Mehler et al. *EPJ Data Science* (2024) 13:35
https://doi.org/10.1140/epjds/s13688-024-00475-0

EPJ Data Science
a SpringerOpen Journal

EPJ.org

RESEARCH Open Access

Who makes open source code? The hybridisation of commercial and open source practices

Peter Mehler[1,3], Eva Iris Otto[1,2] and Anna Sapienza[1,4*]

*Correspondence:
ansa@sodas.ku.dk
[1]Copenhagen Center for Social Data Science, University of Copenhagen, Øster Farimagsgade 5, 1353, Copenhagen K, Denmark
[4]Università del Piemonte Orientale, V.le Teresa Michel, 11, 15121, Alessandria, Italy
Full list of author information is available at the end of the article

Abstract

While Free and Open Source (F/OSS) coding has traditionally been described as a separate commons linked to values of openness and sharing, recent research suggests an increasing integration of private corporations into F/OSS practices, blurring the boundaries between F/OSS and commodified coding. However, there is a dearth of empirical, and especially quantitative studies exploring this phenomenon. To address this gap, we model the power dynamics and infrastructural aspects of software production within GitHub, a central hub for F/OSS development, using a large-scale, directed network. Using various network statistics, we detect the ecosystem's most impactful actors and find a nuanced picture of the influence of individuals, open source organizations, and private corporations in F/OSS practices. We find that the majority of public repositories on GitHub depend on a small core of specialized repositories and users. In accordance with expectations, individuals and open source organizations are more prevalent in this core of elite GitHub users, however, we also find a significant amount of private organizations with an indirect, yet consistent influence within GitHub. In addition, we find that directly influential individuals tend to facilitate sponsorship methods more often than indirectly or non-influential individuals. Our research highlights a hybridization of F/OSS and sheds light on the complex interplay between influence, power, and code production in the multi-language dependency ecosystem of GitHub.

Keywords: Software production; Dependency; Network; Open-source; Hybridization

1 Introduction

Free and Open Source (F/OSS) coding and its environments have long been celebrated within developer communities for embodying a unique set of values and principles. In this distinct sphere, openness and sharing stand as central tenets, fostering a collaborative ethos that supposedly sets it apart from the realm of commodified coding processes [1–3]. This contrast is emblematic of a fundamentally conceived divide in the software development world.

Within the F/OSS domain, the concept of "free" extends beyond cost; it encompasses the freedom to access, use, modify, and distribute software [4]. Developers and enthu-

Springer

siasts who participate in F/OSS projects are supposedly driven by a shared belief in the importance of transparency and community-driven innovation, promoting software as a public good, free from proprietary restrictions. In contrast, commodified coding usually revolves around proprietary software. Proprietary, or closed-source development models, prioritize safeguarding intellectual property and maximizing revenue, often at the expense of openness and sharing.

This tension between open and closed source approaches has been a defining marker of the software industry and it is repeatedly found as a near mythical tale within open-source communities. Take, for instance, the recurring stories of "Nebraska Developers" (named after the well-known comic [5])—individuals who, feeling the social values of F/OSS compromised or lacking financial compensation, withdraw their widely used F/OSS code [6–8].

These tales problematize the intricate interplay between Free and Open Source (F/OSS) coding and the considerable earnings generated by commercial software reliant on F/OSS components. Further, such tales point to the idea that particular developers, within infrastructural dependencies of code going across F/OSS and commoditized code-production, have a significant influence. However, the actual extent of this influence has not been studied yet.

While such tales highlight the tension between F/OSS ideals and commodification of code within lay software communities, recent political economic studies of software production argue that F/OSS practices are increasingly incorporated into private corporations and that private corporations encroach on F/OSS spheres of code [9]. Some scholars point to this as an encroachment of commodification processes into F/OSS spheres and a capitalist exploitation of labour, others instead, point to a growing hybridization of F/OSS and commodified coding practices influencing processes of code-production [10].

This hybridization of F/OSS coding has, however, not been studied extensively, empirically (for exceptions see [10, 11]), nor quantitatively. In short, there is a need to attend to this interplay empirically, and consider the influence of both open-source developers and private corporations.

In this study, we therefore turn our attention to a well-known environment for F/OSS coding: GitHub. Specifically, we contribute to questions about the growing hybridization of relations between F/OSS and commodified coding by asking: (1) Who makes influential open source code? (2) And what does the composition of actors making influential Open Source Code mean for our understanding of influence within infrastructures of code production today?

To answer these questions we operationalise the double meaning of dependency as both a technical relation within software development and a relation that carries implications for infrastructural power in software relations. In infrastructural terms, dependency gives influence to actors upstream to set standards and control access, and thereby gives a view to infrastructural power relations. To achieve this goal, we combine two different strands of research in a novel way: We build on and extend software engineering studies of dependency ecosystems and use this to engage a nascent field on infrastructural power in code-production.

We thus contribute a study that takes a point of departure in a multi-language dependency ecosystem and shifts the focus from typical questions regarding vulnerabilities, package developments etc. to considering influence and power-relations within GitHub.

We leverage state-of-the-art methods from network science to explore the dependency network of Github, contributing an empirically based study to a nascent literature on the infrastructures of code-production and their concomitant power-relations [9, 12, 13]. A literature that points to the need of quantitative studies, especially of F/OSS relations, to explore actual on-the-ground software production relations.

The following sections are organized as follows. Section 1 introduces previous work on power relations in F/OSS, on GitHub and its dependencies. Section 2 provides a description of the data, including the data collection process, the pre-processing and network definitions. Section 3 shows the results of our analysis, which we further discuss in Sect. 4. Section 5 illustrates the conclusions, including the discussion of limitations and implications for future work. Finally, Sect. 6 includes additional declarations, i.e. ethical statement, data availability, etc.

2 Related work

2.1 Power relations in free and open source code-production

Social scientific approaches, in which building code also is fundamentally considered a way of building social structures [14], have contributed three perspectives especially relevant to our study.

First, studies analyzing the social values of F/OSS coding, qualitatively, have pointed to how F/OSS is repeatedly envisioned and described as a "gift economy", motivated by other forms of value than commodification [1, 9, 14]. In this vein, F/OSS has been analyzed as a modern type of commons, in which resources are freely shared, as opposed to being bought or sold [9, 15]. Others point to how particular values of openness, sharing and "hacking" influence coding practices and expectations within F/OSS communities [2, 3]. F/OSS code production is thus considered as different and "free" from processes of commodification, even though, as will be expanded below, this is not actually so.

Second, a nascent political economic literature, focusing on macro-power relations within coding, considers the relations between F/OSS and commodified code production. Early on, Barbrook (1998) [1] pointed out the difference between the idealised divide and the de facto interconnection of F/OSS and commodified coding. This point has recently gained more scholarly consideration in the context of corporations changing their strategies and practices.

In this recent literature, Birkinbine (2020) [9], for instance, provides a political economic analysis describing the encroachment of private corporations into open source practices, a process by which companies absorb "free labour" into their process of capital accumulation. His analysis aligns with other macro-scale analyses that describe the relation between F/OSS and corporations as a growing capture of F/OSS work by capitalization and commodification processes, in which corporations are converting the value of F/OSS code into economic value [16, 17].

A different approach to the interrelation between F/OSS and processes of commodified code, has instead considered it a fundamental hybridization of economic forms (between commodified and commons) within digital production, in which F/OSS and commoditized code production are intertwined [10, 11].

