https://pubsonline.informs.org/journal/isre

INFORMATION SYSTEMS RESEARCH
Articles in Advance, pp. 1–20
ISSN 1047-7047 (print), ISSN 1526-5536 (online)

Workflow Automation in Open-Source Software Development: Accelerating Innovation Through Mechanization and Orchestration

Ao Huang,[a] **Ni Huang,**[a,*] **Yili Hong**[a]

[a] Miami Herbert Business School, University of Miami, Coral Gables, Florida 33146

*Corresponding author

Contact: alanhuang@miami.edu, https://orcid.org/0009-0004-7530-1251 (AH); nhuang@miami.edu, https://orcid.org/0000-0003-3416-513X (NH); khong@miami.edu, https://orcid.org/0000-0002-0577-7877 (YH)

Received: September 29, 2024
Revised: June 18, 2025; October 2, 2025; November 12, 2025
Accepted: November 22, 2025
Published Online in Articles in Advance: January 12, 2026

https://doi.org/10.1287/isre.2024.1551

Abstract. This study develops a conceptual framework distinguishing two foundational mechanisms of workflow automation, *mechanization* and *orchestration*, and empirically examines how workflow automation accelerates innovation in the context of open-source software (OSS) development. Drawing on the process automation, innovation, and OSS development literature, we propose that (a) workflow automation accelerates OSS innovation and (b) distinct workflow automation mechanisms—*mechanization* and *orchestration*—differentially affect incremental innovation (i.e., maintenance) and substantive innovation (i.e., new development), respectively. Based on longitudinal observations of more than 4,500 GitHub repositories and 280,000 issues from 2019 to 2020, we combine machine learning with econometrics in an embedded interlayering multimethod design. To test our hypotheses, our econometric analyses further employ look-ahead rolling entry matching (LA-REM) in tandem with different estimation strategies, including difference-in-differences (DiD) and instrumental variables (IVs). Our analyses first estimate that workflow automation (specifically GitHub Actions) accelerates issue resolution by 10.1%, translating to 4.3 days saved per issue, demonstrating significant improvement in innovation speed. Second, particularly notably, we distinguish between mechanization and orchestration to further investigate how these workflow automation mechanisms differentially affect two types of innovation in software development. Our analyses uncover that mechanization significantly accelerates the incremental innovation of maintenance by 7.8%, saving an average of 3.0 days per issue; however, orchestration shows an inconsequential effect. In contrast, for the substantive innovation of new development, mechanization shows a limited effect, whereas orchestration accelerates issue resolution speed by 16.6%, saving an average of 9.1 days per issue. We further leverage counterfactual estimations to analyze the downstream effects of workflow automation. We observe that the speed gain does not compromise quality, as workflow automation significantly increases the contribution volume (closed issues) and project quality (stars, forks, and releases). This study unpacks the distinct mechanisms of workflow automation, showing that mechanization accelerates routine maintenance-oriented incremental innovation, whereas orchestration catalyzes creative development-oriented substantive innovation. Our study contributes to research and practice on automation, innovation, and software development by demonstrating that workflow automation not only accelerates innovation but also reshapes the underlying processes through which innovation unfolds in OSS development.

History: Gal Oestreicher-Singer, Senior Editor; Min-Seok Pang, Associate Editor.
Funding: This work was supported by OpenAI's Researcher Access Program and the Google Cloud Researcher Award.
Supplemental Material: The online appendices are available at https://doi.org/10.1287/isre.2024.1551.

