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

Jinci Liu *

March 9, 2025

Preliminary Draft –please do not circulate or cite without permission of the author

Abstract

Does division of labor increase team productivity? This paper provides novel evidence that challenges traditional views on specialization. Using comprehensive data from GitHub, which covers 35 million task allocations across 64,400 software development teams over seven years, I document a negative relationship between organizational specialization and various productivity metrics, including output quality, quantity, and user issue resolution time. To establish causality, I exploit the adoption of an automatic task assignment feature on GitHub that evenly distributes tasks among certain team members. Using a matching design, I find that after this feature adoption, teams become less specialized and show improvements in both output quality (measured by platform users' evaluations) and quantity (measured by lines of code), along with a 10% increase in team communication. These findings reveal a trade-off in non-routine production: while specialization increases task-specific human capital, it impedes critical cross-task knowledge spillovers crucial for innovation.

*IIES, Stockholm University. Email: jinci.liu@iies.su.se.

1 Introduction

Significant differences in productivity and organizational structure exist both between and within firms. A central question in economics is how to optimally allocate tasks among workers. The seminal insight from Smith (1776), illustrated through the pin factory example, suggests that the division of labor increases productivity by allowing workers to specialize. Subsequent theoretical work has built on this idea, showing that specialization helps to accumulate human capital and increase productivity[1].

While extensive literature builds on the theory that specialization improves productivity (Deming, 2017; Bassi et al., 2023), direct empirical evidence remains limited. The effect of specialization on team productivity is ex-ante ambiguous, particularly 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 in the returns to specialization: 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 work (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 addresses these gaps by leveraging a unique dataset from GitHub—the world's largest software collaboration platform—to analyze how task allocation specialization affects team productivity. GitHub provides a remarkably extensive dataset of 35 million code files contributed by 292,840 team members across 64,400 teams over seven years. This granular data captures individual activity, specifying who engaged in particular files and at which moments. It enables the construction of novel measures for team specialization and productivity metrics at a team-month level. Using a state-of-the-art topic classification model, millions of code files are classified into 10 distinct task types (e.g., Frontend, Backend). The actual task allocations among team members and these classified tasks are compared against hypothetical benchmarks of "most specialized" and "most generalized" structures, taking into account team size and task types. A higher specialization index indicates a team is more specialized, as it is further from the "most generalized" structure the team could achieve. Team specialization index is then linked to various productivity metrics, including output quality measuring platform users' appreciation, output quantity indicating by lines of code written by team members, problem-solving speed assessing by how quickly the team addresses user questions, and the code submission acceptance rate of members. These productive metrics reflect different dimensions of software team performance and can also be applied in other contexts.

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

To examine the impact of specialization on productivity, I explore the relationship between within-team specialization and productivity metrics. Two main findings stand out. First, teams with higher specialization produce lower output quality and quantity: a one-unit increase in the specialization index is associated with a 14.3% decline in output quality and a 55.4% decrease in output quantity. The results are robust after controlling for task types and include team size, team age, as well as team and member fixed effects to control for unobserved heterogeneity at both the team level and in project dynamics. Additionally, while specialized teams exhibit a 1.82% increase in code acceptance rates—suggesting enhanced task-specific human capital—they require 8.06% more time to resolve user-reported issues, highlighting potential challenges in problem-solving capabilities for specialized teams.

The negative relationship between specialization and productivity within team may not solely stem from specialization itself, as task allocation is often endogenous. Managers may adopt specialized structures in response to poor performance. To address this concern, I exploit GitHub's auto-assignment feature, which evenly distributes tasks among some team members. Using a matched difference-in-differences design, I compare teams that adopt automatic task assignment (treatment) to those that do not (control), ensuring both groups share similar characteristics. This matching approach mitigates selection bias and pre-existing differences in specialization. I find that automatic task assignments lead to a 1.7% decrease in specialization, a 15% increase in output quality after 3 months, a 17.7% increase in output quantity, a 2.4 % decline in the code acceptance rate (from a baseline of 0.83), and a 10% increase in team communication.

This paper makes three key contributions to understanding how organizational structure influences productivity. First, this paper provides the first large-scale empirical evidence on specialization patterns in software development. While previous studies often focus on controlled experiments, such as Gong and Png (2024), which find that cashiers specializing in scanning tasks perform faster than those handling multiple functions, Kohlhepp (2024) finds a positive correlation between specialization and firm productivity in hair salon, these findings primarily apply to routine setting. In contrast, this paper investigates specialization in software development—a non-routine setting where the trade-offs of specialization are more nuanced. By analyzing detailed task-level data, this study offers a comprehensive examination of how specialization evolves, its impact on productivity, and its broader implications for the future of work.

Second, this paper provides empirical evidence for fundamental theories of the division of labor. While classical research emphasizes the productivity gains from specialization (Rosen, 1983; Baumgardner, 1988; Becker and Murphy, 1992), empirical validation has long been hampered by measurement challenges and limited data. To overcome these constraints, I exploit a

platform feature designed to evenly assign tasks among team members and apply a matched difference-in-differences approach to evaluate how specialization influences team productivity.

Third, this paper identifies a new cost of specialization—learning cost—which has critical implications for non-routine work. Both routine and non-routine production incur coordination costs[2] from specialization(Becker and Murphy, 1992; Lazear, 1995), but learning costs are especially important in non-routine contexts such as R&D, where project trajectories are uncertain and cross-task spillovers are crucial. Excessive specialization can hinder the flow of new ideas and adaptive feedback. While Garicano and Rossi-Hansberg (2006) and Garicano and Hubbard (2009) argue that organizations should structure themselves as knowledge hierarchies with specialized agents, this framework does not fully capture the needs of innovation-driven work. Unlike settings where specialists can function independently, software development requires continuous interaction and feedback across domains. This insight helps explain my finding that higher specialization correlates with slower problem-solving. As described by Levinthal and March (1993), excessive specialization can lead to organizational myopia, where teams prioritize short-term efficiency at the expense of long-term adaptability.

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 used in this paper and the different sources from which we retrieve and combine data to create the final panel. The first source is data about developer activities from the GitHub Archive, which includes timeline and event log. The second source is public profiles of GitHub users, which include their profile pictures, names, locations, email addresses, and biographies.

2.1 GitHub: World's Largest Web Platform for Software Developers

GitHub is a cloud-based platform for software developers to store, share, and collaborate on code projects. As of April 2024, GitHub reports having over 129 million users and more

[2]Becker and Murphy (1992) suggests three types of coordination costs when a team becomes more specialized: 1. Principal-agent conflicts: Conflict among members because members have greater incentives to shirk when they get a smaller share of output. 2. Hold-up problem as other members perform complementary tasks 3. Poor coordination of the tasks and functions performed by different members, or to communication of misleading information among members, also tends to expand as the number of separate specialists grows.

than 205 public million repositories [3]. Repositories (projects) [4], where source code and related documents are stored, can be owned by users or organizations. As user accounts may be either a person or an automated process (a "bot"), this study will focus on the former. Each project has only one owner, but the owner can invite other users to collaborate by offering them to become members and granting them specific permissions. As Figure 1a shows, some teams operate under organizations like Microsoft and DeepSeek, while others are independent, such as users' personal websites. This analysis focuses on organizational projects, as these are more likely to be driven by a firm's profit motives than individual hobbies.

[Figure 1 about here.]

To start with GitHub, each user must create a unique login and can provide personal information that will be displayed on their profile page. This allows users to showcase their programming skills and enhance their visibility on the platform, which can be beneficial for their careers. Many developers are motivated to share their real information on GitHub; some even provide additional information, such as links to their LinkedIn profiles and personal websites.

GitHub allows developers to save snapshots of files, thereby effectively recording the history of changes and facilitating collaborative development. Furthermore, GitHub activities are available from GHArchive, which records the public GitHub timeline and event log.

2.2 Data Construction

To construct my dataset, I combine the GitHub Archive database with developer profile information via the GitHub API. The GHArchive provides time-variant information on the events for all public projects, with a comprehensive record of the GitHub timeline since February 2011. For instance, the system records events related to modifying and suggesting code files, such as pull request. It also provides the number of lines of code contributed by each user, the time required for a pull request to be opened and closed, and the number of stars each project received from users of the universe platform.

To gather demographic information, I use the GitHub API to access each developer's profile page, which includes their profile image, location, name, email, and biography. Career factors significantly motivate developers to contribute to online platforms such as GitHub and 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 enhance their career prospects. I focus on developers who are affiliated with repositories and have permission

[3] See https://github.com/search?q=type:user and https://github.com/search?q=is:public, accessed 2024-04-30

[4] In this paper, I use "projects" and "repositories" interchangeably, though I recognize that on GitHub, "projects" might have varied meanings.

to access repository 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 GitHub Team Members

A team is defined as a project-based entity. Ideally, I prefer to focus on full-time employees within each team. However, GitHub lacks a specific indicator for such categorization. Consequently, I rely on another category related to author association: collaborator, contributor, mannequin, member, none, and owner [5]. This category has been available since 2017 and is tied to access permissions.

A team member is defined as anyone classified as a member, owner, or collaborator in the author association[6]. Since author association information is only recorded when a developer participates in specific events, months without participation result in missing author association information. To address this issue of missing values, I document the earliest and latest instances when each developer is recognized as a team member, assuming they were part of the team during the time. It is rare for a member to leave and rejoin a team midway through the observation period.

2.4 Mapping Code Files to Tasks

I link every code file with team members who have worked on it, whether by writing, providing feedback, or evaluating its quality. I do not focus on the code's original authors, as receiving feedback and requesting reviews from other members is also crucial. Such collaborative efforts reduce the bugs and conflicts when merging code files into the main project.

The data contains 34,762,813 code filenames from 2017 to 2023. I use the Latent Dirichlet Allocation (LDA) topic modeling method to categorize these files into 10 task categories (Blei et al. (2003)). Figure 2a illustrates an example of code file names used as input for LDA. Having too many categories can lead to sparse observations, which leads to measurement problems. Conversely, too few categories will obscure important variations, making it difficult to capture differences in task composition across teams. Therefore, I choose 10 task categories for the whole analysis. Table 1 presents a list of the 10 task groups and the results of the LDA model keywords and 2b shows a member task snapshot in the real dataset. The assumption is that code files within the same category have similar skill requirements and lower coordination costs. The appendix section B provides additional details about the process and examples.

[5]Detailed information from GitHub Docs: author association, accessed 2024-04-26

[6]I don't use MemberEvent because not every user becomes a member through the MemberEvent. Additionally, some users added via MemberEvent do not work after their addition.

[Table 1 about here.]

[Figure 2 about here.]

[Table 2 about here.]

2.5 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 about the process and word cloud.

2.6 Gender Prediction

To predict the gender of developers based on their avatars, I select a state-of-the-art model that closely matches the resolution and cropping methods of the training images. This chosen model[7] trains vision transformers on male and female portraits and develops a corresponding image classifier. The model predicts the probability that an image represents a female or a male. To find the optimal cutoff for gender classification, I use a sample labeled by some annotators and find that the highest accuracy is achieved with a cutoff of 0.85. If the predicted probability is above 0.85, the image will be classified as male or female; otherwise, it will be labeled as "others".

2.7 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: Hispanic (of any race), and non-Hispanic 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 four categories, with the category of highest probability being designated as the predicted race or ethnicity.

[7]https://huggingface.co/rizvandwiki/gender-classification, accessed 2024-04-23

3 Measuring Task Assignment and Team Productivity

This section outlines the measurement approach for task specialization and team productivity. I introduce a novel specialization metric that quantifies how a team's task allocation deviates from theoretical benchmarks based on team size and task distribution. Specifically, I compare each team's actual task assignment to two counterfactual benchmarks: a generalized benchmark, where tasks are evenly distributed among members, and a specialized benchmark, where each member focuses on a single task[8]. This relative metric enables comparison of specialization across teams with varying sizes and project types.

For team productivity, I develop four distinct metrics that capture various dimensions of performance in software development. 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. While these measures are designed for software teams, they can be applied to other contexts.