Lastly, recent developments in the fields of internet and communication studies, have turned to an infrastructure approach that links (material) software and data relations with questions of power and control in code and code-production via big data (i.e. see Helles

et al. 2018 [18]). As part of this "infrastructural turn" [19] scholars have, amongst other things, mapped out the ecosystems of software development kits (SDK's) in apps [12, 13, 20], mapped cookies on the internet [21], as well as considered the material infrastructures of the internet more broadly [22].

We take inspiration from the infrastructure approach in communication and internet studies to contribute an empirically grounded quantitative analysis, exploring the dependency relations of F/OSS code. Through this approach we contribute a complementary perspective to existing qualitative studies of F/OSS code production and political economic analyses of commodification and hybridisation of code-production.

2.2 Package dependency ecosystems

Software development heavily relies on package dependency ecosystems to build complex applications using external libraries and frameworks. In recent years, researchers have studied the topology of several language-based ecosystems, including Python, R, and JavaScript, to understand dependencies and their relationships [23]. Large-scale analyses of the npm ecosystem [24] and the R ecosystem [25] have shown that, despite the growth in package dependencies, developers mainly depend on a core set of packages, and the majority of packages do not have any dependencies [26].

However, as the number of dependencies in a project increases, so does the complexity of managing and maintaining them, and ensuring the security and stability of software systems becomes challenging [27]. Researchers have proposed different approaches to prevent dependency conflicts and reduce the potential for errors when automatically downloading and installing dependencies [28, 29]. However, automated dependency updates can still lead to build failures, and developers have been found to use dependency downgrades to react to or prevent these issues [30].

Another significant challenge associated with package dependency ecosystems [31, 32] is the emergence of vulnerabilities. Researchers have highlighted the potential for running vulnerable or malicious code due to third-party dependencies and the lack of maintenance, which causes many packages to rely on vulnerable code [33]. In particular, it has been shown how removing the most popular packages has an increasing impact on vulnerabilities [34].

Overall, research on package dependency ecosystem is mainly related to software engineering studies, where the main focus is either addressing issues of security and stability or mapping and characterizing different ecosystems. Here, we build on the latter to uncover the inner structure of dependency relations of F/OSS code through GitHub. Further, we detach from applications that are typical in software engineering studies to propose a broader perspective where dependencies are used as a proxy to understand influence dynamics in F/OSS code production.

2.3 GitHub as an essential space for code-production and F/OSS code

As a vastly popular platform, GitHub has been considered a particularly interesting context for exploring different topics and social dynamics, including collaborations/coordination behaviors [35], the model and prediction of user behaviors [36], and the structure of developers' social networks [37].

Here, we take an empirical point of departure by studying dependency relations on GitHub, a large-scale open source code development platform used by more than 94M users and including about 330M repositories [38].

While previous research on package dependency ecosystems has mainly focused on official sources, including CRAN for R, PyPI for Python, etc., developers have the flexibility to create and distribute packages on other platforms as well. Among these platforms, GitHub has recently emerged as a popular choice for package development and distribution.

Moreover, we focus on the fact that GitHub is not only commonly associated with F/OSS projects, but is also increasingly a space for corporations, who engage in F/OSS-like practices of code-building and sharing [9, 11, 17]. Mackenzie [17] describes GitHub as a "platformizing" platform, a specific space of F/OSS coding that also traverses to others spaces of F/OSS and commodified code (see also [14]). As such Github is an interesting space to consider questions of hybridity.

A few studies focus on understanding package ecosystems in GitHub through dependencies. The authors of [39] investigate the use of GitHub for R package development and distribution, focusing on inter-repository package dependencies. Their findings reveal that more R packages are hosted on GitHub, and many are exclusively distributed through GitHub. However, the lack of support for dependency constraints in R and the absence of centralized package listings on GitHub lead to inter-repository dependency issues, such as update and compatibility problems.

In Ma et al. [40], the authors examine the GitHub Python ecosystem to identify influential projects in relation to their centrality and popularity. The authors find that influential projects are often custom libraries, and only a small number of projects are considered central. Moreover, they show that dominant projects are not always popular among the GitHub users.

Finally, Blinco et al. [41] introduce Reference Coupling, a method for detecting technical dependencies between projects using user-specified cross-references. The method uncovers untracked dependencies, and identifies ecosystems in GitHub, highlighting the interconnectedness of popular ecosystems centered around software development tools.

Overall, these studies did not consider the power relations characterising F/OSS nor try to identify the main actors in this ecosystem as a whole. Instead, they have mainly focused on mapping the structure and evolution of language-specific dependencies. Our study extends previous literature on package dependency ecosystems, by also considering the possible interconnections that packages and projects within cross-language dependencies might have.

Moreover, the very definition of dependency varies across different studies. While some propose methodologies to parse source files to find dependencies [40], others propose new definitions, such as the reference-coupling proposed by Blinco et al. [41], which, however, mostly captures collaborations between developers instead of direct package dependencies.

We propose to use the technical dependencies tracked by GitHub, thus, using its technical capacities to capture dependencies. This dependency relation allows us to investigate the structure of influence relations within GitHub, thereby uncovering the role and types of its influential owners. As such, our work contributes to a nascent field uncovering the infrastructures and power relations of code-production, while connecting it to the software engineering studies on package dependencies.

3 Data collection and description

Dependency definition To map the package dependency ecosystem of GitHub, we first need to provide a definition for dependencies.

Contrary to previous work on GitHub, where dependencies are defined via reference-coupling [41, 42], and thus capturing a more collaborative relation between coding projects, we rely on the dependency definition provided by the platform to capture technical dependencies between coding projects. This approach also allows to capture dependencies across different coding languages by following a unique approach, as opposed to infer them by the repository source code, as previously done in studies focusing on a specific language ecosystem.

GitHub describes a dependency as a "summary of the manifest and lock files stored in a repository and any dependencies that are submitted for the repository using the Dependency submission API" [43]. Such dependencies are automatically identified by the GitHub's Dependabot [44], which scrapes all repositories and searches for common file types housing information about a repository's dependencies. Dependabot then aggregates such files to compile a list of all dependencies. We collect the resulting list of dependencies via the GitHub's GraphQL API [45], which provides information about the dependency graph of repositories.

Data collection process To build the GitHub dependency ecosystem, we query the GitHub GraphQL API, using a recursive approach.

We first use the GH Archive [46], which records the public GitHub timeline and more than 20 types of events, to detect the set of all repositories that were active, i.e. repositories on which users made some actions, during the observation period spanning from the 1st to the 31^{st} of January 2020. This results in $2.3M$ repositories, of which we use an initial sample of $200K$ repositories as seeds in our recursive approach.

We iterate through the list and query the GitHub GraphQL API to return the list of dependencies of each seed. Then, we repeat this step on the list of new repositories we found. The process continues until it converges, i.e. no new repositories are found.

Note that, for some repositories, the GitHub GraphQL API does not return any dependencies. This could be due either to the repository not depending on any other repository on GitHub, or to missing dependencies. As we cannot distinguish between the two scenarios, we consider repositories without incoming dependencies in our analysis only to study the overall structure of the package dependency network (see the Results section), as directly filtering them would cause to miss some or all of their dependency relations.

Finally, for each repository in our data, we also collect their metadata via the GitHub REST API.