Keywords: workflow automation • process automation • mechanization • orchestration • innovation speed • incremental innovation • substantive innovation • open-source software (OSS) development

1. Introduction

Open-source software (OSS) development has become a cornerstone of the global software industry, driving large-scale collaborative innovation (Osterloh and Rota 2007, Levine and Prietula 2014). With the meteoric rise of data science and artificial intelligence (AI), OSS is increasingly important given its potential to develop transparent and responsible systems (Mikalef et al. 2025). In this era of rapid technological advancement, automation is profoundly reshaping software development. Much recent attention has focused on generative AI coding tools like GitHub Copilot, which automate content creation by generating code and facilitating auto-completion, thereby impacting developer productivity (Cui et al. 2024, Song et al. 2024, Yeverechyahu et al. 2024). As automation diffuses across industries and permeates nearly every aspect of work—spurring debates over augmentation versus displacement (Acemoglu and Restrepo 2019, Brynjolfsson 2023, Li et al. 2025)—this paper focuses on a less discussed but increasingly critical frontier: automating the *processes* that coordinate and sustain collaboration, commonly known as *workflow automation*. We seek to understand the nature of workflow automation and its implications for OSS development innovation, a domain that heavily depends on distributed creativity and collaborative problem solving.

The origins of workflow automation trace back to seminal information systems (IS) research on process innovation, workflow management systems, and business process reengineering (Davenport 1993, Hammer and Champy 1993). Workflow management systems are designed to manage and optimize sequences of tasks, often integrating diverse tools and human inputs to achieve target outcomes (Georgakopoulos et al. 1995, Basu and Kumar 2002). Contemporary software development has shifted from managing workflows to automating them, exemplified by systems that enable continuous integration (CI) and continuous delivery/deployment (CD) pipelines, which automate the process from code commit to deployment (Maruping and Matook 2020). Beyond CI/CD, platforms like GitHub democratize workflow automation through GitHub Actions (GHA), enabling developers to define, execute, and automate custom workflows within their repositories through configurable, reusable units of code known as actions.

In modern software development, maintaining a competitive edge depends crucially on the speed of innovation (Kessler and Chakrabarti 1996). Rapid technological change and evolving market dynamics make it essential to rapidly innovate by developing new solutions while continuously refining existing code bases. The need to accelerate innovation speed (herein defined as the time required to address crowd-based issue requests) is even more critical in OSS development, because slow bug-fixing or failure to introduce new features can lead to community disengagement and ultimately project abandonment. OSS projects, characterized by decentralized governance, voluntary contributions, and often loosely coupled collaboration structures (Shah 2004, Howison and Crowston 2014, Safadi et al. 2021), offer a rich setting for the investigation of innovation speed, encompassing both incremental innovation (i.e., maintenance) and substantive innovation (i.e., new development). Although OSS characteristics like autonomy and flat hierarchies foster creativity and innovation (Amabile 1988, Perry-Smith and Shalley 2003), they simultaneously impose significant coordination challenges (Lindberg et al. 2016, Mockus et al. 2002), an essential prerequisite for effective collaborative work systems (Crowston 1997). Workflow automation systems (e.g., GHA) hold significant potential to accelerate both maintenance- and development-oriented innovation in OSS projects.

This study conceptually proposes and empirically examines whether—and through what mechanisms—workflow automation accelerates innovation in OSS development. We introduce a new typology of workflow automation mechanisms: *mechanization*, which focuses on automating the execution of discrete tasks that are well defined, self-contained, and repeatable (e.g., compiling code, running tests), and *orchestration*, which focuses on automating the coordination of sequences of tasks or different contributors, management of dependencies between them, and facilitation of information flow across stages (e.g., sending notifications, managing dependency bumps). We consider that these mechanisms have differential impacts on maintenance- and development-oriented innovation. Understanding this distinction is crucial, as the nature of automation best suited for predictable, routine tasks may differ significantly from that required for uncertain, creative endeavors, especially in the less hierarchical, more distributed, and creativity-conducive context. Bearing the above discussion in mind, this work seeks to address the following research questions.

a. *How does workflow automation impact innovation speed in OSS development?*

b. *How do distinct workflow automation mechanisms—mechanization and orchestration—differentially impact the speed of maintenance-oriented incremental innovation and new development-oriented substantive innovation?*

To address these research questions, we analyze longitudinal observations from more than 4,500 GitHub repositories and 280,000 associated issues spanning 2019–2020, combining machine learning with econometrics in an embedded interlayering multimethod bridge design (Sarker et al. 2025). Our econometric analyses further employ look-ahead rolling entry matching (LA-REM) in tandem with complementary estimation strategies: DiD and IVs estimation for the issue-level analysis of innovation speed and matrix completion counterfactual estimations for the project-month panel data analysis of downstream outcomes.