3.1 Measure Team Specialization

Denote $\mathbb{J}$ the set of all task types. Each team $m = 1, \ldots, M$ consists of $N_m \in \mathbb{N}$ 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 share 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 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 shares $\alpha_m = \begin{bmatrix} 3/2 & 1/2 & 1 \end{bmatrix}$. Suppose the actual assignment(A_m)

[8]Some teams could not assign one task to one member due to tasks requiring multiple units of labor

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 shares (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. To measure the degree of specialization in A_m, we calculate the distance between A_m and G_m, normalized by the distance between S_m and G_m as defined in Definition 3. The specialization index $\text{SPE}_m \in [0, 1]$ and is higher if A_m is farther from the counterfactual generalized allocation G_m.

[Table 3 about here.]

There are many ways to measure the distance between two matrices. In this paper, I use Euclidean distance[9] and

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

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

To apply the specialization index SPE_m to empirical data, each code file is categorized into one of the ten 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 forms the matrix $A_m(i, j)$. [10] For example, consider a team with two tasks: Front-end Development and Back-end Development. If a member exclusively works on files related to

[9] I only focus on teams with more than 3 members and 1 task type. Mathematically, when $|K_m|$=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.

[10] 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 due to the challenge of precisely observing the time spent on each file.

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 for teams with more than three members, illustrating both the overall distribution and its variation by team size. Many teams exhibit highly specialized structures (index values near the upper bound), while others opt for no specialization (index value of 0). The majority of teams concentrate around the mid-range values (0.4–0.7), indicating a balanced degree of specialization. Figure 3b further decomposes the specialization distribution by team size. Each subplot represents a different team size, ranging from 4 to 15 members. While the overall distribution remains similar across team sizes, larger teams tend to adopt more specialized structures. Figure D1 provides additional details on other dimensions.

[Figure 3 about here.]

3.2 Measure Team Productivity

Measuring productivity is inherently challenging. I develop 4 different measures that capture different dimensions of team-level productivity, drawing from the existing literature and the context of software engineering. These measures include output quality and quantity, submission acceptance rate, and problem-solving speed.

Output Quality. - 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. Studies show that 75% of developers check a project's star count before deciding to use it (Borges and Valente, 2018). Additionally, stars have a monetary value, with each star estimated to be worth approximately 0.88 USD (Eldeeb and Sikora, 2023). To measure a team output quality, I track the number of stars it receives each month. Figure E1 provides an example of the cumulative stars received by a team 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 request (PR), similar to measuring product passing rate in manufacturing. It is calculated as the ratio of successfully merged PRs to the total number of PR submitted in a given month. Developers create branches to isolate development work without affecting others and then use PRs to merge those branches once complete. PR can be rejected when conflicts arise, such as when both the PRs and target branches modify the same part of a file differently, GitHub is unable to automatically decide which changes to keep. Additionally, PRs can fail if developed do not adhere to branch protection rules or if they do not pass required checks. Reviewers may also reject PRs due to issues with code quality, design, or functionality issues. This requires developers to address feedback and update their PRs accordingly. A low PR success rate indicates teams frequently encounter conflicts or require significant rework before merging code. Figure E5 provides examples of the success and failure of 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 E6 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 in software development:

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 task distribution 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 5 displays a binned scatter plot illustrating the relationship between organizational specialization and output quality, measured by the log of monthly stars received from platform users. To mitigate omitted variable bias, I include several controls and fixed effects. First, I control for task share distribution to account for variations in project complexity and scope. Second, I add team age fixed effects because project dynamics could potentially affect both specialization and output quality[11]. Third, I include fixed effects for teams, members, and team size to account for time-invariant heterogeneity in team composition, member characteristics, and project attributes. These controls help mitigate potential confounding factors, such as differences in team structure, project characteristics, and lifecycle effects.

After accounting for these factors, the results indicate a negative relationship between team specialization and project quality: as the team structure becomes more specialized, project quality declines.

[Figure 5 about here.]

Table 5 presents the results from OLS and Poisson pseudo-maximum likelihood (PPML) regressions. In the OLS models, where the dependent variable is the log of monthly stars, team specialization (SPE) is negatively associated with output quality across all specifications. Column (1) shows that a one-unit increase in the specialization index is associated with a 32.0% decrease, implying a substantial reduction in output quality. The magnitude of this negative effect decreases to 8.6% but remains statistically significant in Column (4) when controlling for team and member fixed effects.

In the Poisson models, where the dependent variable is the count of monthly stars, the negative relationship between specialization and quality is even clearer. The coefficient in Column (1) indicates that a one-unit increase in the specialization index is associated with a 42.2% decrease in the expected value of monthly stars. When team and member fixed effects are included, the magnitude of the effect decreases but remains statistically significant: a one-unit increase in specialization is still associated with a 14.3% decrease in output quality.

[Table 5 about here.]

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

[11]For example, during the development stage, teams work actively, attract platform attention, and use one type of task allocation. In the maintenance stage, they perform regular updates, which can affect both outcomes and task allocation.

Figure 6 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 5, revealing that more specialized teams tend to produce less coding output.

[Figure 6 about here.]

Table 6 reports the regression results. In the OLS models, where the dependent variable is the log lines of code, specialization (SPE) shows a negative association with output quantity across all specifications. Column (4) indicates that a one-unit increase in the specialization index corresponds to a 93.4% decrease in lines of code. Similarly, in the Poisson models, which measure the count of lines of code, Column (4) shows that a one-unit increase in the specialization index results in a 55.4% decline in the expected value of lines of code.

[Table 6 about here.]

Fact 4 *Specialized teams achieve higher acceptance rates but require more time to resolve user problems.*

Figures 7a and 7b illustrate the trade-off between task-specific and general human capital. Unlike Figure 5, these analyses additionally include fixed effects for the number of code submissions and the number of questions, respectively. This ensures that the observed relationships account for variations in coding activity and inquiry frequency. Figure 7a shows that code acceptance rates increase with increased team specialization, suggesting that developers who focus on specific tasks produce higher-quality code. However, Figure 7b reveals that specialized teams take longer to resolve platform users' problems compared to less specialized teams. This delay likely arises because effective problem-solving often requires integrating knowledge across multiple domains, such as code architecture, user interface design, and database management.

Additionally, Figure 7c examines the relationship between team specialization and the number of comments. The results indicate that highly specialized teams tend to have fewer comments and less communication, suggesting lower levels of knowledge exchange within the team. This finding supports the hypothesis that while specialization increases task-specific human capital, it also leads to a decrease in general human capital and reduced knowledge diffusion.

[Table 7 about here.]

[Table 8 about here.]

Table 7 and Table 8 present the regression results. After controlling for team task distribution, and adding team age, team size, team and member fixed effects, a one-unit increase in

the team specialization index is associated with a 1.82% increase in the code acceptance rate and an 8.06% increase in the time required to solve a problem. These findings suggest that while specialization may improve certain aspects of code quality, it can compromise problem - solving efficiency.

[Figure 7 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' represents gender diversity index, S is the total number of different categories (e.g., male and female), and p_i is the proportion of individuals in the i-th gender category. This index measures both the abundance and evenness of gender within the team. A higher gender diversity index indicates greater diversity, meaning that the team has a more balanced gender distribution (Krebs, 1989).

Figure 8 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. This result suggests teams coordinate members from similar backgrounds which could avoid communication and coordination costs generated by diversity (Lazear, 1999; Prat, 2002; Hong and Page, 2004; Kahane et al., 2013).

[Figure 8 about here.]

5 Empirical Strategies

The descriptive evidence shows a negative correlation between specialization and team productivity. The choice of task allocation structure is often endogenously determined. Managers might adopt specialized structures in response to poor performance or workforce disruptions. To mitigate these concerns, I exploit GitHub's auto-assignment feature, which evenly distributes tasks among team members. Using a matched difference-in-differences design, I compare teams that adopt auto-assignment (treatment) with those that do not (control) but share similar characteristics. This matching approach mitigates selection bias and pre-existing differences

in specialization, allowing for a more credible assessment of how specialization affects team productivity.

5.1 GitHub Automatic Assignment Feature

I exploit GitHub's auto-assignment feature, which introduces exogenous variation in team specialization by evenly distributing tasks. This system automatically assigns team members to Pull Requests (code submissions requiring review) and Issues (bug reports or feature requests). Unlike traditional workflows where certain members specialize in specific tasks, auto-assignment randomly selects reviewers or assignees. Teams enable this feature via configuration files[12], ensuring a balanced workload and reducing biases in task allocation. By making all members equally likely to engage in different tasks, this mechanism shifts teams toward a more generalized structure. Figure E7 provides an example of such a configuration file.

This setting offers a unique opportunity to study how decreased specialization affects team performance, as adopting teams effectively shifts toward a more generalized task allocation structure. To identify the causal effect, I employ a matched difference-in-differences design comparing teams that adopt automatic assignment (treatment) with those that do not (control). Following the approach from the first identification strategy, I implement a 1:1 matching procedure based on pre-treatment characteristics measured from $t-5$ to $t-1$: deciles of average team size, task types, and output levels.

I successfully matched 3,864 teams in the treatment and control groups, achieving a 98.2% success rate, which confirms that treated teams have closely comparable counterparts. Table 9 presents the summary statistics of both groups at the time of adopting auto assignment. The treatment and control teams are closely aligned in terms of team size, task types, lines of code, and specialization index, consistent with the matching procedure. According to the table, at the time of adoption, treatment teams had an average team size of 7.86, compared to 6.93 for control teams, and engaged in 5.84 task types, slightly higher than the 4.79 task types in the control group. Despite not being directly matched on all characteristics, both groups remain balanced in key productivity measures. The specialization index (0.53–0.56) and code acceptance rate (0.83) are nearly identical across groups. Overall, the table highlights that treatment and control groups are well-balanced across core characteristics related to team composition, specialization, and performance. This strengthens the credibility of the matching process, providing a robust empirical foundation for assessing the impact of even task assignment adoption.

[Table 9 about here.]

[12]For example, .github/workflows/ or .github/auto_assign.yml

5.2 Dynamic Effects

Building on the identification strategy in Jaravel et al. (2018); Sharoni (2024), I use OLS regressions with a full set of leads and lags around the adoption of even task assignments. This approach allows me to examine how adoption affects team specialization and productivity while also validating the research design by testing for pre-existing trends before adoption, addressing concerns about potential common shocks. Using the constructed treatment and control groups, I align adoption events so that the time index k represents time relative to adoption, with $k = 0$ marking the adoption event. This stacking method follows Roth (2022) to ensure a robust estimation of pre-trends. I then estimate equation (1),

$$Y_{mk} = \gamma_m + \beta_k + \psi_{j(m)} + \sum_{k=-5, k\neq -1}^{6} \alpha_k \times \mathbb{I}\left(\text{period}_k\right) \times \text{Treated}_m + \epsilon_{m,k} \tag{1}$$

where Y_{mk} denotes the outcome for team m in month $k = t - d$ (e.g., monthly stars received from platform users, code acceptance rate, problem-solving time) in the treatment group and are normalized to zero in $k = -1$. The specification includes team size fixed effects (γ_m), time fixed effects (β_k), and firm fixed effects ($\psi_{j(m)}$). Treated_m indicates teams that adopt auto-assignment. I cluster the standard errors at the matching level.

If the adoption of even task assignment serves as an exogenous variation, the coefficients α_k estimate its causal effect on even task assignment and productivity, measured k periods relative to adoption. A positive α_k suggests that even task assignment improves productivity, while a negative coefficient indicates a decline. If there is no effect, the coefficient would be zero. However, a key identification concern is selection bias—if teams that adopt even task assignment systematically differ from those that do not, the estimated effect may not reflect true causality. To mitigate this issue, I construct a matched control group of teams with similar pre-treatment characteristics but no adoption. In the analysis below, I demonstrate that pre-trends are not statistically significant, addressing concerns about potential biases. Specifically, the lag coefficients α_{-5} to α_{-1} show no significant differences before adoption, suggesting that prior trends in specialization and productivity are not driving the observed effects. This finding supports the validity of the estimates and reinforces the causal interpretation of the results.

Figure 9 presents estimates from regression equation (1) that show no significant pre-trends. As shown in Figure 9a, adopting auto task assignment reduces team specialization by about 2%, as team members—especially reviewers and assignees—are randomly allocated to tasks, resulting in a more generalized task allocation structure.