Metadata We use the GitHub REST API [47] to collect additional repository metadata: the *user type* and the *repository language*. The user type is the type of user that owns a certain repository. GitHub distinguishes between two user types: the type *user*, which indicates individuals owning repositories with their personal accounts, and the type *organization*, which is related to shared accounts where an unlimited number of people can work from one account at once. The repository language is the primary programming language used to develop the project associated with a certain repository.

Data preprocessing and network representation The dataset includes a total of 529,430 repositories having 32,084,227 dependency relationships and 358,922 unique users. To answer our research questions, we study the GitHub dependency ecosystem at two levels: the repository level and the owner level.

Table 1 Summary statistics of the repository (RDN) and the owner (ODN) dependency network

		RDN		ODN	
	N. repos/owner	In degree	Out degree	In degree	Out degree
std	6.8	89.7	1576.3	180.8	1473.8
min	1.0	0.0	0.0	0.0	0.0
25%	1.0	0.0	0.0	1.0	0.0
median	1.0	8.0	0.0	27.0	0.0
75%	2.0	104.0	1.0	148.0	1.0
max	408.0	1098.0	154,745.0	38,808.0	142,459.0

We start by modelling and preprocessing the *repository dependency network* (RDN). Here, we represent the GitHub ecosystem as a dependency network, whose nodes are repositories and links are dependencies. Links in this network are directed: given the repositories i and j, we create a link e_{ij} starting from i and pointing to j if repository j is dependent on repository i.

Starting from this network, we remove self-loops (a total of 8200), i.e. dependency relations between a repository and itself, and extract the weakly connected component (WCC) of the network, to avoid considering disconnected parts of the ecosystem in the analysis. Our final repository network has a total of 527,422 repositories and 32,076,027 dependency relationships.

To uncover the influence relation of GitHub users and their repositories we aggregate the RDN at the level of the repositories owners and build a *owner dependency network* (ODN). The ODN allows us to uncover dependency relations among GitHub developers, and thus highlights the overall influence each developer has in the GitHub ecosystem.

We first aggregate all repositories belonging to the same user into a single node. Given the set of repositories R_A and R_B, belonging to owner A and B respectively, we then define a new link e_{AB} going from A to B, if any repository $r' \in R_B$ is dependent on any repository $r \in R_A$. Finally, we create the weighted directed network by weighting the link e_{AB} with the count of all the dependencies going from A to B, i.e. $w_{AB} = |\{r' \in R_B | \exists r \in R_A : r' \text{depends on } r\}|$.

For instance, let us examine the repository tensorflow/tensorflow in the RDN, which relies on two packages from the same keras-team user: keras-applications and keras-preprocessing. In the RDN representation, we observe three nodes, with both keras nodes having an unweighted outgoing edge to the tensorflow/tensorflow node. In contrast, the ODN condenses these three nodes into two, representing the owner of tensorflow/tensorflow and the owner of keras-applications and keras-preprocessing. These nodes are linked by a single edge with a weight of 2, indicating the degree to which the first owner depends on (and thus is influenced by) the code developed by the second owner.

Our final owner network has a total of 358,922 users and 16,313,939 dependencies. Table 1 reports an overview of the summary statistics for the degree distributions (in and out) of the two networks.

We use high performance network analysis packages *graph-tool* [48] and *igraph* [49] in Python in the following analysis.

4 Results

4.1 The structure and organization of GitHub dependency ecosystem

We first map the structure of the GitHub ecosystem as a whole by studying the RDN at the macroscopic level.

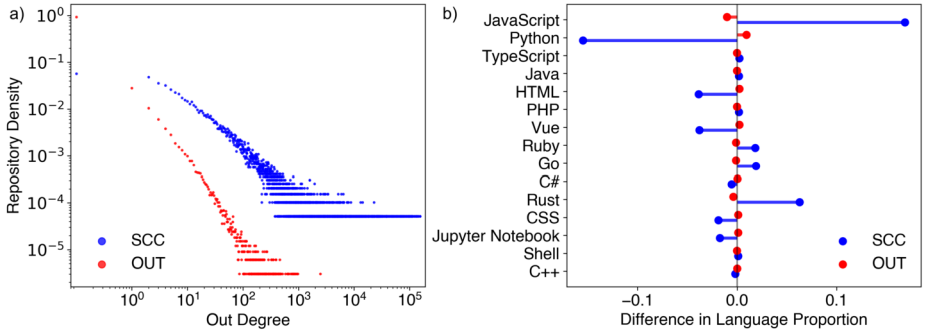

Figure 1 Degree distributions and programming languages proportion of the two main components of the network.: (a) out degree distribution of SCC and OUT; (b) difference in proportion of programming languages in SCC and OUT with respect to the whole network

We are interested in understanding how dependencies tie different parts of the network together and detecting its main components. To this aim, we analyse the WCC via a bow-tie analysis, following the procedure described by Broder et al. [50], which aims at identifying three main components in the network: (1) the strongly connected component (*SCC*); (2) the *IN* component, which includes nodes that give dependencies to the SCC but do not depend on the SCC; and (3) the *OUT* component, which includes nodes that are dependent on the SCC but do not provide dependencies to it.

Note that some nodes in the WCC might not belong to any of these components. As we are interested in the overall structure of the RDN, we disregard this set of nodes from our component analysis.

The bow-tie analysis on the RDN yields two main components: a core, i.e. the strongly connected component (SCC), where dependencies are mutually given between repositories, and an out component (OUT), i.e. a group of repositories receiving dependencies from either the core or other repositories in the group but that do not provide any dependencies to the core.

Note that we also find repositories belonging to the IN component. However, these are repositories for which we do not have any information related to their dependencies. As this could be due to missing information, we disregard the IN component in the following analysis.

We find that the OUT component includes the vast majority of repositories in the network (i.e. 94.5% which is about 331.6K repositories), while only 5.5% of repositories (i.e., 19.4K) make part of the SCC. Both components display power-law-like distributions of the out degree (Fig. 1(a)), indicating that the minority of repositories provide the majority of dependencies.

Moreover, the repositories in OUT heavily rely on the SCC: 84.1% of all dependencies are given by repositories in SCC to repositories in the OUT component. These dependencies are not provided by a limited set of repositories that act as gatekeepers between the SCC and the OUT component. Instead, about 90.1% of all nodes in SCC provides at least a dependency to nodes in OUT.

We further characterize these main structures of the network in light of the repository's most prevalent language.

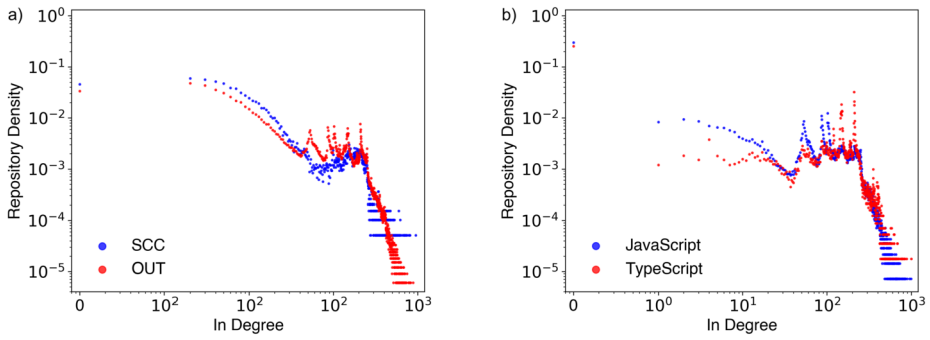

Figure 2 Degree distributions. (a) in degree distribution of SCC and OUT; (b) in degree distributions of JavaScript and Typescript repositories in the network

We compare the language distribution of each component to the one of the overall network (Fig. 1(b)). While the OUT component mostly resembles the overall network, we find that the SCC is mainly organized around JavaScript, having about 15% more repositories in this language than the overall network. This is almost the opposite for Python repositories, which are about 15% less likely to be in the SCC than in the whole ecosystem.