We report novel empirical evidence on the significant impact of workflow automation on OSS innovation speed and its nuanced mechanisms. To begin with, we find that GHA enrollment significantly accelerates overall OSS innovation, reducing average issue resolution time by 10.1% (4.3 days). Further and more notably, through longitudinal observations of GHA configurations, we find differential impacts of the two mechanisms—mechanization and orchestration—on maintenance- and development-oriented innovation.

For maintenance-oriented innovation that involves routine tasks like bug fixes and ongoing patches, mechanization leads to a 7.8% (3.0 days) reduction in resolution time. This suggests that automating repeatable, self-contained, and well-defined jobs is effective for accelerating speed in maintenance-oriented innovation. However, for development-oriented innovation, characterized by novel, complex development work such as creating new features, mechanization shows a limited and insignificant impact. Instead, orchestration yields a substantial 16.6% (9.1 days) reduction in resolution time for development-oriented innovation. This result demonstrates that accelerating novel and complex innovation in OSS development necessitates orchestration, which encompasses automated coordination and communication between steps, developers, and stages. Furthermore, our analysis of downstream effects indicates that workflow automation positively influences community contribution volume, reflected in an 18.2% increase in issues closed. It also enhances project innovation quality, accompanied by a 16.5% increase in stars, a 9.9% increase in forks, and a 15.6% increase in releases.

This work makes several notable contributions to the related literature. First, we add to the emerging literature on software development automation by extending the focus beyond content automation, such as the effects of GitHub Copilot (Song et al. 2024, Yeverechyahu et al. 2024), to examine workflow automation. Our research answers the call from Jain et al. (2021) and extends the work of Anthony et al. (2023) and Adjerid et al. (2023) by exploring the emerging forms of process automation in software development. Distinct from prior research that primarily relies on levels-of-automation frameworks (Parasuraman et al. 2000, Kaber 2018), our study distinguishes between mechanization and orchestration as two pertinent mechanisms of workflow automation and examines their differential effects on incremental and substantive innovation in OSS development. Our findings highlight the importance of aligning workflow automation strategies with specific process needs, particularly emphasizing orchestration in collaborative, innovation-intensive contexts.

Second, for the innovation literature, particularly regarding information systems and digital innovation (Swanson 1994, Fichman et al. 2014), we report novel empirical evidence on how the emerging technology of workflow automation systems can accelerate the speed of both incremental and substantive innovation. Although foundational research has established the importance of information technology (IT) and various enabling factors at the organizational level (Davenport 1993), and recent work has highlighted the importance of coordination for innovation in online collaborative settings (Sherif et al. 2006, Nambisan et al. 2017), the specific effects and mechanisms of these emergent workflow automation technologies on digital innovation remain understudied. By conceptualizing and empirically distinguishing between two workflow automation mechanisms, mechanization and orchestration, we show that, whereas mechanization accelerates maintenance-oriented innovation speed, orchestration is particularly effective for accelerating development-oriented innovation.

Third, for the OSS development literature, our findings offer pioneering evidence on how the workflow automation technology affects OSS innovation speed and quality. Prior work on OSS development emphasized its unique collaborative and often loosely coupled innovation environment (Howison and Crowston 2014, Lindberg et al. 2016), and focused on traditional human collaboration and governance (Shah 2004, Singh et al. 2011). This work adds to the extant literature by furthering our understanding of workflow automation's role in OSS development. In terms of managerial implications, this work also provides actionable takeaways for OSS contributors, maintainers, and platforms on designing workflow automation strategies and toolkits. Although our empirical setting is OSS, the insights into how distinct automation mechanisms accelerate different types of innovation hold potential for generalizability to other innovation-intensive contexts, such as startups or agile teams, which are characterized by high levels of creativity, intensive collaboration, flexible work structures.

