first on the kind of file, then on the programming style of each project and finally on an individual analysis of each change. Thus, adding informative and reliable weights to the network is semantically unclear (how should we interpret those weights?) and operationally out of reach.

Nestedness. The concept of nestedness appeared, in the context of complex networks, over a decade ago in Systems Ecology[42]. In structural terms, a perfect nested pattern is observed when specialists (nodes with low connectivity) interact with proper nested subsets of those species interacting with generalists (nodes with high connectivity), see Fig. 2 (left). Several works have shown that a nested configuration is signature feature of cooperative environments –those in which interacting species obtain some benefit[42–44]. Following this example in natural systems, scholars have sought (and found) this pattern in other kinds of systems[32,45–47]. In particular, measuring nestedness in OSS contributor-file bipartite networks helps to uncover patterns of file development. For instance, in a perfectly nested bipartite network the most generalist developer has contributed to every file in

the project, i.e. a core developer. Other contributors will exhibit decreasing amounts of edited files. On top of this hierarchical arrangement, we find asymmetry: specialist contributors (those working on a single file) develop precisely the generalist file, i.e. the file that every other developer also works on. Here, we quantify the amount of nestedness in our OSS networks by employing the global nestedness fitness $\mathcal{N}$ introduced by Solé-Ribalta *et al.*[30]:

$$\mathcal{N} = \frac{2}{N+M}\left\{\sum_{i,j}^{N}\left[\frac{O_{i,j} - \langle O_{i,j}\rangle}{k_j(N-1)}\Theta(k_i - k_j)\right] + \sum_{l,m}^{M}\left[\frac{O_{l,m} - \langle O_{l,m}\rangle}{k_m(M-1)}\Theta(k_l - k_m)\right]\right\}, \tag{1}$$

where $O_{i,j}$ (or $O_{l,m}$) measures the degree of links overlap between rows (or columns) node pairs; k_i, k_j corresponds to the degree of the nodes i,j; $\Theta(\cdot)$ is a Heaviside step function that guarantees that we only compute the overlap between pair of nodes when $k_i \geq k_j$. Finally, $\langle O_{i,j}\rangle$ represents the expected number of links between row nodes i and j in the null model, and is equal to $\langle O_{i,j}\rangle = \frac{k_i k_j}{M}$. This measure is in the tradition of other overlap measures, i.e. NODF[48,49].

Modularity. A modular network structure (Fig. 2, center) implies the existence of well-connected subgroups, which can be identified given the right heuristics to do so. Unlike nestedness (which apparently emerges only in very specific circumstances), modularity has been reported in almost any kind of systems: from food-webs[50] to lexical networks[51], to the Internet[27] and social networks[52]. Applied to OSS developer-file networks, modularity helps to identify blocks of developers working together in a set of files. High Q values in OSS projects would reveal some level of specialisation (division of labour) in the development of the project. However, if an OSS project is only modular (i.e., any trace of nestedness is missing), it may reveal that, beyond compartmentalisation, no further organisational refinement is at work. Here, we search a (sub)optimal modular partition of the nodes through a community detection analysis[26,27]. To this end, we apply the extremal optimisation algorithm[53] (along with a Kernighan-Lin[54] refinement procedure) to maximise Barber's[26] modularity Q,

$$Q = \frac{1}{L}\sum_{i=1}^{N}\sum_{j=N+1}^{N+M}(\tilde{a}_{ij} - \tilde{p}_{ij})\,\delta(\alpha_i, \alpha_j) \tag{2}$$

where L is the number of interactions (links) in the network, $\tilde{a}_{ij}$ denotes the existence of a link between nodes i and j, $\tilde{p}_{ij} = k_i k_j / L$ is the probability that a link exists by chance, and $\delta(\alpha_i,\alpha_j)$ is the Kronecker delta function, which takes the value 1 if nodes i and j are in the same community, and 0 otherwise.

In-block nestedness. Nestedness and modularity are emergent properties in many systems, but it is rare to find them in the same system. This apparent incompatibility has been noticed and studied, and it can be explained by different evolutive pressures: certain mechanisms favour the emergence of blocks, while others favour the emergence of nested patterns. Following this logic, if two such mechanisms are concurrent, then hybrid (nested-modular) arrangements may appear. Hence, the third architectural organisation that we consider in our work refers to a mesoscale hybrid pattern, in which the network presents a modular structure, but the interactions within each module are nested, i.e. an in-block nested structure, see Fig. 2 (right). This type of hybrid or "compound" architectures was first described in Lewinsohn *et al.*[28]. Although the literature covering this types of patterns is still scarce, the existence of such type of hybrid structure in empirical networks has been recently explored[29,30,55], and the results from these works seem to indicate that combined structures are, in fact, a common feature in many systems from different contexts.