The impact on team performance varies across different outcomes. Figure 9b shows that, relative to the control group, teams with reduced specialization experience a 15% statisti-

cally significant increase in output quality —measured by monthly stars—approximately three months after adoption. In terms of output quantity as shown in Figure9c, teams adopting auto-assignment increase output quantity by 17.7% (measured by the number of lines of code written), as measured by the number of lines of code written. However, despite this increase in coding activity, the code acceptance rate declines immediately by roughly 2.4% with a baseline of 0.83, suggesting the loss of task-specific human capital. There is no significant effect on problem-solving speed(e.g., time required to resolve issues). Finally, Figure 9f indicates a 10% increase in team communication following adoption, providing evidence that reduced specialization promotes greater knowledge sharing and cross-task feedback.

[Figure 9 about here.]

6 Conclusion

In this paper, I investigate how specialization in team task allocation affects productivity and knowledge sharing in software development. Although the traditional view, dating back to Smith (1776), suggests that specialization increases productivity, my findings reveal a negative relationship between team specialization and productivity. They highlight that excessive specialization can inhibit the general human capital necessary for innovative work where the next steps are often uncertain and cross-task feedback is essential.

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

Another significant contribution of this study is the development of new empirical indicators for both specialization and productivity. I create a novel dataset measuring team task allocations based on the code developers produce, alongside various productivity metrics. My specialization measure captures deviations from fully generalized or fully specialized benchmarks, allowing for comparisons within teams over time. For productivity, I establish metrics that encompass both output quantity (lines of code) and quality (project popularity), along with finer indicators such as code acceptance rates and problem-solving speed. Together, these metrics provide a comprehensive view of how specialization affects innovation processes.

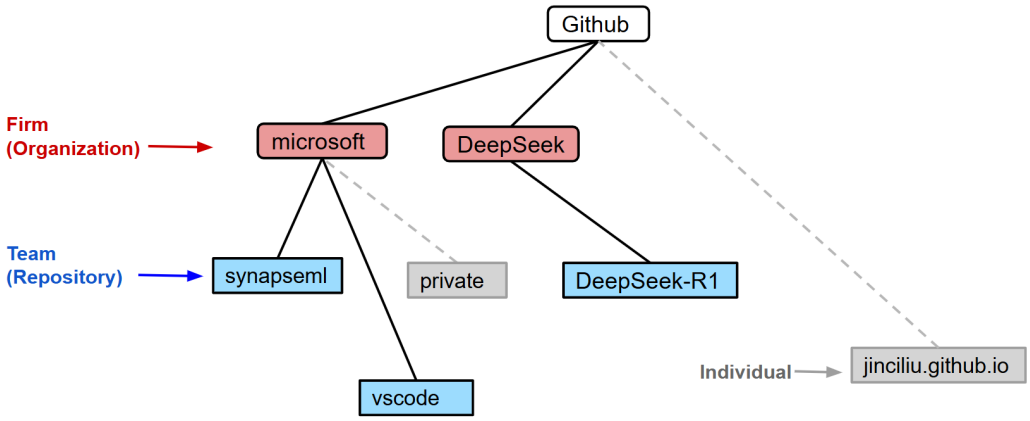
These findings enhance our understanding of organizational design in innovative industries. While previous research has emphasized the benefits of specialization through task-specific human capital accumulation, I illustrate an important trade-off: excessive specialization can hinder the development of general human capital essential for complex problem-solving and knowledge integration. This distinction is especially relevant as economies increasingly shift

toward knowledge work, where solutions often require integration across multiple domains.

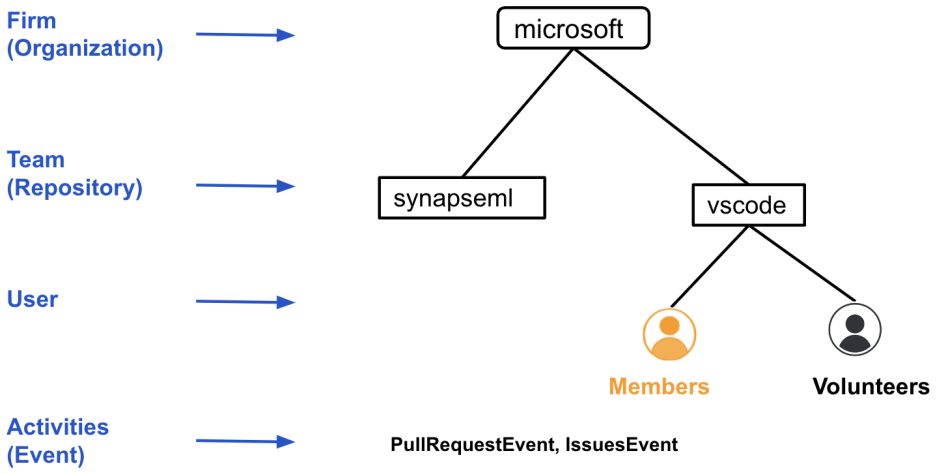
Moreover, the results contribute to the broader literature on firm organization and productivity variation. The persistent differences in organizational structures among firms may reflect not only coordination costs or information frictions but also different optimal positions along the specialization spectrum, depending on the nature of production processes. Future research could explore how these trade-offs vary across industries and types of knowledge work, potentially using the measurement framework developed in this paper.

Figure 1: Schematic Representation of GitHub Structure

(a) Organization and Team



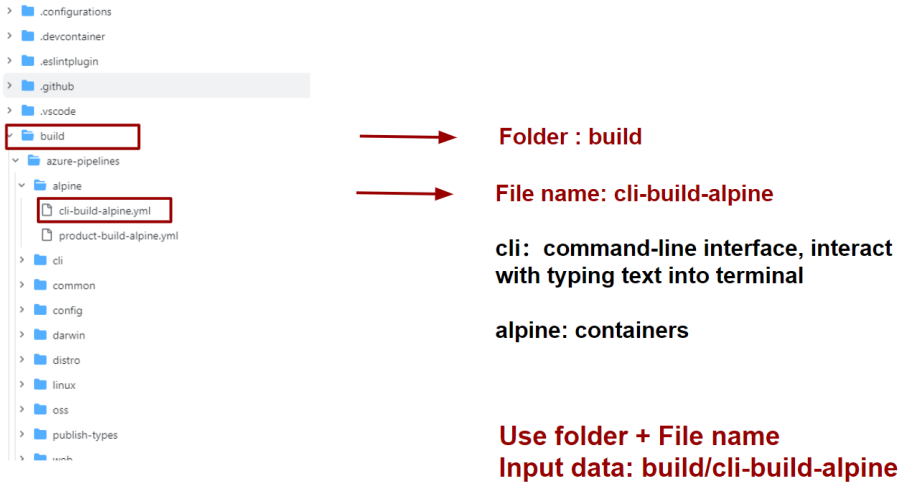
(b) Team members and Volunteers



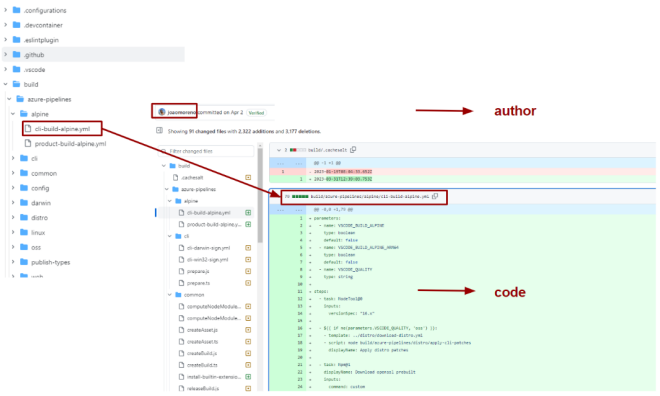
Notes: As shown in Figure 1a, some repositories belong to the organization, while others do not. In this project, I include only repositories affiliated with the organization to exclude those intended for individual purposes or hobbies, such as personal websites. Figure 1b illustrates the structure within a single organization. For instance, within Microsoft, there are multiple teams, each with various users. Some users are official team members (see Section 2.3), 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

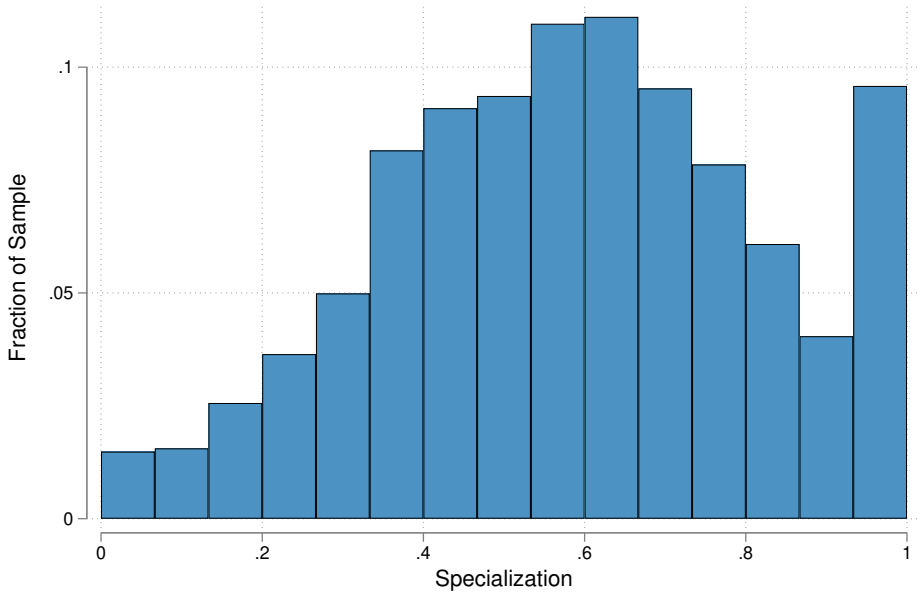


Team	Month	Member	Task Type	Labor Share
vscode	2021-05	joaomoreno	1	0.714286
vscode	2021-05	joaomoreno	4	0.285714
vscode	2021-05	isidorn	4	0.018809
vscode	2021-05	...	...	...

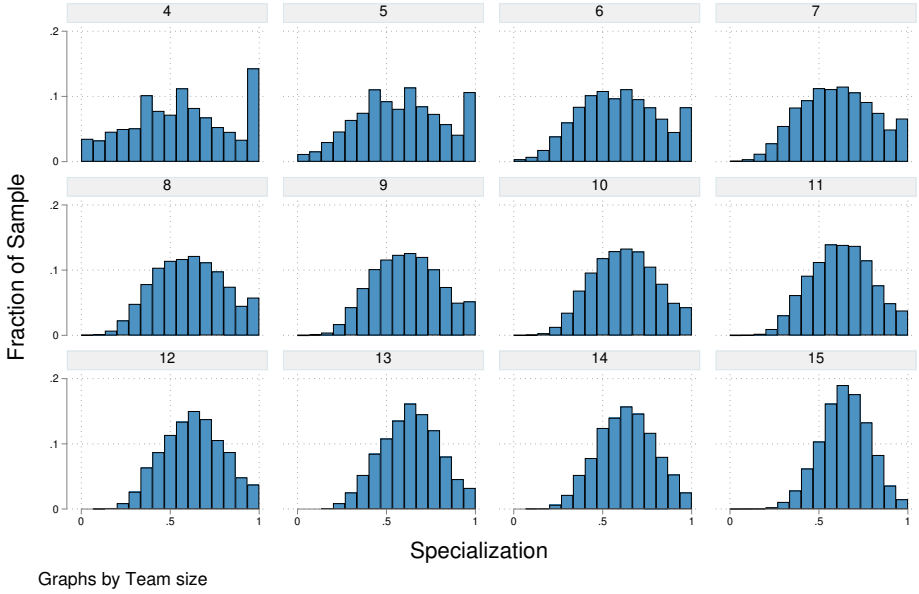
Notes: Figure 2a illustrates an example of code file names used for task type classification. The input data consists of the folder name combined with the file name, such as "build/cli-build-alpine" in the example above. Figure 2b presents a data snapshot of Member "joaomoreno" from Team "vscode" highlighting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

