The Strength of Weak Ties Between Open-Source Developers

In the Python Ecosystem GitHub Stars Predict Innovation Better Than Commits

Hongbo Fang,[★☆] Patrick Park,[★] James Evans,[☆] James Herbsleb,[★] and Bogdan Vasilescu[★]
[★]Carnegie Mellon University [☆]University of Chicago
{hongbofang,jevans}@uchicago.edu,{patpark,jim.herbsleb,vasilescu}@cmu.edu

ABSTRACT

In a real-world social network, weak ties (reflecting low-intensity, infrequent interactions) act as bridges and connect people to different social circles, giving them access to diverse information and opportunities that are not available within one's immediate, close-knit vicinity. Weak ties can be crucial for creativity and innovation, as they introduce ideas and approaches that people can then combine in novel ways, leading to innovative solutions. Do weak ties facilitate creativity in software in similar ways? This paper suggests that the answer is "yes." Concretely, we study the correlation between developers' knowledge acquisition through three distinct interaction networks on GitHub and the innovativeness of the projects they develop, across over 37,000 Python projects hosted on GitHub. Our findings suggest that the topical diversity of projects in which developers engage, rather than the volume, correlates positively with the innovativeness of their future code. Notably, exposure through weak interactions (e.g., starring) emerges as a stronger predictor of future novelty than via strong ones (e.g., committing).

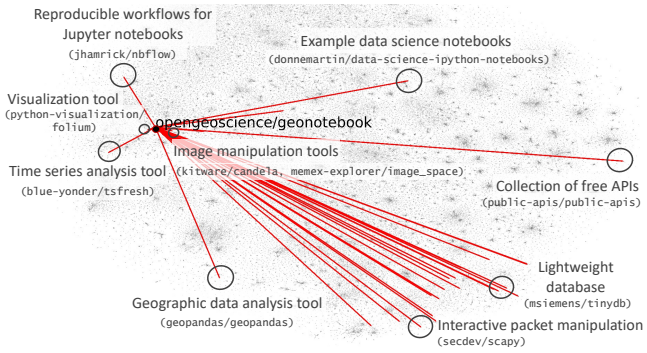


Figure 1: t-SNE visualization of the embedding space for weak ties (details in Section 3.4), depicting all the weak ties of the GeoNotebook Python project in our sample. We highlight some that seem influential for the design of the focal project.

1 INTRODUCTION

Think of examples of big software innovations. You might name the Netscape browser (created the visual web and fueled mass internet adoption), the Git version control system (revolutionized collaborative software development), Hadoop MapReduce (enabled "big data" processing), Photoshop (made digital image manipulation mainstream), Netflix (killed video rental stores), Uber ("revolutionized" transportation), or perhaps more than anything else, the World Wide Web (the global information network that connected humanity and fundamentally changed how we access knowledge).

Besides their undeniable impact, what all these innovations have in common is, perhaps surprisingly, that *none was highly novel in a pure sense*. Instead, they succeeded by recognizing latent potential in existing components that others had overlooked or dismissed as insufficient. This reveals a fundamental paradox in software innovation: breakthroughs can emerge not only from inventing entirely new concepts, but from developing the insight to see how familiar pieces can fit together in unexpected ways.

As he remarkably candidly notes in his book "Weaving the Web" [7], Berners-Lee didn't invent hypertext, markup languages, or networking protocols, but he saw how HTTP, HTML, and URLs could create a self-reinforcing web that previous hypertext systems had failed to achieve. The web browsers didn't invent hypertext or networking either, but they made the connection between them obvious in retrospect. Git didn't create new concepts around file differencing, cryptographic hashing, or distributed systems, but made distributed version control finally practical. Photoshop didn't invent image processing algorithms or digital manipulation, but unified traditional darkroom techniques with mathematical transforms in an intuitive interface. Uber didn't invent transportation technology, but cleverly orchestrated GPS, mobile payments, and dispatch algorithms that had existed separately for years. Finally, Netflix didn't invent video streaming or recommendation systems – they combined content delivery networks with collaborative filtering at a moment when bandwidth made it viable.

What distinguishes many impactful software innovations, then, isn't pure novelty but rather a form of *combinational creativity* [9] – the ability to perceive new relationships between existing elements and to pursue combinations that may seem counterintuitive. Many innovators aren't necessarily the first to create the individual components, but they are the first to see the gestalt that emerges from their combination. This suggests that thinking of breakthrough innovations as pure inventions may actually mislead us about where a lot of transformational software comes from. Perhaps a highly valuable skill isn't creating something from nothing, but developing the pattern recognition to spot which existing pieces, when properly orchestrated, can produce emergent behaviors that transcend their individual capabilities. Much real innovation, thus, lies not in the novelty of the parts, but in the non-obvious wisdom of their assembly. In fact, this process of innovation through novel recombination is not specific to software – it appears to be a fundamental pattern in virtually all domains of human innovation, and it has been theorized and studied empirically in many contexts, including business, medicine, science, and technology [38, 66, 69, 79].

But while it's clear that innovation drives the software industry [27], and that great software engineers (among many other attributes) should be "*[able to] generate novel and innovative solutions based on the context and its limitations*" [49], it's still unclear how to develop one's ability to combine existing things in clever ways. Such combinational creativity "*typically requires a very rich*

store of knowledge, of many different kinds, and the ability to form links of many different types" [9]. How to develop these? Although empirical evidence is scarce [35, 57], one might theorize that mixing knowledge depth with knowledge breadth is the key [20, 78], since these characteristics make software professionals more successful in general [25, 72]. According to this thinking, deep engagement with the software development process, ideally through "deliberate practice" [5], should help build expertise, while broad exploration in many different contexts should help build vocabulary.

Our paper goes one step further in this line of thinking, revealing a stronger and more nuanced effect of breadth than one might expect. Across over 37,000 Python projects on GitHub, while we confirm an association between knowledge breadth and software innovation, we find that **exposure through minimal-effort interactions like starring repositories emerges as a stronger predictor of future innovation than high-effort engagements like writing code**. This counterintuitive result suggests that casual, passive observation of diverse projects – essentially *digital lurking* – may be more valuable for sparking creative recombination than direct, intensive engagement and active learning.

Much like the unexpected advantage of weak ties in job searches, which challenged conventional wisdom of social networks at the time [34], this "strength of weak ties on GitHub" represents a paradox worth exploring: Sometimes, developers' least-effort interactions with projects may contribute the most to their innovation capacity. Our research investigates this surprising mechanism, examining how the structural characteristics of developers' interaction networks correlate with the innovativeness of the software artifacts they subsequently create.

Below, we draw from established theory to formulate our hypotheses (§2); propose a novel software innovativeness measure, reconstruct strong- and weak-tie interaction networks (via commits, issues, and starring) for a project's core team, and estimate the amount and diversity of knowledge accessible to the core developers via these networks (§3); build regression models to test the association between network structural properties and software innovativeness (§4); and discuss the implications of our results (§5).

2 THEORY AND HYPOTHESES

In a nutshell, developers on the GitHub platform engage in a wide range of interactions, corresponding to ties of varying strength in a network sense, that create opportunities for knowledge acquisition via social learning. This knowledge can influence the innovativeness of the solutions they create. Before diving into the details of the mechanism, let us clarify some key constructs.

Innovation as Novel Recombination. As discussed above, we view software innovation as often emerging from *novel recombinations* of existing components, libraries, and patterns, following the long-standing view that "[business] innovation combines factors in a new way, or that it consists in carrying out new combinations" [66], which has fueled much research in the social sciences (e.g., [38, 69, 79]). As a result, the line between *novelty* and *innovation* becomes fuzzy, thus we will use the two terms interchangeably, in contrast to some prior work that distinguishes them [6, 18].

Creativity as Antecedent of Innovation. How to facilitate the emergence of innovation is unclear, although a common argument centers on enabling creativity as a precursor. *Creativity* and *innovation* are closely related but distinct constructs: creativity is the ability to generate novel ideas, while innovation transforms novel concepts into tangible outcomes. It is expected that a software team that is more creative has access to richer knowledge (e.g., knowledge about the problem and technical approaches to solving it), and is better able to combine pieces of that knowledge to create an innovative solution [26]. Of course, there are many facets of creativity, including as an individual personality trait [43]. In this paper we focus only on one. Specifically, we argue that creativity at the team level is a function of the knowledge networks of the team members and, while we cannot measure creativity directly, we can expect to see differences in the teams' creative outputs (i.e., more novel software), that are associated with differences in the structure of those knowledge networks.

Knowledge Acquisition via Social Learning. Historically, and especially with the advent of "social coding" platforms like GitHub, participation in open source has been rife with opportunities for social learning [19], i.e., the process by which individuals acquire new knowledge, behaviors, skills, or attitudes through observation, imitation, and interaction with others. Learning, including by observing what others are doing [86], has been [46, 47] and remains [33] among the most important motivations for people to contribute to open source. In addition, modern code hosting platforms offer many opportunities for users to interact (e.g., collaboration on a shared codebase, issue discussions, code review). Through social media-like functionality (e.g., "following" [8]) and many available signals that offer a high degree of transparency [21] (e.g., repository badges [76]), the platforms also facilitate users quietly observing and being influenced by others' behaviors.

These interactions with other individuals [81] and with the artifacts they create [15] serve as channels for information dissemination and knowledge exchange. Indeed, prior studies have documented many such social learning effects, including choosing an open-source license [60, 71], discovering and adopting emerging tools [48, 70], and learning new software design principles and programming skills by reviewing code authored by others [3, 58].

Therefore, we expect that an increase in the volume of interactions among developers should correspond to a higher amount of knowledge transfer, thus better preparing developers for future innovations. Thus, we hypothesize that:

H_1. *The more interactions developers have with other developers and projects, the more innovative their projects are.*

Still, if innovation requires an increased "vocabulary" of knowledge bits accessible for recombination, even high volumes of interaction might not be sufficient if the corresponding information is redundant. This can happen, e.g., when one works primarily within a narrow domain. Instead, we expect that accessing diverse knowledge acts as a catalyst for innovation, enabling individuals to integrate disparate concepts and to develop unconventional and novel products [67]. Outside of software engineering, Tortoriello et al. [75] observed a positive association between employees' access to diverse knowledge within the research departments of high-tech companies and their innovation levels. Similarly, Abdul Basit and

Medase [1] found that the integration of knowledge from both internal research teams and customers in the public sector enhances innovation. Thus, we hypothesize that:

$\mathbf{H}_2$. *The greater the informational diversity of developers' past interactions, the more innovative their projects are.*

The Strength of Weak Ties. In a social network, ties can have varying strength, reflective of real-world factors like the duration of shared interactions, emotional depth, level of intimacy, and amount of reciprocal exchanges [34]. In that sense, "weak" ties involve infrequent and less intimate interactions, while "strong" ties are characterized by higher frequency of interaction and intimacy.