In order to compute the amount of in-block nested present in networks, in this work, we have adopted a new objective function[30], that is capable to detect these hybrid architectures, and employed the same optimisation algorithms used to maximise modularity. The in-block nestedness objective function can be written as,

$$\mathcal{I} = \frac{2}{N+M}\left\{\sum_{i,j}^{N}\left[\frac{O_{i,j} - \langle O_{i,j}\rangle}{k_j(C_i - 1)}\Theta(k_i - k_j)\,\delta(\alpha_i, \alpha_j)\right] + \sum_{l,m}^{M}\left[\frac{O_{l,m} - \langle O_{l,m}\rangle}{k_m(C_l - 1)}\Theta(k_l - k_m)\,\delta(\alpha_l, \alpha_m)\right]\right\}, \tag{3}$$

Note that, by definition, $\mathcal{I}$ reduces to $\mathcal{N}$ when the number of blocks is 1. This explains why the right half of the ternary plot (Fig. 6) is necessarily empty: $\mathcal{I} \geq \mathcal{N}$, and therefore $f_{\mathcal{I}} \geq f_{\mathcal{N}}$. On the other hand, an in-block nested structure exhibits necessarily some level of modularity, but not the other way around. This explains why the lower-left area of the simplex in Fig. 6 is empty as well (see Palazzi *et al.*[33] for details).

The corresponding software codes for nestedness measurement, and modularity and in-block nestedness optimisation (both for uni- and bipartite cases), can be downloaded from the web page http://cosin3.rdi.uoc.edu/, under the Resources section.

Stationarity test. Figures 1 and 5 visually suggest that some quantities do not vary as a function of project size –or vary very slowly. As convincing as this visual hint may result, a statistical test is necessary to confirm that, indeed, there is a limit on the quantity at stake. The idea of stationarity on a time series implies that summary statistics of the data, like the mean or variance, are approximately constant when measured from any two starting points in series (different project sizes in our case). Typically, statistical stationarity tests are done by checking for the presence (or absence) of a unit root on the time series (null hypothesis). A time series is said to have a unit root if we can write it as

$$y_t = a^n y_{t-n} + \sum_i \varepsilon_{t-i} a^i \tag{4}$$

where ε is an error term. If $a = 1$ the null hypothesis of non-stationarity is accepted. On the contrary, if $a < 1$ there is not unit root, and the process is deemed stationary. In this work, we have employed the Augmented Dickey-Fuller (ADF) test[56], as implemented in the *statsmodels.tsa.stattools* Python package. The results of the analysis indicate that, if the test statistic is less than the critical values at different significance levels, then, the null hypothesis of a unit root is rejected, and we can conclude that the data series is stationary.

