How does the Division of Labor Affect Team Productivity? Evidence from GitHub

Jinci Liu *

October 28, 2025

(Preliminary Draft-Latest Version Available Here)

Abstract

Does the division of labor increase team productivity? This paper provides new evidence challenging the conventional view that specialization increases productivity. I create a panel dataset from GitHub, covering 35 million task allocations across 64,400 software development teams from 2017 to 2023. My result shows a negative relationship between team specialization and various productivity metrics, including output quality, quantity, and user issue resolution time. To identify causal effects, I exploit GitHub's introduction of an automatic task assignment feature, which evenly distributes tasks across team members. Using a matched difference-in-differences design, I find that adoption of this feature reduces specialization and leads to significant gains in productivity: output quality rises by 4%, output quantity by 21%. Team communication also increases by 13%, suggesting that improved interaction and knowledge exchange are a key mechanism behind these productivity gains. These findings highlight a trade-off in non-routine production: while specialization increases task-specific human capital, it impedes cross-task knowledge spillovers that are essential for innovation.

JEL Classification: C55, D23, L23, L86, J24, M54
Keywords: task allocation, human capital, team organization, digital economy

*IIES, Stockholm University. Email: jinci.liu@iies.su.se.
I thank David Card, Sahiba Chopra, Mitch Downey, Jie Gong, Patrick Kline, Yu Ching Lam, Danielle Li, Zhongyi Liu, Christos Makridis, Arash Nekoei, Jonatan Riberth, David Schönholzer, Jósef Sigurdsson, Carolyn Stein, David Strömberg, Hui Sun, Dominik Wehr, Horng Chern Wong, and Michael Wong for their valuable feedback. I also thank seminar participants at the Stockholm University, UC Berkeley, the 1st Asian Conference on Organizational Economics, and CEPR-PSE. I am grateful to Yundi Xiao and Zhehao Zhu for their excellent research assistance and to many of my software developer friends for their valuable insights. I acknowledge support from the Google Cloud Research and thank the Mannerfelt Foundation for its generous financial support. All errors are my own.

1 Introduction

Significant differences in productivity and organizational structure exist both across and within firms. A central question in economics is how to optimally allocate tasks among workers. The seminal answer from Smith (1776), illustrated through the pin factory example, suggests that the division of labor increases productivity. Later theoretical work builds on this idea, showing that specialization helps human capital accumulation and raises productivity. [1]

While prior studies often assume that specialization improves productivity (Deming, 2017; Bassi et al., 2023), direct empirical evidence remains limited. This paper studies task allocation specialization within software development teams and finds that specialization is detrimental to team productivity, reducing both code output quantity and quality.

The effect of specialization on team productivity is ambiguous in non-routine production. Unlike routine production, which follows explicit rules, non-routine production requires problem-solving and complex communication (Autor et al., 2003). This distinction creates a trade-off: focusing on a narrow set of tasks helps workers accumulate task-specific skills and increase productivity, but it concurrently restricts knowledge sharing and discussion across tasks, which are essential in non-routine production (Levinthal and March, 1993). As Hayek (1945) argued, organizations exist primarily to coordinate and combine diverse information—a function that becomes indispensable in complex, uncertain environments.

This paper uses novel data from GitHub—the world's largest online coding platform. GitHub, valued at over 8 trillion USD (Hoffmann et al., 2024), plays a central role in the software industry. I construct a panel of 35 million code files, 292,840 developers, and 64,400 teams over seven years. This granular data captures individual activity, specifying who engages in particular files and at which point in time. It enables the construction of novel measures for team specialization and productivity at a team-month level.

Building on this dataset, I measure team task specialization by comparing how tasks are distributed across team members. Using a topic classification model, I assign millions of files to 10 task types (e.g., frontend, backend) and compute the task distribution within each team. I then benchmark observed allocations against two extremes: "most specialized" and "most generalized" structures, controlling for team size and task type distribution. A higher specialization index indicates that a team is further from the "most generalized" counterfactual. I link this index to multiple productivity outcomes: output quality (user appreciation), output quantity (lines of code), problem-solving speed (issue resolution time), and code acceptance rates. These metrics capture distinct aspects of productivity and are broadly applicable across team-based work.

I find a negative relationship between team specialization and productivity. One standard

[1] For example: Rosen (1983); Baumgardner (1988); Becker and Murphy (1992)

deviation increase in the specialization index corresponds to a 2.2% decline in team output quality and a 24.3% decline in team output quantity. These results hold after controlling for team size, task type, team age, and team and member fixed effects. Specialized teams show a 0.47% higher acceptance rate for submitted code, suggesting stronger task-specific skills, but also take 1.08 days longer to resolve user issues. The longer resolution time indicates that members in specialized teams are less familiar with tasks beyond their assigned roles, reducing the team's functional flexibility and slowing its response to problems.

OLS estimates show that specialized teams are associated with less productivity. However, task allocation is often endogenous, as managers may assign tasks based on past performance. To address this concern, I exploit GitHub's automatic task assignment feature, which randomly distributes tasks across team members. Adoption of this feature reduces within-team specialization by enforcing a more balanced allocation.

I apply a one-to-one matched differences-in-differences design, comparing teams that adopt automatic task assignment (treatment) to observationally similar teams that do not (control), ensuring both groups share similar characteristics. This matching approach helps to mitigate selection bias and pre-existing differences in specialization.

Adoption of the feature decreases specialization and increases output quality by 4% and output quantity by 21%. Code acceptance rate decreases by 1.2% (from a baseline of 0.83)and increases team communication by 13%. These results are consistent with the OLS findings, reinforcing the conclusion that specialization harms team productivity.

This paper makes three contributions to the understanding of how organizational structure affects productivity. First, it provides new evidence on the productivity effects of specialization in non-routine production. Prior studies focus on routine settings, such as cashiering or salon services, where tasks are repetitive and codified, and specialization tends to improve performance (Gong and Png, 2024; Kohlhepp, 2024). In contrast, software development requires problem-solving and coordination, and I find that specialization reduces productivity by limiting cross-task learning and communication. This finding also revisits classical theories of the division of labor (Rosen, 1983; Baumgardner, 1988; Becker and Murphy, 1992), which emphasize productivity gains from specialization. Empirical tests of these theories have been constrained by data and identification challenges. I address this by leveraging GitHub's automatic task assignment feature—which randomly distributes tasks across team members—and applying a matched difference-in-differences design to estimate the causal effect of specialization on team performance.

Second, the paper identifies the knowledge spillover lost with particular relevance for non-routine work. Both routine and non-routine production incur coordination costs (Becker and

Murphy, 1992; Lazear, 1995),[2] but learning costs matter more in contexts like R&D, where project trajectories are uncertain and task interdependence is high. Excessive specialization can limit idea exchange and adaptive feedback. While Garicano and Rossi-Hansberg (2006); Garicano and Hubbard (2009) argue that firms should adopt hierarchical knowledge structures with specialized agents, such frameworks may not suit innovation-driven work. In software development, specialists cannot operate in isolation and require continuous interaction across domains. This helps explain why higher specialization slows problem-solving, consistent with Levinthal and March (1993)'s argument that specialization can lead to organizational myopia.

Third, the paper contributes to recent literature on task complexity and coordination in the labor market. While returns to technical skills in cognitive occupations have declined (Beaudry et al., 2014), demand for coordination and adaptability has risen, especially in complex, fast-evolving environments (Lee and Makridis, 2023). Complexity improves outcomes only when paired with autonomy and effective coordination (Caines et al., 2017). My findings provide micro-level evidence on how task structure affects performance in such settings, underscoring the importance of cross-task learning and communication.

The remainder of this paper is structured as follows. Section 2 describes the data and sample construction. Section 3 details the measurements. Section 4 presents the stylized facts. Section 5 reports the empirical strategy, and Section 6 concludes.

2 Data

This section describes the data sources used to construct the final panel. The first source is the GH Archive, which provides event-level timelines of developer activity. The second source is developer profile pages on GitHub, which include names, profile pictures, locations, email addresses, and brief biographies. The panel covers all public activities from 2017 to 2023.

2.1 GitHub: World's Largest Online Coding Platform

GitHub is the largest online coding platform, where developers store code, share projects, and collaborate. To give a sense of its scale today, as of June 2025, it hosts over 259 million public repositories and more than 161 million users.[3] Its estimated global value exceeds 8 trillion USD (Hoffmann et al., 2024). Technology firms such as Google and Microsoft maintain thousands of public projects on Github. These firms increasingly rely on GitHub for innovation

[2]Becker and Murphy (1992) identifies three types of coordination costs: (1) principal-agent conflicts as members specialize and face weaker output incentives; (2) hold-up problems across interdependent tasks; and (3) miscommunication or poor coordination as the number of specialists increases.

[3]See user and repo, accessed 2025-06-29.

and productivity (Nagle, 2019).[4]

Beyond code storage, GitHub facilitates large-scale, distributed collaboration. It records the full version history of each file, allowing developers to coordinate asynchronously across time zones and teams. These platform-level features are further enhanced by GH Archive, a third-party project that captures all public GitHub activity in near real time. GH Archive preserves event-level traces even when repositories are modified or deleted, making it possible to study open source development with precision over time.

2.1.1 Team: Repository under Organization

A GitHub repository is a project workspace where developers store code, files, and version history.In this paper, I define each repository as a team—a group of developers jointly contributing code toward a shared project, much like researchers co-authoring a paper. This definition follows Jones (2021), who studies "team science" through patterns of scientific collaboration and co-authorship. [5]As Figure 1a shows, some teams operate under organizations, such as Microsoft or DeepSeek, while others are managed by individual users. This study focuses on teams under organizations for three main reasons. First, organizational teams are more likely to involve structured collaboration, with coordinated workflows across multiple developers. Second, they are typically aligned with firm-level goals such as product development or commercialization. Third, organizations often use these teams to signal technical capability and attract talent. Together, these features make organizational teams more representative of real workplace environments. In contrast, individual teams often reflect personal interests or side projects.

[Figure 1 about here.]

2.1.2 Team Members: Defined by GitHub Access Permissions

GitHub user accounts can be either human users ("developers") or machine users ("bot"). This paper focuses only on human users. To join GitHub, each developer must create a unique login ID and can publicly display personal information—such as name, profile image, location,

[4]As of June 2025, Microsoft maintains 6.9k public repositories, Google 2.8k, Apple 323, Meta 149, and Amazon 162.

[5]The concept of a "team" has been defined in various ways in the economics literature. Marschak (1955) defines a team as "an organization the members of which have only common interests," highlighting the alignment of incentives. Holmstrom (1982) defines a team "rather loosely as a group of individuals who are organized so that their productive inputs are related," emphasizing joint production. In more recent work, such as Jones (2021), teams are observed through patterns of co-authorship in science. Following this tradition, I define each GitHub repository as a team—a group of developers jointly contributing code toward a shared project, much like a group of researchers co-authoring a paper. Conceptually, any repository can be viewed as a team; in the empirical analysis I focus on those owned by organizations

firm, bio, and email—on the profile page. Many use this space to highlight programming skills and increase visibility to potential employers. Career-related motivations often drive developers to provide accurate personal details and include links to LinkedIn profiles or personal websites to signal credibility and professional identity (El-Komboz and Goldbeck, 2024).

GitHub teams often include both internal members and external volunteers. While anyone can contribute or comment on a team's work, only internal members hold formal roles with project-level permissions. It is important to distinguish members with internal access from external volunteers. To identify internal team members, I rely on the *author association* field defined by GitHub. This field classifies users as *collaborator*, *contributor*, *member*, *none*, or *owner*,and has been available since 2017. It reflects a developer's access level and formal relationship to the team.

I define team members as users labeled *owner*, *member*, or *collaborator*, as these roles imply access to the team's codebase and active involvement in development.[6] These users typically hold write-level or higher permissions, enabling them to push code, review pull requests, manage issues, and coordinate work. *Owner* and *member* are formally affiliated with organizations, while *collaborator* are invited contributors with similar technical access. In contrast, users labeled *contributor* or *none* represent occasional or external engagement, without sustained involvement or permissions. As shown in Figure 1b, each team may include both team members and external volunteers. This paper focuses exclusively on team members. [7]

2.2 Data Construction

To construct my dataset, I combine the GH Archive with developer profile information from the GitHub API. The GH Archive offers time-stamped records of events from all public projects, providing a comprehensive timeline of GitHub activity since February 2011. It includes all public information on code-related actions, such as pull request events, and tracks changes for each file. For each event, it records the user who contributed, the number of lines of code added, the time it took to open and close a pull request, and project-level metrics such as the number of stars received.

To gather demographic information, I use the GitHub API to access each developer's profile page, which includes their profile image, location, name, email address, and biography. Career factors significantly motivate developers to contribute to online platforms such as GitHub and

[6]This measure is preferable to GitHub's MemberEvent, which captures only a subset of membership changes and may include users who never actively contributed.

[7]Because *author association* is recorded only when a team member performs an observable action, the data are missing for inactive months. I address this by imputing continuous membership between the first and last month in which a user is observed as a team member. In practice, it is rare for a member to leave and rejoin a team within the observation window.