In the late 1960s, sociologist Mark Granovetter uncovered a counterintuitive finding that would revolutionize our understanding of social networks: When searching for jobs, people relied more on casual acquaintances (weak ties) than close friends or family (strong ties). His groundbreaking theory on the "strength of weak ties" [34] revealed that these seemingly superficial connections often serve as critical bridges between different social circles, providing access to novel information unavailable within one's immediate network. This pattern arises because individuals within the same network neighborhood have numerous opportunities for interaction and often share many mutual ties. This shared social context leads to increased similarity in behaviors, interests, and other characteristics. Consequently, the information disseminated via strong ties risks becoming less novel and less valuable (the discussion of echo chambers on social media [45] is a prime example of this degradation of information quality). In contrast, information acquired via weak ties, or cross-group connections, tends to originate more from individuals with diverse backgrounds and knowledge bases, making it more likely to be novel and of greater value [34]. This mechanism has been extensively validated in social contexts, including generation of innovative ideas [13], the diffusion of news feed content [4], and job seeking in the digital age [61].

Similarly, we expect that the software-related knowledge transferred through weak ties is more valuable for the creation of innovative software projects. Thus, we hypothesize that:

$\mathbf{H}_3$. *The more the informational diversity of developers' past interactions is due to weak ties, the more innovative their projects are.*

3 METHODS

Next we give a high-level overview of our approach, before diving into operationalization details.

3.1 Overview / Intuition

Network Construction. We theorized above that (1) the interactions developers have with each other and with each other's artifacts, i.e., the ties they form, create opportunities for knowledge transfer and facilitate social learning; and (2) "strong" and "weak" ties may play different roles in this knowledge diffusion process. To operationalize these concepts, we first construct three interaction networks where every node is a project (i.e., GitHub repository), and the links between two nodes encode different interactions the developers of one project had with the other project.

There are many ways in which two open-source developers can interact, both in and outside the GitHub platform. Clearly, it's not possible to capture all interactions at scale, as projects may use, e.g., a diversity of communication channels, including private ones. Instead, we consider three representative examples of interactions that (a) vary substantially in effort, thus can be expected to reflect varying levels of knowledge flow from the target project to the author of the action (i.e., to encode person–to–project ties of varying strength, which we later project to construct our project–to–project networks), (b) are core features of the GitHub platform, thus are commonly used: making *commits* to a codebase (direct pushes or merged pull requests), posting or participating in *issue discussions*, and *starring* repositories.

Among the three, commits typically require the most effort and starring the least. This ranking should correspond to the depth of the knowledge that may transfer as a result of each action. Intuitively, while commits vary in size and content [2], an "average" commit should require a significant understanding of the project's technical details [81], indicating a path for considerable knowledge flow between the project and the commit author. Issue threads primarily discuss feature suggestions, bug reports, and user support [53]. Thus, while engaging in issue discussions can reflect a deep understanding of the project, on average we expect that it requires (and reflects) a less deep understanding of the project compared to making changes to its codebase. Finally, starring a project is often done as a sign of appreciation, a bookmarking attempt, or an intent for later use [10]. Starring a repository indicates at least some awareness of the project, but on average probably much more basic understanding than the other two.

Note that we consider only the actions of core developers in a focal project, identified heuristically as those contributors who authored at least five percent of all commits, with a minimum of 10 commits in total. Our operationzalization is validated with alternative threshold of core developer identification in the appendix. Conceptually, in open source, core developers wield the most influence over a project's technical decisions and development trajectory [63]. Empirically, in our sample core developers also authored the vast majority (close to 90 percent) of commits importing new packages into projects (which we use to compute innovativeness, as described below). Thus, we expect that the knowledge they had access to prior to working on a focal project (as opposed to peripheral contributors) has the most influence on the design and innovativeness of that project.

As possible alternatives [62], we considered reflecting the past interactions of only the founder of a project (which would have missed influential people joining a project later), as well as network centrality-based operationalizations of "core" versus "periphery" status (which are more precise but computationally more complex and, on average, correlate highly with count-based measures like ours [41]). Finally, we considered capturing the past interactions of all project contributors. However, at the scale of our study (over 37,000 projects) this was infeasible considering that popular projects may have tens of thousands of peripheral contributors.

Network Measures. Using our network data, we compute variables related to the network position of each project to represent the *amount* and *diversity of knowledge* that may be accessible to its

core developers through each network. For example, consider the commit-based project–to–project network. A node (project) in this network may have many outgoing connections, indicating that its core developers have collectively committed to many other projects prior to joining the focal project. That is, they have first-hand experience with the technologies used in those other projects, and they can probably draw from that experience (knowledge) now to influence the development of the focal project. Similarly, we can reason about the other two networks constructed from issue discussions and starring repositories.

How to measure the diversity of knowledge is less obvious. Conceptually, the amount and diversity of knowledge are related yet distinct. For example, a project whose core developers are connected to a large number of other projects within the network *may* have access to diverse knowledge. However, the actual diversity of this knowledge remains contingent on the nature of these connections – if all linked projects contain similar knowledge, diversity may be limited; e.g., if they're all related to visualization, that doesn't say much about experience with TENSORFLOW.

To capture this diversity, we adopt an approach analogous to word embeddings, wherein semantic similarity between words is inferred from their vector representations [55]. The key ideas is that we compute project graph embeddings and use embedding distance to quantify the similarity of gainable knowledge among projects. The intuition is that nodes that frequently appear together in the same random walk while learning the embeddings represent projects that are often contributed to or interacted with by the same developers, i.e., are related in some informational sense. This could be because they share similar functionality, are used in similar contexts, or are part of the same development ecosystem.

Dimensionality Reduction. Thus far we have been treating our three networks (commits, issues, and stars) as separate. However, one can expect that the variables we compute based on these networks are correlated to some extent. Moreover, one can imagine considering other types of interactions (i.e., networks) besides our three, which would result in yet more variables. To prevent issues of multicollinearity in our subsequent regressions, which can both complicate the interpretation of the estimated coefficients and reduce statistical power [22], we use Principal Component Analysis (PCA) [84] to decompose the variables representing the *amount* and *diversity of knowledge* into orthogonal components. As we show below, this produces a clear decomposition of our original variables into two components capturing strong and weak ties, which we use in our models instead of the original variables.

Project Innovativeness Measure. Fang et al. [31] introduced a measure of a software project's novelty, or innovativeness, as a function of the combinations of packages the project imports. For example, while certain libraries, such as NUMPY and TENSORFLOW, are frequently used together, others, like NUMPY and REQUESTS, are less commonly combined. The measure estimates how atypical combining two packages is, and by extension how atypical is the overall set of packages being used as dependencies in a project, by comparison to what could be expected by random chance. Projects that import more atypically paired packages are deemed more innovative as they reuse packages in nontraditional ways.

We maintain this framing of **innovation through novel recombination**, and similarly consider **packages as the unit of recombination**. Packages are designed to be reused; they're typically substantial enough to provide real functionality while remaining composable (individual functions are too small – they lack sufficient context and capability to be meaningfully recombined; entire applications are too large); and they're usually semantically coherent, i.e., they tend to be organized around conceptual domains (e.g., REQUESTS encapsulates the entire complexity of HTTP communication in a reusable form, while PANDAS bundles decades of data manipulation knowledge into a coherent interface). Alternatively, one could consider something like "concepts" as the unit of recombination, as in some prior sience-of-science work [38]. However, it wasn't clear how to automatically extract concepts from software, so we chose packages as the unit instead.

Our measure of project innovativeness, while similar in spirit to Fang et al's [31], is based on learning embeddings of packages and addresses two important limitations of the original measure. First, the original measure only captures pairwise combinations between packages (and aggregates up to the project level from that), neglecting the possible interactions among related packages. However, packages often form "stacks" comprised of more than two libraries that complement each other and are often reused together. Our embeddings-based approach considers more context and is expected to better capture a package's technical use, function, and "stack." Second, computing the original measure requires random reshuffling of the global dependency network, which is computationally prohibitive at the scale of our study. In contrast, we compute the cosine similarity of embeddings, which is faster.

Regression Analysis. Putting everything we discussed so far together, our analysis involves modeling the variation in project innovativeness across our sample as a function of the variables we discussed above, controlling for known covariates. While not causal,[1] this analysis will still allow us to test the extent to which the observational data is consistent with the causal paths theorized above (about determinants of combinatorial creativity).

In the remainder of this section we give lower-level operationalization details for the steps above, including validation checks.

3.2 Sample Selection

We combine data from World of Code (WoC) [51] and the GitHub API. WoC is arguably the most comprehensive record of open-source projects and their commit history, and it also contains historical import-based dependency information for projects in the most popular programming languages, which we use when computing project innovativeness. To keep our analysis tractable, we focus only on Python projects, i.e., the project contains more than 10 Python source code scripts. We chose Python due to its immense popularity [74] and its widespread use across various domains by developers from diverse backgrounds [64]. While not allowing us to directly generalize our findings, the considerable variability in the Python ecosystem across many characteristics should ensure points of overlap with many other ecosystems.

[1] Testing our hypotheses experimentally is infeasible, and identification is otherwise unclear from observational data like ours, since there is no clear "intervention."

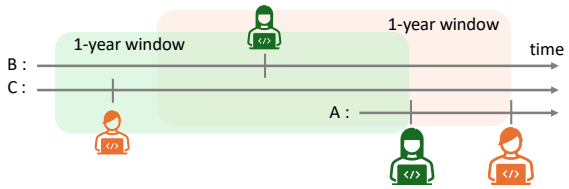


Figure 2: Two core developers Green and Orange started contributing to a focal project A on different dates. We record an edge from A to B (i.e., B could be a source of knowledge for A), because Green interacted with project B in the previous year. We don't record an edge from A to C, because Orange interacted with C too far into the past.

Because many public repositories are intended as code dumps rather than active development projects, and to account for more of the intricacies of GitHub data [42], we excluded forks and repositories with fewer than ten commits in total. In addition, we excluded repositories created before 2008 (pre GitHub) and after 2022 (which allows for at least one year of history in WoC, which we need to make inferences about project innovativeness).

For this remaining set of projects we queried the GitHub API for historical information on all their commits, issues, and stars (the latter two are not available in WoC); for each such event we recorded its author and timestamp. We used this data for our network construction, detailed in Section 3.3 below.