2. Related Literature

2.1. Automation and OSS Development

Automation transforms work organization (Acemoglu and Restrepo 2019). Within organizations, process automation—the use of technology to restructure, augment, and expedite business processes and functions—has been investigated across various contexts, including office systems (Millman and Hartwick 1987), automated sales agents (Adam et al. 2023), algorithm-enabled process innovation in healthcare (Adjerid et al. 2023), and workflow management in e-business (Georgakopoulos et al. 1995, Basu and Kumar 2002). Historically, much of this automation can be characterized as mechanization—the substitution of technology for manual labor in executing discrete, routine tasks. This form of automation dominated discussions in manufacturing and other domains throughout the twentieth century (Gunn 1982, Smith 1982). As work grew more complex and interconnected, practitioners began to view automation not only as mechanization but also as orchestration. In fields such as information security and networked technologies, orchestration—coordinating and managing information flows and interdependencies across processes and stakeholders—has gained prominence, whereas mechanization automates individual tasks, orchestration structures workflows, and automates information exchange.

Alongside these developments in process automation, recent years have seen growing attention to content automation driven by advances in AI. Generative AI tools have demonstrated potential to enhance creative productivity and output quality in domains such as digital art (Zhou and Lee 2024) and writing (Doshi and Hauser 2024), albeit outcomes depend on how humans and AI interact and user skill level (Chen and Chan 2024). Within software development, a growing body of research investigates the effects of AI-assisted coding tools on developer productivity and code quality (Cui et al. 2024, Song et al. 2024, Yeverechyahu et al. 2024). Our work approaches automation from a different perspective, focusing not on content automation or levels-of-automation frameworks (Parasuraman et al. 2000, Kaber 2018) but on *process automation*—specifically, how workflow automation systems impact OSS development through the conceptual mechanisms of mechanization and orchestration.

The OSS context provides a particularly valuable setting for examining process automation's impact on collaborative innovation. OSS projects depend heavily on distributed, voluntary participation with limited formal authority structures, making coordination and workflow design especially important (Mockus et al. 2002, Howison and Crowston 2014). Prior IS research has extensively documented how various factors shape OSS trajectories and outcomes, including contributor roles (Setia et al. 2012, Safadi et al. 2021), social network structures (Fershtman and Gandal 2011, Singh et al. 2011), and coordination mechanisms (Lindberg et al. 2016). More recently, research has investigated how organizational engagement and structural characteristics influence OSS development. Medappa and Srivastava (2019) demonstrate that work structure characteristics such as the degree of task superposition, significantly impact project success, with organizational ownership moderating these effects. Similarly, Chen et al. (2022) find that technological acquisitions reshape internal and external OSS contributions, with effects contingent on the acquirer's OSS experience and project similarity. These studies highlight that OSS environments are highly sensitive to coordination challenges. Also building on this literature, we examine how process automation influences innovation speed in OSS, focusing on the differential effects of two mechanisms across two types of innovation.

2.2. Innovation and OSS Development

IS research has long established IT as a critical engine for innovation, providing typologies of IS innovations and recognizing IT as a source of both process and product innovation (Swanson 1994). The contemporary field is being fundamentally reshaped by *digital innovation*—new IT-enabled "products, processes, and business models" that entail substantial changes for organizations, individuals, and other stakeholders (Fichman et al. 2014, p. 2). As software becomes the backbone of innovation across industries, the mechanisms that accelerate software development have become a matter of major economic significance (Branstetter et al. 2019).

The nature and dynamics of innovation evolve within digital platforms and online communities, where sociotechnical factors shape collaborative outcomes (Ramaraju et al. 2025). Knowledge exchange through helping behaviors influences individuals' capacity to generate innovative ideas (Hwang et al. 2019), whereas communication visibility improves collective "metaknowledge," leading to more innovation and less duplicated work (Leonardi 2014). Communication capabilities such as transmission velocity and reprocessability enhance peer connections and access to shared knowledge pools, fostering sustained innovation (Liao et al. 2024), although diverse linkages remain critical for integrating knowledge and managing coordination challenges (Kane and Ransbotham 2016).