Stack Exchange (El-Komboz and Goldbeck, 2024; Forderer and Burtch, 2024). This suggests that developers have an incentive to disclose accurate personal information that may improve their career prospects. I focus on developers who are affiliated with repositories and have permission to access team files, defining these individuals as team members. For these developers, GitHub functions more as a workspace than a hobby space, providing a greater incentive to disclose accurate personal information that may benefit their careers.

2.3 Mapping Code Files to Tasks

I link each code file to all team members who interacted with it—whether by writing, reviewing, or commenting. Rather than focusing solely on the original author, I treat collaboration as the unit of analysis, reflecting the importance of feedback and code review in modern software development. These interactions help reduce bugs and integration conflicts when merging code into the main project.

The dataset includes 34,762,813 code filenames from 2017 to 2023. To group these files into interpretable task types, I apply Latent Dirichlet Allocation (LDA), a widely used topic modeling method (Blei et al., 2003). Figure 2a shows an example of the raw filenames used as input to the model.

[Figure 2 about here.]

Selecting the number of task types involves a trade-off: too many create sparsity and measurement error; too few obscure meaningful variation in task composition. I select 10 task types to balance interpretability and precision, based on the assumption that files within the same type require similar skills and involve lower coordination costs.[8] Appendix B provides additional details and robustness checks. The results are robust to alternative classifications using 8 and 12 task types.

Table 1 lists the 10 task types along with representative keywords. These include Front-end, Back-end, server management, Android mobile development, cloud infrastructure, UI design, and other common software tasks. Figure 2b illustrates an example of task assignments at the team member level, while Table 2 reports summary statistics for the share of labor allocated to each task type across all team-months.

[Table 1 about here.]

[Table 2 about here.]

[8]In the remainder of the paper, "task" refers to the 10 task types classified using LDA.

2.4 Project Type

Given that each team may work on different projects, I collect information from each team's descriptions, README files and labels. For example, the Visual Studio Code team has the description "Visual Studio Code" and labels including "electron," "microsoft," "editor," "typescript," and "visual-studio-code". Using LDA, I categorized the team into 6 distinct project types: App Development, Container, Data Tools, Education, Library&Framework and Web Development. The appendix section C provides additional details.

2.5 Gender Prediction

To predict the gender of developers based on their profile page avatars, I select a deep learning model that closely matches the images in the training sample in terms of resolution and cropping methods. The chosen model trains vision transformers on male and female portraits and produces, for each image, predicted probabilities for both genders. To determine the optimal cutoff for gender classification, I use a sample labeled by human annotators and evaluate accuracy across different thresholds. The highest accuracy is achieved with a cutoff of 0.85. For each image, I first identify the gender with the higher predicted probability and then assign that gender only if this maximum probability exceeds 0.85; otherwise, the observation is coded as missing. Classification accuracy is computed on the labeled subset where a gender is assigned, excluding cases left unclassified. I then create a variable *Male*, which equals 1 if the image is classified as male, 0 if it is classified as female, and 3 if the classification is missing.

2.6 Race Prediction

I predict developers race and ethnicity based on their name and I choose Ethnicolr, as developed by Laohaprapanon et al. (2017). Ethnicolr uses both first and last names to classify individuals into four combined race and ethnicity categories: Asian, Black, and White. The algorithm is trained on data from the US Census, Florida voter registration, and Wikipedia. Each name is assigned probabilities for belonging to each of the three categories, with the category of highest probability being designated as the predicted race or ethnicity.

3 Measuring Task Assignment and Team Productivity

This section describes the measurement of task specialization and team productivity. I construct a specialization metric that captures how a team's task allocation deviates from two theoretical benchmarks based on team size and task distribution. Specifically, I compare each team's actual task assignment to a generalized benchmark, where tasks are evenly distributed

across members, and a specialized benchmark, where each member focuses on a single task[9]. This relative metric allows comparison of specialization across teams with different sizes and project types.

To measure productivity, I construct four metrics that capture different dimensions of software team performance. These metrics address both the quantity and quality of team outputs, making it easier to distinguish between the effects of specialization on task-specific human capital and general problem-solving capability. Although designed for software development, these measures can be adapted to other settings.

3.1 Team Specialization Measure

Denote $\mathbb{J}$ the set of all task types. Each team $m = 1, \ldots, M$ consists of $N_m \in \mathbb{N}$ homogeneous team members and handles a set of task $J_m \subseteq \mathbb{J}$. Denote $A_m \in \mathbb{R}_+^{N \times |J_m|}$ is the task assignment for team m, where $A_m(i,j)$ represents worker i's labor input on task j.

$$A_m = \begin{pmatrix} A_m(1,1) & A_m(1,2) & \cdots & A_m(1,|J_m|) \\ A_m(2,1) & A_m(2,2) & \cdots & A_m(2,|J_m|) \\ \vdots & \vdots & \ddots & \vdots \\ A_m(N_m,1) & A_m(N_m,2) & \cdots & A_m(N_m,|J_m|) \end{pmatrix}$$

Denote $l_m(i) := \sum_j A_m(i,j)$ to be worker i's labor share and $\alpha_m(j) := \sum_i A_m(i,j)$ to be the task demand of task j. Assume each member has one unit of inelastic labor supply.

Definition 1 *Define G_m to be the most generalized task allocation matrix of team m, where the task assignment of member i to task j is $G_m(i,j) = \frac{\alpha_m(j)}{N_m}$.*

Definition 2 *Define S_m to be the most specialized task allocation matrix of team m.*

Definition 3 *Define SPE_m to be the team specialization index where $SPE_m = \frac{d(A_m, G_m)}{d(S_m, G_m)}$*

Consider three task assignments in Table 3. There are three members (A, B, C) and three task types (1, 2, 3), with task demand $\alpha_m = \begin{bmatrix} 3/2 & 1/2 & 1 \end{bmatrix}$. Suppose the actual assignment(A_m) shows how each member allocates their labor supply across tasks. The most generalized (least specialized) assignment is when all members evenly split their labor across all tasks (G_m) as defined in the Definition 1. In contrast, the most specialized assignment aims to assign each member to a single task, with any residual workload allocated to another member due to unequal task demand (S_m). Since members are homogeneous, it doesn't matter who handles the leftover tasks. One interpretation of the member handling the leftover tasks (e.g., member B) could act as a manager overseeing various tasks.

[9]Some teams cannot assign one task per member because some tasks require multiple units of labor.

[Table 3 about here.]

To measure the degree of specialization in A_m, I calculate the distance between A_m and the fully generalized benchmark G_m, normalized by the distance between the equal split benchmark S_m and G_m, as defined in Definition 3. This yields the team specialization index $\text{SPE}_m \in [0, 1]$, which captures how far the actual task allocation A_m deviates from an equal division of labor, relative to the fully specialized benchmark. A higher value of SPE_m indicates that the team's task allocation is closer to full specialization.

There are many ways to measure the distance between two matrices. In this paper, I use Euclidean distance. [10] I calculate the team specialization index SPE_m

$$\text{SPE}_m = \frac{d(A_m, G_m)}{d(S_m, G_m)}$$

where $d(\cdot, \cdot)$ denotes the Euclidean distance between two task allocation matrices. The numerator is:

$$d(A_m, G_m) = \sqrt{\sum_{i,j} \left(A_m(i,j) - G_m(i,j)\right)^2}$$

The denominator, $d(S_m, G_m)$, captures the distance between equal and fully specialized allocations. Let $\alpha_m(j)$ be the total demand for task j in team m, and N_m the number of team members. Then:

$$\begin{aligned} d(S_m - G_m) = \sqrt{\sum_j} & \underbrace{\lfloor \alpha_m(j) \rfloor}_{\text{\# of 1s}} \cdot \left(1 - \frac{\alpha_m(j)}{N_m}\right)^2 \\ & + \underbrace{\left(N_m - \lceil \alpha_m(j) \rceil\right)}_{\text{\# of 0s}} \cdot \left(\frac{\alpha_m(j)}{N_m}\right)^2 \\ & + \left(\lceil \alpha_m(j) \rceil - \lfloor \alpha_m(j) \rfloor\right) \cdot \left(\underbrace{\alpha_m(j) - \lfloor \alpha_m(j) \rfloor}_{\text{residual workload}} - \frac{\alpha_m(j)}{N_m}\right)^2 \end{aligned}$$

To apply the specialization index SPE_m to empirical data, each code file is categorized into one of the 10 task types outlined in Table 1. Using this classification, I compute the fraction of code files associated with each task type that each team member handles every month, which

[10] I only focus on teams with more than 3 members and more than 1 task type. Mathematically, when $|J_m|$=1 or N_m=1,SPE_m is not well-defined. I also used Kullback-Leibler divergence to measure the distance between the matrices, and the correlation between the specialization indices calculated with Euclidean distance and Kullback-Leibler divergence was 0.971.

forms the matrix $A_m(i, j)$. [11]

For example, consider a team with two tasks: Front-end Development and Back-end Development. If a member exclusively works on files related to Front-end Development, her task assignment would be represented as $A_m(i) = \begin{bmatrix} 1 & 0 \end{bmatrix}$.

Figure 3a presents the distribution of the team specialization index SPE_m for teams with more than 3 members and more than 1 task type, illustrating both the overall distribution and its variation by team size. I restrict the sample to teams to ensure that specialization is both possible and meaningful. In very small teams or single-task settings, there is limited scope for division of labor, and the specialization index becomes mechanically constrained.

Team specialization index SPE ranges from 0.36 at the 25th percentile to 0.76 at the 75th percentile across teams and year-months, with a mean of 0.56 and a standard deviation of 0.28. Many teams adopt highly specialized structures (values near 1), while others exhibit no specialization (value of 0). Most teams fall within the 0.4–0.7 range, suggesting a moderately specialized division of labor.

Figure 3b shows the distribution of specialization by team size. Each panel corresponds to a different team size, ranging from 4 to 15 members. While the overall distribution is stable across sizes, larger teams tend to specialize more. Figure D1 explores related patterns along other organizational dimensions.

[Figure 3 about here.]

3.2 Team Productivity Measure

Measuring productivity is challenging. I construct four metrics to capture different dimensions of team-level productivity, drawing on insights from the software engineering literature. The metrics include output quality, output quantity, submission acceptance rate, and problem-solving speed.

Output Quality. I measure team output quality using GitHub stars, and track the number of stars each team receives monthly. GitHub users can give a project a star to express their interest and appreciation. Each user can only give one star to a project, and this action is not anonymous. GitHub uses these stars to recommend content to users on their dashboards, and they also serve as an indicator of community satisfaction. Prior work finds that 75% of developers consider a project's star count before adoption (Borges and Valente, 2018). Additionally,

[11] An alternative approach would be to measure task assignment based on the fraction of time each member allocates to different tasks. However, I rely on the number of code files, which are then converted into shares, due to the difficulty of precisely observing the time spent on each file.

each star is estimated to carry a market value of roughly 0.88 USD (Eldeeb and Sikora, 2023). To measure a team output quality, Figure E1 shows an example of cumulative stars over time.

Output Quantity. I measure output quantity using the number of lines of code produced by team members each month. The relationship between quantity and productivity is difficult to determine, as coding techniques like loops can reduce the number of lines without affecting functionality. Despite the limitation, quantity metrics still offer valuable insights into the volume of code generated, with the number of lines serving as a general indicator of a team's overall output (Vasilescu et al., 2015; Casalnuovo et al., 2015; Wagner and Ruhe, 2018).

Submission Acceptance Rate. This metric evaluates the success rate of code submissions through pull requests (PRs), similar to a product passing rate in manufacturing. It is defined as the ratio of PRs that are successfully merged in month t to the total number of PRs opened in the same month. Both the numerator and denominator are therefore restricted to events that occur during month t. Developers create branches to isolate development work without affecting others and then use PRs to merge those branches once complete.

A PR can be rejected when merge conflicts arise, for example when both the PR and the target branch modify the same part of a file differently and GitHub cannot automatically decide which version to keep. PRs may also fail if developers do not follow branch protection rules or if their submissions do not pass required checks. Reviewers can reject PRs due to issues with code quality, design, or functionality, requiring developers to revise and resubmit. A low submission acceptance rate indicates that teams frequently encounter conflicts or require substantial rework before merging code. Figure E4 shows examples of successful and failed PRs.

Problem-Solving Speed. This metric captures the duration between the creation of an issue on GitHub and its initial closure. Users create issues to raise questions or report problems, which are then addressed and closed by team members. Once resolved, the issue is closed, indicating successful problem resolution. This metric is constructed by tracking issues opened within a given month and recording the time to their first closure. It serves as an indicator of the team's efficiency in handling issues. Figure E5 provides an example of one issue from opening to closing.

Descriptive statistics are provided in Table 4.

[Table 4 about here.]

4 Stylized Facts

This section presents stylized facts about team specialization and productivity.

Fact 1 *Teams become slightly more specialized over time*

Figure 4 shows a binned scatter plot of team specialization over time. I only include teams active for at least 10 months and calculate their average specialization over that period. I also control for 10 task distribution [12]and team size to avoid mechanical effects from new members or changing tasks. Over 10 months, the average team specialization index increases from 0.575 to 0.595, a 2 percentage point rise. This suggests that most of the variation in specialization is due to differences across teams, not time trends.

[Figure 4 about here.]