References

1. Open source initiative, https://opensource.org/.
2. Schuwer, R., van Genuchten, M. & Hatton, L. On the impact of being open. *IEEE Softw.* **32**, 81–83 (2015).
3. Dabbish, L., Stuart, C., Tsay, J. & Herbsleb, J. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. In *ACM Conf. on Computer-Supported Cooperative Work and Social Computing*, 1277–1286 (2012).
4. Padhye, R., Mani, S. & Sinha, V. S. A Study of External Community Contribution to Open-source Projects on GitHub. In *Working Conf. on Mining Software Repositories*, 332–335 (2014).
5. Lima, A., Rossi, L. & Musolesi, M. Coding Together at Scale: GitHub as a Collaborative Social Network. In *Int. Conf. on Weblogs and Social Media*, **10** (2014).
6. Dabbish, L., Stuart, C., Tsay, J. & Herbsleb, J. Leveraging Transparency. *IEEE Softw.* **30**, 37–43 (2013).
7. Fitz-Gerald, S. Book review of:'internet success: a study of open-source software commons' by cm schweik and rc english. *Int. J. Inf. Manag.* **32**, 596–597 (2012).
8. Cosentino, V., Izquierdo, J. L. C. & Cabot, J. Assessing the bus factor of git repositories. In *Int. Conf. on Software Analysis, Evolution, and Reengineering*, 499–503 (2015).
9. Yamashita, K., McIntosh, S., Kamei, Y., Hassan, A. E. & Ubayashi, N. Revisiting the Applicability of the Pareto Principle to Core Development Teams in Open Source Software Projects. In *Int. Workshop on Principles of Software Evolution*, 46–55 (2015).
10. Avelino, G., Passos, L., Hora, A. & Valente, M. T. A novel approach for estimating truck factors. In *Int. Conf. on Program Comprehension*, 1–10 (2016).
11. Pham, R., Singer, L., Liskin, O., Figueira Filho, F. & Schneider, K. Creating a Shared Understanding of Testing Culture on a Social Coding Site. In *Int. Conf. on Software Engineering*, 112–121 (2013).
12. Yamashita, K., Kamei, Y., McIntosh, S., Hassan, A. E. & Ubayashi, N. Magnet or Sticky? Measuring Project Characteristics from the Perspective of Developer Attraction and Retention. *J. Inf. Process.* **24**, 339–348 (2016).
13. Hata, H., Todo, T., Onoue, S. & Matsumoto, K. Characteristics of Sustainable OSS Projects: a Theoretical and Empirical Study. In *Int. Workshop on Cooperative and Human Aspects of Software Engineering*, 15–21 (2015).
14. Bertholdo, A. P. O. & Gerosa, M. A. Promoting Engagement in Open Collaboration Communities by Means of Gamification. In *Int. Conf. on Human-Computer Interaction*, 15–20 (2016).
15. Steinmacher, I., Conte, T. U., Treude, C. & Gerosa, M. A. Overcoming open source project entry barriers with a portal for newcomers. In *Int. Conf. on Software Engineering*, 273–284 (2016).
16. Steinmacher, I., Silva, M. A. G., Gerosa, M. A. & Redmiles, D. F. A systematic literature review on the barriers faced by newcomers to open source software projects. *Inf. & Softw. Technol.* **59**, 67–85 (2015).
17. Cosentino, V., Izquierdo, J. L. C. & Cabot, J. A systematic mapping study of software development with github. *IEEE Access* **5**, 7173–7192 (2017).
18. Valverde, S. & Solé, R. V. Self-organization versus hierarchy in open-source social networks. *Phys. Rev. E* **76**, 046118 (2007).
19. Bird, C., Pattison, D., D'Souza, R., Filkov, V. & Devanbu, P. Latent social structure in open source projects. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 24–35 (2008).
20. Hong, Q., Kim, S., Cheung, S. C. & Bird, C. Understanding a developer social network and its evolution. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, 323–332 (IEEE, 2011).
21. Dunbar, R. Neocortex size as a constraint on group size in primates. *J. Hum. Evol.* **22**, 469–493 (1992).
22. Gonc¸alves, B., Perra, N. & Vespignani, A. Modeling users' activity on twitter networks: Validation of dunbar's number. *PloS One* **6**, e22656 (2011).
23. Patterson, B. D. & Atmar, W. Nested subsets and the structure of insular mammalian faunas and archipelagos. *Biol. J. Linnean Soc.* **28**, 65–82 (1986).
24. Atmar, W. & Patterson, B. D. The measure of order and disorder in the distribution of species in fragmented habitat. *Oecologia* **96**, 373–382 (1993).
25. Newman, M. E. & Girvan, M. Finding and evaluating community structure in networks. *Phys. Rev. E* **69**, 026113 (2004).
26. Barber, M. J. Modularity and community detection in bipartite networks. *Phys. Rev. E* **76**, 066102 (2007).
27. Fortunato, S. Community detection in graphs. *Phys. Reports* **486**, 75–174 (2010).
28. Lewinsohn, T. M., Inácio Prado, P., Jordano, P., Bascompte, J. & Olesen, J. M. Structure in plant–animal interaction assemblages. *Oikos* **113**, 174–184 (2006).
29. Flores, C. O., Valverde, S. & Weitz, J. S. Multi-scale structure and geographic drivers of cross-infection within marine bacteria and phages. *The ISME journal* **7**, 520–532 (2013).
30. Solé-Ribalta, A., Tessone, C. J., Mariani, M. S. & Borge-Holthoefer, J. Revealing in-block nestedness: detection and benchmarking. *Phys. Rev. E* **97**, 062302 (2018).
31. Lee, S. H. *et al.* Network nestedness as generalized core-periphery structures. *Phys. Rev. E* **93**, 022306 (2016).
32. Borge-Holthoefer, J., Baños, R. A., Gracia-Lázaro, C. & Moreno, Y. Emergence of consensus as a modular-to-nested transition in communication dynamics. *Sci. Reports* **7**, 41673 (2017).
33. Palazzi, M., Borge-Holthoefer, J., Tessone, C. & Solé-Ribalta, A. Antagonistic structural patterns in complex networks. *arXiv preprint arXiv:1810.12785* (2018).
34. Dunbar, R. *How many friends does one person need?: Dunbar's number and other evolutionary quirks* (Faber & Faber, 2010).
35. Derex, M. & Boyd, R. Partial connectivity increases cultural accumulation within groups. *Proc. Natl. Acad. Sci.* **113**, 2982–2987 (2016).
36. Derex, M., Perreault, C. & Boyd, R. Divide and conquer: intermediate levels of population fragmentation maximize cultural accumulation. *Phil. Trans. R. Soc. B* **373**, 20170062 (2018).
37. Olsson, U. Confidence intervals for the mean of a log-normal distribution. *J. Stat. Educ.* **13** (2005).
38. Penzenstadler, B. Towards a definition of sustainability in and for software engineering. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 1183–1185 (2013).
39. https://github.com.
40. Using the request, https://api.github.com/search/repositories?q=stars:>1&sort=stars&order=desc&per_page=100.
41. Cosentino, V., Cánovas Izquierdo, J. L. & Cabot, J. Gitana: A SQL-Based Git Repository Inspector. In *Int. Conf. on Conceptual Modeling*, 329–343 (2015).