OSS development exemplifies this collaborative innovation mode, with decentralized governance and voluntary contribution creating an environment conducive to innovation. However, these characteristics also generate significant coordination challenges—including development interdependencies (e.g., features requiring simultaneous changes across modules) and developer interdependencies (e.g., specialists synthesizing distinct expertise)—that can impede innovation speed (Lindberg et al. 2016). We extend the OSS literature by conceptually proposing and empirically analyzing how workflow automation systems accelerate OSS innovation through two distinct mechanisms: mechanization, which automates discrete routine tasks, and orchestration, which coordinates information flows and task interdependencies. We further examine their differential impacts on maintenance-oriented *incremental* innovation versus development-oriented *substantive* innovation.

3. Hypotheses

This section provides the conceptual background for innovation speed, the nature of maintenance-oriented *incremental innovation*, and new development-oriented *substantive innovation* in the context of OSS development. We also discuss workflow automation and its associated mechanisms in the forms of mechanization and orchestration. Based on these discussions, we propose three hypotheses: the overall effect of workflow automation on OSS innovation speed (Hypothesis 1) and the differential impacts of alternative automation mechanisms on two types of innovation (Hypothesis 2a and Hypothesis 2b).

3.1. Workflow Automation and OSS Innovation Speed

Whereas much of the innovation literature has emphasized innovation volume—the quantity of new products and features introduced over time—speed is an equally critical dimension of innovative performance. Innovation speed refers to the rate at which ideas are transformed into deployed products or features (Kessler and Chakrabarti 1996, Markman et al. 2005). In software development, it parallels the notion of industry clockspeed in the IT sector, where rapid technological change and compressed product cycles demand accelerated innovation processes (Mendelson and Pillai 1998, Carrillo 2005). Innovation speed is particularly vital in software development due to the dynamic nature of digital technologies and the near-zero marginal cost of deployment. Unlike physical products, software can be updated continuously, enabling rapid responses to user feedback, bug fixes, and the rollout of new features. Delays in addressing user needs can lead to dissatisfaction, eroded trust, and eventual migration to alternative solutions, whereas fast iterations help sustain user engagement and long-term competitiveness.

OSS further presents a unique setting characterized by crowd-based, decentralized, voluntary participation (Raymond 1999, AlMarzouq et al. 2005). Contributors are often geographically dispersed, operate without employment relationships, and are driven by a diverse array of motivations beyond financial incentives, including skill enhancement (Hann et al. 2013), reputation building (Wasko and Faraj 2005), and ideological commitments (Lerner and Tirole 2002). Such environments, marked by minimal formal management and hierarchical oversight, inherently possess characteristics—such as autonomy, reduced bureaucracy, and loose coupling—that are widely recognized as conducive to creativity, learning, and the birth of novel ideas (Amabile 1988, Woodman et al. 1993, Perry-Smith and Shalley 2003).

Despite its promise, OSS innovation processes often entail large-scale collaboration (Levine and Prietula 2014) involving heterogeneous and distributed contributors (Setia et al. 2012, Safadi et al. 2021), complicating the innovation process and potentially leading to delays and failure. OSS projects often struggle with heavy reliance on repetitive and burdensome manual work, such as testing, updating, and managing builds and deployments, which creates bottlenecks that slow handoffs and hinder progress. Even when pipelines are introduced, they are frequently fragmented and not unified across contributors, leaving inefficiencies and even conflicts (e.g., “it works on my machine” issue). Furthermore, projects also face broader coordination challenges, including notifications, updates, triggers, triage, managing working sequences and information flows across development stages, and among distributed contributors. It is within this dynamic interplay of burdensome workloads and coordination challenges that workflow automation can make a potentially significant impact.

Although some concerns about initial overhead or setup effort are possible, we contend that modern workflow automation systems such as GHA are well suited to address this challenge. In contrast to traditional systems, the initial cost is often mitigated by the modular and reusable nature of actions, which allow gradual adaptation, and the standardized YAML (YAML Ain’t Markup Language) configuration makes setup and debugging considerably more straightforward. More importantly, the potential efficiency gains from automating repetitive tasks and streamlining coordination are substantial.