Fact 2 *Higher team specialization is associated with lower output quality.*

Figure 5a presents a binned scatter plot showing the relationship between organizational specialization and output quality, measured by the log of monthly GitHub stars. To address potential omitted variable bias, I control for several confounding factors. First, I include task demand distribution to account for variation in project scope and complexity. Second, I add fixed effects for team age to capture lifecycle dynamics that may jointly influence specialization and output quality.[13] Third, I include fixed effects for teams, members, and team size.

After accounting for these factors, the results indicate a negative relationship between team specialization and project quality: teams with more specialized structures exhibit lower quality.

[Figure 5 about here.]

Table 5 reports results from OLS and Poisson pseudo-maximum likelihood (PPML) regressions. In the OLS models, where the dependent variable is log monthly stars, team specialization (SPE) is negatively associated with output quality across all specifications. Column (1) shows that a one standard deviation increase in specialization (0.26 points) is associated with a 7.98% decrease in monthly stars. The estimated effect decreases to 2.21% in Column (4) after controlling for team and member fixed effects, but remains statistically significant. The decline in the coefficient after adding team fixed effects indicates that most of the raw negative correlation is explained by differences across teams. However, the persistent negative effect even

[12]Task types are defined based on code file names using a topic classification model. See Section 2.3 for details.

[13]For example, teams in active development may attract more user attention and adopt different task structures than those in maintenance.

after controlling for both team and member fixed effects suggests that within-team increases in specialization over time are still associated with lower output quality.

In the Poisson models, where the dependent variable is the count of monthly stars, the negative relationship between specialization and output quality is pronounced. Column (1) shows that a one standard deviation increase in the specialization index is associated with a 13.0% decrease in expected monthly stars. After controlling for team and member fixed effects, the effect remains statistically significant: a one standard deviation increase in specialization is still associated with a 4.3% decrease in output quality.

[Table 5 about here.]

Fact 3 *Higher team specialization corresponds to lower output quantity.*

Figure 5b illustrates the relationship between team specialization and the number of lines of code produced by team members each month. The findings align with the specification in Figure 5a, revealing that more specialized teams tend to produce less coding output.

Table 6 reports the regression results. In the OLS models, where the dependent variable is the log of lines of code, team specialization (SPE) is negatively associated with output quantity across all specifications. Column (4) shows that a one standard deviation increase in specialization (0.26 points) is associated with a 24.3% decrease in lines of code. The Poisson models yield consistent results. Column (4) indicates that a one standard deviation increase in specialization corresponds to a 15.8% decrease in the expected number of lines of code.

[Table 6 about here.]

Fact 4 *Specialized teams achieve higher acceptance rates but take longer to resolve user issues.*

Figures 6a and 6b illustrate the trade-off between task-specific and general human capital. Unlike Figure 5a, these analyses include additional fixed effects: the number of code submissions (for acceptance rates) and the number of questions (for resolution time). This adjustment ensures that the relationships are not driven by differences in coding activity or inquiry frequency.

Figure 6a shows that code acceptance rates increase with team specialization. This may reflect better code from focused work—or that others are too unfamiliar with the code to evaluate and reject it. In contrast, Figure 6b shows that specialized teams take longer to resolve user issues. This delay likely occurs because effective problem-solving often requires knowledge from multiple areas, such as system design, user interface, and data management.

Figure 6c examines the relationship between team specialization and level of discussion. Specialized teams tend to exchange fewer comments, suggesting lower levels of internal communication and knowledge sharing. This pattern supports the idea that while specialization builds task-specific skills, it may reduce cross-task knowledge and limit diffusion within the team.

[Figure 6 about here.]

Table 7 and Table 8 present the regression results. In Column (4), a one standard deviation increase in specialization is associated with a 0.39 percentage point increase in the code acceptance rate, equivalent to a 0.47% increase relative to the mean acceptance rate of 82.4%. However, it also leads to a 1.08-day delay in problem resolution time. These results suggest that while specialization may improve code quality, it slows down problem-solving.

[Table 7 about here.]

[Table 8 about here.]

Fact 5 *Team members have similar demographic compositions regardless of team specialization.*

To measure diversity, I follow the literature on ecological diversity and use the Shannon - Wiener Index to quantify team composition diversity. For example, the Gender Diversity Index is calculated as:

$$H' = -\sum_{i=1}^{S} p_i \ln(p_i),$$

where H' is the gender diversity index, S is the total number of gender categories, and p_i is the proportion of individuals in the i-th category within a team. This index captures both the abundance and evenness of gender representation. A higher value indicates a more balanced gender composition (Krebs, 1989). $S = 2$ (male and female), and developers whose gender cannot be classified with high confidence are excluded from the team count.

Figure 7 illustrates the relationship between organizational specialization and diversity across two dimensions: gender and race. There is little difference in gender and race diversity among teams with different levels of specialization. Teams with high specialization and those with low specialization tend to have a similar demographic composition of people.

Gender Diversity Index.

[Figure 7 about here.]

5 Empirical Strategies

Descriptive evidence shows a negative correlation between specialization and team productivity. However, task allocation is often endogenous: managers may change team structures in response to performance problems or shifts in workload. To address this concern, I exploit

a platform feature that automatically distributes tasks among eligible team members. This assignment reduces task allocation specialization by assigning work more evenly. I implement a matched difference-in-differences design, comparing teams that adopt this feature (treatment) with similar teams that do not (control). Matching on pre-treatment characteristics reduces selection bias and improves balance, enabling a more credible estimate of the effect of specialization on team productivity.

5.1 GitHub Automatic Assignment Feature

I exploit GitHub's auto-assignment feature, which introduces exogenous variation in team specialization by randomly distributing tasks among eligible team members. This system automatically assigns contributors to both PRs (for code review) and Issues (such as bug reports or feature requests), replacing manual assignment with randomized allocation. These activities enter the A_m matrix used to compute the specialization index SPE_m. Unlike traditional workflows, where some members repeatedly handle specific types of tasks, this feature ensures a more balanced workload and uniform task exposure. Adopting the feature shifts teams toward a more generalized task structure, reducing persistent specialization.

Teams enable this feature via configuration files, as illustrated in Figure E6. Table 9 lists these files and explains their functions.

[Table 9 about here.]

Figure 8 compares average team characteristics by adoption status. Teams that adopt the auto-assignment feature are significantly larger (mean size 5.47 vs. 3.96) and more active across all metrics. They handle more tasks (4.41 vs. 3.46), submit more pull requests (25.62 vs. 15.38), contribute more code (30,076.81 vs. 24,055.74 lines), and attract more attention from users, as indicated by higher monthly stars (44.54 vs. 15.92). They also engage in more platform activities (610.83 vs. 312.60 actions). These patterns suggest that adoption is concentrated among larger and more collaborative teams, potentially to support scalable and balanced workflows.

[Figure 8 about here.]

5.2 Matched Sampling Procedure to Select Comparison Group

Given substantial baseline differences across teams, a key challenge is to identify an appropriate comparison group for teams that adopt the auto-assignment feature. I address this by using a matched sampling procedure to construct a control group of placebo-adopting teams—teams that did not adopt the feature but have lagged characteristics similar to those of one of the treatment group teams with adoption. This procedure is similar to Jäger et al. (2024).

I let t denote calendar time, d the event time (i.e., the month of adoption), and $k = t - d$ the number of months relative to the event. For each event month d from 2017 to 2023 (For example, 2017-12), I identify the set of teams that first adopt the auto-assignment feature, defined as the first appearance of an automation-related configuration file described in Table 9. Those teams are in the treatment group. For each team m that adopted the feature in d, I record a rich set of baseline characteristics from $d-5$ to $d-1$. For each event month d, the comparison group is sampled from the set of teams that never adopted the feature during the whole sample period. I can also record baseline characteristics in from $d-5$ to $d-1$ for this comparison group pool.

I implement a matched sampling procedure separately for each event month d. For each treated team m, I select a team from the comparison group pool with similar lagged characteristics. Specifically, I compute the mean of each team's characteristics over this window and then match on deciles of average team size, task types, and output levels. Rather than matching on deciles in each period, I match on the mean value of each variable from $d-5$ to $d-1$. This strategy captures persistent team characteristics while increasing the number of matched pairs. When no exact match can be found, the adopted team m will not be included in the sample. When multiple potential matches for an adopted team m are available, I randomly select one team as the matched control group.

I successfully match 3,864 treated teams to control teams, achieving a 98.2% matching success rate. Table D1 reports summary statistics for both groups at the time of auto-assignment adoption. The two groups are well balanced across key team and performance characteristics, including variables not directly used in the matching procedure.

5.3 Dynamic Effects

Building on the identification strategy in Sharoni (2024); Jäger et al. (2024), I estimate the effect of automatic task assignment based on the following dynamic difference-in-differences framework. This stacking method follows Roth (2022) to ensure a robust estimation of pre-trends. I then estimate equation (1),

$$\begin{aligned} Y_{mdt} - Y_{md,d-1} = &\sum_{\substack{k=-5 \\ k\neq -1}}^{6} \delta_k \times 1(t - d_{md} = k) + \sum_{\substack{k=-5 \\ k\neq -1}}^{6} \beta_k \times 1(t - d_{md} = k) \times \text{Treated}_{md} \\ &+ \lambda_{f(m)} + \theta_{tsize_{mt}} + \mathbf{X}'_{mt}\Gamma + \varepsilon_{mdt} \end{aligned} \tag{1}$$

where Y_{mdt} is the outcome of interest for team m observed in month t in which the team adopted (or matched a placebo team) occurring in month d. The model includes fixed effects for relative time δ_k. The model in (1) does not include calendar year fixed effects, as calendar

time is balanced between the comparison and treatment group as a consequence of the matched sampling procedure. Treated_{md} is an indicator function for treatment status, whether team m actually adopted the feature in month d or was chosen as a matched control. I also include firm fixed effect $\lambda_{f(m)}$, team size fixed effect $\theta_{tsize_{mt}}$ and control for the 10 task demand $\mathbf{X}_{mt}$.

The coefficients of interest β_k, capture the effect of an actual automatic assignment feature adoption in time $k = t - d$ in the treatment group and are normalized to zero in $k = -1$ given the difference specification. Standard error is clustered at the matching-pair level, as suggested by Abadie and Spiess (2022) to account for correlated shocks within observables used in the matching.

Figure 9 reports the point estimates from equation (1). No pre-trends are found. It also reveals that the specialization decreases when teams adopt the automatic task assignment features. The impact on team performance varies across different outcomes, consistent with the descriptive patterns in Section 4. Compared to the matched control group, treated teams exhibit higher productivity, as reflected in both output quality and quantity. At the same time, teams that adopt automatic assignment experience lower code acceptance rates and increased discussion activity.

[Figure 9 about here.]

5.4 Baseline Regression

As a way of summarizing the results, I use a second specification that directly compares the pre- and post-adoption periods, rather than tracing dynamic effects over time. In this alternative approach, the treatment indicator T_m turns to one after the time of the real or placebo adoption. This allows for a more concise assessment of the impact without focusing on the dynamic changes over time. This specification is given by

$$\begin{aligned} Y_{mdt} - Y_{md,d-1} =& \delta \times 1(t - d_{md} \geq 0) + \beta \times 1(t - d_{md} \geq 0) \times \text{Treated}_{md} \\ &+ \lambda_{f(m)} + \theta_{tsize_{mt}} + \mathbf{X}'_{mt}\Gamma + \varepsilon_{mdt} \end{aligned} \tag{2}$$

I include firm fixed effect $\lambda_{f(m)}$, team size fixed effect $\theta_{tsize_{mt}}$ and control for the 10 task demand $\mathbf{X}_{mt}$. The standard error is clustered at the matching-pair level.

Table 10 reports the regression results. Column (1) shows a negative and statistically significant coefficient on the specialization index ($\beta = -0.01$), indicating that teams become less specialized following the adoption of the auto-assignment feature. Columns (2) and (3) show that, relative to placebo teams, treated teams experience a 4% increase in output quality and a 21% increase in output quantity—both statistically significant and suggesting substantial productivity gains. Consistent with Fact 4, the code acceptance rate decreases 1.2% (from a

baseline of 0.83). The effect on problem-solving speed is statistically insignificant. Finally, Column (6) shows that team discussion increases significantly, supporting the interpretation from Figure 6c that generalized teams engage in broader communication.

[Table 10 about here.]

6 Conclusion

This paper investigates how specialization in team task allocation affects productivity and knowledge sharing in software development. Although the traditional view, dating back to Smith (1776), argues that specialization raises productivity, my findings reveal a negative relationship between team specialization and productivity. Excessive specialization appears to inhibit the general human capital needed for innovative work, where next steps are uncertain and cross-task feedback is critical.

To address endogeneity in task allocation, I exploit GitHub's automatic task assignment feature, which distributes tasks evenly among team members and reduces specialization. A matched difference-in-differences analysis shows that teams become more productive when they become less specialized.