Figure 3: Distribution of Team Specialization Index

(a) Specialization

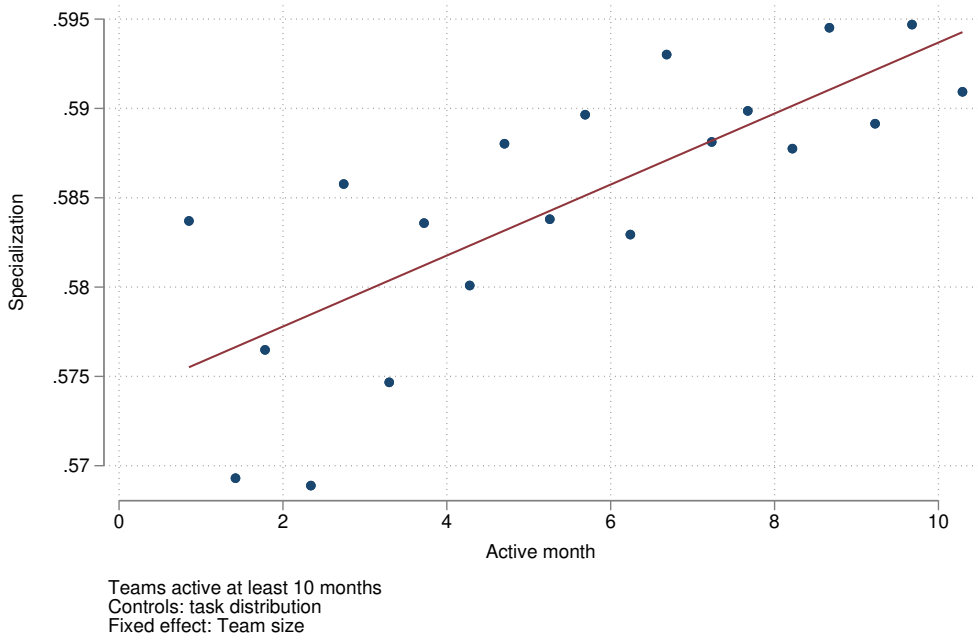


(b) By Team Size



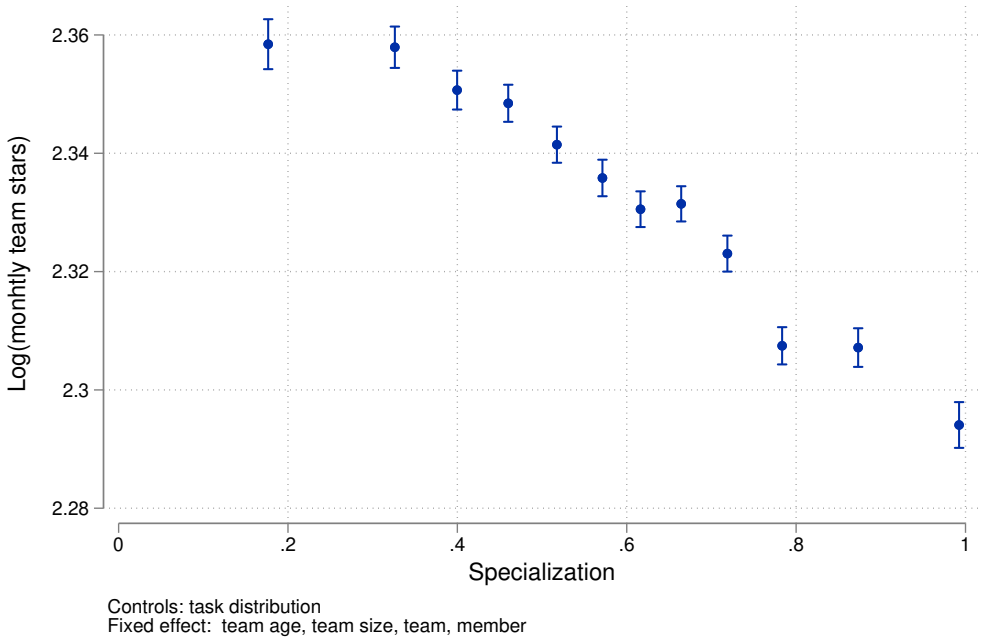
Notes: All team-month observations are included, except for teams with fewer than four workers (team size). A higher index value reflects a greater degree of specialization. FigureD1 provides more details about the distribution of the team specialization index.

Figure 4: Team Specialization Over Time



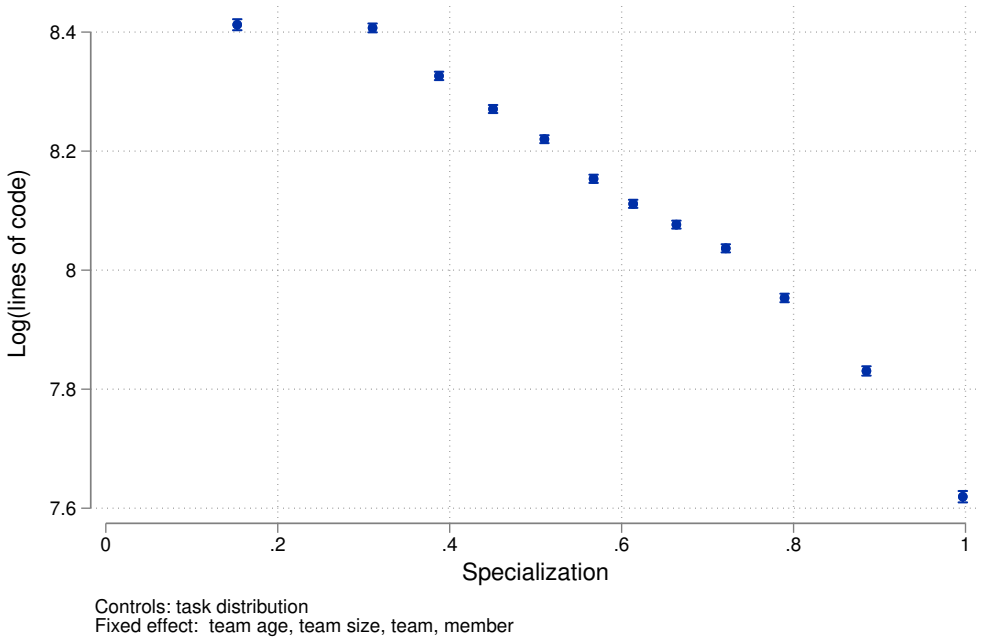
Notes: The figure shows a binned scatter plot illustrating the evolution of team specialization over time, specifically for teams that are active for at least 10 months, while controlling for task distribution and team size fixed effect.

Figure 5: Team Specialization and Output Quality



Notes: This figure presents a binned scatter plot, accounting for task distribution. The analysis also includes team age, 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.

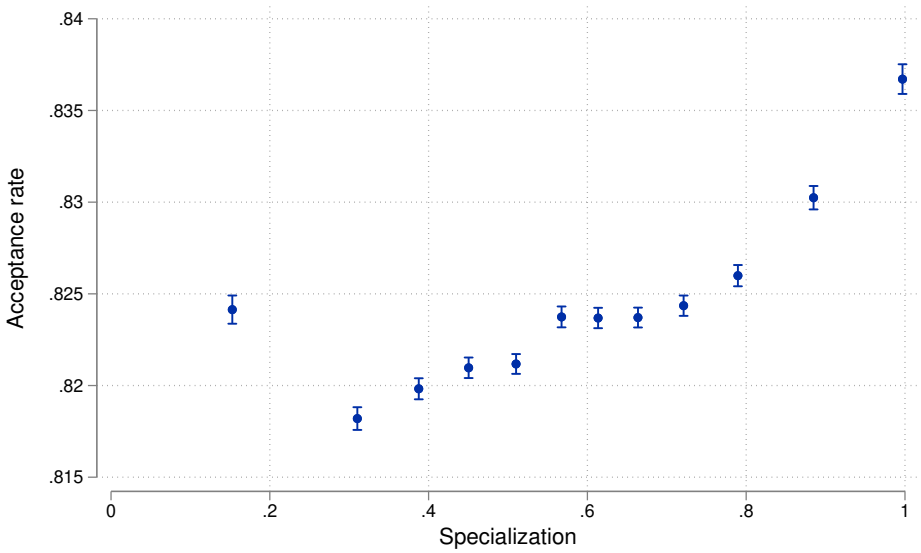
Figure 6: Team Specialization and Output Quantity



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 quantity.

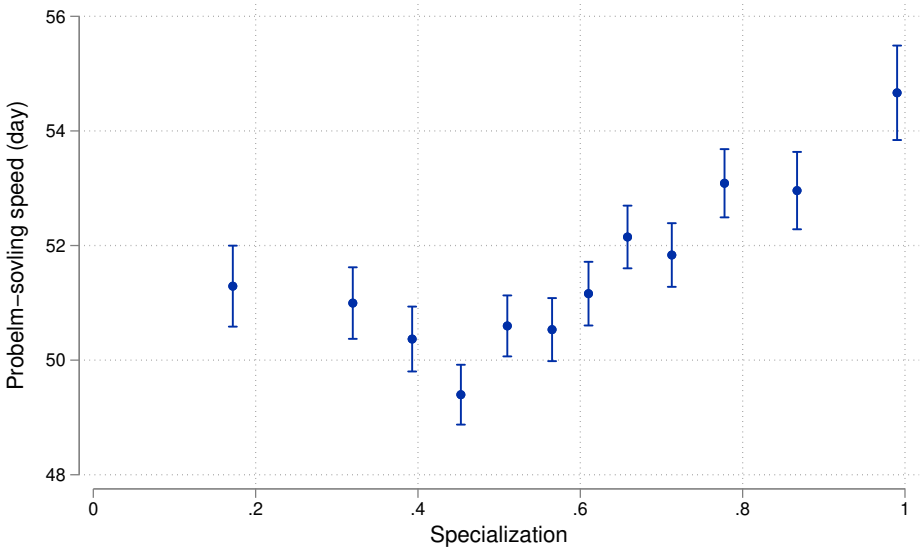
Figure 7: Team Specialization Trade-off

(a) Code Acceptance Rate



Controls: task distribution, team age
Fixed effect: team size, #submissions, team, worker

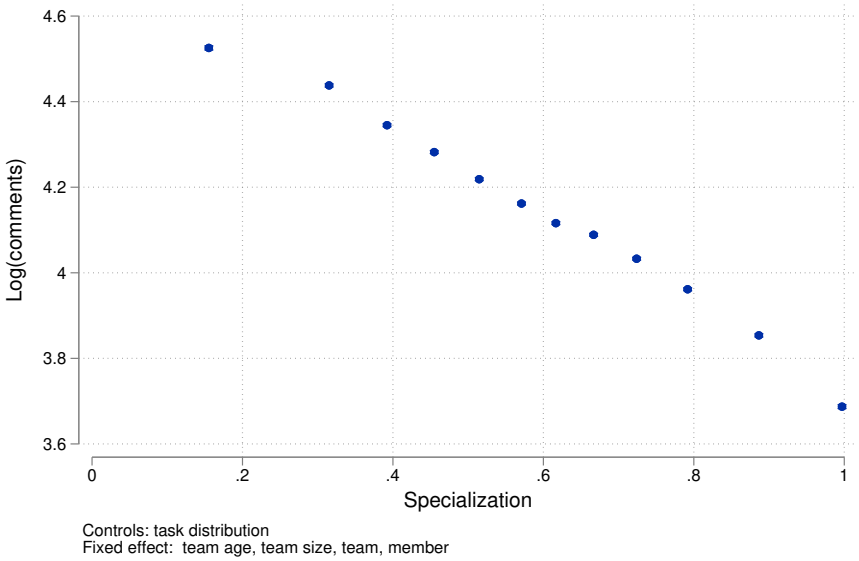
(b) Problem Solving Speed



Controls: task distribution
Fixed effect: team age, team size, #issues, team, member