The difference in language distributions could suggest a relation between the typical use of different coding languages and their accessibility. While Python frameworks are widely used in different settings (e.g. Instagram, TensorFlow, etc.) it is often a beginner-choice when approaching certain projects and as such it is generally more accessible than for instance JavaScript.

This result is further supported by the analysis of the in degree distributions of the two components (Fig. 2(a)). Here, we observe some irregularities in the shape of the distributions of both the SCC and OUT component. While both components show a higher probability of importing a number of dependencies for certain degree ranges (e.g. between 100 and 500), the OUT component also displays some peaks.

By investigating the repositories generating these peaks, we find that both JavaScript and TypeScript offer pre-built project skeletons, which create a pre-set number of dependencies to the same repositories on GitHub (Fig. 2(b)). This phenomenon is only related to the OUT component, suggesting that repositories in JavaScript and TypeScript in this part of the network are more likely to be made from pre-built and accessible code.

Finally, we study the two components in light of the their user type. We find that, in proportion, organizations have a much higher presence than individual developers in the core of the GitHub ecosystem: organizations own 47.8% of repositories in the SCC, while this is the case for only 19.1% of the repositories in OUT.

Taken together, these results show that despite the open source organization of GitHub, projects and, thus, dependencies are not equally distributed. The majority of projects ultimately depend on a very small subset of the entire ecosystem: a core, which consists of a different language and user composition and that tends to be more professionally organized compared to the rest of the network. For now we home in on this core: If this core creates the code on which the rest of the network depends, and it is differently organised as well, it begs the question of a closer look at exactly how this core is organised, to which we turn now.

Table 2 Embeddedness. Average Embeddedness and standard deviation of nodes for the partition achieved by different community detection algorithms. The highest value is reported in bold

	Infomap	Walk-Trap	Leiden
Embeddedness	0.857 ± 0.22	$\mathbf{0.939 \pm 0.12}$	0.719 ± 0.23

Table 3 Percent of a community's repositories which are organizations and the number of repositories in each community. We report the community size together with the prevalence of organizations (here Percent Organization). Communities are sorted by size

Community index	2	1	3	0	6	5	7	4	8	9	10	11	12	13
Community size	12,018	2517	1534	911	876	673	498	284	86	8	7	3	3	2
Percent Organization	43.9	53.4	41.5	62.8	53.4	64.3	74.5	67.7	38.4	100.0	100.0	0.0	100.0	0.0

4.2 The modular structure of the GitHub ecosystem core

Having identified that there is a marked difference in structure and composition of languages and actors between the SCC and rest of the GitHub ecosystem, we are now interested in understanding more closely how the core of the GitHub dependency ecosystem is structured. To this aim, we identify the building modules of the SCC via state-of-the-art community detection algorithms.

Previous research studying package dependency networks has typically used the Louvain method to identify communities. Despite Louvain being one of the most popular community detection algorithms, it may find badly connected communities [51]. Thus, we consider Leiden as a modularity optimization approach, which is proposed as a better alternative for Louvain [51] and compare it to Walk-Trap [52] and Infomap [53], which are dynamical processes approaches [54]. For each method, we follow a grid-search approach and find, when relevant, the set of best performing parameters.

We then compare the quality of the partition identified by the methods by computing the embeddedness score [55]. We compute the embeddedness of a given community detection algorithm as the average of the embeddedness of its nodes, i.e. the proportion of internal and total degree of a node in the community. It is defined as:

$$embeddedness = \frac{1}{N} \sum_{i=1}^{N} \frac{k_{ic}}{k_i}, \tag{1}$$

where n is the number of nodes in the network, k_i is the degree of a node and k_{ic} is the within community degree of a node. Here, degrees are the sum of both in and out edges.

Table 2 shows the embeddedness score and its standard deviation for each community detection algorithm. We find that the methodology achieving the highest embeddedness score is Walk-Trap, followed by Infomap, and Leiden. We thus select Walk-Trap, which divides the core of the RDN in 14 communities of different sizes spanning from a few nodes, i.e. five communities under 10 nodes, to a few thousand, i.e. the biggest community has 12,018 nodes. Table 3 shows the community size breakdown.

To shed light on the composition of these communities, we consider two aspects characterizing their repositories: the programming language and the ownership.

Starting from the programming language, we find that, the uncovered communities are mainly language-based, i.e. only one language characterizes the majority of repositories in the community, and could map to more typical language ecosystems. In particular, we

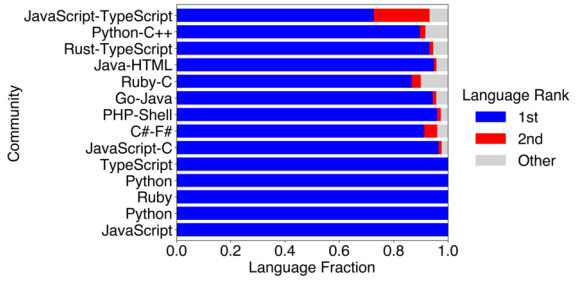

Figure 3 Coverage of top two languages in each community. We report the fraction of repositories in each communities that are characterized by the first (blue) and the second (red) most frequent languages. Labels on the x-axis refer to the first and second largest languages in each community respectively. Other languages are shown in grey. Communities are displayed in order of size

find that only the largest JavaScript community, has its first language covering less than 80% of repositories in the community (see Fig. 3).

However, if we consider its second top language, we find that JavaScript together with TypeScript make up 93.2% of the total number of repositories in the community. This relation is consistent with TypeScript being superset of JavaScript: code developed in JavaScript is valid TypeScript code, making these two languages interdependent.

While communities are primarily language-based, all communities (with more than 10 nodes) are characterized by more than one language. The presence of multiple languages in the communities could be due to the way projects and packages are developed in GitHub. For example, the Python ecosystem has many popular repositories which are based on C++, like Tensorflow. Additionally, C# and F# both exist within the .NET ecosystem and have a Common Intermediate Language allowing streamlined connections between the two languages.

For our purposes of studying dependencies and thereby structural power-relations in code production, these interactions between languages show that it is not sufficient to consider one-language ecosystems. A point we return to in the discussion.

Finally, we investigate the composition of communities. Table 3 shows that organizations are present in all communities to different levels.

Considering only communities with more than 10 repositories, the percentage of repositories which are organizations ranges from 38% to 74%. Four out of five of the smallest communities have a single owner, and thus have either 0% or 100% ownership by an organization.

We then select the top owners of each community and consider the number of repositories they have in the community compared to the average user in that community (Table 4). While the top owners tend to be mostly companies, we find the presence of both individuals and companies. Among the top owners which are companies we find well-known private tech companies such as Google, and open-source companies, such as Apache.

The percentage of community owned by the top owner varies from 1.6% for bigger communities to 100.0% for smaller communities. Despite the small absolute percentage observed, the number of repositories owned by top owners is substantial when compared to the average number of repositories owned (Table 4).