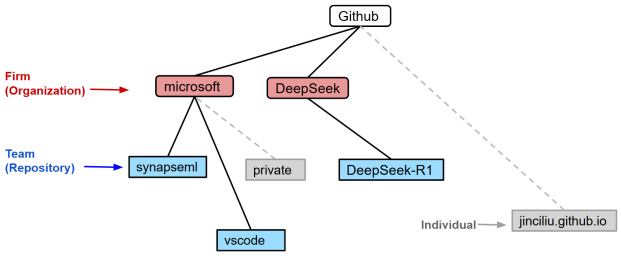
This study also develops new empirical measures for specialization and productivity. I construct a dataset linking team task allocations—based on the code developers produce—to various productivity metrics. The specialization measure captures deviations from fully generalized or fully specialized benchmarks, allowing for comparisons across teams and over time. The productivity measures capture both output quantity (lines of code) and quality (project popularity), along with finer indicators such as code acceptance rates and problem-solving speed. These measures provide a comprehensive view of how specialization shapes innovation.

My result also highlights the organizational design in innovative industries. While earlier research emphasizes the benefits of specialization through task-specific human capital accumulation, I document an important trade-off: excessive specialization can limit the general skills critical for complex problem-solving and knowledge integration. This distinction matters as economies increasingly shift toward knowledge work.

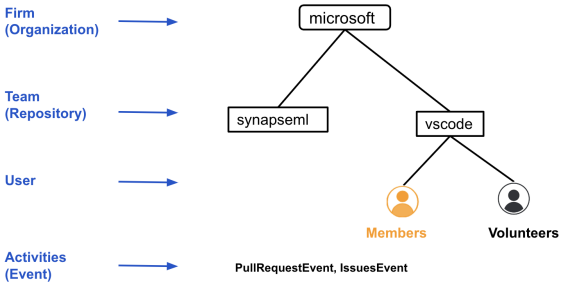
Finally, the results contribute to the broader literature on firm organization and productivity variation. Persistent differences in organizational structures may reflect not only coordination costs or information frictions but also differences in optimal specialization, depending on the production environment. Future work could extend the measurement framework developed here to study these trade-offs across industries and types of knowledge work.

Figure 1: Schematic Representation of GitHub Structure

(a) Organization and Team



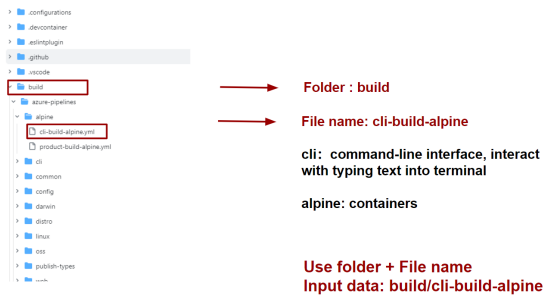
(b) Team Member and Volunteers



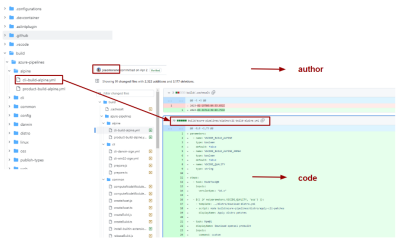
Notes: As shown in Figure 1a, some repositories belong to the organization, while others do not. In this project, I include only repositories affiliated with the organization to exclude those intended for individual purposes or hobbies, such as personal websites. Figure 1b illustrates the structure within a single organization. For instance, within Microsoft, there are multiple teams, each with various users. Some users are official team members (see Section 2.1.2), while others are volunteers who also participate and contribute. This project focuses solely on the activities of team members.

Figure 2: Code File and Task-Member Snapshot

(a) Code File Example



(b) Task-Member Example

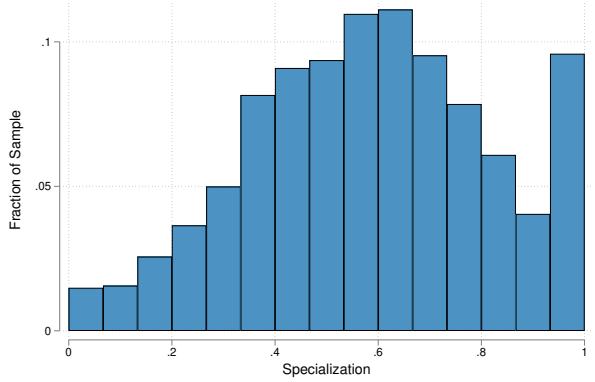


Team	Month	Member	Task Type	Labor Share
vscode	2021-05	joaomoreno	1	0.714286
vscode	2021-05	joaomoreno	4	0.285714
vscode	2021-05	isidorn	4	0.018809
vscode	2021-05	...	...	...

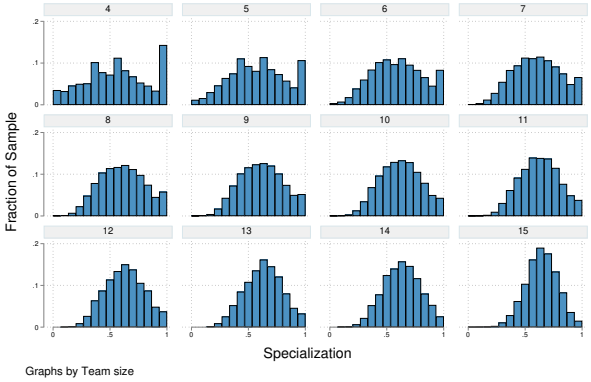
Notes: Figure 2a illustrates an example of code file names used for task type classification. The input data consists of the folder name combined with the file name, such as "build/cli-build-alpine" in the example above. Figure 2b presents a data snapshot of Member "joaomoreno" from Team "vscode" highlighting their labor share in task types 4 and 1.

Figure 3: Distribution of Team Specialization Index

(a) Specialization

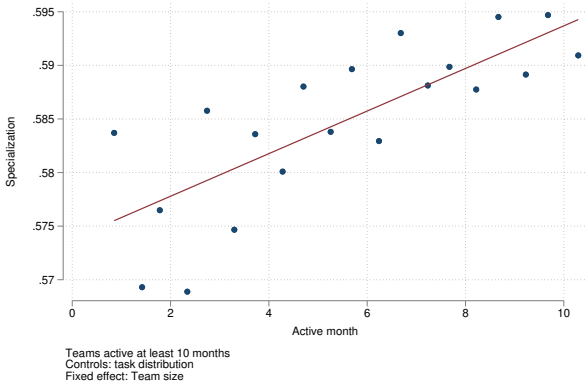


(b) By Team Size



Notes: I include team-month observations with more than 3 members and more than 1 task type in a given month. A higher index value reflects a greater degree of specialization. FigureD1 provides more details about the distribution of the team specialization index.

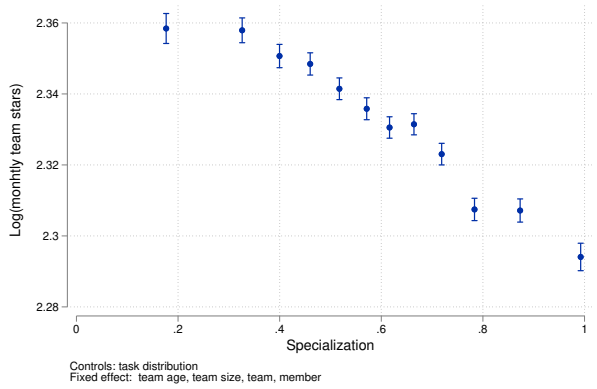
Figure 4: Team Specialization Over Time



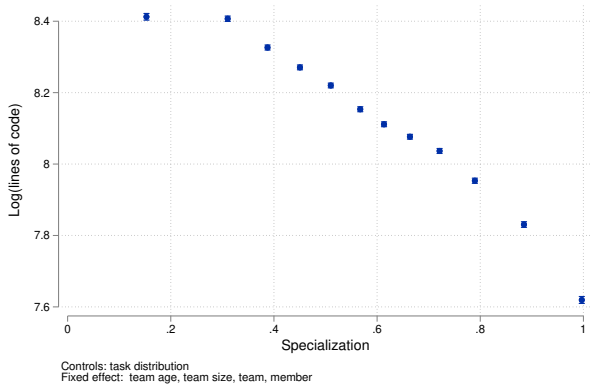
Notes: The figure shows a binned scatter plot illustrating the evolution of team specialization over time, specifically for teams that are actisumve for at least 10 months, while controlling for task distribution and team size fixed effect.

Figure 5: Team Specialization and Output Quality and Quantity

(a) Output Quality



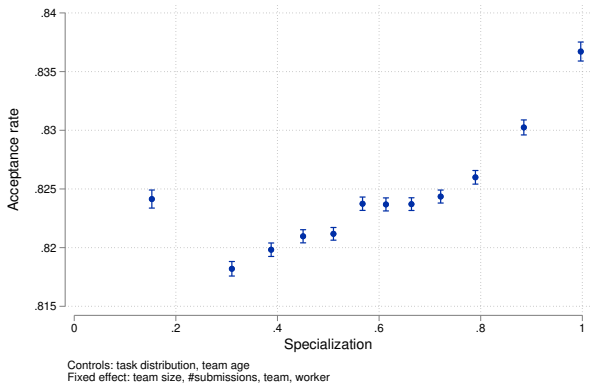
(b) Output Quantity



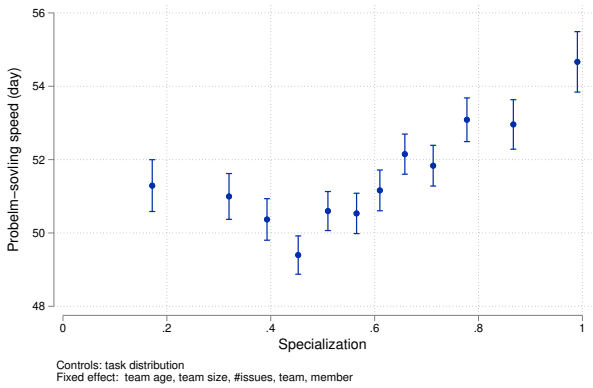
Notes: This figure presents a binned scatter plot, accounting for task distribution and team age. The analysis also includes team size, member fixed effects, and team fixed effects. These fixed effects help control for differences in member composition, team size, and the impact of team project types on team productivity. Observations are at the individual level and are weighted by 1/team size. The graph highlights a stronger negative relationship between specialization and output quality (panel (a)) and output quantity (panel (b)).

Figure 6: Team Specialization Trade-off

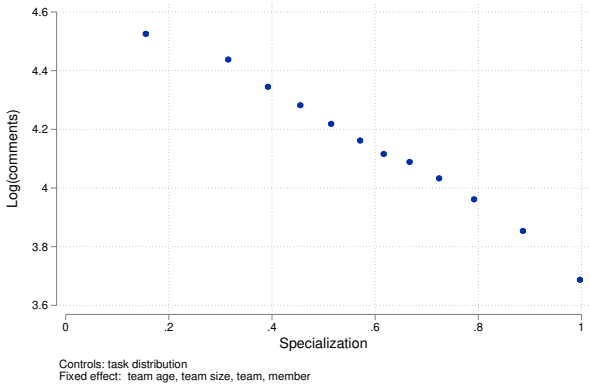
(a) Code Acceptance Rate



(b) Problem Solving Speed



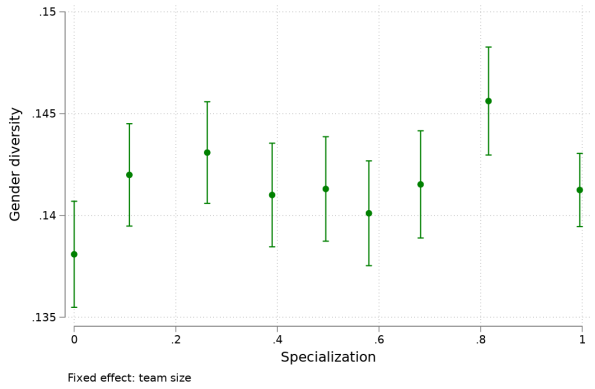
(c) Discussion



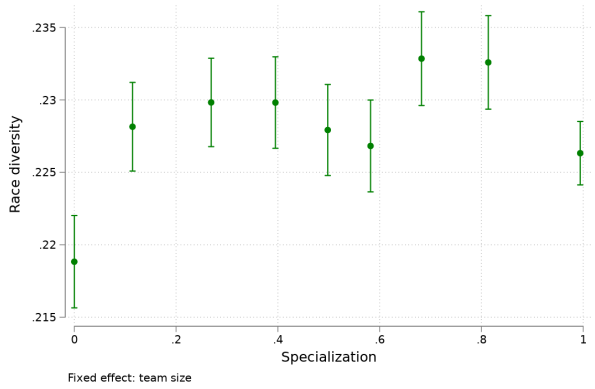
Notes: This figure shows the correlation between team specialization and code acceptance rate (Panel 6a), problem-solving speed (Panel 6b), average comments per question (Panel 6c) after controlling for task demand distribution as well as team age, team size, team, member fixed effects. The observations are at the individual level and weighted by 1/team size.