In crowd-based environments such as OSS, innovation is typically user driven, initiated through issue requests and culminating in their resolution. Accordingly, an increase in innovation speed manifests in faster resolution of issues. By systematically reducing manual overhead and mitigating coordination bottlenecks, workflow automation is poised to provide a net positive impact, translating these efficiencies into faster issue resolution. Therefore, we propose the following.

Hypothesis 1. *Workflow automation (i.e., GHA enrollment versus no enrollment) accelerates overall innovation speed (i.e., issue resolution) in OSS development.*

Having proposed the overall effect of workflow automation on OSS innovation speed, we now turn to how two distinct mechanisms, mechanization and orchestration, may differentially accelerate specific types of innovation. Maintenance and new development represent two distinct forms of work in software engineering, a distinction well established in the software development and information systems literature (Kriebei and Raviv 1980, Banker et al. 1991). We follow this view and treat both as forms of OSS innovation, consistent with the perspective advanced by Gupta et al. (2006). Building on established distinctions in the innovation literature, we conceptualize maintenance-oriented innovation as aligning with incremental innovation, which emphasizes localized improvements and the preservation of system stability, whereas development-oriented innovation corresponds to substantive innovation, which involves more radical enhancements or expansions in functionality (Dewar and Dutton 1986, Henderson and Clark 1990, Garcia and Calantone 2002). This conceptual framework provides the basis for examining how mechanization and orchestration may shape distinct OSS innovation trajectories.

3.2. Mechanization and Incremental Innovation

Mechanization is defined as using automation tools (e.g., GHA, Travis CI) to execute discrete tasks within the software innovation cycle. These discrete tasks are often well defined, self-contained, and repeatable in nature. The goal of mechanization is typically to enhance consistency, reliability, and efficiency by reducing or replacing manual input for routine development operations, ensuring standardized execution. For example, in the GHA workflow automation system, certain actions can automate tasks such as executing test suites upon code commits, compiling source code into executable artifacts, linting code for style consistency, and automatically deploying code to staging or production environments after successful tests.

Incremental innovation refines existing features, products, and processes through routine and predictable improvements. Incremental innovation sustains the status quo by exploitation (Gupta et al. 2006), ensuring the reliability and stability of the software, maintaining robust and dependable functionality, and achieving efficiencies through established practices and standardization. In software development, incremental innovation manifests through maintenance, typically including issues related to bug fixes, patching, and code refactoring, tasks that typically follow well-defined solution paths. Given the low levels of creative and strategic uncertainty, the speed of incremental innovation is constrained less by intellectual challenges and more by operational inefficiencies. The primary bottleneck lies in the cumulative time, effort, and manual burden required to carry out the routine, mechanizable tasks involved in validating and implementing these changes.

Addressing such operational bottlenecks is precisely where mechanization excels. Rather than requiring large-scale brainstorming, complex coordination, or creative problem-solving, incremental innovation involves discrete, self-contained, and repeatable tasks that machines can execute reliably, such as running test suites, performing style checks, or compiling code. By focusing on the efficient and consistent execution of such known tasks, mechanization emphasizes standardization, automation, and streamlined processing of mechanizable operations. The innovation acceleration here stems from removing the execution bottleneck: absent mechanization, a developer's progress is frequently stalled by the need to manually execute and monitor these essential yet routine tasks. Consider a developer implementing a one-line bug fix: Although the work is minimally intellectual, running a comprehensive regression test suite remains essential for quality assurance. This testing represents pure execution overhead—a time-consuming, context-switching activity that impedes progress despite being routine and repeatable. Mechanization eliminates this bottleneck by offloading such rate-limiting tasks to automated systems. Whereas implementing mechanization may introduce adaptation overhead, we expect the efficiency gains to outweigh the transitional costs. Therefore, we propose the following.