We show the relative difference calculated as $(n_{\text{top}} - \overline{n})/\overline{n}$ to represent the number of times more repositories that the top user has (n_{top}) in comparison with the average user

Table 4 Top owners in each community. We show the number of repositories owned by the first, second, and third largest owners in the respective community as well as statistics regarding the prevalence of the top owners in each community. Communities are displayed in order of size

Community index	Top owner	First owner	Second owner	Third owner	Other owners	Top Owner percent	Average N. repos	Relative difference
2	Individual 1	408	198	94	11,318	3.4	2.6	157.0
1	googleapis	40	22	22	2433	1.6	1.6	23.5
3	rust-lang	34	33	24	1443	2.2	1.7	18.8
0	apache	110	27	20	754	12.1	1.7	63.4
6	ruby	37	21	21	797	4.2	1.9	18.9
5	hashicorp	45	22	17	588	6.7	1.8	24.2
7	symfony	52	24	22	400	10.4	2.7	18.2
4	dotnet	22	19	13	228	7.7	1.7	11.9
8	Individual 2	50	16	6	13	58.1	7.8	5.4
9	ProseMirror	8	0	0	0	100.0	8.0	0.0
10	blockdiag	7	0	0	0	100.0	7.0	0.0
11	Individual 3	2	1	0	0	66.7	1.5	0.3
12	ethereum	3	0	0	0	100.0	3.0	0.0
13	Individual 4	2	0	0	0	100.0	2.0	0.0

in a community ($\overline{n}$). Take for instance the relative difference between the top owner of community 2 corresponding to 3.4 percent ownership. Here, 408 repositories belong to the same owner, which corresponds to almost 157 times more than the average owner within the community (owning 2.6 repositories).

4.3 Influential owners in the ecosystem

Having found that there are specific owners that own relative large portions of core communities, we want to focus more directly on the influential owners in the network. To understand who the most influential developers are and how they shape the open source environment of GitHub, we compute different measures of influence, i.e. centrality measures, on the ODN and select the top 50 users. In particular, we consider:

- *Out degree centrality:* it ranks nodes by their out degree, so that the higher the out degree the more influential is the node. In the present case, an owner is influential if a high number of users in GitHub depend on them. We consider this type of influence a *direct influence metric*, as the user importance is visible within dependent repositories.
- *Betweenness centrality:* it is based on the concept of shortest paths and it is computed by considering the proportion of shortest paths that pass through nodes in the network. The higher the proportion the more influential is the node. We consider an owner with high betweenness to act as an *intermediary*, connecting different parts of the network by both depending on and providing dependencies to other nodes.
- *Eigenvector centrality:* it considers a node influential if it connects to other influential nodes in the network. Thus, an owner's influence is not solely determined by the number of dependencies they give, but also by the other influential owners that depend on them. As such, this can be interpreted as an *indirect influence metric*.
- *PageRank:* it is based on the concept that important nodes are likely to be connected to other important nodes. It assigns a score to each node based on the number and quality of incoming links it receives from other nodes and the rank of these nodes. To compute this metric on the owner network, we switch the direction of links so that, owners that give many dependencies to other highly ranked owners are considered more important. While PageRank is connected to eigenvector centrality, in that it

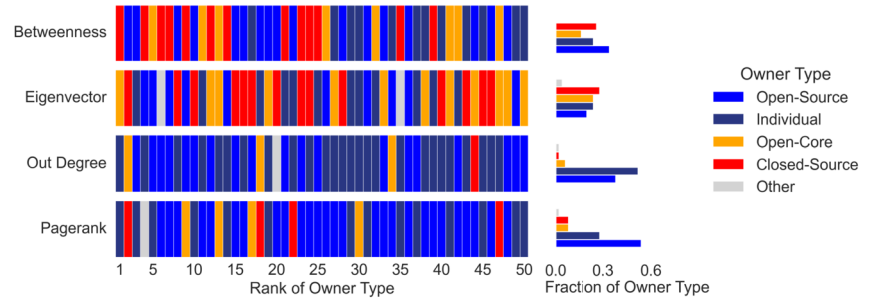

Figure 4 Top 50 owners by centrality measure Colors refer to different types of owners. For each centrality metric, we report the fraction of owners of a certain type, i.e. individuals, open source, open-core, and closed-source organizations, and others as a bar plot summarizing the owner types of each top 50 ranking

considers the importance of a node's neighbors, it also heavily relies on the out degree of nodes.

While GitHub provides a high-level characterization of users into two types (i.e. individuals and organizations), we provide a fine-grained characterization of the top 50 owners identified by the centrality measures through the following annotation task. We asked 3 independent annotators to inspect each owner profile, gather information about the type of owner, and identify the presence of sponsorship methods (if any).

The annotators are instructed to classify owner types in *individuals*, i.e. GitHub users that do not belong to any organization, *open source* organizations, not selling any products *open core* organizations, i.e. organizations selling services and features extending their open source code, *closed source* organizations, private companies that mainly sells proprietary software and do not typically have open source products and *other*, e.g., universities, governmental organizations, etc.

Sponsorship methods are defined as any service provided on the users GitHub page, which is used to facilitate the payment of someone for their open source work.

Annotators are told to first look for the GitHub sponsor link with the pink heart in the main owners page or in the sponsor section of one of their top repositories. If this exists, they are listed as using GitHub as a sponsorship method. Thereafter, other sponsorship methods are checked for in the same way. We did not provide a fixed list of sponsorship methods, and annotators were free to create new labels for the new sponsorship method found.

Through this approach we most repeatedly found the sponsor methods *GitHub*, *Tidelift*, and *Open Collective*, which we separate out within our analysis, while less recurring methods are aggregated together in the analysis under the label *Other*.

Figure 4 shows that individuals are the most prevalent central owners when considering direct influence (i.e. out degree) and open source organizations are the most prevalent when considering direct influence of their connections (i.e. PageRank).[1] The same individual is the most influential user for both Out degree and Pagerank, and has the highest number of repositories in the largest community.

Despite the relatively large prevalence of closed and open-core users in eigenvector and betweenness ranks, open-source organizations make up the largest proportion of influen-

[1] note that the closed source companies in second position of PageRank is actually GitHub with GitHub actions.

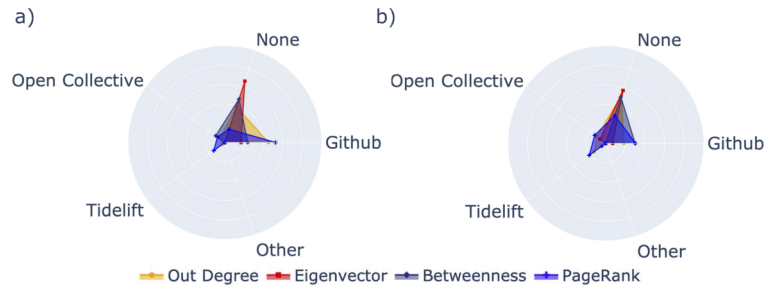

Figure 5 Sponsorship method by type of owner. The radar plots show the percentage of (a) individuals and (b) open source organizations with sponsorship methods in a certain ranking. Circles in the radar refers to the fractions 0.2, 0.4, 0.6, 0.8, and 1. Closed source organizations and Other types of organizations are here omitted as no type of sponsorship method has been found

tial users when considering all users within the top 50 of all metrics with 73 users, closely followed by individuals with 64 users. These are relatively large compared to the 32 and 27 closed sourced and open-core users. This result indicates that open source organizations and individuals are the main actors that provide dependencies to general repositories in GitHub.