Figure 7: Organizational Specialization Trade-off

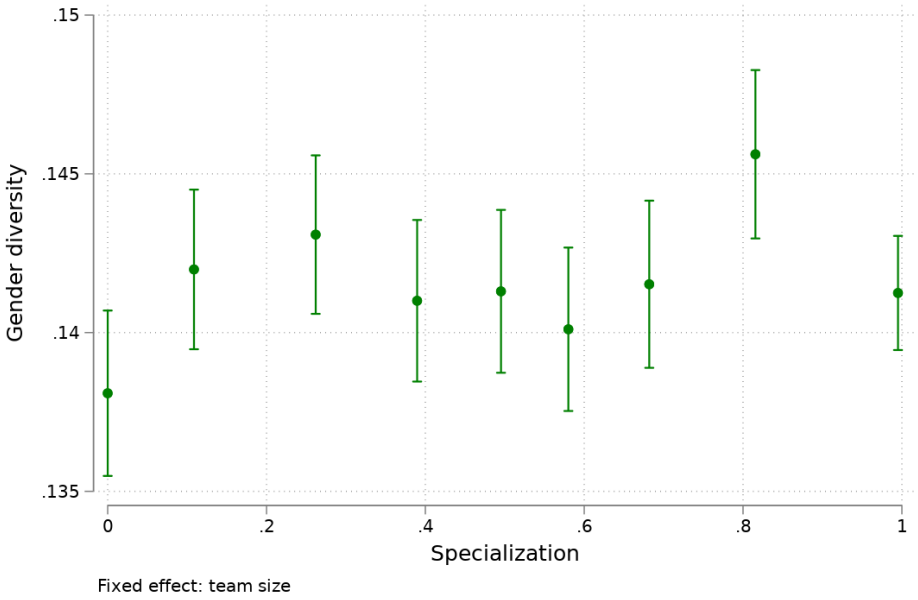
(c) Discussion



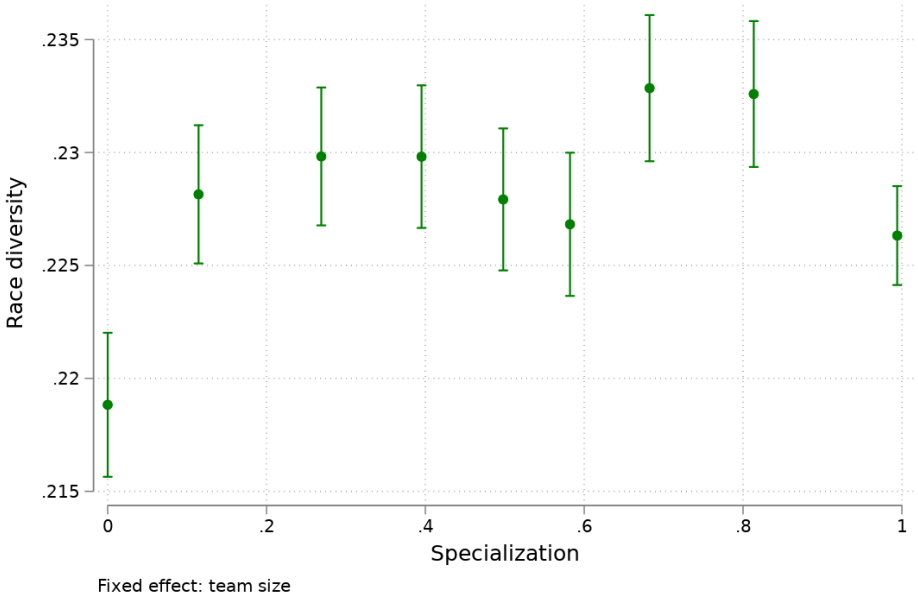
Notes: This figure shows the correlation between team specialization and code acceptance rate (Panel 7a), problem-solving speed (Panel 7b), average comments per question (Panel 7c) after controlling for task share distribution as well as team age, team size, team, member fixed effects. The observations are at the individual level and weighted by 1/team size.

Figure 8: Team Specialization and Demographic Composition

(a) Gender



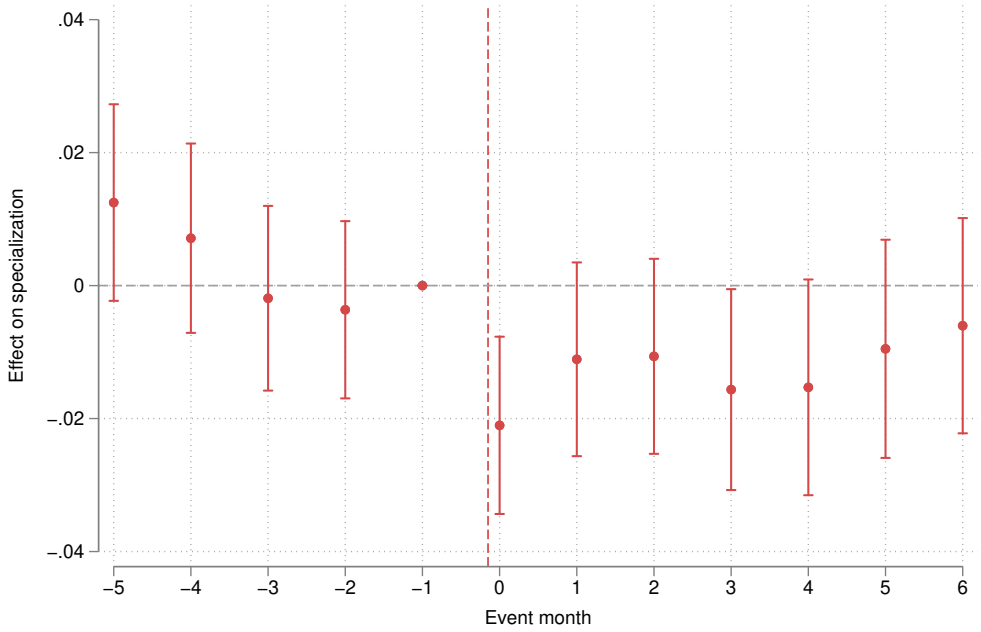
(b) Race



Notes: The positive relationship between organizational specialization and diversity in gender (Panel 8a), and race (Panel 8b) 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 9: Dynamic Effects for Task Assignment Adoption

(a) Team Specialization Index



(b) Output Quality

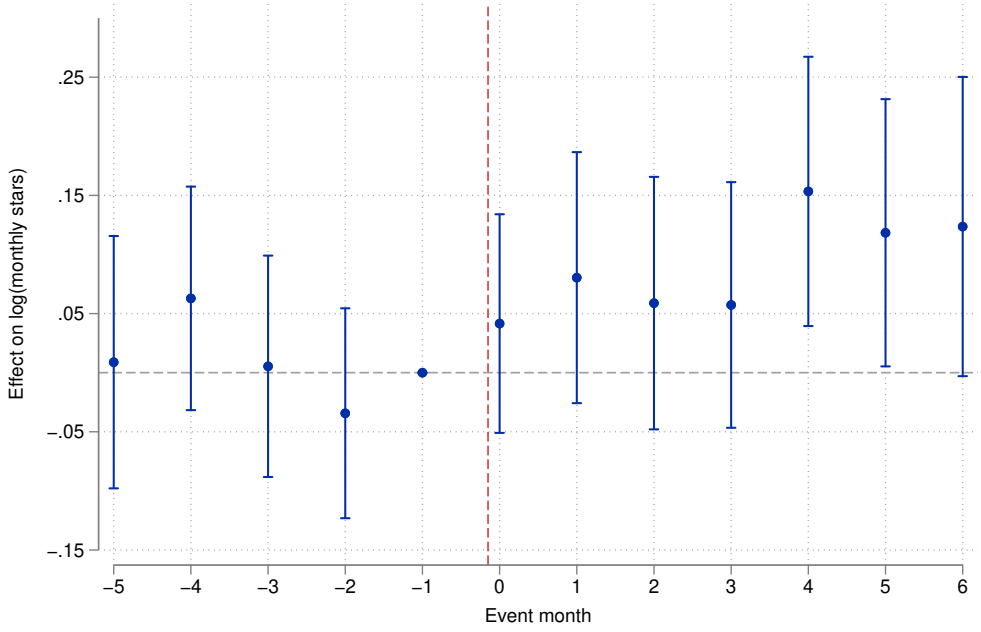
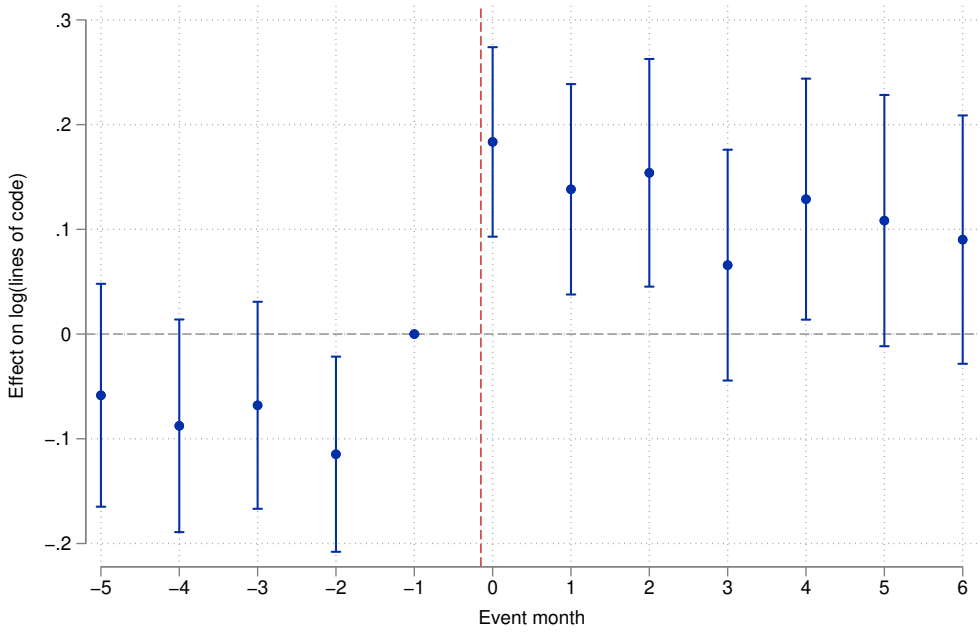