Figure 7: Team Specialization and Demographic Composition

(a) Gender

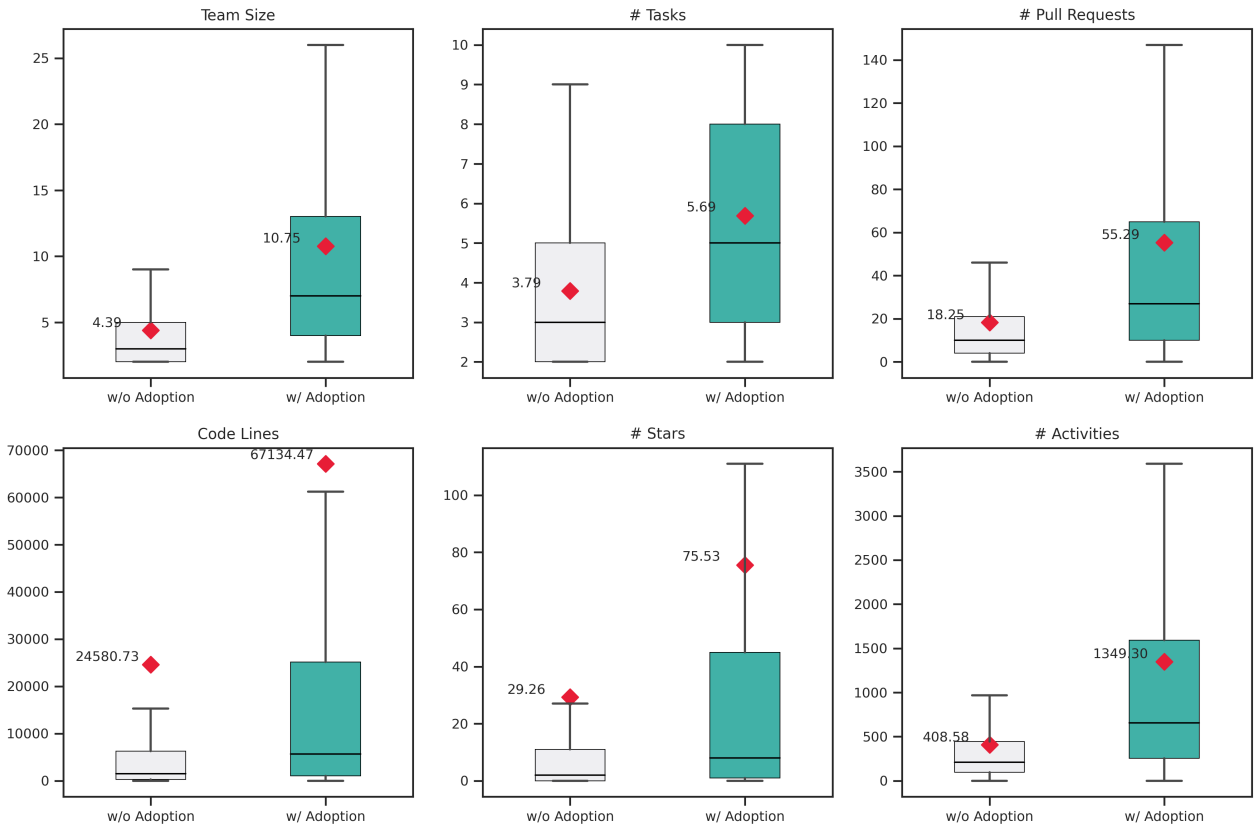


(b) Race



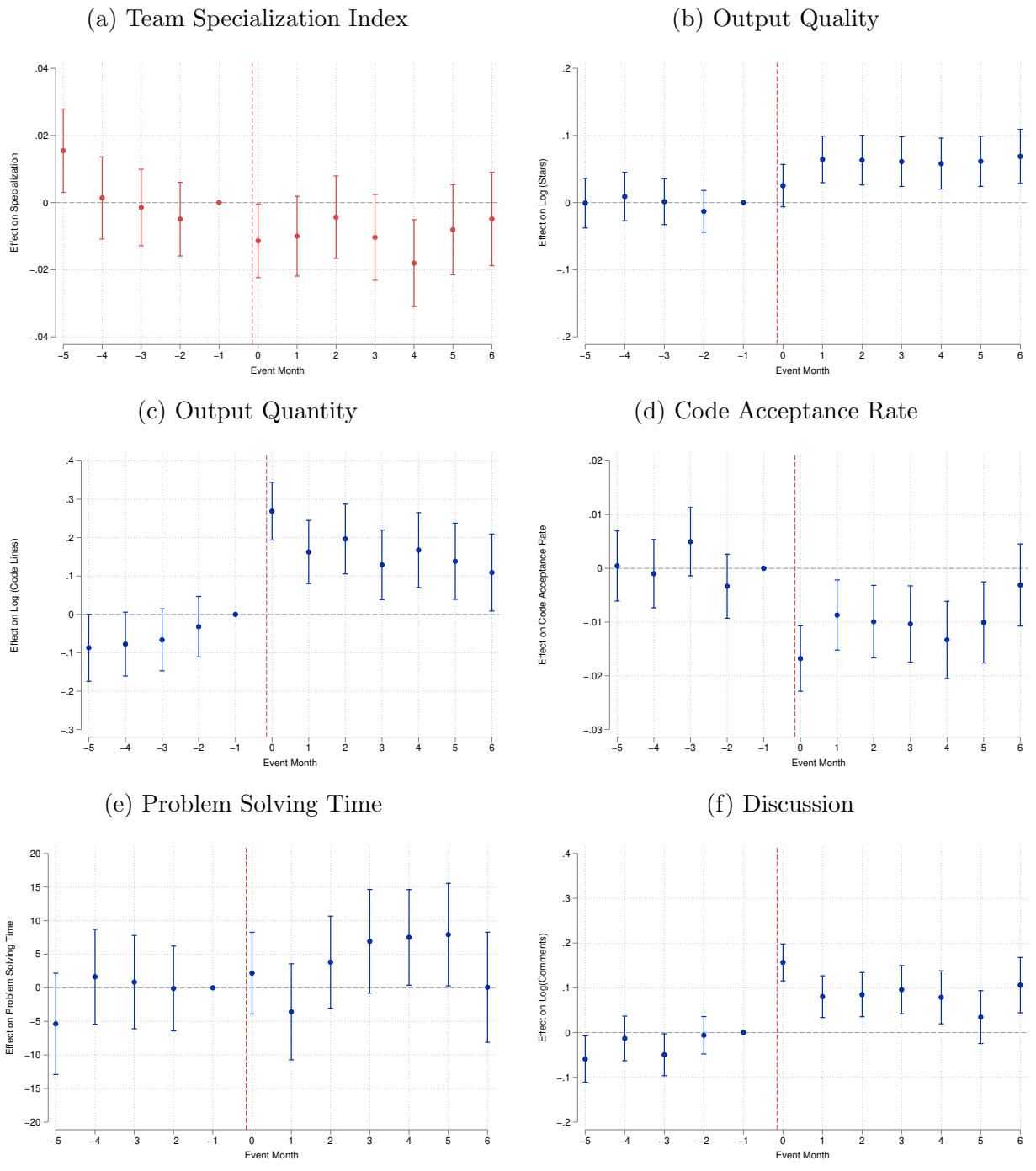
Notes: The positive relationship between organizational specialization and diversity in gender (Panel 7a), and race (Panel 7b) is presented in a binned scatter plot. I calculate each diversity index separately and exclude teams with only one member. Gender is predicted based on developers' profile images and names. Race is predicted using the name. Teams with a single task are also excluded. Section 2 provides more details.

Figure 8: Comparison of Team Characteristics by Automatic Feature Adoption



Notes: This figure compares average team characteristics between teams that adopted the auto assignment feature and those that did not, using data from 2017 to 2023.

Figure 9: Dynamic Effects for Task Assignment Adoption



Notes: This plot presents the effect of auto-assignment feature adoption on team specialization and productivity. The vertical lines represent a 90% confidence interval with standard errors clustered at the matching level. The dependent variables—team specialization index (SPE), team output quality, quantity, problem-solving time, code acceptance rate, and communication are defined in Section 3.

Table 1: Task Classification

	Task Type	Keywords (LDA result)	Occupational Role
1	Frontend development	Frontend, UI	Front-end engineer
2	Server management, Platform migration	Client, server	DevOps engineer
3	Android mobile development	Kotlin, runtime	Mobile engineer
4	Cloud feature implementation	Feature, sdk	Cloud engineer
5	Data management	Data, web	Data engineer
6	Internal system management	Internal, apache	System Administrator
7	CLI Development and Framework	User, cli	Technical Writer
8	API and Backend Services	API, controller	Back-end engineer
9	Integration system	Integration, function	System Integrator
10	App Development and UI Design	App, style	App Developer

Notes: This table shows the results of task classification using LDA on 34,762,813 code filenames from 2017 to 2023. Keywords are from the LDA model. Task summary and job role are based on keywords.

Table 2: Task Share Statistics

Task Type	N	Mean	St. Dev.	Min	P75	Max
Frontend development	1054274	0.461	0.291	0	0.711	0.998
Server management	1054274	0.056	0.143	0	0.019	0.994
Android mobile development	1054274	0.056	0.138	0	0.033	0.994
Cloud feature implementation	1054274	0.046	0.126	0	0.010	0.993
Data management	1054274	0.059	0.143	0	0.032	0.997
Internal system management	1054274	0.047	0.128	0	0.008	0.998
CLI Development and Framework	1054274	0.051	0.129	0	0.028	0.993
API and Backend Services	1054274	0.078	0.164	0	0.069	0.998
Integration system	1054274	0.055	0.135	0	0.030	0.993
App Development and UI Design	1054274	0.091	0.171	0	0.103	0.997

Notes: This table provides summary statistics for the share of labor required by each task type across all team-months from 2017-01 to 2023-12. Sample includes only teams with valid specialization index (Team size > 1 & Task types > 1).

Table 3: Task Assignment Example

Actual (A_m)

	Task 1	Task 2	Task 3	
A	1/2	1/6	1/3	1
B	1/2	1/6	1/3	1
C	1/2	1/6	1/3	1
	3/2	1/2	1	

Generalized (G_m)

	Task 1	Task 2	Task 3	
A	1/2	1/6	1/3	1
B	1/2	1/6	1/3	1
C	1/2	1/6	1/3	1
	3/2	1/2	1	

Specialized (S_m)

	Task 1	Task 2	Task 3	
A	1	0	0	1
B	1/2	1/2	0	1
C	0	0	1	1
	3/2	1/2	1	

Notes: Each row sum in the matrix represents a member's task assignment, $l_m(i)$ and each column sum is the task share $\alpha_m(j)$. Given three members (A, B, C) and three task types (1, 2, 3), Actual (A_m) shows an example of how each member allocates their labor supply across the tasks. The most generalized (G_m) assignment is when all members evenly split their labor across all tasks as defined in Definition 1, and the most specialized assignment aims to assign each member to a single task, with any remaining task share allocated to another member due to unequal task shares (S_m) as defined in Definition 2.

Table 4: Descriptive Statistics

	N	Mean	St. Dev.	Min	Median	Pctl(75)	Pctl(95)
SPE	439079	0.59	0.24	0.00	0.59	0.75	1.00
Team size	439079	7.39	8.77	4	5	7.0	17
Task type	439079	4.69	2.24	2	4	6	9
Lines of code	439079	43558.76	372784.61	0.00	3325	12926	119168
Activities	439079	750.20	2322.29	0.00	381	784	2379
Monthly Stars	439079	44.55	254.21	0.00	3	18	197
Comments	439079	144.98	362.40	0.00	54	145	533
Solving time	300034	51.18	122.35	0.00	13.86	41.36	224.82
Edited files	439079	570.47	2985.87	0.00	123	363	1917
Code acceptance rate	433833	0.82	0.17	0.00	0.86	0.94	1.00
Create year	439079	2018.14	2.82	2011	2018	2020	2022

Notes: This table provides the summary statistics for the main variables of interest at the team-month level. Data is from 2017-01 to 2023-12. Only include teams with more than 3 members. Code acceptance rate, activities, edited files, lines of code, and communication metrics are measured for team members only.

Table 5: Specialization and Team Output Quality

	(1)	(2)	(3)	(4)
OLS: Dep Var- Log(Monthly Team Stars)				
SPE	-0.320***	-0.305***	-0.078***	-0.086***
	(0.021)	(0.021)	(0.007)	(0.007)
Dependent mean	2.13	2.13	2.13	2.13
R-squared	0.648	0.650	0.903	0.910
Observations	1,823,750	1,823,750	1,823,750	1,770,310
Poisson: Dep Var- Monthly Team Stars				
SPE	-0.424***	-0.368***	-0.140***	-0.143***
	(0.060)	(0.058)	(0.042)	(0.035)
Dependent mean	44.550	44.550	44.550	44.550
Observations	3,192,051	2,968,104	3,076,921	2,996,394
10 Task type share controls	✓	✓	✓	✓
Team age FE	✓	✓	✓	✓
Team size FE	✓	✓	✓	✓
Firm FE	✓	✓		
Project type FE		✓		
Team FE			✓	✓
Member FE				✓

Notes: This table presents OLS and Poisson pseudo maximum likelihood (PPML) regression results on the impact of team specialization. The outcome variable for OLS is the log of monthly stars received by each team, while the outcome variable for Poisson is the monthly star count, which captures the extensive margin. Team age is calculated as the difference between the calendar year and the year the team was created. Observations are at the individual level and weighted by 1/team size. Standard errors are clustered at the team level. *$p < .10$, ** $p < .05$, *** $p < .01$.