However, the presence of certain types of owners is more balanced in the case of eigenvector centrality. Here, we find a higher presence of closed source and open core organizations than in the other rankings. Among the most known private companies we find Microsoft (2nd), Google (8th), Facebook (23rd), and Amazon (24th). The eigenvector centrality measures how influential owners are in giving dependencies to other influential owners. Thus, while private organizations do not provide the most direct dependencies in the network they are generally better connected to other influential nodes than open source users.

These results show that, while GitHub is typically imagined as an open-source space for individuals and open source organisations, closed and open core organizations are present and important to the ecosystem.

Finally, we find that most influential developers and open organizations on GitHub (by Out degree and PageRank) provide ways for direct sponsorship.

In the case of individual developers, the main sponsor method found, is the function directly provided by GitHub, but we also find Tidelift, Open Collective and other forms of sponsorship in smaller proportions (Fig. 5(a)). While individuals dominate the use of digital sponsorship links, open source organizations use sponsor methods as well, although in different proportions (Fig. 5(b)).

While GitHub remains the main sponsor method used, open source organizations tend to use Tidelift and Open Collective slightly more than individuals. This observation is in line with the characteristics of the sponsor types, where GitHub allows users to sponsor individual developers directly, while Tidelift and Open Collective are platforms providing ways to fund open source projects development and maintenance.

While the presence of sponsorship methods can be a good way for platforms like GitHub to directly support and encourage developers, this indicates again a level for more professional organization surrounding their coding practices and it indicates the prevalent hybridisation of commodified and open source code.

5 Discussion

In this study, we contribute a state-of-the-art network analysis to two growing fields of literature by studying: (1) power relations of code production, through (2) package dependency ecosystems. To this aim, we consider GitHub, a platform that plays a large role for both F/OSS and commodified code production, and study its dependencies.

We find a small core of repositories that delivers the majority of dependencies to projects in the whole of GitHub. The majority of GitHub projects, thereby, rely on a small subset of the overall ecosystem. This result is in line with previous studies on package dependencies, that have shown how, despite a large growth in the number of dependencies, the majority of packages in different language-based ecosystems depend on a few core packages [26].

In contrast to most dependency studies, however, we do not limit our dependency analysis to one programming language. It is, thus, interesting to note how this underlying structure is preserved in the context of multiple programming languages and their inter-relations. Our results clearly demonstrate that not all F/OSS repositories on GitHub are equal in terms of influence, which contributes a quantitative perspective on power in larger F/OSS ecosystems to studies focused on values or specific F/OSS communities.

Looking more closely at the repositories in the core in comparison to the others, we find a difference in the language distribution. Some of this difference can be interpreted through the roles different languages play within coding.

Take for instance Python which we pointed to in the results section. Python is less present in the core than in the rest of the ecosystem; it is commonly cast as being a more accessible and beginner-friendly language. As such, it is not far-fetched to think that one common use for Python repositories are individually oriented learning- or beginner repositories, not explicitly aimed at developing code that others would rely on.

Conversely, we find a strong presence of JavaScript and TypeScript within the core. This is an interesting find, and could point to the popularity of open-source tweaked or "custom" solutions within these common languages for web-development. In other words, there might be a large demand for such solutions outside Github that drives the development of these repositories in JavaScript/TypeScript within the core. Such a hypothesis points to further questions concerning the relation between open-source spaces and commercialised coding spaces, for further studies.

In addition to the language difference, we observe a much higher proportion of the owner type "organization" within the core. Such a finding might indicate that individual projects turn into organizations as they become more popular, as for instance the Apache Incubator. While this is a hypothesis, calling for further studies, we can establish that a higher concentration of organizations within the core contributes to the conclusion that the core is more specialized in comparison with the overall network.

In short, we find that this core is not only special in terms of facilitating a relative small proportion of projects others rely on, but crucially, also that it is specialized (in terms of language) and differently structured, with a higher weight of organizations (in terms of ownership).

Overall, we thereby find what we could interpret as a Github-elite. While open source project organization has been studied on a project basis, this finding of an elite within the larger GitHub ecosystem is novel. It points to the necessity of understanding more of the role that this Github-elite plays in code production within F/OSS spaces. Further, it

nuances the view of the "Nebraska developer" as probably more than just a "random guy". It begs the question of further enquiry of who this elite is?

To consider this, we turn to uncover the communities within the ecosystem's core. When considering the owner distribution of repositories in different communities of the core, we find both organizations and individuals as top owners of repositories within these communities. In addition, we find that specific individuals own relatively high proportions of communities, and that communities tend to map onto languages, although not entirely.

Considering the implications of these dependency structures in terms of influence within code-production, we see a network organization in which specific individuals and organizations play an influential role in a whole language base. This points to further questions for empirical studies of influence within relations of code-production, such as studies considering why these owners are dominant. Is it because they were the first on the scene? What would it mean for the chances of newcomers to become influential owners, and what possibilities and restraints does this offer for private tech companies also present within the F/OSS ecosystem?

Looking closer at the organizations that own a large proportion of repositories across both communities and languages, as mentioned, we find well-known tech companies, such as Google. As such, the interplay between languages is not only given by the direct dependencies of projects, but also by the presence of certain actors in different language-based ecosystems. This result provides insights on how private companies seem to incorporate themselves in F/OSS dependency relations.

As noted, the literature considering the encroaching privatization—or hybridization—of F/OSS and commodified code-relations, mentions these tendencies of private companies, but lacks an empirical attention to the ways in which hybridisation occurs. Here we find one such way, through which large private companies seemingly successfully insert themselves as large owners within the communities comprising the core of the GitHub ecosystem.

In terms of infrastructural influence, we would expect the presence of large tech companies in and across different language communities, to overlap with the strategies and software development of these companies—at least to some degree. The finding of these companies, as significant actors within different communities, therefore opens up a host of new questions concerning the interplay and influence that such actors might have in these communities—and in forming the relations between these communities—over time.

Taking the above finding as a point of departure for future studies looking at the development of this network over time, would give important insights into how such actors can or cannot influence these dependency relations. Are they stable structures once built? Which translated into questions of power and influence, would give insights into the reach and limits of power that private large tech companies have to influence these relations.

Finally, we have found individuals owning significant portions of repositories in core communities within languages. This leads us to a last question: are individuals also important in terms of the influence they wield through their dependency connections? To look at this, we turn to different types of influence measures.

When looking at direct influence, through the out-degree centrality, i.e. the amount of dependencies given, we note how open source organizations and individuals are the most influential. Relatedly, individuals also act as important intermediaries within the core.

However, this changes when looking at eigenvector centrality. Here, closed source and open core organizations are more present and provide indirect dependencies; what can be considered an indirect power, not immediately obvious to the common user depending on popular repositories. Looking at the specific closed organizations these entail the global biggest tech companies (Google, Amazon, Microsoft etc.).

Overall, we observe an interesting dynamic. Open source organizations and individuals are most prevalent when considering direct influence, while closed source organizations become more prevalent when considering indirect forms of influence. This demonstrates the specific ways in which encroachment of private companies into F/OSS space can happen. It is not immediately as visible as for instance the type of influence based on giving many direct dependencies. However, it can potentially carry quite a big influence behind the scenes, we thereby consider this an indirect form of power.