As a final preprocessing step, we heuristically identified and filtered out bot accounts [24, 83], which would distort our networks and subsequent measures, as they tend to have a lot of activity. To this end, we searched for keywords in user logins, such as *-bot* or *-robot*, and labeled those users as bots after manual review. Additionally, we manually inspected the profile descriptions of the top 100 most active user accounts in our sample, ordered by the number of commits, to ensure the removal of highly active bot accounts that may have escaped the naming convention heuristics.

3.3 Network Construction

Each of the three actions we consider (making commits, participating in issue discussions, and starring) can be thought of as a person–to–project tie. Aggregating across all developers and projects in our sample leads to three person–to–project bipartite networks, one for each action type. However, since our analysis is at the project level, we construct three directed project–to–project networks by way of projection, to represent possible flow of knowledge across projects. As discussed previously, we consider only actions initiated by core developers (e.g., which other repositories they committed to or starred in the past). In addition, we restrict the scope to actions in the recent past (e.g., within the last year, we validate with alternative period length in the appendix), as we expect older actions to be less relevant. This time window is developer-dependent. That is, we record an edge $A \rightarrow B$ if and only if a core developer of project A interacted with project B within one year prior to their initial commit to project A , as illustrated in Figure 2. The weight of each edge represents the number of core developers from project A who engaged with project B via a specific type of interaction.

As expected, the structural properties of the three networks support our reasoning for why commits, issues, and stars, respectively, capture ties of decreasing strength. Comparing the transitivity[2] values of our three networks, we observe approximately an order of magnitude (10×) difference between each pair of networks. Specifically, the commit network displays the highest levels of transitivity, followed by the issue network, while the star network exhibits the lowest level; see Supplementary Materials for details.

3.4 Network Measures

Next, we compute variables related to the network position of each project to represent the *amount* and *diversity of knowledge* that may be accessible to its core developers through each network.

First, we use the number of connected projects as a proxy for the **amount of knowledge accessible** to a given project. More formally, for every project, we compute its out-degree centrality within each of the three networks, defined as the total sum of the weights of all its out-edges. This step resulted in computing the three variables listed in Table 2 under ***Degree Variables (original)***.

Second, we use the Node2Vec graph embedding algorithm [36] to generate vector representations (embeddings) for each node in the network based on their topological position. The process begins by generating sequences of random nodes from the graph using random walks. Starting from a random node in the network, we sample the next node by randomly selecting a direct neighbor. The likelihood of selecting each node is proportional to the edge weight between the candidate next node and the current node. This process continues until we have selected enough nodes to complete the walk (in our study, each walk comprises 20 nodes). We repeat this process to generate multiple walks (or sequences of nodes). Subsequently, we use a skip-gram model [54], commonly used to generate embeddings for words in natural language, on the generated walks to learn embeddings for each node.

Next, for each project in the network, we examine all other projects that receive a directed edge from the focal project (denoted as set P). Our **knowledge diversity index**[3] is the average pairwise distance between any two projects in P: $D = \frac{\sum_{i,j \in P, i \neq j} -sim(v_i, v_j)}{|P| * (|P|-1)}$, where i and j are projects in P, and v_i and v_j are their vector representations from the Node2Vec model. The distance between v_i and v_j is the negation of the cosine similarity of the two vectors. This step resulted in computing the three variables listed in Table 2 under ***Diversity Variables (original)***.

3.5 Dimensionality Reduction

Since they are designed to capture different concepts, we run PCA separately for the three degree variables and the three diversity variables. As standard, we first log-transformed all input variables and scaled them to a mean of zero and a standard deviation of one.

The PCA resulted in three components for the degree variables and another three for the diversity variables, with the proportion of variance explained by each component listed in Table 1 (top). Inspecting the table, we observe that the first two PCs cumulatively explain over 80% of the variance in both groups, thus we decided

[2] In network science, *transitivity* is a measure of the ratio of triangles to triads.
[3] The index is undefined for projects with out-degree centrality below two. We exclude these from our regression when considering diversity metrics as independent variables.

Table 1: *Top*: Proportion of variance explained by each principal component. *Bottom*: Loadings of the principal components onto the original degree and diversity variables.

	Out-deg. centrality			Diversity index		
	PC1	**PC2**	**PC3**	**PC1**	**PC2**	**PC3**
Variance Explained	0.64	0.22	0.14	0.51	0.29	0.19
Cumulative Variance	0.64	0.86	1.00	0.51	0.81	1.00
D_{commit}	0.60	−0.45	0.67	0.63	−0.36	0.69
D_{issue}	0.61	−0.28	−0.74	0.65	−0.24	−0.72
D_{star}	0.52	0.85	0.11	0.43	0.90	0.08

to retain only two components each for the degree and diversity variables. That is, we will use these PCs (listed under ***Degree Variables (post-PCA)*** and ***Diversity Variables (post-PCA)*** in Table 2) instead of the original variables in our regression models below.

To interpret the two PCs we turn to Table 1 (bottom), which lists the loadings of the principal components onto the original variables.[4] Inspecting the table we make the following observations. First, the loadings onto the first principal component (PC1) are relatively consistent across networks for both the degree and the diversity variables. Thus, we interpret PC1 to represent the average degree (or diversity of ties) of the project across the three networks. For example, drawing on our theoretical framework (Section 2), a project for which $PC1_{\text{degree}}$ is high is expected to have many ties (on average, across the three networks), i.e., its core developers should have more sources from which to draw inspiration. Similarly, the ties of a project for which $PC1_{\text{diversity}}$ is high are expected to span more of the knowledge embedding space we used to estimate tie diversity, i.e., its core developers could have access to more varied sources to draw inspiration from.

Second, the loadings onto the second component (PC2) show a descending trend across networks, again similarly for both sets of variables. Notably, the highest loading on the degree (or diversity) comes from the star network, followed by the issue network, while the lowest comes from the commit network. Thus, we interpret PC2 to represent the strength of network ties in the degree (or diversity of ties) metric, i.e., the **strength of weak ties**. A project for which $PC2_{\text{degree}}$ is high is expected to get more of its connectivity through the star network. Analogously, relatively more of the diversity of knowledge accessible to core developers in a project for which $PC2_{\text{diversity}}$ is high can be attributed to the star network (weak ties) compared to the commit or issue networks (stronger ties).

It may seem counterintuitive to reason about PC1 and PC2 jointly, especially in a regression modeling framework. What does it mean to vary PC2 from low to high, for instance, while holding PC1 fixed? Can PC1 and PC2 even vary independently? They can in theory, because they are orthogonal by construction, but are the combinations of low-high values of PC1 and PC2 observable in the real-world data? After all, regression is an interpolation mechanism. We illustrate this space of interpretations of PC1 and PC2 for the diversity variables with the artificial examples in Figure 3

[4]Principal components are linear combinations of the original variables, with the loadings indicating the contributing coefficients. For example, the first PC for the degree variables, $PC1_{\text{degree}}$, uses the coefficients in the first column in Table 1 (bottom): $PC1_{\text{degree}} = 0.60 * Deg_{\text{commit}} + 0.61 * Deg_{\text{issue}} + 0.52 * Deg_{\text{star}}$.

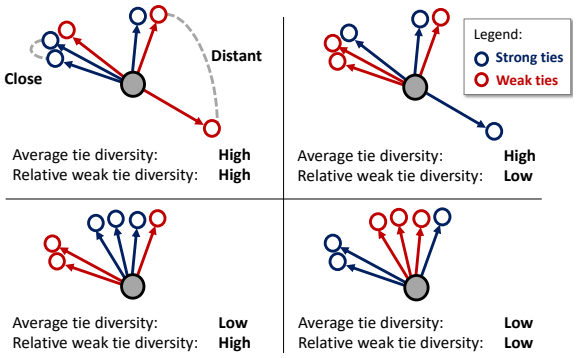


Figure 3: Illustration of the four possible combinations of low-high values for $PC1_{\text{diversity}}$ (corresponding to average tie diversity) and $PC2_{\text{diversity}}$ (corresponding to the relative diversity attributable to weak ties).

(see Supplementary for real-world examples in each quadrant). In the top left quadrant (PC1 and PC2 both high), the project core developers can tap into a diverse knowledge space via either strong or weak ties. In the top right (PC1 high and PC2 low), the overall diversity of information accessible is just as high, but it's the strong ties that are the source of it. In the bottom left (PC1 low and PC2 high), the overall diversity of information available is lower, but whatever diversity there is, it can be attributed mostly to the weak ties. Finally, in the bottom right (PC1 and PC2 both low), weak ties contribute little to an otherwise low diversity of information.

Going back to our hypotheses (Section 2), we expect the left quadrants, where weak ties are strong, to be most associated with innovativeness. As an anecdote, consider the example in Figure 1 on the first page. The GeoNotebook project is a Jupyter notebook-based environment for interactive visualization and analysis of geographic data. Interestingly, GeoNotebook's core developers have previously starred (without otherwise contributing to) many GitHub projects spanning a variety of seemingly related topics, including a collection of free APIs, example data science notebooks, reproducible workflows for Jupyter notebooks, visualization tools, time series analysis tools, geographic data analysis tools, and infrastructure tools for interactive packet manipulation. One can expect that this diverse space of ideas provided some inspiration, knowingly or unknowingly, for the design and implementation of GeoNotebook. Our analysis below tests to what extent there is evidence supporting this mechanism at scale, beyond this anecdote.

3.6 Project Innovativeness Measure

For each project, we extract the sequence of packages imported[5] (precomputed in World of Code) and use a skip-gram model to generate positional embeddings for each package based on these dependency relationships. These embeddings place the package in an optimal location relative to all other packages imported in the same project, based on their co-import relationships. Unlike the random walks used in the Node2Vec model to generate embeddings for the three interaction networks, there is no inherent order in the

[5]We ignore built-in Python libraries like Fang et al. [31] did, and consider only packages published in the PyPI registry, using the explicit links back to GitHub recorded there.

Table 2: The definitions of the variables in our models.