Figure 9: Dynamic Effects for Task Assignment Adoption

(c) Output Quantity



(d) Code Acceptance Rate

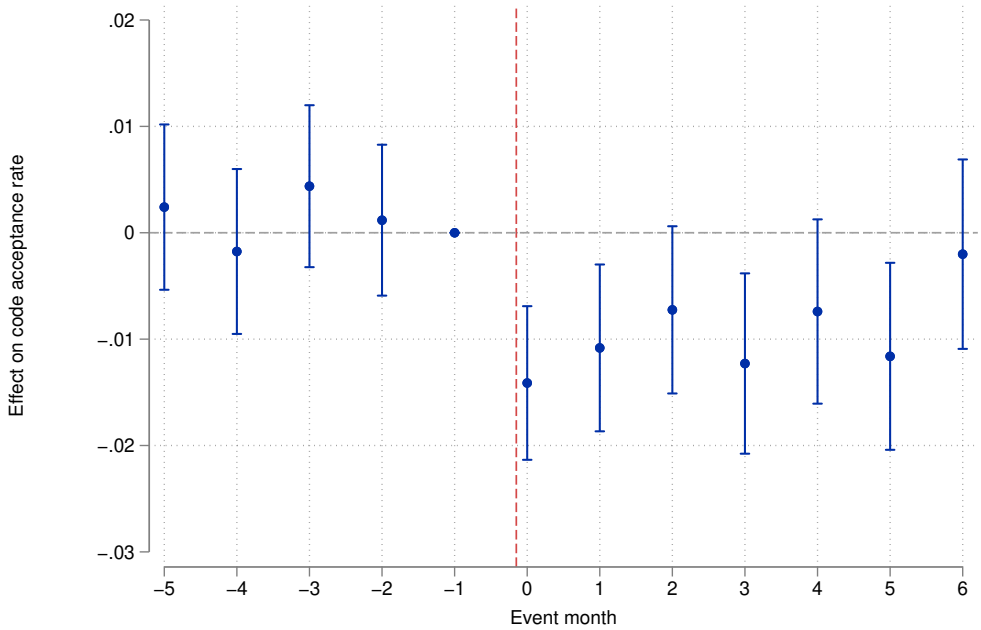
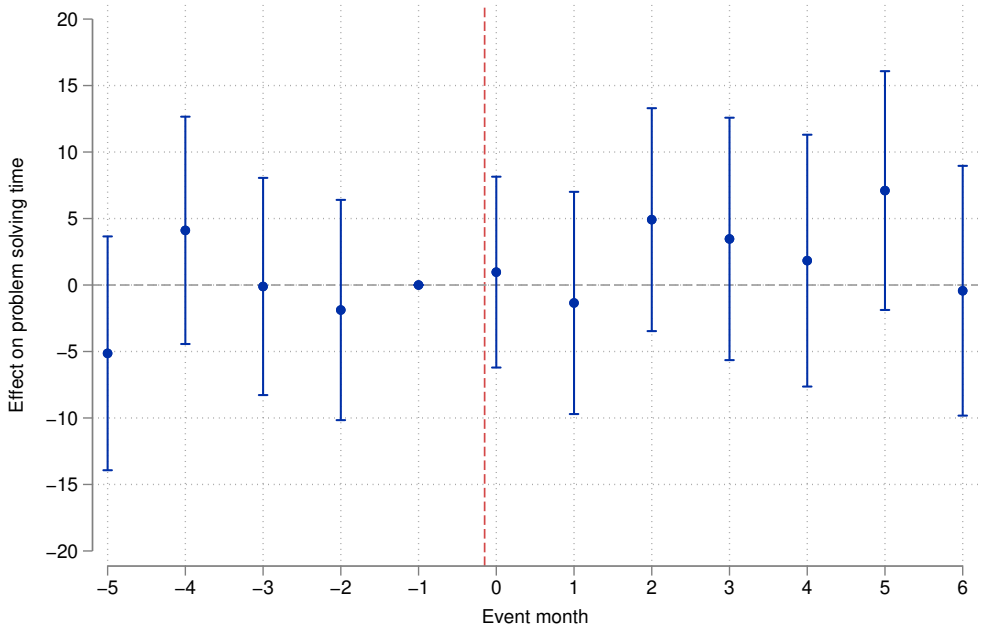
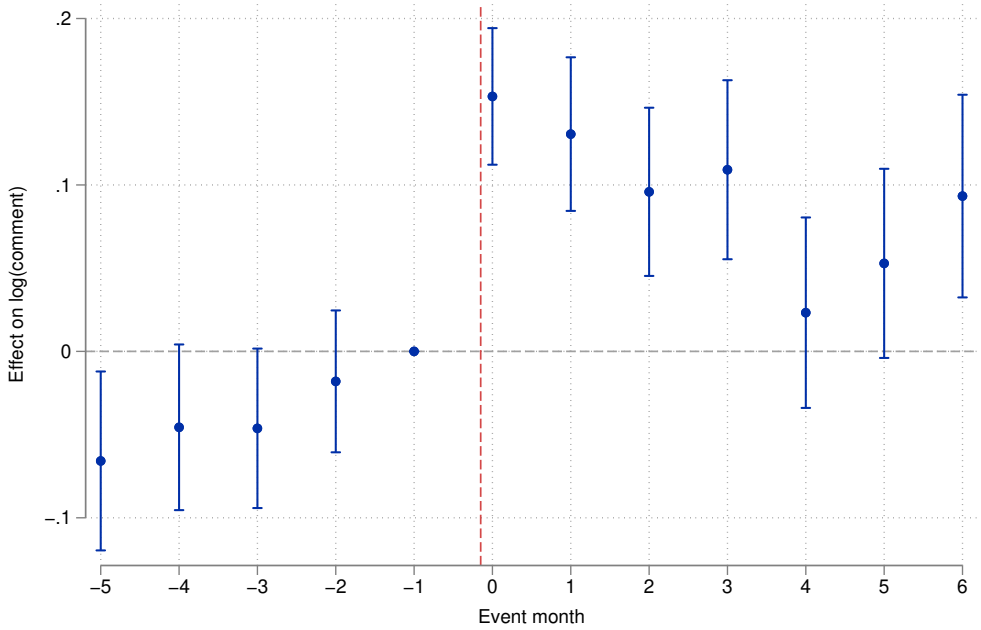


Figure 9: Dynamic Effects for Task Assignment Adoption

(e) Problem Solving Time



(f) Communication



Notes: This plot presents the effect of auto-assignment feature adoption on team organizational specialization and productivity. The vertical lines represent a 95% 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 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: Dependent var- log(monthly team star)				
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: Dependent var- monthly team star				
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
Task type share control	Y	Y	Y	Y
Team age fixed effect	Y	Y	Y	Y
Team size fixed effect	Y	Y	Y	Y
Firm fixed effect	Y	Y		
Project type fixed effect		Y		
Team fixed effect			Y	Y
Member fixed effect				Y

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: Dependent var - log (lines of code)				
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: Dependent var - lines of code				
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	Y	Y	Y	Y
Team age fixed effect	Y	Y	Y	Y
Team size fixed effect	Y	Y	Y	Y
Firm fixed effect	Y	Y		
Project type fixed effect		Y		
Team fixed effect			Y	Y
Member fixed effect				Y

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	Y	Y	Y	Y
Team size fixed effect	Y	Y	Y	Y
Firm fixed effect	Y	Y		
Project type fixed effect		Y		
# code submission		Y	Y	Y
Team fixed effect			Y	Y
Member fixed effect				Y
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	Y	Y	Y	Y
Team age fixed effect	Y	Y	Y	Y
Team size fixed effect	Y	Y	Y	Y
Firm fixed effect	Y	Y		
Project type fixed effect		Y		
# question fixed effect		Y	Y	Y
Team fixed effect			Y	Y
Member fixed effect				Y
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: Balance Table for Even Task Assignment Adoption

	Control		Treatment		Difference
	Mean	# Obs.	Mean	# Obs.	
Team size	6.93	3864	7.86	3864	-0.92
					(0.21)
Task types	4.79	3864	5.84	3864	-1.05
					(0.05)
Lines of code	49824.41	3864	55258.12	3864	-5433.72
					(11440.01)
Monthly stars	24.11	3864	59.73	3864	-35.61
					(3.71)
Specialization index	0.56	3864	0.53	3864	0.02
					(0.01)
Code acceptance	0.83	3818	0.83	3834	0.01
					(0.00)
Problem-solving time	49.21	2281	43.02	2969	6.19
					(2.69)
Create year	2018.38	3864	2018.16	3864	0.22
					(0.06)

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.

References

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.

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.

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.

Hong, L. and S. E. Page (2004). Groups of diverse problem solvers can outperform groups of high-ability problem solvers. *Proceedings of the National Academy of Sciences 101*(46), 16385–16389.

Jaravel, X., N. Petkova, and A. Bell (2018). Team-specific capital and innovation. *American Economic Review 108*(4-5), 1034–1073.

Kahane, L., N. Longley, and R. Simmons (2013). The effects of coworker heterogeneity on firm-level output: assessing the impacts of cultural and language diversity in the national hockey league. *Review of Economics and Statistics 95*(1), 302–314.

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*.

Lazear, E. P. (1999). Globalisation and the market for team-mates. *The Economic Journal 109*(454), 15–40.

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

Prat, A. (2002). Should a team be homogeneous? *European Economic Review 46*(7), 1187–1207.

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 Division of Labor affect Team Productivity?

Jinci Liu

March 9, 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

$$\max_{A \in \mathbb{R}_+^{N \times K}} Y - wN - c_m(A) \tag{2}$$

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.

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

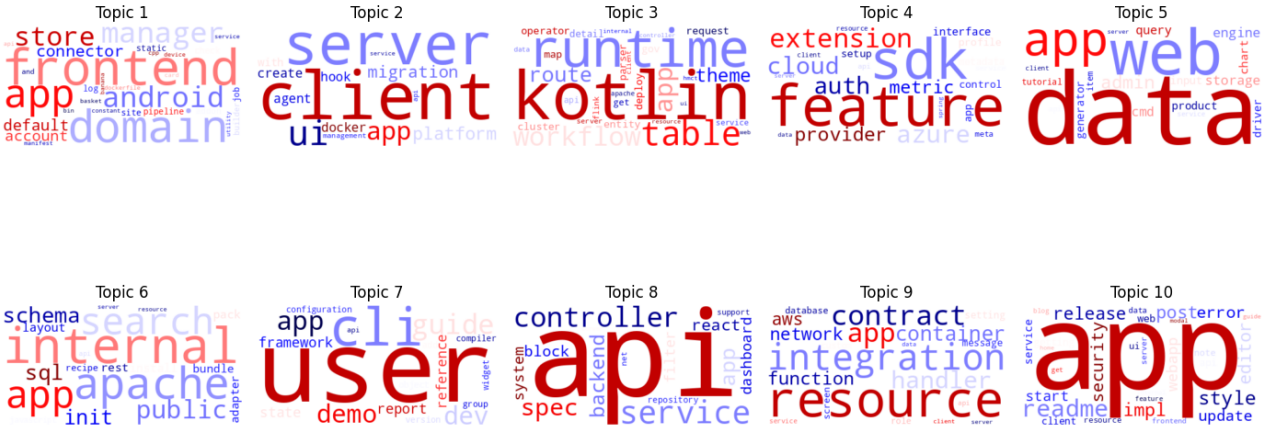
$$y(j) = \min\{1 - A(1,j), A(1,j)\} A(1,j) + \min\{A(1,j), 1 - A(1,j)\} (1 - A(1,j)$$

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

B Task Classification Process

[Figure B1 about here.]

Figure B1: Word Clouds for LDA 10 Task Types



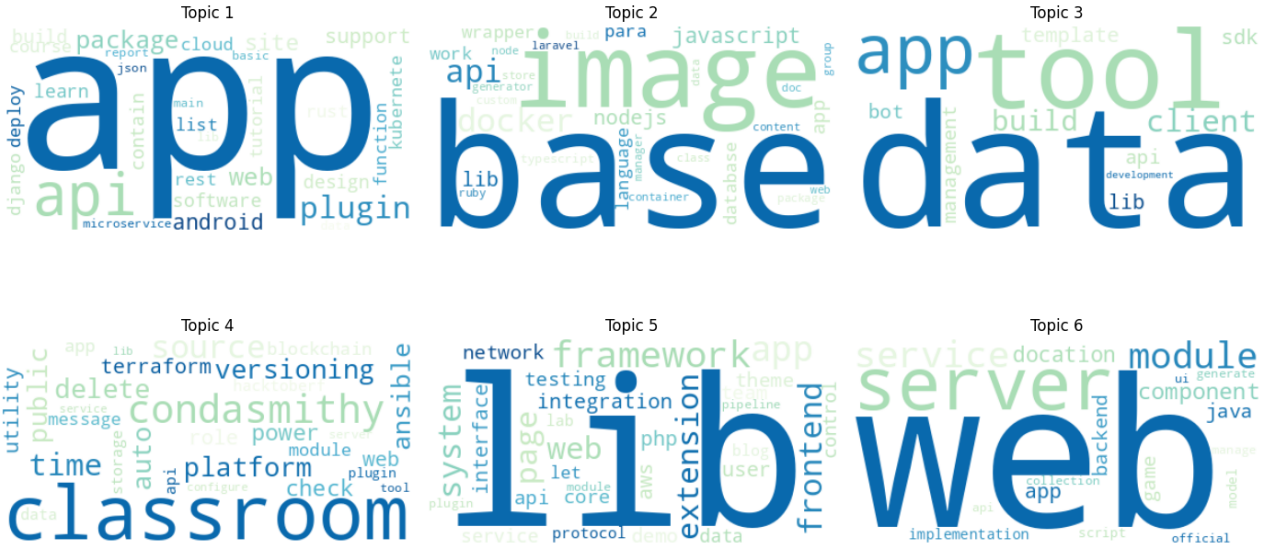
Notes: This figure presents word clouds visualizing the 10 task types identified through Latent Dirichlet Allocation (LDA) topic modeling. Each panel shows frequently occurring terms within different development domains: data management (Topic 5), server-side development (Topic 2), runtime environments (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 about here.]