Table 6: Specialization and Team Output Quantity

	(1)	(2)	(3)	(4)
OLS: Dep Var- Log(Code Lines)				
SPE	-1.224***	-1.260***	-0.954***	-0.934***
	(0.021)	(0.022)	(0.015)	(0.015)
Dependent mean	8.124	8.128	8.124	8.118
R-squared	0.398	0.403	0.597	0.616
Observations	3,212,871	2,982,107	3,212,871	3,124,751
Poisson: Dep Var - Code Lines				
SPE	-0.369***	-0.406***	-0.576***	-0.554***
	(0.108)	(0.119)	(0.072)	(0.073)
Dependent mean	43558.76	43558.76	43558.76	43558.76
Observations	3,243,291	3,000,021	3,235,747	3,147,523
Task type share control	✓	✓	✓	✓
Team age FE	✓	✓	✓	✓
Team size FE	✓	✓	✓	✓
Firm FE	✓	✓		
Project type FE		✓		
Team FE			✓	✓
Member FE				✓

Notes: This table presents OLS and Poisson pseudo-maximum likelihood (PPML) regression results on the impact of team specialization on code quantity. The outcome variable for OLS is the log of lines of code additions, while the outcome variable for Poisson models is the raw count of lines of code, which captures the extensive margin. Team age is calculated as the difference between the calendar year and the year the team was created. Observations are at the individual level and weighted by 1/team size. Standard errors are clustered at the team level. $p < .10$, ** $p < .05$, *** $p < .01$.

Table 7: Specialization and Code Acceptance Rate

	(1)	(2)	(3)	(4)
SPE	-0.010***	0.020***	0.015***	0.015***
	(0.002)	(0.001)	(0.001)	(0.001)
Task type share control	✓	✓	✓	✓
Team size FE	✓	✓	✓	✓
Firm FE	✓	✓		
Project type FE		✓		
# code submission		✓	✓	✓
Team FE			✓	✓
Member FE				✓
R-squared	0.217	0.214	0.417	0.442
Dependent mean	0.824	0.825	0.824	0.824
Observations	3,213,677	2,982,846	3,213,677	3,125,570

Notes: This table presents OLS regression results examining the relationship between team specialization and code submission acceptance rate. The dependent variable is the average code submission rate by team members. Team age is calculated as the difference between the calendar year and the year the team was created. Observations are at the individual level and weighted by 1/team size. Standard errors are clustered at the team level. $p < .10$, ** $p < .05$, *** $p < .01$.

Table 8: Specialization and Problem-Solving Speed

	(1)	(2)	(3)	(4)
SPE	17.655***	11.247***	4.411**	4.155**
	(1.496)	(1.540)	(1.385)	(1.332)
Task type share control	✓	✓	✓	✓
Team age FE	✓	✓	✓	✓
Team size FE	✓	✓	✓	✓
Firm FE	✓	✓		
Project type FE		✓		
# question FE		✓	✓	✓
Team FE			✓	✓
Member FE				✓
R-squared	0.166	0.166	0.355	0.398
Dependent Mean	51.182	51.372	51.182	51.579
Observations	2,364,271	2,267,892	2,364,271	2,294,737

Notes: This table presents OLS regression results examining the relationship between team specialization and issue solving time. The dependent variable is the average time a team takes to resolve an issue raised by users. Team age is calculated as the difference between the calendar year and the year the team was created. Observations are at the individual level and weighted by 1/team size. Standard errors are clustered at the team level. $p < .10$, ** $p < .05$, *** $p < .01$.

Table 9: Configuration Files Matching Auto-Assignment Detection Pattern

File Pattern	**Purpose**
`codeowners`	Specifies default reviewers by file path. When a matching file changes, GitHub automatically requests review from listed users or teams.
`.github/auto_assign.yml` or `.github/auto-assign.yml`	Defines reviewer or assignee rules (e.g., random selection) for the auto-assign GitHub Action. Used to automate task allocation.
`.github/dependabot.yml`	Configuration for Dependabot, which automates dependency updates. Not reviewer-specific, but reflects general automation practices.
`.github/reviewer_lottery.yml` or `.github/reviewer-lottery.yml`	Specifies pools and rules for bots that randomly assign reviewers to pull requests.
`.github/assign_reviewers.yml` or `.github/assign-reviewers.yml`	Generic reviewer configuration file supporting round-robin, random, or weighted assignment logic.
`.github/workflows/assign_reviewers.yml` or `.github/workflows/assign-reviewers.yml`	Triggers assignment logic via GitHub Actions, often referencing reviewer config files.
`.github/workflows/auto_assign.yml` or `.github/workflows/auto-assign.yml`	Runs the auto-assign workflow for automated reviewer selection on pull request events.
`.github/workflows/.mergify.yml`	Configures the Mergify bot to auto-merge pull requests after reviewer approval or status checks.

Notes: This table summarizes the set of configuration files captured by the regular expression used to detect GitHub teams adopting automated task and reviewer assignment. These files reflect a combination of reviewer-focused settings and general automation workflows.

Table 10: Baseline Regression Results

	(1) SPE	(2) Output Quality	(3) Output Quantity	(4) Acceptance Rate	(5) Problem-Solving Speed	(6) Discussion
Coefficient	–0.01***	0.04***	0.21***	–0.01***	2.56	0.13***
	(0.00)	(0.01)	(0.03)	(0.00)	(2.04)	(0.02)
R^2	0.28	0.72	0.43	0.50	0.25	0.63
N	79,184	65,692	77,624	77,644	54,505	71,054
10 Task type share controls	✓	✓	✓	✓	✓	✓
Team size FE	✓	✓	✓	✓	✓	✓
Firm FE	✓	✓	✓	✓	✓	✓

Notes: Each column reports results from a separate regression of the outcome variable in a matched difference-in-differences design. The unit of observation is team-month. Standard errors are clustered at the matching level.
* $p < 0.1$; ** $p < 0.05$; *** $p < 0.01$

References

Abadie, A. and J. Spiess (2022). Robust post-matching inference. *Journal of the American Statistical Association 117*(538), 983–995.

Autor, D. H., F. Levy, and R. J. Murnane (2003). The skill content of recent technological change: An empirical exploration. *The Quarterly journal of economics 118*(4), 1279–1333.

Bassi, V., L. Jung Hyuk, A. Peter, T. Porzio, R. Sen, and E. Tugume (2023). Self-employment within the firm. *Working paper 123*, 1–18.

Baumgardner, J. R. (1988). The division of labor, local markets, and worker organization. *Journal of Political Economy 96*(3), 509–527.

Beaudry, P., D. A. Green, and B. M. Sand (2014). The declining fortunes of the young since 2000. *American Economic Review 104*(5), 381–386.

Becker, G. S. and K. M. Murphy (1992). The division of labor, coordination costs, and knowledge. *The Quarterly journal of economics 107*(4), 1137–1160.

Blei, D. M., A. Y. Ng, and M. I. Jordan (2003). Latent dirichlet allocation. *Journal of machine Learning research 3*(Jan), 993–1022.

Borges, H. and M. T. Valente (2018). What's in a github star? understanding repository starring practices in a social coding platform. *Journal of Systems and Software 146*, 112–129.

Caines, C., F. Hoffmann, and G. Kambourov (2017). Complex-task biased technological change and the labor market. *Review of Economic Dynamics 25*, 298–319.

Casalnuovo, C., B. Vasilescu, P. Devanbu, and V. Filkov (2015). Developer onboarding in github: the role of prior social links and language experience. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering*, pp. 817–828.

Deming, D. J. (2017). The growing importance of social skills in the labor market. *The Quarterly Journal of Economics 132*(4), 1593–1640.

El-Komboz, L. A. and M. Goldbeck (2024). Career concerns as public good the role of signaling for open source software development. Technical report, ifo Working Paper.

Eldeeb, Y. and A. Sikora (2023). How much are github stars worth to you? (the guild). Accessed: 2024-10-11.

Forderer, J. and G. Burtch (2024). Estimating career benefits from online community leadership: Evidence from stack exchange moderators. *Management Science*.

Garicano, L. and T. N. Hubbard (2009). Specialization, firms, and markets: The division of labor within and between law firms. *The Journal of Law, Economics, & Organization 25*(2), 339–371.

Garicano, L. and E. Rossi-Hansberg (2006). Organization and inequality in a knowledge economy. *The Quarterly journal of economics 121*(4), 1383–1435.

Gong, J. and I. P. Png (2024). Automation enables specialization: Field evidence. *Management Science 70*(3), 1580–1595.

Hayek, F. A. (1945). The use of knowledge in society. *The American Economic Review 35*(4), 519–530.

Hoffmann, M., F. Nagle, and Y. Zhou (2024). The value of open source software. *Harvard Business School Strategy Unit Working Paper* (24-038).

Holmstrom, B. (1982). Moral hazard in teams. *The Bell journal of economics*, 324–340.

Jäger, S., J. Heining, and N. Lazarus (2024). How substitutable are workers? evidence from worker deaths. Technical report, American Economic Review.

Jones, B. F. (2021). The rise of research teams: Benefits and costs in economics. *Journal of Economic Perspectives 35*(2), 191–216.

Kohlhepp, J. (2024). The inner beauty of firms.

Krebs, C. J. (1989). Ecological methodology. *(No Title)*.

Laohaprapanon, S., G. Sood, and B. Naji (2017). ethnicolr algorithm.

Lazear, E. P. (1995). Personnel economics. *Massachusetts Institute of Technology*.

Lee, D. and C. Makridis (2023). A task-interdependency model of complex collaborative work for advancing human-centered crowd work. *Available at SSRN 4472585*.

Levinthal, D. A. and J. G. March (1993). The myopia of learning. *Strategic management journal 14*(S2), 95–112.

Marschak, J. (1955). Elements for a theory of teams. *Management science 1*(2), 127–137.

Nagle, F. (2019). Open source software and firm productivity. *Management Science 65*(3), 1191–1215.

Rosen, S. (1983). Specialization and human capital. *Journal of Labor Economics 1*(1), 43–49.

Roth, J. (2022). Pretest with caution: Event-study estimates after testing for parallel trends. *American Economic Review: Insights 4*(3), 305–322.

Sharoni, B. (2024). The effect of inventor mobility on network productivity. Technical report, Working Paper. Harvard University.

Smith, A. (1776). *The wealth of nations [1776]*, Volume 11937. na.

Vasilescu, B., Y. Yu, H. Wang, P. Devanbu, and V. Filkov (2015). Quality and productivity outcomes relating to continuous integration in github. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering*, pp. 805–816.

Wagner, S. and M. Ruhe (2018). A systematic review of productivity factors in software development. *arXiv preprint arXiv:1801.06475*.

Online Appendix of:

How does the Division of Labor Affect Team Productivity? Evidence from GitHub

Jinci Liu

October 28, 2025

A Working on Model

In this theoretical section, I provide a simple framework that sets the stage for understanding the potential directions of effect and the mechanisms driving it. It is used to form predictions and to guide the empirical tests. The model incorporates two key drives of specialization on team productivity: task-specific human capital accumulation and cross-task knowledge spillover. While both parts are integral components of the team production function, the channels through which they affect team's productivity are distinct. Therefore, the model allows me to explore scenarios where the benefits of specialization outweigh the cost.

A.1 Setting

Each team produces a single product (e.g., a website) that requires a fixed set of tasks to be completed by its members. Members have no preference over tasks, and their wages are determined by the firm rather than by the team. Teams cannot choose which members to hire or which tasks they must undertake; however, they do decide how to allocate members across these tasks. That is, both the set of members and the set of tasks are exogenous, while the assignment of members to tasks is endogenous. Following Becker and Murphy (1992), assume all members are homogeneous, supply labor inelastically, and receive a constant wage w. Let $\mathbb{J}$ denote the set of all possible tasks. Each team $m \in \{1, \ldots, M\}$ consists of $N_m \in \mathbb{N}$ members and require to complete a subset of tasks $J_m \subseteq \mathbb{J}$. The task allocation within team m is represented by

$$A \in \mathbb{R}_+^{N_m \times J_m},$$

where $A(i, j)$ is the amount of labor member i allocates to task j as discussed in Section3.1. For notational simplicity, the subscript m is suppressed when no confusion arises.

Each member i's output on task j is given by

$$q(i,j) = \left[\underbrace{z(i,j)^{\frac{\sigma-1}{\sigma}}}_{\text{Knowledge Spillover}} + \underbrace{A(i,j)^{\frac{\sigma-1}{\sigma}}}_{\text{Task-specific HC}} \right]^{\frac{\sigma}{\sigma-1}} A(i,j)$$

where $\sigma > 1$ is the elasticity of substitution between knowledge spillovers and task-specific human capital.

We define

$$z(i,j) = \alpha(j) - A(i,j),$$

where $\alpha(j) = \sum_i A(i,j)$ to capture the idea that the more labor other members devote to task j, the greater the feedback and assistance (knowledge spillover) available to member i.

The task-level output is the sum of all members yields the total output for task j:

$$y(j) = \sum_{i=1}^{N} q(i,j).$$

and the team's final output is given by a production function

$$Y = F\big(y(1),\, y(2),\, \ldots,\, y(J)\big),$$

which is strictly concave and increasing in each $y(j)$.