Outcome Variables	
Innovativeness	The extent to which a project imports packages that were typically not combined in the past, computed based on all the packages imported until the end of 2021.
Degree Variables (original)	
Deg_{Commit}	The sum of edge weights from the focal project to the others in the commit network.
Deg_{Issue}	The sum of edge weights from the focal project to the others in the issue network.
Deg_{Star}	The sum of edge weights from the focal project to the others in the star network.
Degree Variables (post-PCA)	
Deg_{Ave} ($\mathbf{H}_1$)	The first principal component resulted from the PCA analysis over three original degree variables, representing average network degree across the three networks.
Deg_{Weakness}	The second principal component resulted from the PCA analysis over three original degree variables, representing the degree from weak ties.
Diversity Variables (original)	
Div_{Commit}	The diversity of projects to which the focal project has a directed edge in the commit network.
Div_{Issue}	The diversity of projects to which the focal project has a directed edge in the issue network.
Div_{Star}	The diversity of projects to which the focal project has a directed edge in the star network.
Diversity Variables (post-PCA)	
Div_{Ave} ($\mathbf{H}_2$)	The first principal component resulted from the PCA analysis over three original diversity variables, representing average diversity of connections across the three networks.
Div_{Weakness} ($\mathbf{H}_3$)	The second principal component resulted from the PCA analysis over three original diversity variables, representing the diversity from weak ties.
Control Variables	
$Year_{\text{creation}}$	A fixed effect variable indicating the year in which the project received its first commit.
Org_owned	A binary variable indicating whether the project was owned by an organizational account or not.
$N_{\text{Owner\_Stars}}$	The total number of stars that the project owner received on all other repositories they owned before the creation of the focal project.
$N_{\text{Core\_Devs}}$	The total number of core developers in the project, as computed before the end of 2021.
N_{Packages}	The number of software packages that the project imported before the end of 2021.

sequence of packages imported within the same project. Therefore, we set the window size in the skip-gram model to be sufficiently large, enabling it to consider all packages in the same sequence as its context when generating embeddings. Similar approaches to compute embeddings of software packages have been used before in software engineering research [23, 29].

To estimate the atypicality of a combination of two packages, we compute the negation of their embeddings' cosine similarity. Then, our **measure of project innovativeness** is the average pairwise atypicality of all pairs of packages imported by a project.

Validation. Although measuring project novelty in terms of the atypicality of package combinations therein has solid theoretical foundation (Section 2) in addition to application in past empirical studies of open-source software [31], this likely remains the most controversial part of our methodology. Therefore, we perform two validation checks, comparing to an existing dataset of "awesome" projects and to the previous measure by Fang et al. [31].

"Awesome" lists are community-curated lists of software. Many exist for different programming languages, platforms, application domains, etc. One can propose new entries via a pull request (PR), and typically some minimum amount of support (+1s) is required for the PR to be merged. Being innovative isn't formally a requirement for being "awesome," although as part of the same PRs submitters usually argue that the proposed entries are novel (e.g., PRs to the Python list contain a section titled "What's the difference between this Python project and similar ones?")

We scraped software mentions from a popular aggregator of "awesome" lists [73] (350k+ GitHub stars at the time of writing), and identified 760 Python projects matching our sampling criteria (§3.2), and for which we could compute atypicality scores (e.g., having at least two import-based dependencies we could identify), corresponding to about 2% of our sample. Comparing the atypicality scores between the "awesome" and remaining projects in our sample (Figure 4), we observe a statistically significant difference in means (two-sample independent t-test $p < 8.8e^{-13}$), with a non-trival effect size: a project with a median atypicality score in the non-selected group will only be among the bottom 35% ordered by atypicality score among the "awesome" projects. Further validation suggests this effect is not confounded by variables such as project size (Supplementary). Therefore, we conclude that **our innovativeness measure reflects to some extent the developers' perception of "awesome" projects, which usually includes a dimension of novelty.**

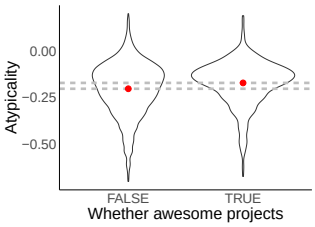


Figure 4: "Awesome" projects have statistically significantly higher atypicality scores than the rest of our sample. The red dots represent the distribution means.

In addition, we compute the correlation between the atypicality scores obtained using our skip-gram model and those generated using the original measure by Fang et al. [31]. Results show a moderate-to-strong [65] linear (Pearson) correlation of 0.65. Therefore, we conclude that **our innovativeness measure correlates to that of prior work to a large degree.** Furthermore, we test the robustness of our conclusions to variations in the innovativeness measure, and find that the estimated regression coefficients for the *degree* and *diversity* variables behave consistently with both measures. Thus, we only present the results obtained with our proposed measure below, but include the corresponding regression results with the original measure by Fang et al. [31] in Supplementary.

3.7 Regression Modeling Considerations

To test our three hypotheses, we use fixed-effects linear regression (one data point per project), where the response variable is our measure of *innovativeness*, and our main explanatory variables are the degree and diversity principal components discussed above: Deg_{Ave} captures $\mathbf{H}_1$ (volume of interactions the core developers had with other projects in the previous year), Div_{Ave} captures $\mathbf{H}_2$ (diversity

Table 3: Summary of regression analysis of factors associated with differences in project innovativeness.

	Model I	Model II	Model III	Model IV
Variables of interest				
Deg_{ave} ($\mathbf{H}_1$)	0.001^{***}	-0.001		-0.002^{**}
	(0.0002)	(0.0006)		(0.0006)
$Deg_{weakness}$	-0.0003	-0.001		-0.005^{***}
	(0.0002)	(0.0007)		(0.0008)
Div_{ave} ($\mathbf{H}_2$)			0.006^{***}	0.007^{***}
			(0.0006)	(0.0006)
$Div_{weakness}$ ($\mathbf{H}_3$)			0.005^{***}	0.007^{***}
			(0.0008)	(0.0008)
Controls				
Org_owned	0.027^{***}	0.017^{***}	0.018^{***}	0.018^{***}
	(0.0006)	(0.0016)	(0.0016)	(0.0016)
$N_{\text{Owner\_Star}}$ (log)	0.002^{***}	0.001^{*}	0.001	0.001^{*}
	(0.0002)	(0.0004)	(0.0004)	(0.0004)
$N_{\text{Core\_Dev}}$ (log)	0.014^{***}	0.018^{***}	0.017^{***}	0.018^{***}
	(0.0006)	(0.0016)	(0.0015)	(0.0016)
N_{Packages} (log)	0.042^{***}	0.023^{***}	0.023^{***}	0.024^{***}
	(0.0003)	(0.0011)	(0.0011)	(0.0011)
Fixed effect				
$\text{Year}_{\text{creation}}$	✓	✓	✓	✓
Observations	589,999	37,451	37,451	37,451
Adjusted R^2	0.054	0.059	0.063	0.064

*p<0.05; **p<0.01; ***p<0.001

of information accessible), and Div_{Weakness} captures $\mathbf{H}_3$ (strength of weak ties). The remaining principal component, Deg_{Weakness}, does not directly map to one of our hypotheses, but can help distinguish whether it's the volume of weak ties, or their strength, that explains more variance in our outcome measure.

We also control for several important confounding variables (Table 2): the repository owner's social standing ($N_{\text{Owner\_Stars}}$), the project team size ($N_{\text{Core\_Devs}}$), the complexity of the codebase in terms of number of packages imported (N_{Packages}), whether the project is owned by an organizational account (*Org_owned*), and the project age ($Year_{\text{creation}}$, modeled as a categorical variable). For model fit and diagnostics we followed standard practice, e.g., we first log-transformed the variables with skewed distributions to reduce heteroskedasticity [44]; see replication package.

4 RESULTS

Next we present the main results from the regression analysis formally testing our hypotheses, summarized in Table 3, as well as a series of robustness checks.

Main Regression Results. Model I (all projects) and Model II (projects with a minimum out-degree centrality of two across all three networks) shed light on the impact of degree variables. Model I reveals a significant positive effect from average degree centrality, whereas Model II shows a negative effect. Nevertheless, effect sizes for both models are relatively small. For instance, in Model I, transitioning a project's Deg_{ave} from the 25th percentile (-1.09) to the 75th (0.53) results in a mere 0.002 increase in project innovativeness. Considering that the 25th percentile of a project's innovativeness score is -0.33, and the 75th is -0.12 in our sample, the change in project innovativeness due to variations in average degree centrality is practically negligible. Therefore, $\mathbf{H}_1$ is not supported.

> We do not find supporting evidence that the more interactions developers have with other developers and projects, the more innovative their projects are ($\mathbf{H}_1$).

In Model III, we observe a positive association between the average diversity (Div_{ave}) of ties originating from the focal project and project innovativeness. Moreover, this effect is notably stronger – nearly seven times greater – compared to that of average degree in Models I and II. Transitioning a project from the 25th percentile (-0.89) to the 75th (0.86) in terms of Div_{ave} corresponds to an increase of approximately 0.012 in project innovativeness, representing roughly a 4% change in the distribution (e.g., a shift from the median project in terms of novelty to the 54th percentile). We conclude that $\mathbf{H}_2$ is supported.

> On average, the greater the informational diversity of developers' past interactions, the more innovative their projects are ($\mathbf{H}_2$).

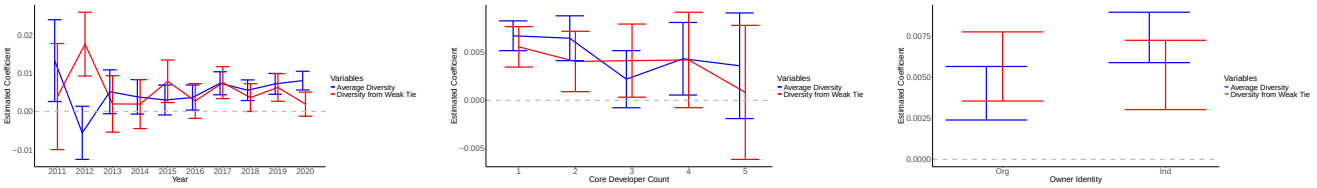
Additionally, we observe a significant effect for the $Div_{weakness}$ variable. This suggests that, when controlling for the average diversity across the three networks, higher diversity in the "weak ties network" (i.e., star network), or stronger weak ties, corresponds to increased project innovativeness. In practical terms, the effect size for the $Div_{weakness}$ variable is comparable, albeit somewhat smaller, to that of the Div_{ave} variable above. We interpret this as supporting evidence for $\mathbf{H}_3$.

> On average, the more the informational diversity of developers' past interactions is due to weak ties, the more innovative their projects are ($\mathbf{H}_3$).

When incorporating both degree and diversity variables in the same regression model, as in Model IV, the positive effects of Div_{ave} and $Div_{weakness}$ remain significant. This suggests a consistent effect stemming from tie diversity and strength of weak ties. Nevertheless, we also note a significant negative effect for Deg_{ave} and $Deg_{weakness}$ after adjusting for project diversity. These negative effects suggest that, while holding tie diversity constant, higher out-degree centrality and more ties in the "weak network" are associated with decreased project innovativeness. This observation could be attributed to the increased constraints within a local community associated with higher out-degree, as discussed by Burt [12]. Such constraints can lead to a reduced inclination for combining packages of diverse functions within the project, consequently contributing to the reduced novelty of the project. More research is needed to further test these effects.