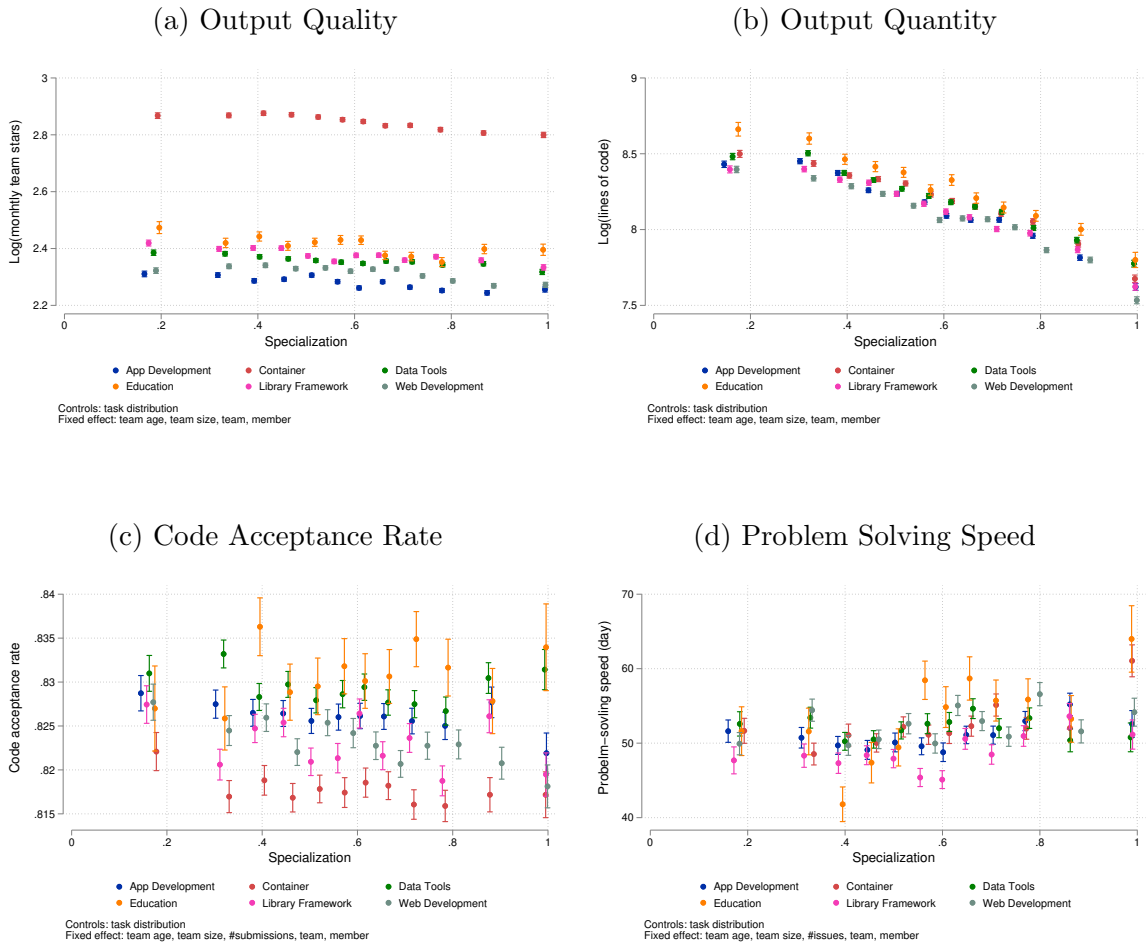
[Figure C2 about here.]

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



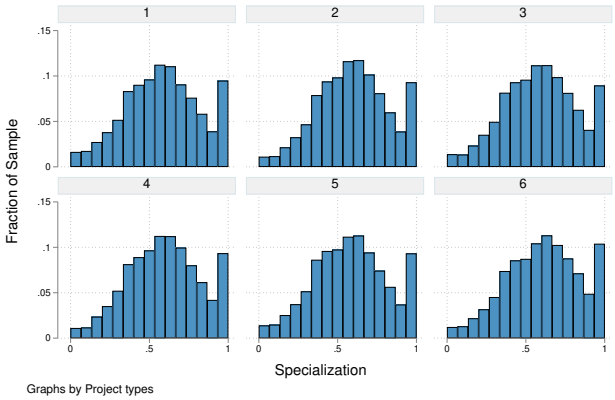
Notes: This figure presents the team specialization and productivity metrics by project types, accounting for task distribution and team age. The analysis also includes team size, member fixed effects, and team fixed effects. These fixed effects help control for differences in worker composition, team 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

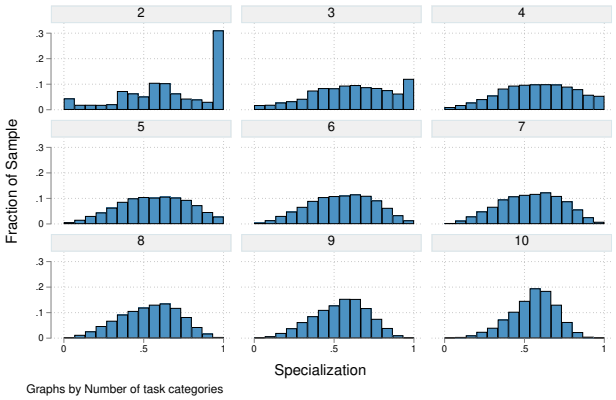
[Figure C1 about here.]

Figure D1: Distribution of Team Specialization Index

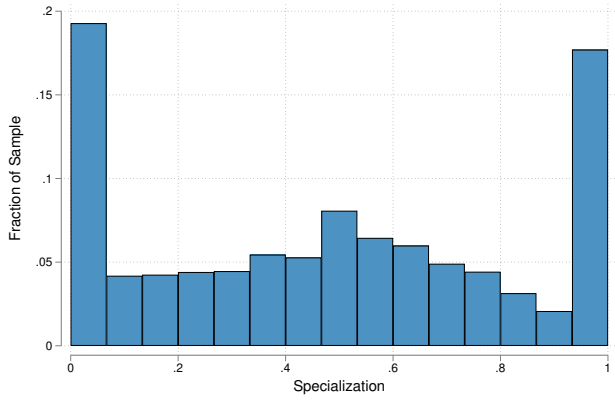
(a) By Project Type



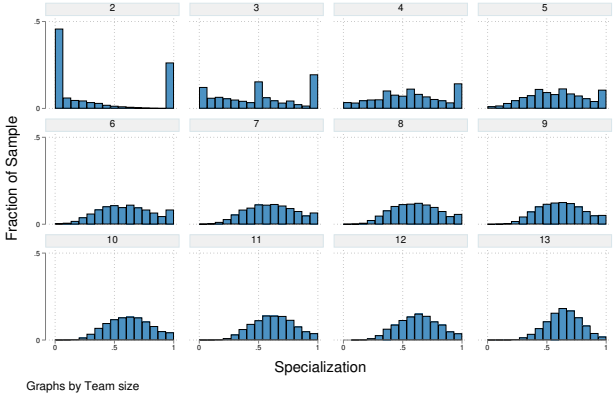
(b) By Number of Task Type



(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 2.5 and 2.4. 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.

E Examples

[Figure E1 about here.]

[Figure E2 about here.]

[Figure E3 about here.]

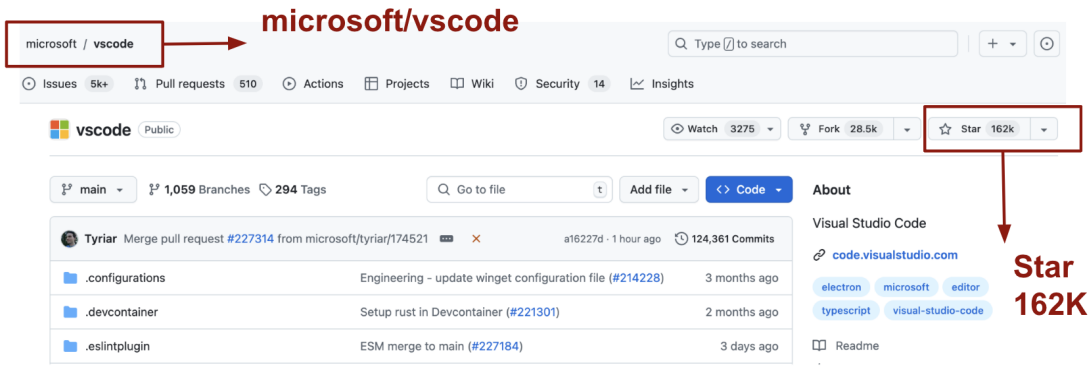
[Figure E4 about here.]

[Figure E5 about here.]

[Figure E6 about here.]

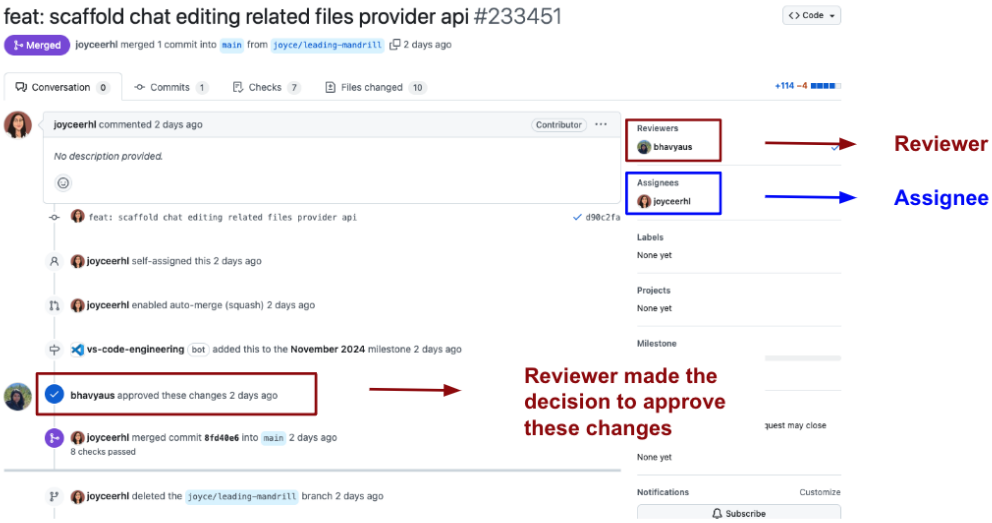
[Figure E7 about here.]

Figure E1: Star Example



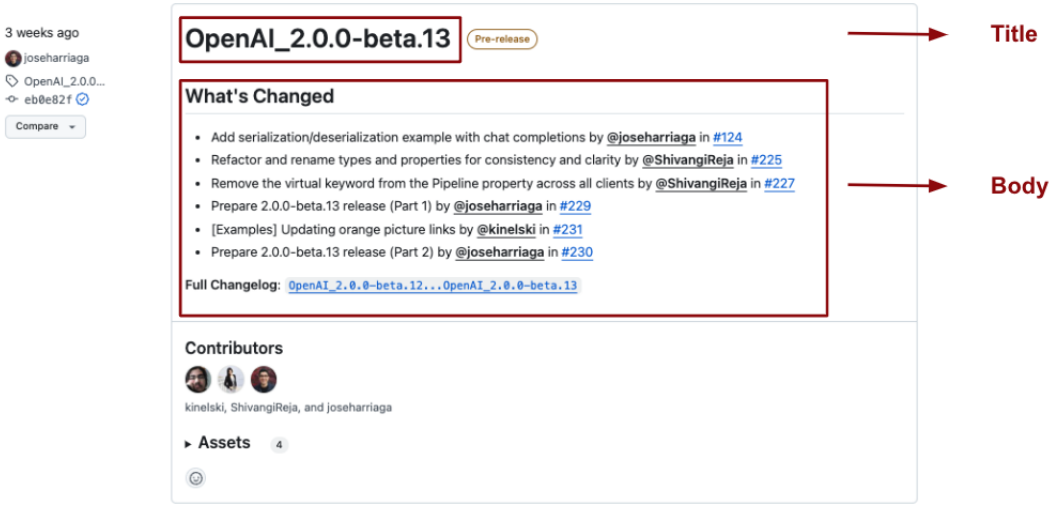
Notes: This figure illustrates how project popularity is measured using GitHub's star system, using Microsoft's Visual Studio Code repository as an example. Stars represent a form of peer evaluation where users can bookmark and show appreciation for repositories they find valuable. In this example, Visual 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