Noting this, we also study the sponsorship methods that owners have. Here, we find an interesting correlation between the fact that many of individuals ranking highly in terms of out degree and PageRank, also make use of sponsorship methods. This is, however, not so evident for individuals that rank highly in terms of eigenvector centrality. We therefore see a relation between direct dependencies—what most closely relates to visibility and ways to turn F/OSS work into paid work.

6 Conclusions

In conclusion, our analysis reveals a hybridization of F/OSS and commodified coding practices at different levels. We find private tech organizations are integrated into the core of the GitHub dependency ecosystem and exert an indirect influence vie their dependencies. Additionally, individuals facilitating most direct dependencies within this core engage in sponsorship to support their work, prompting a nuanced understanding of commodification processes.

These findings nuance critiques of commodification processes within F/OSS as a unidirectional capitalization of "free labour" (e.g., [16]) and show a close entanglement of influence and commodification that both private organisations and individuals engage with in different ways, thereby opening up a new avenue of questions for studies of commodification processes.

Broader perspective Our analysis underscore the hybridization of F/OSS and commercial coding practices, prompting a broader discourse on corporate encroachment in open source. Politically, corporate influence over F/OSS projects raises concerns about community-driven practices and participatory ideals. Economically, integration of F/OSS components into commercial products may restrict access, impacting the freedom traditionally associated with F/OSS and the sustainability of independent contributors. Increased corporate involvement risks power imbalances, potentially limiting diversity, innovation, and meaningful participation of smaller contributors in open source development.

Implications for future studies We propose three avenues for future research. Firstly, we suggest broadening the scope of software dependency studies to include multi-language communities and exploring dependencies as indicators of infrastructural power.

Secondly, investigating the historical development of dependency relations could shed light on how certain individuals and companies gained the influence within the network

that they have today. This perspective could highlight possibilities and limits for gaining influence in code-production. This study thus provides a stepping stone for the nascent field of infrastructural power within internet and communication studies [19, 21].

Lastly, we bring together discussions about value spheres and hybridization in software production commonly "abstracted away" [16] by empirically going "close to the code" (for other examples see Fourcade and Kluttz 2020 [10], Otto et al. 2023 [11]). By framing the study of hybridisation as a multi-sphere field, we show how it is fruitful for future studies of software power to consider F/OSS as entangled with commodified code production.

Limitations We acknowledge the following limitations. First, while the use of GitHub's public data is reasonable given our focus on open source, it captures only part of the ecosystem. Second, self-selection bias may also affect our findings, as users marking dependencies likely have a professional interest. Finally, we excluded repositories without dependencies, potentially missing influential entities. However, we note that, while conservative, this approach maintains the reliability of our findings, highlighting the unequal distribution of influence and the interplay between individuals, open source, and corporations on GitHub.

Abbreviations
F/OSS, Free and Open Source; RDN, Repository Dependency Network; ODN, Owner Dependency Network; WCC, Weakly Connected Component; SCC, Strongly Connected Component; IN, IN Component; OUT, OUT Component; GDPR, General Data Protection Regulation.

Acknowledgements
We would like to thank the annotators at the Copenhagen Center for Social Data Science (SODAS) of the University of Copenhagen, and the ERC DISTRACT team, in particular Professor Morten Axel Pedersen and Professor Anders Blok for their valuable input to previous versions of this article.

Author contributions
EO and AS designed the study and the research approach. PM performed the data analyses. All authors analysed the results and wrote the paper. All authors read and approved the final manuscript.

Funding
This work is funded by H2020 European Research Council (grant number 834540) as part of the project "The Political Economy of Distraction in Digitized Denmark". This work has further been supported by DATAFIED LIVING (grant no. 947735).

Data availability
The dataset generated and analysed during the current study is not publicly available due to the general data protection regulation (GDPR), but we share the minimal dataset that would be necessary to interpret, replicate and build upon the findings reported in the article in a repository whose link will be shared upon acceptance.

Declarations

Ethics approval and consent to participate
The data collection process was approved by the European Research Council as part of running reviews of the overall project of which this research forms part.
Following the minimization principle of the General Data Protection Regulation (GDPR), we only collected data related to dependencies and metadata needed for the study on public repositories and users.
Based on the size of GitHub, and the wide recognition of such public repositories as part of a public online forum, it is reasonable to assume that users do not consider GitHub as a private space, which would require a different ethical consideration [56]. However, we do identify particular owners, and also manually annotate their information. As a number of these owners are individuals, we consider our data as personal data according to GDPR. As such, we have followed the requirements concerning personal data collection and storage.
Because of size of the data-set and the nature of the data, it was impossible to inform data subjects of the study directly. Instead, we mitigate this by publishing the data processing on the research project website, in accordance with GDPR protocols.
Further parts of this evaluation rest on the basis that the data processing was evaluated as non-intrusive, and we also anonymize names of individual repository owners that we refer to in the paper.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]Copenhagen Center for Social Data Science, University of Copenhagen, Øster Farimagsgade 5, 1353, Copenhagen K, Denmark. [2]Center for Tracking and Society, Department of Communication, University of Copenhagen, Karen Blixens Plads 8, 2300, Copenhagen, Denmark. [3]Department of Computer Science, IT University of Copenhagen, Rued Langgaards Vej 7, 2300, Copenhagen, Denmark. [4]Università del Piemonte Orientale, V.le Teresa Michel, 11, 15121, Alessandria, Italy.