Robustness Checks. In addition to the regressions above, we test how robust our conclusions about the effects of Div_{ave} and $Div_{weakness}$ are to differences in operational decisions.

Does Time of Measurement Matter? "No." We segmented our dataset by project inception year and conducted regression analyses with the specification of Model III in Table 3 on projects initiated in each respective year. Figure 5a shows the resulting estimated coefficients for the two diversity variables. As illustrated in the graph, the positive relationship between the average diversity of knowledge access across the three networks (represented by the blue line) remains consistent over the years, with the average estimated coefficient



(a) Effects of knowledge diversity across years.

(b) Effects of knowledge diversity on different team-size projects.

(c) Effects of knowledge diversity on individual vs org projects.

Figure 5: The effects of the two diversity of knowledge variables (Div_{ave} testing H_2 and $Div_{weakness}$ testing H_3) are generally robust. Error bars denote 95% confidence intervals.

being statistically significantly above zero in most years. The estimated coefficient for Div_{ave} appears to also increase in recent years, though this difference is not statistically significant.

The positive relationship between diversity from weak ties and project innovativeness is generally consistent (the red line in Figure 5a). The estimated coefficient is above zero in all ten years observed, and the statistically insignificant coefficient (i.e., the lower bound of the 95% confidence interval is below zero) is likely attributable to the limited number of projects in each annual cohort (i.e., fewer than 4,000 projects in each year before 2016).

Does Project Size Matter? "No." Similarly, we split projects into subsets based on the number of core developers and assessed the impact of the diversity variables in each cohort. As shown in Figure 5b, the effect of Div_{ave} remains consistent across projects of varying team sizes, confirming the robustness of our findings. It is noteworthy that the effect of $Div_{weakness}$ becomes insignificant for projects with four or more core developers. This may be attributed to the limited number of project samples within these categories.

Does Project Ownership Matter? "No." Lastly, we re-estimated the effects of the diversity variables separately for projects owned by organizational and individual accounts. Our analysis (Figure 5c) shows that both *average diversity across networks* and *diversity from weak ties* have significantly positive effects for both categories. Furthermore, the difference in effect size between organization-owned and individual-owned projects is not statistically significant.

In conclusion, the positive associations between average knowledge diversity and project innovativeness, as well as between diversity from weak ties and innovativeness, are robust.

5 DISCUSSION

Next we discuss our results in the broader context of the literature and highlight budding directions for future research and practice.

Summary of Main Results. Established social science theory describes the mechanisms through which social network weak ties can be particularly "strong" – they can bridge disjoint parts of the network, providing access to diverse information, which in turn can be used towards better outcomes. Our study provides the first empirical link (that we know of) between this theoretical mechanism and an important software engineering outcome – the emergence of innovation in a software project. We hypothesized that the many user-to-user and user-to-artifact interactions possible on the GitHub platform, ranging from high-effort ones like making changes to a codebase to relatively trivial ones like starring a repository, result in network ties of varying strength, along which knowledge can flow. And, in particular, that weak ties are instrumental for this knowledge flow, as the GeoNotebook anecdote in the Introduction (Figure 1) would suggest.

Using a sophisticated methodology to reconstruct interaction networks, estimate tie strength, compute network informational diversity, and estimate project innovativeness, coupled with a robust statistical analysis, we found clear evidence (albeit only correlational) supporting two of our three hypotheses: on average, it's not the amount ($\mathbf{H}_1$), but rather both the diversity of information available to a focal project's core developers ($\mathbf{H}_2$), as well as the extent to which this information diversity comes from weak ties ($\mathbf{H}_3$), that explain some of the variance in how novel (atypical) the combination of software packages imported in a focal project is.

Effect Sizes. Some readers may question whether our main variables explain *enough* variance, but small effect sizes are exactly what we should expect when studying innovation in complex socio-technical systems. Software development operates in an extraordinarily high-dimensional space where countless factors influence outcomes, making any single mechanism necessarily modest in isolation. The theoretical mechanism of weak ties operates through subtle informational advantages that accumulate over time rather than dramatic singular influences, and these effects must compete against an already information-rich environment of documentation, tutorials, and online resources. Moreover, as discussed in the beginning of our paper, the weak ties theory has decades of robust empirical support across diverse domains, consistently showing modest but meaningful effects that compound over time. We're far from the first to discover this effect – at best, we're the first to offer some supporting evidence from the open-source software development context. In addition, most software projects engage in predictable recombination of existing packages, with genuinely novel combinations occurring primarily in the tail of the distribution where even small probability shifts can have outsized impacts. The consistency of effects across multiple strata, coupled with statistical significance despite the noisy environment, actually strengthens confidence in the underlying mechanism – large effect sizes in network research are often suspicious because they suggest implausibly crude influences that overwhelm all other factors, which rarely occurs in real complex systems. Next, we discuss the implications of our results.

Platform Design. One way to interpret the strength-of-weak-ties theory in our context is that lurking on the GitHub platform (i.e., we see starring repositories as lurking rather than actively contributing) has quantifiable benefits. If indeed people draw inspiration from things they've starred and apply those ideas to something they will build next, as the theory would suggest and our quantitative results support, platform designers should consider facilitating this process more. For example, GitHub has a way of highlighting "what the community is most excited about today" on its Trending page. It could be beneficial if this algorithm that determines what is trending (allegedly based on the star growth rate) also took into account and increased the overall informational diversity of the highlighted repositories (as computed, e.g., using an embedding-based approach like ours). Better still, the recommendations could be personalized, i.e., the algorithm could highlight repositories that enhance information diversity *for a given user*.

Our results also raise questions about the GitHub culture of (implicitly) rewarding code contributions the most, since one's commit history tends to be highly visible in the platform. Much has been written about how other developers [52] and even recruiters [14] use such signals. In contrast, we are finding evidence that well-informed but not necessarily highly active developers may also be experts at their craft, at least insofar as the novelty of their output. It's worth thinking about how to feature (and reward) such lurking more prominently in the platform design.

It's also worth thinking about what tracking and giving credit to ideas, not just code artifacts, might look like in this domain. In scientific research, authors are expected to cite prior work that influenced their thinking. In turn, these citations form networks encoding invaluable information about scientific progress and scientific collaboration [32]. What might such a system look like in software development? Anecdotally, it seems like developers already use many ad-hoc ways to credit ideas, e.g., including links in source code comments [37] and even referring to research papers in repository README files [82] or source code [39]. But we still don't understand how common the practice is, how complete the mentions are, and what the supply chains of ideas look like. Nor do we have ways of systematically tracking ideas as part of the platform design, which is worth exploring.

Theory. More broadly, there are many opportunities for developers for relatively low-effort interaction with each other and with new technology besides the one we measured here (starring). These include using social media to stay current [30, 70, 85], listening to podcasts [28], creating or consuming video content online [16], and participating in developer conferences [77] or trainings. Since our study is not exploratory (bottom-up), but rather tests hypotheses drawn from a much more general, widely-validated theory (top-down), it's reasonable to expect, given the theory, that the same "strong weak ties" mechanism could also apply in these scenarios. The famous "water-cooler effect," that advocates of in-person work environments often quote, is another example of the same mechanism – casual encounters, typically with weak ties (work acquaintances rather than close colleagues), can help exchange new ideas that spark creativity [11]. We need more research to empirically investigate these and other types of weak ties and their innovation-enabling potential, as it would be important not only to collect more evidence in support of the mechanism, but also to rank the different interactions in terms of their effectiveness. For example, many technology companies support their engineers regularly attending conferences and these experiences are likely beneficial. Could we quantify these benefits both in general, and relative to, say, consuming technical social media content?

Separately, our work aligns well with Baltes and Diehl's [5] software development expertise theory. For example, our lurking can be considered as an example of *behavior* that contributes to a developer's knowledge base, or as an example of *continuous learning*; similarly, the GitHub platform is the *work context* that influences expertise development, by facilitating exposure to diverse projects. But perhaps more interestingly, while the theory emphasizes *deliberate practice* for expertise development, our work offers a complementary perspective and an opportunity for theory refinement: casual observation of diverse projects may contribute to innovative thinking in ways that focused practice alone might not.

Exploration vs Exploitation. Research has shown that developers are more likely to join projects that are technically more familiar to them [23, 29]. Various automated project recommendation tools have also been proposed, that match projects to developers with the closest technical background [87]. Such matching is likely useful, as open-source projects depend heavily on contributions from volunteers, and contributors can be in short supply [17]. At the same time, our results suggest that maximizing technical similarity between a project and the backgrounds of its contributors might be counterproductive in terms of enabling innovation. More research is needed to explore this potential trade-off between *exploration* (when divergent thinking is likely beneficial) and *exploitation* (when narrow focus is required) [50] in a software development context, particularly open source. For example, while tracking novelty computationally is becoming increasingly feasible (our work demonstrates this), we are still lacking in our understanding of when in the lifecycle of an open-source project innovations occur and who is responsible for them. Future work could start exploring how novel design emerges at the onset of a project and how that compares to, e.g., feature request discussions (by definition opportunities for innovation) occurring throughout a project's lifetime.

Diversity. Questions around the value of having access to diverse information have also come up in research focused on demographic team composition, e.g., around gender [80] and race / ethnicity [68]. This past work posits a similar underlying mechanism (surface-level demographic variables act as a proxy for deeper-level, less observable differences in backgrounds, skills, approaches to solve problems, etc) to the one we measure here more directly. Now that such measurements are becoming possible, it would be interesting to further test the relationships between team surface-level attributes, informational or network diversity, and outcomes.

AI and Creativity. Finally, we can't help but join colleagues [40] in wondering how AI-based assistants will impact the innovativeness of the software being created. A large language model has certainly seen more diverse information in its training than any individual developer. But is the user experience designed to expose that diversity, e.g., when generating code snippets automatically, or might AI-generated code end up looking more like a regression to the mean with current iteraction modalities?

6 CONCLUSION

Much like in Granovetter's jobs search study exposing the strength of weak ties, we conclude that open-source developers may also find useful ideas to help them create innovative projects through passive GitHub interactions, like starring repositories, more often than through active engagement like committing to a repository.

Data Availability. Our replication package[6] (Zenodo post acceptance) includes data and scripts to reproduce our tables and figures.

REFERENCES

[1] Shoaib Abdul Basit and Kehinde Medase. 2019. The diversity of knowledge sources and its impact on firm-level innovation: Evidence from Germany. *European Journal of Innovation Management* 22, 4 (2019), 681–714.

[2] Abdulkareem Alali, Huzefa Kagdi, and Jonathan I Maletic. 2008. What's a typical commit? a characterization of open source software repositories. In *2008 16th IEEE international conference on program comprehension*. IEEE, 182–191.