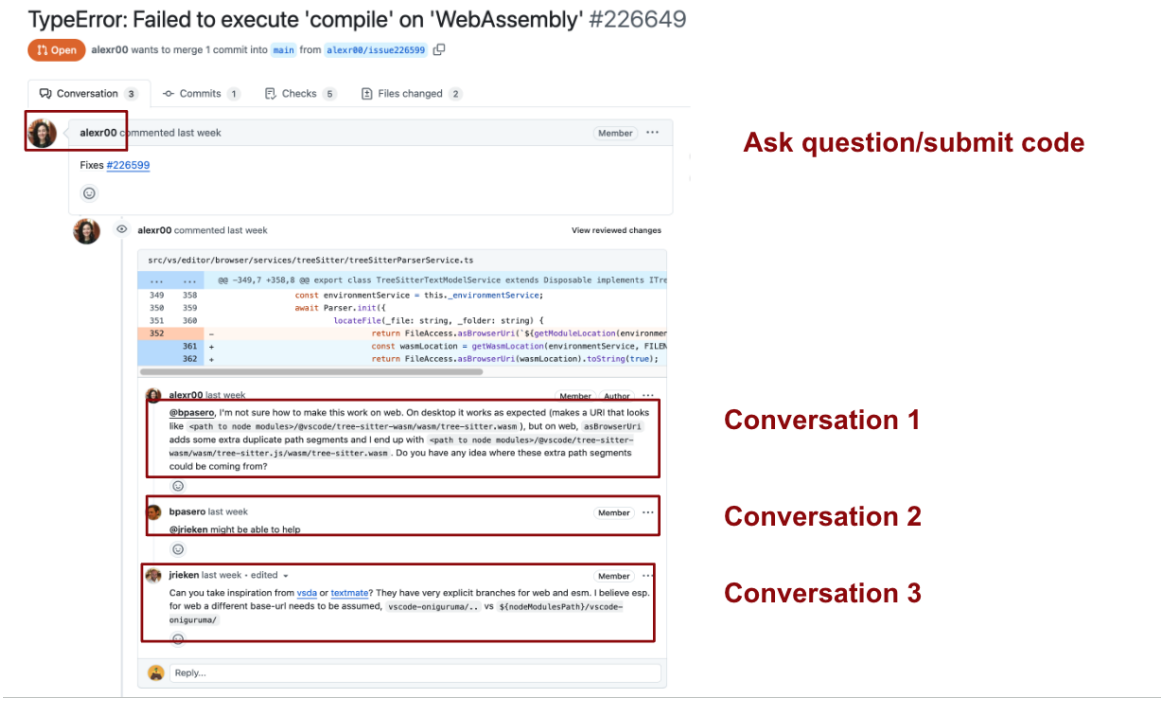
Notes: The example shows a typical Pull Request (#233451) with its associated review thread. Two distinct roles are highlighted: Reviewers (red box) who evaluate code changes and Assignees (blue box) who are responsible for implementing the changes. The bottom of the image demonstrates a completed review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

Figure E3: Project Release Example



Notes: This figure illustrates how software releases are documented on GitHub, using OpenAI_2.0.0-beta.13 as an example. A release consists of two main components: a title indicating the version number (highlighted in red) and a detailed body describing changes (highlighted in red box). The body contains specific improvements, bug fixes, and feature additions.

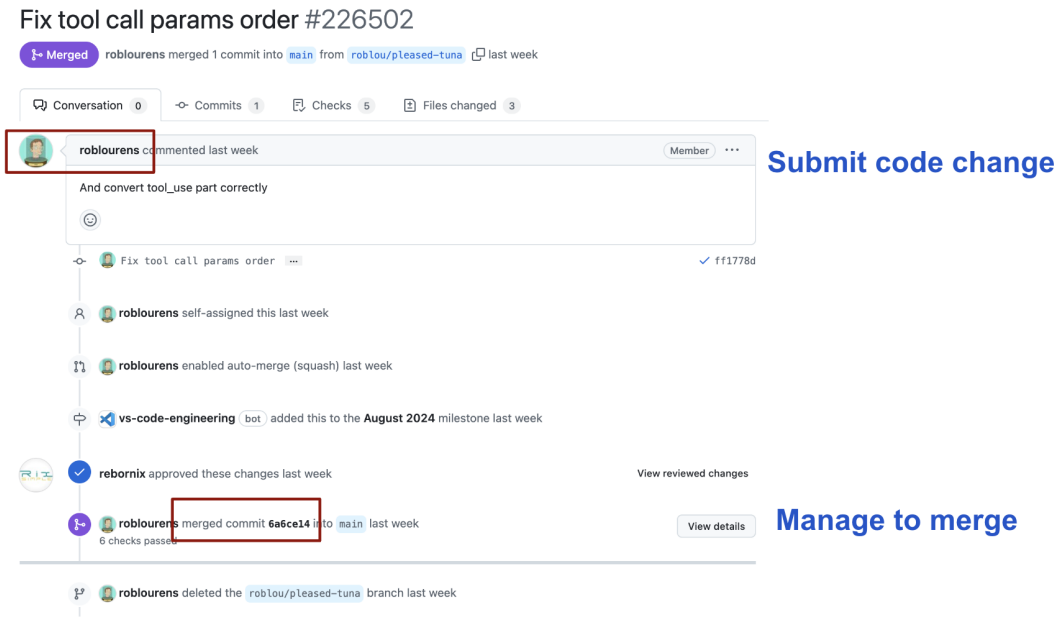
Figure E4: Conversation Example



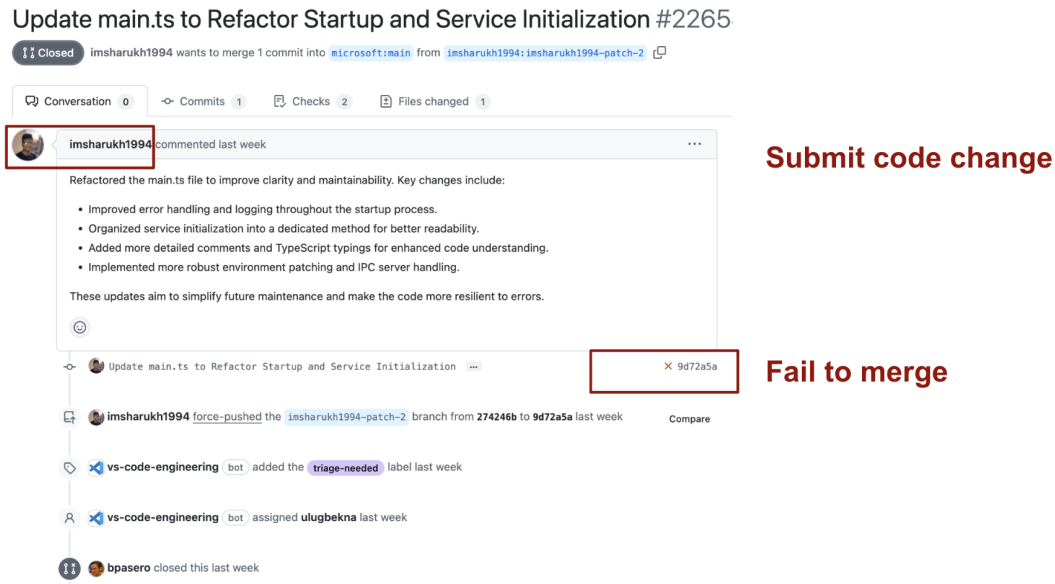
Notes: This figure demonstrates how technical discussions unfold in GitHub's issue tracking system. The example shows a TypeError issue (#226649) and the subsequent problem-solving conversation. The interaction is structured in three parts: (1) initial question/code submission, (2) technical discussions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

Figure E5: Pull Request Merge Example

(a) Manage to merge code

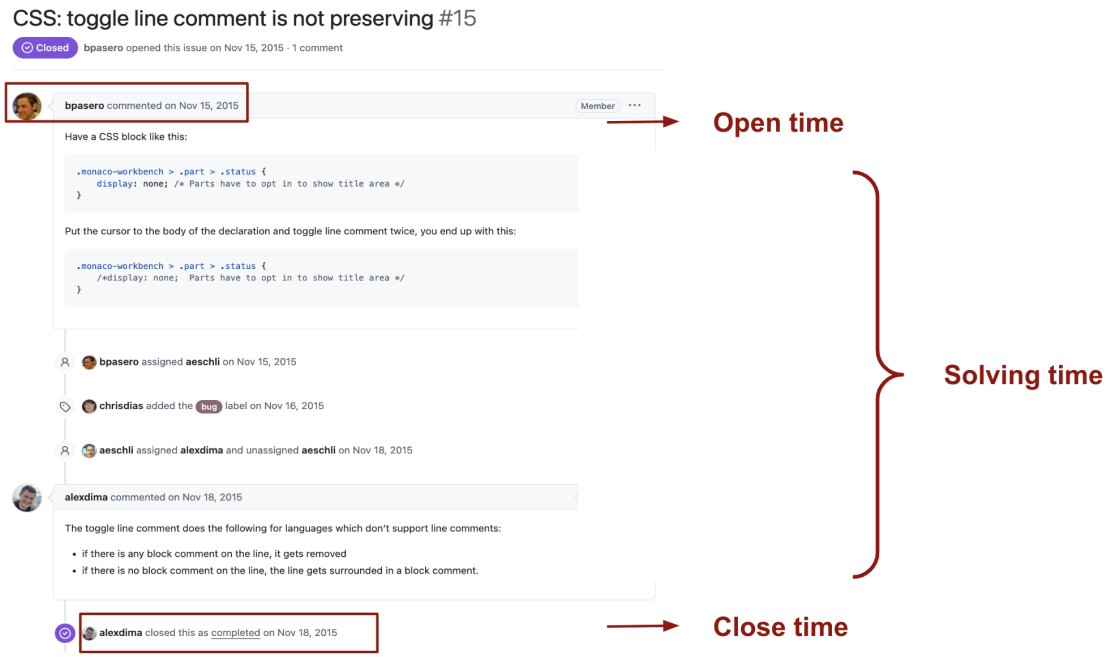


(b) Fail to merge code



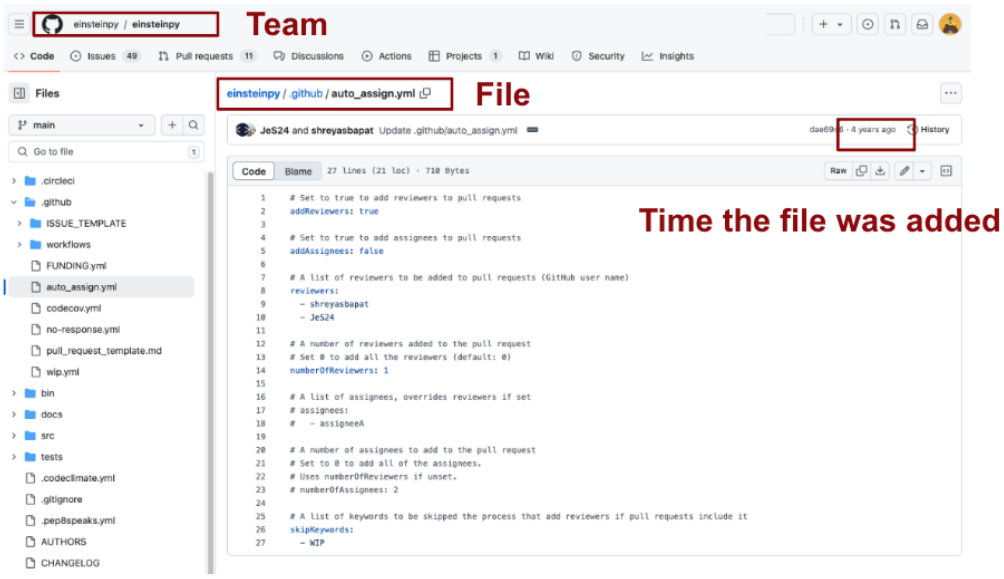
Notes: This figure illustrates two outcomes in the code review process: successful and failed merges. Panel (a) shows a successful merge (#226502) where submitted code changes are approved and integrated into the main codebase. Panel (b) demonstrates a failed merge (#2265) where proposed changes are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

Figure E6: 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 E7: 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.