42. Bascompte, J., Jordano, P., Melián, C. J. & Olesen, J. M. The nested assembly of plant–animal mutualistic networks. *Proc. Natl. Acad. Sci.* **100**, 9383–9387 (2003).
43. Bastolla, U. *et al*. The architecture of mutualistic networks minimizes competition and increases biodiversity. *Nat.* **458**, 1018–1020 (2009).
44. Suweis, S., Simini, F., Banavar, J. R. & Maritan, A. Emergence of structural and dynamical properties of ecological mutualistic networks. *Nat.* **500**, 449 (2013).
45. Saavedra, S., Stouffer, D. B., Uzzi, B. & Bascompte, J. Strong Contributors to Network Persistence Are the Most Vulnerable to Extinction. *Nat.* **478**, 233–235 (2011).
46. Bustos, S., Gomez, C., Hausmann, R. & Hidalgo, C. A. The dynamics of nestedness predicts the evolution of industrial ecosystems. *PloS One* **7**, e49393 (2012).
47. Kamilar, J. M. & Atkinson, Q. D. Cultural assemblages show nested structure in humans and chimpanzees but not orangutans. *Proc. Natl. Acad. Sci.* **111**, 111–115 (2014).
48. Almeida-Neto, M., Guimarães, P., Guimarães, P. R., Loyola, R. D. & Ulrich, W. A consistent metric for nestedness analysis in ecological systems: reconciling concept and measurement. *Oikos* **117**, 1227–1239 (2008).
49. Ulrich, W., Almeida-Neto, M. & Gotelli, N. J. A consumer's guide to nestedness analysis. *Oikos* **118**, 3–17 (2009).
50. Stouffer, D. B. & Bascompte, J. Compartmentalization increases food-web persistence. *Proc. Natl. Acad. Sci.* **108**, 3648–3652 (2011).
51. Borge-Holthoefer, J. & Arenas, A. Navigating Word Association Norms to Extract Semantic Information. In *An. Conf. of the Cognitive Science Society* (2009).
52. Borge-Holthoefer, J. *et al*. Structural and dynamical patterns on online social networks: the spanish may 15th movement as a case study. *PloS One* **6**, e23883 (2011).
53. Duch, J. & Arenas, A. Community detection in complex networks using extremal optimization. *Phys. Rev. E* **72**, 027104 (2005).
54. Kernighan, B. W. & Lin, S. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal* **49**, 291–307 (1970).
55. Beckett, S. J. & Williams, H. T. Coevolutionary diversification creates nested-modular structure in phage–bacteria interaction networks. *Interface Focus.* **3**, 20130033 (2013).
56. Dickey, D. A. & Fuller, W. A. Distribution of the estimators for autoregressive time series with a unit root. *J. Am. Stat. Assoc.* **74**, 427–431 (1979).

Acknowledgements

M.J.P., A.S.-R. and J.B.-H. acknowledge the support of the Spanish MICINN project PGC2018-096999-A-I00 and the Cariparo Visiting Program 2018 (Padova, Italy). M.J.P. acknowledges as well the support of a doctoral grant from the Universitat Oberta de Catalunya (UOC).

Author Contributions

All authors designed research. J.C. and J.C.I. collected and curated the data. M.J.P., A.S.-R. and J.B.-H. and performed research. All authors analysed the results. J.B.-H. and A.S.-R. wrote the paper. All authors approved the final version.

Additional Information

Competing Interests: The authors declare no competing interests.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/.

© The Author(s) 2019