[3] Adam Alami, Marisa Leavitt Cohn, and Andrzej Wąsowski. 2019. Why does code review work for open source software communities?. In *International Conference on Software Engineering (ICSE)*. IEEE, 1073–1083.

[4] Eytan Bakshy, Itamar Rosenn, Cameron Marlow, and Lada Adamic. 2012. The role of social networks in information diffusion. In *Proceedings of the 21st international conference on World Wide Web*. 519–528.

[5] Sebastian Baltes and Stephan Diehl. 2018. Towards a theory of software development expertise. In *International Conference on the Foundations of Software Engineering (FSE)*. 187–200.

[6] Alejandra Beghelli and Sara Jones. 2020. On the novelty of software products. In *International Conference on Design Creativity (ICDC)*. The Design Society, 11–18.

[7] Tim Berners-Lee. 1999. *Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor*. Harper San Francisco.

[8] Kelly Blincoe, Jyoti Sheoran, Sean Goggins, Eva Petakovic, and Daniela Damian. 2016. Understanding the popular users: Following, affiliation influence and leadership on GitHub. *Information and Software Technology* 70 (2016), 30–39.

[9] Margaret A Boden. 2004. *The creative mind: Myths and mechanisms*. Routledge.

[10] Hudson Borges and Marco Tulio Valente. 2018. What's in a github star? understanding repository starring practices in a social coding platform. *Journal of Systems and Software* 146 (2018), 112–129.

[11] Melanie S Brucks and Jonathan Levav. 2022. Virtual communication curbs creative idea generation. *Nature* 605, 7908 (2022), 108–112.

[12] Ronald S Burt. 2000. The network structure of social capital. *Research in organizational behavior* 22 (2000), 345–423.

[13] Ronald S Burt. 2004. Structural holes and good ideas. *Amer. J. Sociology* 110, 2 (2004), 349–399.

[14] Andrea Capiluppi, Alexander Serebrenik, and Leif Singer. 2012. Assessing technical candidates on the social web. *IEEE Software* 30, 1 (2012), 45–51.

[15] Gina N Cervetti, Carolyn A Jaynes, and Elfrieda H Hiebert. 2009. Increasing opportunities to acquire knowledge through reading. *Reading more, reading better* (2009), 79–100.

[16] Souti Chattopadhyay, Thomas Zimmermann, and Denae Ford. 2021. Reel life vs. real life: How software developers share their daily life through vlogs. In *International Conference on the Foundations of Software Engineering (FSE)*. 404–415.

[17] Jailton Coelho and Marco Tulio Valente. 2017. Why modern open source projects fail. In *International Conference on the Foundations of Software Engineering (FSE)*. 186–196.

[18] J Daniel Couger and Geoff Dengate. 1992. Measurement of creativity of IS products. In *Hawaii International Conference on System Sciences (HICSS)*, Vol. 4. IEEE, 288–298.

[19] Kevin Crowston and James Howison. 2005. The social structure of free and open source software development. (2005).

[20] Jose Ricardo da Silva, Esteban Clua, Leonardo Murta, and Anita Sarma. 2015. Niche vs. breadth: Calculating expertise over time through a fine-grained analysis. In *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 409–418.

[21] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *ACM Conference on Computer Supported Cooperative Work (CSCW)*. 1277–1286.

[22] Jamal I Daoud. 2017. Multicollinearity and regression analysis. In *Journal of Physics: Conference Series*, Vol. 949. IOP Publishing, 012009.

[23] Tapajit Dey, Andrey Karnauch, and Audris Mockus. 2021. Representation of developer expertise in open source software. In *International Conference on Software Engineering (ICSE)*. IEEE, 995–1007.

[6] https://github.com/icsesubmission/replication_package_icse26_novelty

[24] Tapajit Dey, Sara Mousavi, Eduardo Ponce, Tanner Fry, Bogdan Vasilescu, Anna Filippova, and Audris Mockus. 2020. Detecting and characterizing bots that commit code. In *International Conference on Mining Software Repositories*. 209–219.

[25] Oscar Dieste, Alejandrina M Aranda, Fernando Uyaguari, Burak Turhan, Ayse Tosun, Davide Fucci, Markku Oivo, and Natalia Juristo. 2017. Empirical evaluation of the effects of experience on code quality and programmer productivity: an exploratory study. *Empirical Software Engineering* 22 (2017), 2457–2542.

[26] Marina Du Plessis. 2007. The role of knowledge management in innovation. *Journal of knowledge management* 11, 4 (2007), 20–29.

[27] Henry Edison, Nauman Bin Ali, and Richard Torkar. 2013. Towards innovation measurement in the software industry. *Journal of systems and software* 86, 5 (2013), 1390–1407.

[28] Jeanette Engzell and Charlotte Norrman. 2023. Podcasts As A Learning Method In Engineering Education. In *51st Annual Conference of the European Society for Engineering Education*. SEFI, 398–405.

[29] Hongbo Fang, James Herbsleb, and Bogdan Vasilescu. 2023. Matching Skills, Past Collaboration, and Limited Competition: Modeling When Open-Source Projects Attract Contributors. In *International Conference on the Foundations of Software Engineering (FSE)*. 42–54.

[30] Hongbo Fang, Hemank Lamba, James Herbsleb, and Bogdan Vasilescu. 2022. "This is damn slick!" Estimating the impact of tweets on open source project popularity and new contributors. In *International Conference on Software Engineering (ICSE)*. 2116–2129.

[31] Hongbo Fang, Bogdan Vasilescu, and James Herbsleb. 2024. Novelty Begets Popularity, But Curbs Participation-A Macroscopic View of the Python Open-Source Ecosystem. In *International Conference on Software Engineering (ICSE)*. ACM, 643–653.

[32] Santo Fortunato, Carl T Bergstrom, Katy Börner, James A Evans, Dirk Helbing, Staša Milojević, Alexander M Petersen, Filippo Radicchi, Roberta Sinatra, Brian Uzzi, et al. 2018. Science of science. *Science* 359, 6379 (2018), eaao0185.

[33] Marco Gerosa, Igor Wiese, Bianca Trinkenreich, Georg Link, Gregorio Robles, Christoph Treude, Igor Steinmacher, and Anita Sarma. 2021. The shifting sands of motivation: Revisiting what drives contributors in open source. In *International Conference on Software Engineering (ICSE)*. IEEE, 1046–1058.

[34] Mark S Granovetter. 1973. The strength of weak ties. *American journal of sociology* 78, 6 (1973), 1360–1380.

[35] Wouter Groeneveld, Laurens Luyten, Joost Vennekens, and Kris Aerts. 2021. Exploring the role of creativity in software engineering. In *International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*. IEEE, 1–9.

[36] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[37] Hideaki Hata, Christoph Treude, Raula Gaikovina Kula, and Takashi Ishio. 2019. 9.6 million links in source code comments: Purpose, evolution, and decay. In *International Conference on Software Engineering (ICSE)*. IEEE, 1211–1221.

[38] Bas Hofstra, Vivek V Kulkarni, Sebastian Munoz-Najar Galvez, Bryan He, Dan Jurafsky, and Daniel A McFarland. 2020. The diversity–innovation paradox in science. *Proceedings of the National Academy of Sciences* 117, 17 (2020), 9284–9291.

[39] Akira Inokuchi, Yusuf Sulistyo Nugroho, Supatsara Wattanakriengkrai, Fumiaki Konishi, Hideaki Hata, Christoph Treude, Akito Monden, and Kenichi Matsumoto. 2019. From academia to software development: publication citations in source code comments. *arXiv preprint arXiv:1910.06932* (2019).

[40] Victoria Jackson, Bogdan Vasilescu, Daniel Russo, Paul Ralph, Maliheh Izadi, Rafael Prikladnicki, Sarah D'angelo, Sarah Inman, Anielle Andrade, and André van der Hoek. 2024. The Impact of Generative AI on Creativity in Software Development: A Research Agenda. *ACM Transactions on Software Engineering and Methodology* (2024).

[41] Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. 2017. Classifying developers into core and peripheral: An empirical study on count and network metrics. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 164–174.

[42] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. 2016. An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering* 21 (2016), 2035–2071.

[43] Christian Kandler, Rainer Riemann, Alois Angleitner, Frank M Spinath, Peter Borkenau, and Lars Penke. 2016. The nature of creativity: The roles of genetic factors, personality traits, cognitive abilities, and environmental sources. *Journal of Personality and Social Psychology* 111, 2 (2016), 230.

[44] Robert L Kaufman. 2013. *Heteroskedasticity in regression: Detection and correction*. Sage Publications.

[45] Brent Kitchens, Steven L Johnson, and Peter Gray. 2020. Understanding echo chambers and filter bubbles: The impact of social media on diversification and partisan shifts in news consumption. *MIS Quarterly* 44, 4 (2020).

[46] Sandeep Krishnamurthy. 2006. On the intrinsic and extrinsic motivation of free/libre/open source (FLOSS) developers. *Knowledge, Technology & Policy* 18, 4 (2006), 17–39.

[47] Karim R Lakhani and Robert G Wolf. 2005. Why hackers do what they do: Understanding motivation and effort in free/open source software projects. (2005).

[48] Hemank Lamba, Asher Trockman, Daniel Armanios, Christian Kästner, Heather Miller, and Bogdan Vasilescu. 2020. Heard it through the Gitvine: an empirical study of tool diffusion across the npm ecosystem. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 505–517.

[49] Paul Luo Li, Amy J Ko, and Andrew Begel. 2020. What distinguishes great software engineers? *Empirical Software Engineering* 25 (2020), 322–352.

[50] Lu Liu, Nima Dehmamy, Jillian Chown, C Lee Giles, and Dashun Wang. 2021. Understanding the onset of hot streaks across artistic, cultural, and scientific careers. *Nature Communications* 12, 1 (2021), 5392.

[51] Yuxing Ma, Chris Bogart, Sadika Amreen, Russell Zaretzki, and Audris Mockus. 2019. World of code: an infrastructure for mining the universe of open source VCS data. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 143–154.

[52] Jennifer Marlow, Laura Dabbish, and Jim Herbsleb. 2013. Impression formation in online peer production: activity traces and personal profiles in GitHub. In *ACM Conference on Computer Supported Cooperative Work (CSCW)*. 117–128.

[53] Thorsten Merten, Bastian Mager, Paul Hübner, Thomas Quirchmayr, Barbara Paech, and Simone Bürsner. 2015. Requirements Communication in Issue Tracking Systems in Four Open-Source Projects.. In *REFSQ workshops*. 114–125.

[54] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[55] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems* 26 (2013).

[56] Audris Mockus, Roy T Fielding, and James D Herbsleb. 2002. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 11, 3 (2002), 309–346.