Teams also incur a coordination cost, $c_m(A)$, that depends on how labor is allocated across tasks. I assume there is no coordination cost when the team adopts the most specialized task allocation S_m and the coordination cost grows as the distance between S and A increases, which essentially corresponds to the specialization index SPE=$\frac{d(A,G)}{d(S,G)}$.

The team maximization problem is

$$\max_{A \in \mathbb{R}_+^{N \times K}} Y - wN - c_m(A) \tag{3}$$

s.t. $\sum_i A(i,j) = \alpha_j \ \forall j$ and $\sum_j A(i,j) = 1$

A.2 Simple Example

Assume $\sigma = -\infty$ and $c = 0$ and there are only two members and two tasks with equally task share α_j.

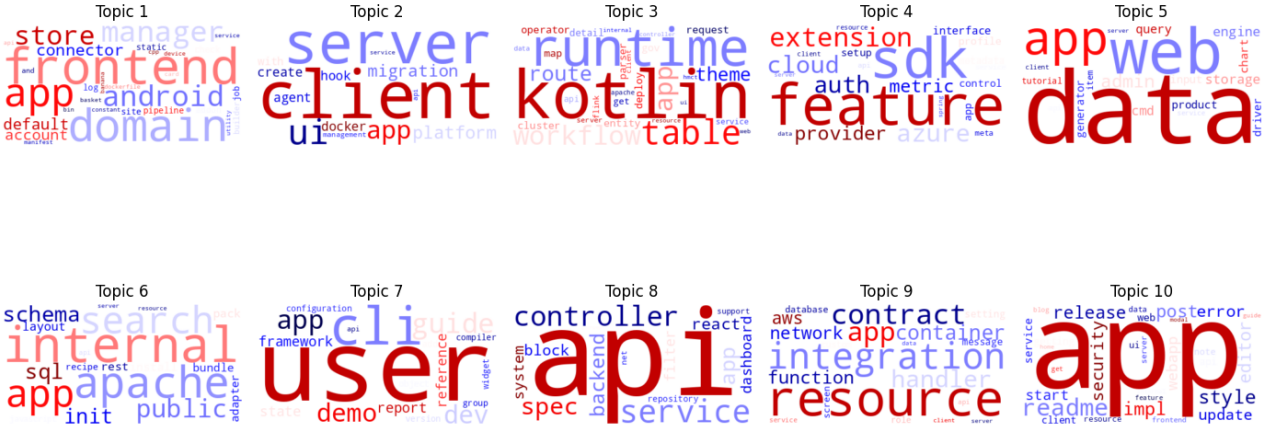
$$q(i,j) = \min\{1 - A(i,j), A(i,j)\}\, A(i,j)$$

$$y(j) = \min\{1 - A(1,j), A(1,j)\} A(1,j) + \min\{A(1,j), 1 - A(1,j)\} (1 - A(1,j)$$

$$A(1,1)^* = A(2,1)^* = 1/2$$

B Task Classification Process

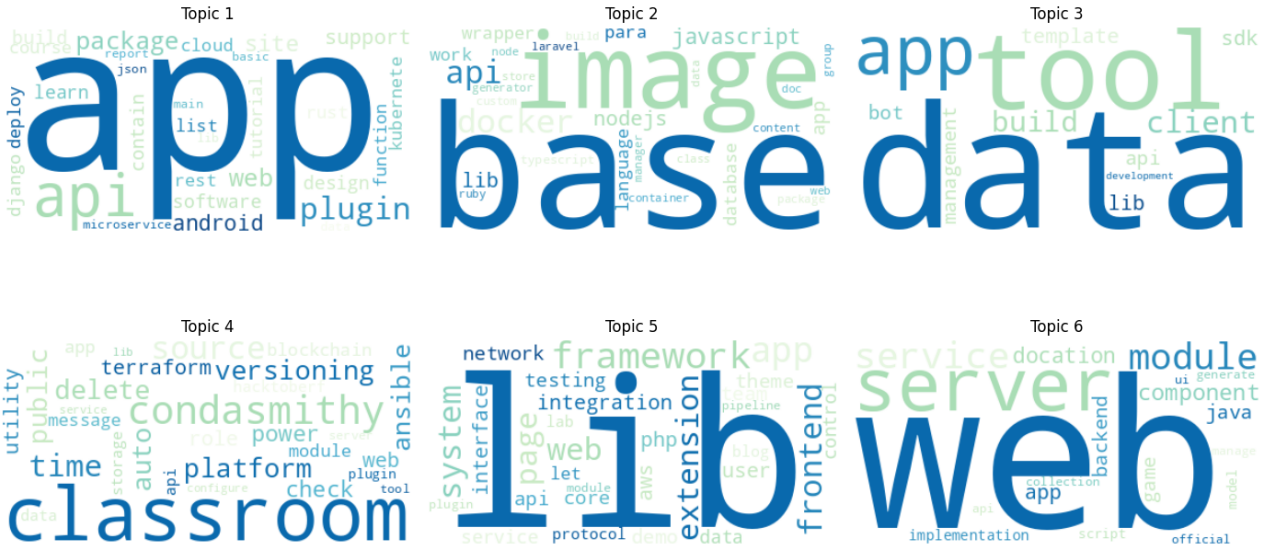
Figure B1: Word Clouds for LDA 10 Task Types



Notes: This figure presents word clouds visualizing the 10 task types identified through Latent Dirichlet Allocation (LDA) topic modeling. Each panel shows frequently occurring terms within different development domains: data management (Topic 5), server-side development (Topic 2), runtime environments (Topic 3), client applications (Topic 7), API development (Topic 8), and web applications (Topic 10). The size of each word represents its frequency within the topic, with terms like "data," "api," "user," and "app" being particularly prominent. This classification helps measure team specialization by categorizing tasks into distinct technical domains. Topics reflect common software development tasks and technological components.

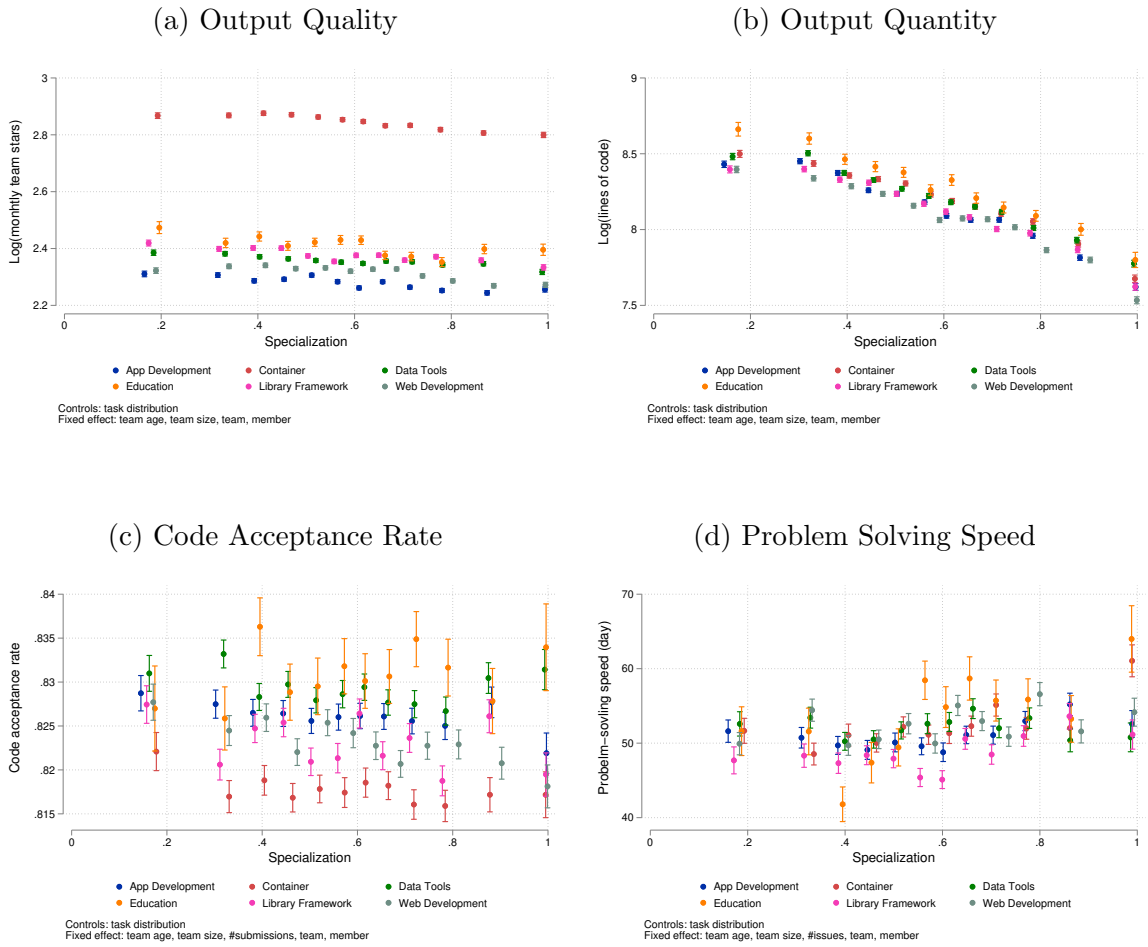
C Project Type

Figure C1: Word Clouds for LDA 6 Project Types



Notes: This figure presents the six project types identified through Latent Dirichlet Allocation (LDA) topic modeling. Each panel represents a distinct project category: **App Development** (Topic 1, dominated by "app"), **Container** (Topic 2, centered on "base"), **Data Tools** (Topic 3, highlighted by "tool"), **Education** (Topic 4, featuring "classroom"), **Library&Framework** (Topic 5, showing "lib"), and **Web Development** (Topic 6, emphasized by "web"). The word clouds display terms scaled by their frequency within each project type, helping classify repositories into broad development domains. Project type classification helps control for heterogeneity across different software development contexts when analyzing the relationship between specialization and productivity. LDA analysis performed on repository metadata, README files, and project descriptions.

Figure C2: Team Specialization and Productivity by Project Type



Notes: This figure presents the team specialization and productivity metrics by project types, accounting for task distribution and team age. The analysis also includes team size, member fixed effects, and team fixed effects. These fixed effects help control for differences in worker composition, team size, and the impact of team project types on team productivity. Observations are at the individual level and are weighted by 1/team size.

D Additional Results

Table D1 presents summary statistics for treatment and control teams at the time of auto-assignment adoption. The two groups are well-balanced across key team and performance characteristics, consistent with the matching design. Treatment teams have an average team size of 6.16, compared to 6.40 for control teams. They perform a similar number of task types (5.09 vs. 5.06) and have comparable specialization indices (0.53 vs. 0.54). Code acceptance rates are nearly identical across groups at 0.83 and 0.82, respectively. Differences in lines of code (43,218 vs. 41,264) and problem-solving time (51.92 vs. 53.01 days) are small and statistically insignificant. Although treatment teams receive fewer monthly stars on average

(36.01 vs. 59.98), this variable is not a matching criterion and may reflect idiosyncratic project visibility. Overall, the balance across core attributes—team composition, specialization, and productivity—reinforces the credibility of the matched sample and supports the identification strategy.

Table D1: Balance Table for Even Task Assignment Adoption

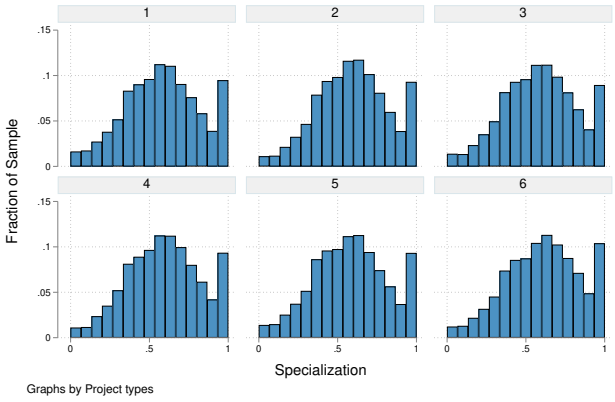
Team size	6.16	32052	6.40	57584	-0.25
					(0.04)
Task types	5.09	32052	5.06	57584	0.03
					(0.02)
Lines of code	43217.96	32212	41264.16	61160	1953.79
					(2705.34)
Monthly stars	36.01	32212	59.98	61160	-23.97
					(1.29)
Specialization index	0.53	32052	0.54	57584	-0.01
					(0.00)
Code acceptance	0.83	31745	0.82	56693	0.01
					(0.00)
Problem-solving time	51.92	21512	53.01	45397	-1.09
					(0.94)
Create year	2017.12	32212	2018.13	61160	-1.01
					(0.02)

Notes: The table presents summary statistics for both the control and treatment groups at event month $k = 0$, each consisting of 3,864 teams. The treatment group includes teams that adopted even task assignments in event month t. Notably, I only matched on decile of average team size, task type, and lines of code but did not specifically match the characteristics of teams.