Received: 19 December 2023 Accepted: 22 April 2024 Published online: 06 May 2024

References

1. Barbrook R (1998) The hi-tech gift economy. First monday
2. Kelty C (2005) Geeks, social imaginaries, and recursive publics. Cult Anthropol 20(2):185–214
3. Coleman EG (2013) Coding freedom: the ethics and aesthetics of hacking. Princeton University Press, Princeton
4. Barbrook R (2002) The regulation of liberty: free speech, free trade and free gifts on the Net. Sci Cult 11(2):155–170
5. xkcd. Dependency; n.d. Available from https://xkcd.com/2347/
6. The Stack (2023) JavaScript library is EVERYWHERE. Its maintainer is broke. The Stack. Available from https://www.thestack.technology/core-js-maintainer-denis-pusharev-license-broke-angry/
7. Stokel-Walker C (2014) The Internet Is Being Protected By Two Guys Named Steve. BuzzFeed. Available from https://www.buzzfeed.com/chrisstokelwalker/the-internet-is-being-protected-by-two-guys-named-st
8. Collins K (2016) How one programmer broke the internet by deleting a tiny piece of code. QUARTZ. Available from https://qz.com/646467/how-one-programmer-broke-the-internet-by-deleting-a-tiny-piece-of-code
9. Birkinbine B (2020) Incorporating the digital commons: corporate involvement in free and open source software. University of Westminster Press
10. Fourcade M, Kluttz DN (2020) A Maussian bargain: accumulation by gift in the digital economy. Big Data Soc 7(1):2053951719897092
11. Otto EI, Salka JH, Blok A (2023) How app companies use GitHub: on modes of valuation in the digital attention economy. J Cult Econ 16(2):242–259
12. Sophus Lai S, Flensburg S (2020) A proxy for privacy uncovering the surveillance ecology of mobile apps. Big Data Soc 7(2):2053951720942543
13. Blanke T, Pybus J (2020) The material conditions of platforms: monopolization through decentralization. Soc Media Soc 6(4):2056305120971632
14. Geiger RS, Howard D, Irani L (2021) The labor of maintaining and scaling free and open-source software projects. In: Proceedings of the ACM on human-computer interactio, CSCW1, vol 5. pp 1–28
15. Ghosh R (2006) CODE: collaborative ownership and the digital economy. MIT Press, Cambridge
16. Parikka J (2014) Cultural techniques of cognitive capitalism: metaprogramming and the labour of code. Cult Stud Rev 20(1):30–52
17. Mackenzie A (2018) 48 million configurations and counting: platform numbers and their capitalization. J Cult Econ 11(1):36–53
18. Helles R, Ørmen J, Jensen KB, Lai SS, Menchen-Trevino E, Taneja H et al (2018) A division of labor: the role of big data analysis in the repertoire of internet research methods. AoIR Selected Papers of Internet Research
19. Flensburg S, Lai SS (2020) Mapping digital communication systems: infrastructures, markets, and policies as regulatory forces. Media Cult Soc 42(5):692–710
20. Flensburg S, Lai SS (2022) Datafied mobile markets: measuring control over apps, data accesses, and third-party services. Mob Media Commun 10(1):136–155
21. Helles R, Lomborg S, Lai SS (2020) Infrastructures of tracking: mapping the ecology of third-party services across top sites in the EU. New Media Soc 22(11):1957–1975
22. Flensburg S, Lai SS (2021) Networks of power. Analysing the evolution of the Danish Internet infrastructure. Int Hist 5(2):79–100
23. Lertwittayatrai N, Kula RG, Onoue S, Hata H, Rungsawang A, Leelaprute P et al (2017) Extracting insights from the topology of the javascript package ecosystem. In: 2017 24th Asia-Pacific software engineering conference (APSEC). IEEE, New York, pp 298–307
24. Wittern E, Suter P, Rajagopalan S (2016) A look at the dynamics of the JavaScript package ecosystem. In: Proceedings of the 13th international conference on mining software repositories, pp 351–361
25. Bommarito E, Bommarito MJ II (2021) An empirical analysis of the R package ecosystem. arXiv preprint. arXiv:2102.09904
26. German DM, Adams B, Hassan AE (2013) The evolution of the R software ecosystem. In: 2013 17th European conference on software maintenance and reengineering. IEEE, New York, pp 243–252
27. Serebrenik A, Mens T (2015) Challenges in software ecosystems research. In: Proceedings of the 2015 European conference on software architecture workshops, pp 1–6
28. Wang Y, Wen M, Liu Y, Wang Y, Li Z, Wang C et al (2020) Watchman: monitoring dependency conflicts for python library ecosystem. In: Proceedings of the ACM/IEEE 42nd international conference on software engineering, pp 125–135
29. Hejderup J, Gousios G (2022) Can we trust tests to automate dependency updates? A case study of Java projects. J Syst Softw 183:111097
30. Cogo FR, Oliva GA, Hassan AE (2019) An empirical study of dependency downgrades in the npm ecosystem. IEEE Trans Softw Eng 47(11):2457–2470
31. Pfretzschner B, Ben Othmane L (2017) Identification of dependency-based attacks on node. js. In: Proceedings of the 12th international conference on availability, reliability and security, pp 1–6
32. Staicu CA, Pradel M (2018) Freezing the Web: a study of {ReDoS} vulnerabilities in {JavaScript-based} web servers. In: 27th USENIX security symposium (USENIX Security 18), pp 361–376

33. Zimmermann M, Staicu CA, Tenny C, Pradel M (2019) Small world with high risks: a study of security threats in the npm ecosystem. In: 28th USENIX security symposium (USENIX Security 19), pp 995–1010
34. Kikas R, Gousios G, Dumas M, Pfahl D (2017) Structure and evolution of package dependency networks. In: 2017 IEEE/ACM 14th international conference on mining software repositories (MSR). IEEE, New York, pp 102–112
35. Romero D, Huttenlocher D, Kleinberg J (2015) Coordination and efficiency in decentralized collaboration. In: Proceedings of the international AAAI conference on web and social media, vol 9, pp 367–376
36. Blythe J, Bollenbacher J, Huang D, Hui PM, Krohn R, Pacheco D et al (2019) Massive multi-agent data-driven simulations of the github ecosystem. In: International conference on practical applications of agents and multi-agent systems. Springer, Berlin, pp 3–15
37. Yu Y, Yin G, Wang H, Wang T (2014) Exploring the patterns of social behavior in GitHub. In: Proceedings of the 1st international workshop on crowd-based software development methods and technologies, pp 31–36
38. GitHub. GitHub Webpage; n.d. Available from https://GitHub.com/about
39. Decan A, Mens T, Claes M, When GP (2016) GitHub meets CRAN: an analysis of inter-repository package dependency problems. In: 2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER), vol 1. IEEE, New York, pp 493–504
40. Ma W, Chen L, Zhou Y, Xu B (2016) What are the dominant projects in the github python ecosystem? In: 2016 third international conference on trustworthy systems and their applications (TSA). IEEE, New York, pp 87–95
41. Blincoe K, Harrison F, Kaur N, Damian D (2019) Reference coupling: an exploration of inter-project technical dependencies and their characteristics within large software ecosystems. Inf Softw Technol 110:174–189
42. Blincoe K, Harrison F, Damian D (2015) Ecosystems in GitHub and a method for ecosystem identification using reference coupling. In: 2015 IEEE/ACM 12th working conference on mining software repositories. IEEE, New York, pp 202–211
43. GitHub. About The Dependency Graph; n.d. Available from https://docs.github.com/en/code-security/supply-chain-security/understanding-your-software-supply-chain/about-the-dependency-graph
44. GitHub. GitHub Dependabot; n.d. Available from https://GitHub.com/dependabot
45. GitHub. GitHub GraphQL; n.d. Available from https://docs.GitHub.com/en/graphql
46. GH Archive; n.d. Available from https://www.gharchive.org/
47. GitHub. GitHub REST API documentation; 2022. Available from https://docs.github.com/en/rest?apiVersion=2022-11-28
48. de Paula Peixoto T (2018) Graph-tool documentation. Available from https://graph-tool.skewed.de/
49. core team I (2022). igraph—The Netwotk Analysis Package. Available from https://igraph.org/
50. Broder A, Kumar R, Maghoul F, Raghavan P, Rajagopalan S, Stata R et al (2000) Graph structure in the web. Comput Netw 33(1–6):309–320
51. Traag VA, Waltman L, Van Eck NJ (2019) From Louvain to Leiden: guaranteeing well-connected communities. Sci Rep 9(1):5233
52. Pons P, Latapy M (2006) Computing communities in large networks using random walks. J Graph Algorithms Appl 10(2):191–218
53. Rosvall M, Axelsson D, Bergstrom CT (2009) The map equation. Eur Phys J Spec Top 178(1):13–23
54. Fortunato S, Hric D (2016) Community detection in networks: a user guide. Phys Rep 659:1–44
55. Lancichinetti A, Kivelä M, Saramäki J, Fortunato S (2010) Characterizing the community structure of complex networks. PLoS ONE 5(8):e11976
56. Eysenbach G, Till JE (2001) Ethical issues in qualitative research on Internet communities. BMJ 323(7321):1103–1105

Publisher's Note