[57] Rahul Mohanani, Prabhat Ram, Ahmed Lasisi, Paul Ralph, and Burak Turhan. 2017. Perceptions of creativity in software engineering research and practice. In *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 210–217.

[58] Jagadeesh Nandigam, Venkat N Gudivada, and Abdelwahab Hamou-Lhadj. 2008. Learning software engineering principles using open source software. In *2008 38th Annual Frontiers in Education Conference*. IEEE, S3H–18.

[59] Mark EJ Newman. 2003. The structure and function of complex networks. *SIAM review* 45, 2 (2003), 167–256.

[60] Gang Peng, Jifeng Mu, and C Anthony Di Benedetto. 2013. Learning and open source software license choice. *Decision Sciences* 44, 4 (2013), 619–643.

[61] Karthik Rajkumar, Guillaume Saint-Jacques, Iavor Bojinov, Erik Brynjolfsson, and Sinan Aral. 2022. A causal test of the strength of weak ties. *Science* 377, 6612 (2022), 1304–1310.

[62] Martin P Robillard, Deeksha M Arya, Neil A Ernst, Jin LC Guo, Maxime Lamothe, Mathieu Nassif, Nicole Novielli, Alexander Serebrenik, Igor Steinmacher, and Klaas-Jan Stol. 2024. Communicating Study Design Trade-offs in Software Engineering. *ACM Transactions on Software Engineering and Methodology* (2024).

[63] Gregorio Robles, Jesus M Gonzalez-Barahona, and Israel Herraiz. 2009. Evolution of the core team of developers in libre software projects. In *International Conference on Mining Software Repositories (MSR)*. IEEE, 167–170.

[64] AS Saabith, MMM Fareez, and T Vinothraj. 2019. Python current trend applications-an overview. *International Journal of Advance Engineering and Research Development* 6, 10 (2019).

[65] Patrick Schober, Christa Boer, and Lothar A Schwarte. 2018. Correlation coefficients: appropriate use and interpretation. *Anesthesia & analgesia* 126, 5 (2018), 1763–1768.

[66] Joseph Alois Schumpeter. 1939. Business cycles: a theoretical, historical and statistical analysis of the capitalist process. (1939).

[67] Joseph A Schumpeter and Richard Swedberg. 2021. *The theory of economic development*. Routledge.

[68] Sheik Shameer, Gema Rodríguez-Pérez, and Meiyappan Nagappan. 2023. Relationship between diversity of collaborative group members' race and ethnicity and the frequency of their collaborative contributions in GitHub. *Empirical Software Engineering* 28, 4 (2023), 83.

[69] Feng Shi and James Evans. 2023. Surprising combinations of research contents and contexts are related to impact and emerge with scientific outsiders from distant disciplines. *Nature Communications* 14, 1 (2023), 1641.

[70] Leif Singer, Fernando Figueira Filho, and Margaret-Anne Storey. 2014. Software engineering at the speed of light: how developers stay current using Twitter. In *International Conference on Software Engineering (ICSE)*. 211–221.

[71] Param Vir Singh and Corey Phelps. 2013. Networks, social influence, and the choice among competing innovations: Insights from open source software licenses. *Information Systems Research* 24, 3 (2013), 539–560.

[72] Sabine Sonnentag. 1995. Excellent software professionals: Experience, work activities, and perception by peers. *Behaviour & Information Technology* 14, 5 (1995), 289–299.

[73] Sindre Sorhus. 2025. Collection of Awesome GitHub Repositories. https://github.com/sindresorhus/awesome.

[74] KR Srinath. 2017. Python–the fastest growing programming language. *International Research Journal of Engineering and Technology* 4, 12 (2017), 354–357.

[75] Marco Tortoriello, Bill McEvily, and David Krackhardt. 2015. Being a catalyst of innovation: The role of knowledge diversity and network closure. *Organization Science* 26, 2 (2015), 423–438.

[76] Asher Trockman, Shurui Zhou, Christian Kästner, and Bogdan Vasilescu. 2018. Adding sparkle to social coding: An empirical study of repository badges in the npm ecosystem. In *International Conference on Software Engineering (ICSE)*. 511–522.

[77] Kimberly Truong, Courtney Miller, Bogdan Vasilescu, and Christian Kästner. 2022. The unsolvable problem or the unheard answer? A dataset of 24,669 open-source software conference talks. In *International Conference on Mining Software Repositories (MSR)*. 348–352.

[78] Scott F Turner, Richard A Bettis, and Richard M Burton. 2002. Exploring depth versus breadth in knowledge management strategies. *Computational & Mathematical Organization Theory* 8 (2002), 49–73.

[79] Brian Uzzi, Satyam Mukherjee, Michael Stringer, and Ben Jones. 2013. Atypical combinations and scientific impact. *Science* 342, 6157 (2013), 468–472.

[80] Bogdan Vasilescu, Daryl Posnett, Baishakhi Ray, Mark GJ van den Brand, Alexander Serebrenik, Premkumar Devanbu, and Vladimir Filkov. 2015. Gender and tenure diversity in GitHub teams. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*. 3789–3798.

[81] Georg Von Krogh, Sebastian Spaeth, and Karim R Lakhani. 2003. Community, joining, and specialization in open source software innovation: a case study. *Research Policy* 32, 7 (2003), 1217–1241.

[82] Supatsara Wattanakriengkrai, Bodin Chinthanet, Hideaki Hata, Raula Gaikovina Kula, Christoph Treude, Jin Guo, and Kenichi Matsumoto. 2022. GitHub repositories with links to academic papers: Public access, traceability, and evolution. *Journal of Systems and Software* 183 (2022), 111117.

[83] Mairieli Wessel, Bruno Mendes De Souza, Igor Steinmacher, Igor S Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco A Gerosa. 2018. The power of bots: Characterizing and understanding bots in OSS projects. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW (2018), 1–19.

[84] Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems* 2, 1-3 (1987), 37–52.

[85] Marvin Wyrich and Justus Bogner. 2024. Beyond Self-Promotion: How Software Engineering Research Is Discussed on LinkedIn. In *International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*. 85–95.

[86] Xuan Yang, Xiao Li, Daning Hu, and Harry Jiannan Wang. 2021. Differential impacts of social influence on initial and sustained participation in open source software projects. *Journal of the Association for Information Science and Technology* 72, 9 (2021), 1133–1147.

[87] Xunhui Zhang, Tao Wang, Gang Yin, Cheng Yang, Yue Yu, and Huaimin Wang. 2017. Devrec: a developer recommendation system for open source repositories. In *Mastering Scale and Complexity in Software Reuse: 16th International Conference on Software Reuse, ICSR 2017, Salvador, Brazil, May 29-31, 2017, Proceedings 16*. Springer, 3–11.

A USING TRANSITIVITY TO VALIDATE THE 'STRENGTH' OF DIFFERENT NETWORKS

Figure 6: In strongly tied social networks, triads are unlikely.

In Section 3.3, we reason that interactions via commits, issues, and stars reflect ties of decreasing strength, and here we provide an empirical validation of this assumption through the comparison of transitivity among the three networks.

Transitivity measures the density of local connections in the local network [59]. Consider the illustration in Figure 6: if node A is connected to B and C, a triad is formed. In social network theory, if A's connections to B and C are both strong ties, the triad is unstable and will likely evolve into a triangle where A, B, and C are all connected. This evolution occurs because B and C have more opportunities to interact and increase their mutual familiarity due to their strong ties with A. Thus, networks characterized by strong ties typically exhibit a greater number of triangles, as opposed to triads, compared to those with weak ties [34]. Formally, this can be computed using the measurement of transitivity, defined as $T = 3 * N_{\text{triangles}}/N_{\text{triads}}$. A higher transitivity value (number of triangles relative to triads) indicates a network of stronger ties.

Similarly, we compare the transitivity values of our three networks, summarized in Table 4.[7] Overall, we observe approximately an order of magnitude (10×) difference in transitivity values between each pair of networks. Specifically, the commit network displays the highest levels of transitivity, followed by the issue network, while the star network exhibits the lowest level. Thus, these findings are consistent with our theoretical understanding of tie strength.

Table 4: Validating the relatively decreasing strength of commit, issue, and star network ties.

Interaction	#Nodes	#Edges	Transitivity ($\times 10^{-2}$)
Commits	763, 062	1, 926, 978	30.04
Issues	278, 945	727, 255	3.42
Stars	480, 394	3, 658, 543	0.23

B REPLICATION WITH THE ATYPICALITY MEASURE BY FANG ET AL

We further test the robustness of our regression results by replicating the analysis presented in Table 3 using the original measurement of atypicality defined by Fang et al. [31] as the outcome variable. The results of this robustness analysis are reported in Table 5. It is important to note that the range of the atypicality score in the main analysis differs from that in the robustness analysis. As a result, the magnitudes of the estimated coefficients may vary; however, the overall qualitative patterns remain consistent.

Table 5: Regression analysis on factors associated with differences in project innovativeness, with innovativeness measured with the atypicality measure by Fang et al. [31]

	Model I	Model II	Model III	Model IV
Variables of interest				
Deg_{ave} ($\mathbf{H}_1$)	-0.055^{***}	-0.009^{*}		-0.017^{***}
	(0.0015)	(0.0044)		(0.0045)
$Deg_{weakness}$	0.009^{***}	0.012^{*}		-0.020^{***}
	(0.0022)	(0.0054)		(0.0059)
Div_{ave} ($\mathbf{H}_2$)			0.042^{***}	0.048^{***}
			(0.0042)	(0.0044)
$Div_{weakness}$ ($\mathbf{H}_3$)			0.045^{***}	0.053^{***}
			(0.0056)	(0.0060)
Controls				
Org_owned	0.125^{***}	0.071^{***}	0.074^{***}	0.076^{***}
	(0.0058)	(0.0118)	(0.0116)	(0.0118)
$N_{\text{Owner\_Star}}$ (log)	0.016^{***}	0.011^{***}	0.010^{***}	0.013^{***}
	(0.0020)	(0.0028)	(0.0027)	(0.0028)
$N_{\text{Core\_Dev}}$ (log)	0.189^{***}	0.093^{***}	0.087^{***}	0.094^{***}
	(0.0053)	(0.0114)	(0.0112)	(0.0114)
N_{Packages} (log)	0.236^{***}	0.150^{***}	0.153^{***}	0.154^{***}
	(0.0031)	(0.0080)	(0.0079)	(0.0080)
Fixed effect				
$\text{Year}_{\text{creation}}$	✓	✓	✓	✓
Observations	589,999	37,451	37,451	37,451
Adjusted R^2	0.044	0.034	0.038	0.038

*p<0.05; **p<0.01; ***p<0.001

C REPLICATION WITH ALTERNATIVE DEFINITIONS OF CORE DEVELOPERS AND INTERACTION PERIODS

In Section 3.3, we constructed knowledge networks between projects based on interactions among their *core developers* within a *12-month* period prior to their first contributions. To assess the robustness of our findings, we conducted replication analyses using alternative definitions of core developers and varying lengths of interaction periods, as reported in Table 3.