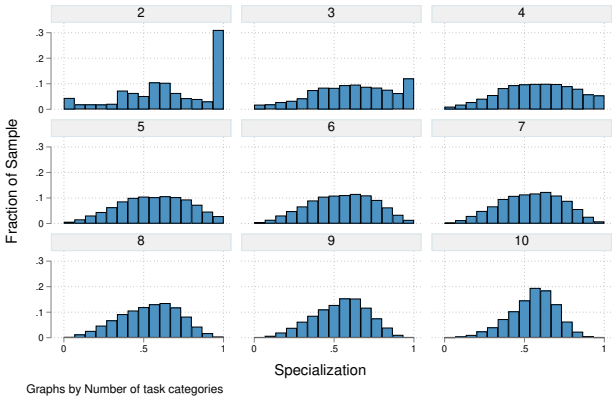
E Examples

Figure D1: Distribution of Team Specialization Index

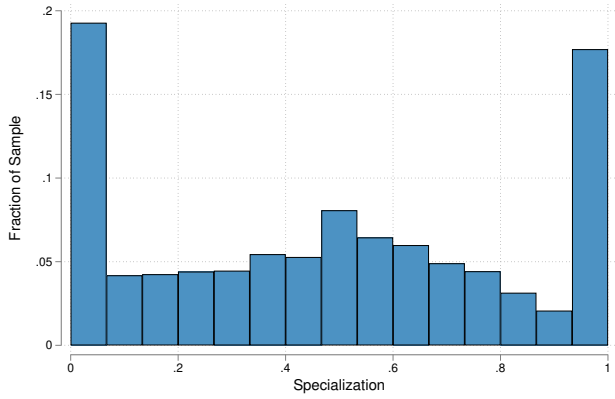
(a) By Project Type



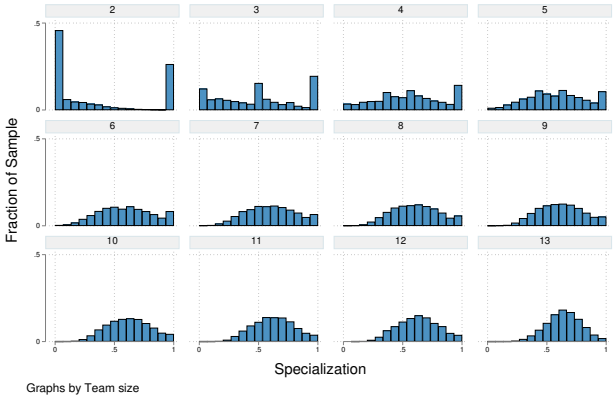
(b) By Number of Task Type



(c) Include Small Teams

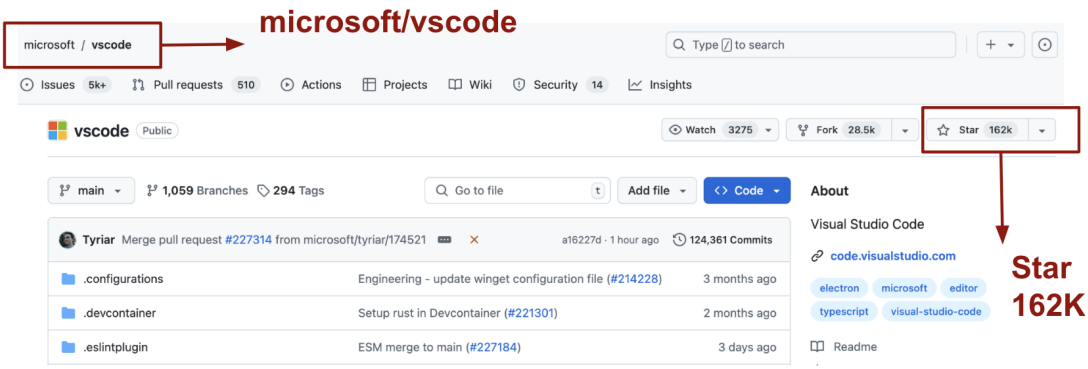


(d) By Team Size



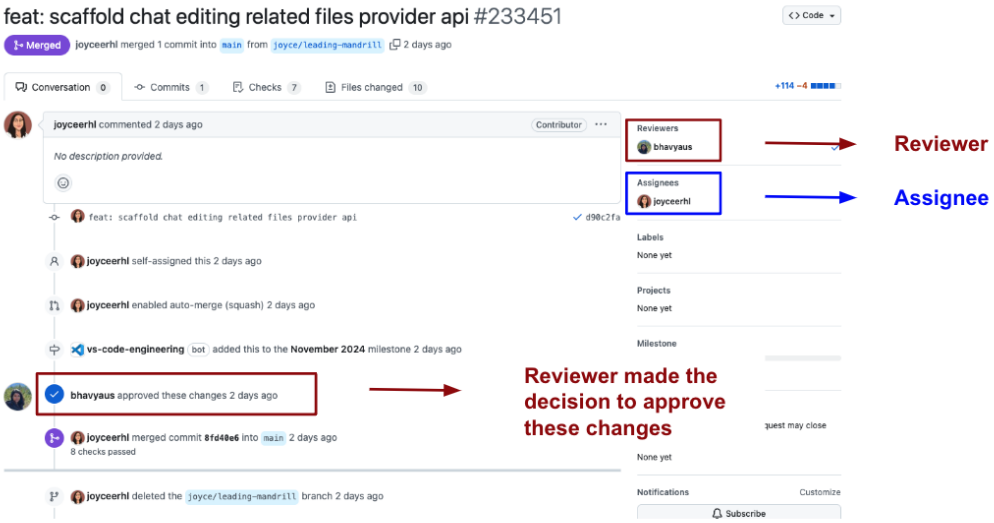
Notes: This figure illustrates the distribution of the team specialization index across various dimensions. Figures D1a and D1b display the distribution by project type and number of tasks for all team-month observations, excluding teams with fewer than four workers. Additional details on project types and task types are provided in Sections 2.4 and 2.3. Figures D1c and D1d also present the distribution for small teams consisting of 2 or 3 workers. These smaller teams are excluded from the main analysis because their limited size restricts their ability to vary their degree of specialization, resulting in most being either highly specialized or generalized.

Figure E1: Star Example



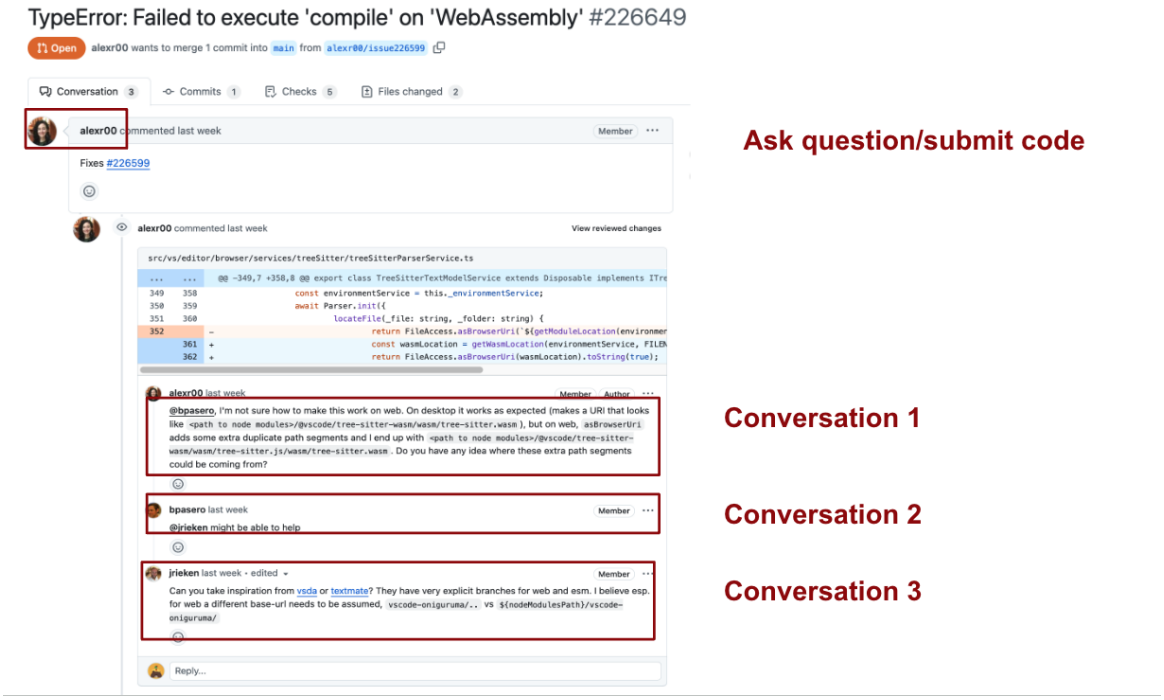
Notes: This figure illustrates how project popularity is measured using GitHub's star system, using Microsoft's Visual Studio Code repository as an example. Stars represent a form of peer evaluation where users can bookmark and show appreciation for repositories they find valuable. In this example, Visual Studio Code has received 162,000 stars, indicating its high popularity within the developer community. This metric serves as one of our key measures of team productivity, as it captures the project's impact and perceived value among potential users and other developers. While not a perfect measure of quality, stars provide a standardized way to compare project impact across different domains and team sizes.

Figure E2: Reviewer and Assignee



Notes: The example shows a typical Pull Request (#233451) with its associated review thread. Two distinct roles are highlighted: Reviewers (red box) who evaluate code changes and Assignees (blue box) who are responsible for implementing the changes. The bottom of the image demonstrates a completed review process where a reviewer has approved the proposed changes.

Figure E3: Discussion Example

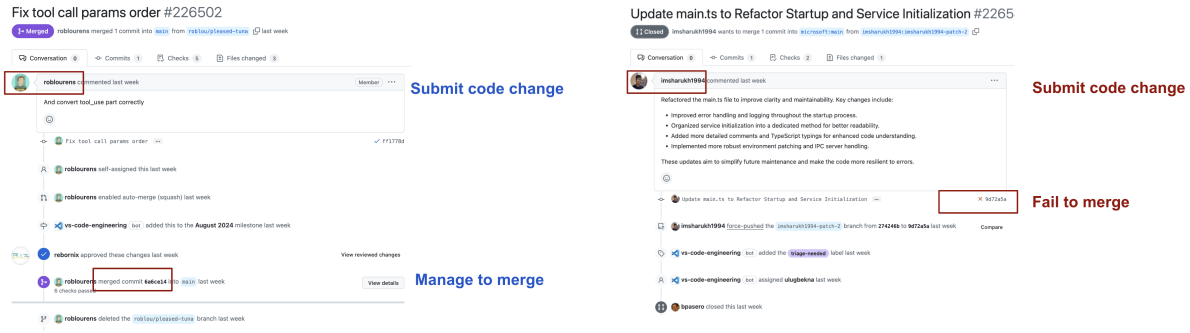


Notes: This figure demonstrates how technical discussions unfold in GitHub's issue tracking system. The example shows a TypeError issue (#226649) and the subsequent problem-solving conversation. The interaction is structured in three parts: (1) initial question/code submission, (2) technical discussions and proposed solutions, and (3) follow-up clarifications. This conversation structure allows us to measure the quantity of team communication (number of comments).

Figure E4: Pull Request Merge Example

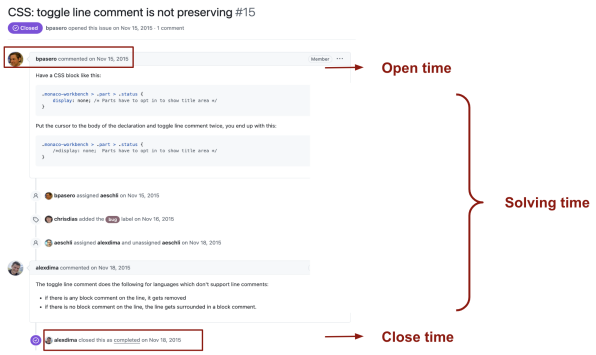
(a) Manage to merge code

(b) Fail to merge code



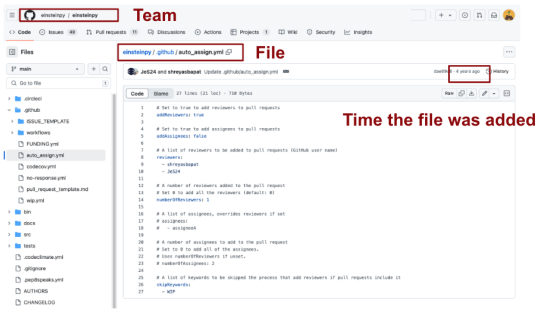
Notes: This figure illustrates two outcomes in the code review process: successful and failed merges. Panel (a) shows a successful merge (#226502) where submitted code changes are approved and integrated into the main codebase. Panel (b) demonstrates a failed merge (#2265) where proposed changes are rejected. Code acceptance rates serve as one of our key productivity metrics.

Figure E5: Problem Solving Example



Notes: This figure demonstrates how we measure problem-solving efficiency in teams. Using issue #15 as an example, we track three key timestamps: (1) Open time - when the issue is first reported, (2) Solving time - duration of technical discussion and solution development, and (3) Close time - when the issue is resolved. The time between open and close serves as our measure of problem-solving speed.

Figure E6: Auto Assignment File Example



Notes: This figure shows the implementation of GitHub's random assignment feature through a configuration file. The left panel shows the team's repository structure, while the right panel displays the auto-assignment configuration. The timestamp of file addition marks when teams transition to more generalist task allocation, providing our second source of identification.