Following established practice [56], we alternatively identify core developers as the top contributors to a project whose cumulative commits account for at least 80% of all project commits. The corresponding regression results using this alternative definition are presented in Table 6. Similarly, we extend the interaction window from 12 to 24 months prior to a core developer's first contribution to the focal project when constructing the networks. The regression results based on this extended interaction period are presented in Table 7.

We also run the same model (as model IV in Table 3) with several other combinations of core developer identification approach

[7] Since network transitivity is usually defined for undirected networks, we first convert our three networks (which are all directed) to their undirected form. That is, there is an undirected edge between nodes A and B if there is either an edge from A to B or from B to A. In addition, the reported number of nodes excludes isolates (i.e., nodes without edges) from each network, and the number of edges is calculated in the undirected version of each network.

and length of interaction periods, and the estimated coefficient of Variables of Interest are presented in Table 8

Table 6: Regression analysis with alternative core developer identification approach

	Model I	Model II	Model III	Model IV
Variables of interest				
Deg_{ave} ($\mathbf{H}_1$)	0.001^{***}	-0.001		-0.002^{***}
	(0.0002)	(0.0006)		(0.0006)
$Deg_{weakness}$	-0.001^{**}	-0.002^{*}		-0.006^{***}
	(0.0002)	(0.0007)		(0.0008)
Div_{ave} ($\mathbf{H}_2$)			0.007^{***}	0.008^{***}
			(0.0006)	(0.0006)
$Div_{weakness}$ ($\mathbf{H}_3$)			0.004^{***}	0.006^{***}
			(0.0007)	(0.0008)
Controls				
Org_owned	0.028^{***}	0.018^{***}	0.020^{***}	0.020^{***}
	(0.0006)	(0.0016)	(0.0015)	(0.0016)
$N_{\text{Owner\_Star}}$ (log)	0.002^{***}	0.001	0.0004	0.0007
	(0.0002)	(0.0004)	(0.0004)	(0.0004)
$N_{\text{Core\_Dev}}$ (log)	0.010^{***}	0.017^{***}	0.015^{***}	0.016^{***}
	(0.0005)	(0.0015)	(0.0015)	(0.0015)
N_{Packages} (log)	0.043^{***}	0.025^{***}	0.025^{***}	0.026^{*}
	(0.0003)	(0.0010)	(0.0010)	(0.0010)
Fixed effect				
$\text{Year}_{\text{creation}}$	✓	✓	✓	✓
Observations	679,930	40,589	40,589	40,589
Adjusted R^2	0.053	0.064	0.068	0.070

*p<0.05; **p<0.01; ***p<0.001

Table 7: Regression analysis with 24-month period length to construct networks

	Model I	Model II	Model III	Model IV
Variables of interest				
Deg_{ave} ($\mathbf{H}_1$)	0.002^{***}	0.0003		-0.001^{**}
	(0.0002)	(0.0005)		(0.0005)
$Deg_{weakness}$	-0.0002	-0.002^{**}		-0.007^{***}
	(0.0002)	(0.0006)		(0.0007)
Div_{ave} ($\mathbf{H}_2$)			0.007^{***}	0.008^{***}
			(0.0004)	(0.0005)
$Div_{weakness}$ ($\mathbf{H}_3$)			0.004^{***}	0.007^{***}
			(0.0007)	(0.0007)
Controls				
Org_owned	0.027^{***}	0.020^{***}	0.022^{***}	0.021^{***}
	(0.0006)	(0.0014)	(0.0014)	(0.0014)
$N_{\text{Owner\_Star}}$ (log)	0.002^{***}	0.001^{*}	0.0010^{**}	0.0012^{***}
	(0.0002)	(0.0003)	(0.0003)	(0.0003)
$N_{\text{Core\_Dev}}$ (log)	0.013^{***}	0.015^{***}	0.015^{***}	0.015^{***}
	(0.0006)	(0.0014)	(0.0013)	(0.0013)
N_{Packages} (log)	0.042^{***}	0.024^{***}	0.025^{***}	0.025^{***}
	(0.0003)	(0.0009)	(0.0009)	(0.0009)
Fixed effect				
$\text{Year}_{\text{creation}}$	✓	✓	✓	✓
Observations	587,710	51,571	51,571	51,571
Adjusted R^2	0.055	0.055	0.059	0.061

*p<0.05; **p<0.01; ***p<0.001

Table 8: Regression analysis with different core developer identification and length of interaction periods

		Variables			
Core developer identifica-tion	**Interaction period length (month)**	Deg_{ave}	$Deg_{weakness}$	Div_{ave}	$Div_{weakness}$
Original	6	-0.001	-0.005^{***}	0.006^{***}	0.006^{***}
		(0.0009)	(0.0011)	(0.0008)	(0.0008)
Alternative	6	-0.003^{**}	-0.006^{***}	0.007^{***}	0.007^{***}
		(0.0008)	(0.0010)	(0.0008)	(0.0010)
Alternative	24	-0.003^{**}	-0.007^{***}	0.010^{***}	0.006^{***}
		(0.0005)	(0.0007)	(0.0005)	(0.0007)

*p<0.05; **p<0.01; ***p<0.001

D RELATIONSHIP BETWEEN AWESOME AND ATYPICAL PROJECTS.

Table 9: Regression analysis on the relationship between awesome projects and project innovativeness. (Outcome variable: Project Atypicality)

	Estimated coefficient
Variables of interest	
Is awesome	0.028^{***}
	(0.0052)
Controls	
Org_owned	0.017^{***}
	(0.0016)
$N_{\text{Owner\_Star}}$ (log)	0.001
	(0.0004)
$N_{\text{Core\_Dev}}$ (log)	0.017^{***}
	(0.0015)
N_{Packages} (log)	0.023^{***}
	(0.0011)
Fixed effect	
$\text{Year}_{\text{creation}}$	✓
Observations	37,451
Adjusted R^2	0.059

*p<0.05; **p<0.01; ***p<0.001

In Section 3.6, we present that the "awesome" projects on average are more atypical per our measurement than other projects in our sample, and we further validate this result with a regression analysis in Table 9, where we show that this relationship is not driven by confounding variables such as the project size and team size.

E REAL-WORLD EXAMPLES OF PROJECTS WITH VARYING LEVEL OF TIE DIVERSITY

In Section 3.5, we present graphs depicting the networks of projects characterized by varying levels of average diversity and relative weak tie diversity. To concretize those abstract diagrams, we also provide real-world examples in Figure 7.

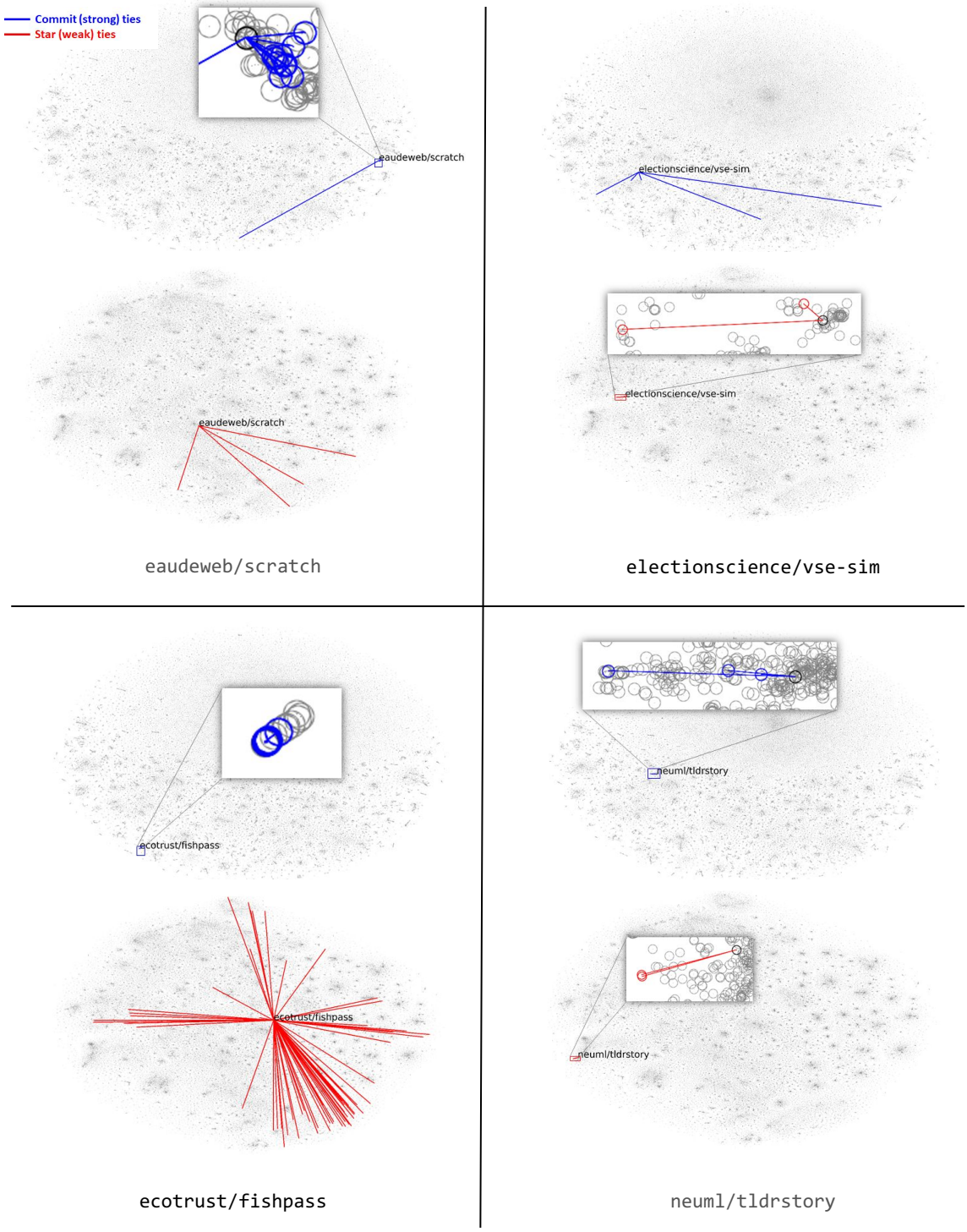


Figure 7: Real-world examples for projects with varying level of diversity, corresponding to the four quadrants in Figure 3.