Hypothesis 2a. *Mechanization (i.e., configuring versus not configuring mechanization actions in GHA) accelerates incremental (maintenance-oriented) innovation in OSS development.*

3.3. Orchestration and Substantive Innovation

Orchestration moves beyond discrete task execution and is defined as using automation tools (e.g., GHA, Airflow) to coordinate sequences of tasks and contributors, triage and manage complex dependencies, and facilitate information flow across development stages and team members. This mechanism addresses the collaborative and coordinative complexities of innovation processes (Dhanaraj and Parkhe 2006), encompassing the management of interdependencies that exceed the coordination capacity of arm's-length mechanisms, such as modularity that OSS communities have established to cope with these complexities (Lindberg et al. 2016). For example, in the GHA workflow automation system, orchestration includes actions that notify the pertinent maintainers when issues or pull requests are opened or updated; detect and flag potential dependency updates or conflicts before they cause problems; apply labels and assign tasks based on predefined rules and access permissions; and trigger follow-up tasks when earlier steps fail, tests break, or further action is required.

Substantive innovation involves pursuing new knowledge and capabilities through exploration (Gupta et al. 2006). In software development, substantive innovation manifests through new development, typically including creating new features, implementing architectural transformations, and experimenting with cutting-edge approaches requiring creative and collaborative problem-solving. Successful substantive innovation requires effective coordination mechanisms to integrate diverse expertise and the autonomy that fosters creative experimentation (Amabile 1988, Woodman et al. 1993). The loosely coupled, open nature of OSS creates exceptionally fertile conditions for substantive innovation (Raymond 1999, Perry-Smith and Shalley 2003, Levine and Prietula 2014). OSS projects attract diverse talent pools from core maintainers to peripheral contributors, each bringing unique perspectives and specialized knowledge that spark transformative innovations (Setia et al. 2012, Safadi et al. 2021). However, this same characteristic also generates significant coordination challenges equally vital for new development success. Unlike maintenance tasks with well-defined solution paths, new development frequently encounters unresolved interdependencies of development and developers (Lindberg et al. 2016), where the primary bottleneck limiting speed may not be intellectual capacity but coordination complexity—orchestrating collaborative processes, managing information flow, and integrating distributed knowledge across loosely connected contributors.

Tackling the coordination bottleneck is where orchestration demonstrates its unique value. Unlike mechanization, which automates discrete tasks, orchestration addresses the challenge of coordinating complex collaborative processes by automating sequences of operations, managing dependencies, and facilitating multichannel communication. Workflow orchestration automates intraproject coordination, enhancing conditions for creative contributions by streamlining collaborative aspects of new development (Dhanaraj and Parkhe 2006). This capacity to ensure information reaches the right people at the right time is especially valuable for projects involving contributors with different expertise and less established interaction patterns (Mockus et al. 2002, Perry-Smith and Shalley 2003). Where formal coordination mechanisms are limited (AlMarzouq et al. 2005, Levine and Prietula 2014), orchestration algorithmically structures and manages collaborative processes critical for integrating diverse contributions, where some might bring novel epistemic insights but require timely feedback from others to effectively implement (Setia et al. 2012, Safadi et al. 2021).

Consider an example from the GitHub platform: an issue proposing a novel feature that would transform the project's data processing architecture—a potentially groundbreaking but architecturally disruptive innovation. In the absence of orchestration, these substantive innovation efforts face coordination challenges from the outset: How should the domain experts be coordinated, and how should their communication be structured and tracked? How would dependencies with external code be tracked and updated? Orchestration systematically dissolves these bottlenecks by activating actions that automatically categorize the issue as a major architectural change, notifying relevant core maintainers and domain specialists through appropriate communication channels, establishing dedicated project boards to track both discussion and implementation progress, triggering comprehensive dependency and security scans while notifying pertinent contributors, and ensuring continuous stakeholder updates throughout the innovation lifecycle.

Whereas mechanization automates certain discrete tasks within new development (e.g., testing), orchestration plays a broader role by automating the coordination, dependencies, and information flows crucial for navigating the substantive change in the innovation cycle. Therefore, we propose the following.

Hypothesis 2b. *Orchestration (i.e., configuring versus not configuring orchestration actions in GHA) accelerates substantive (new development-oriented) innovation in OSS development.*

4. Research Methodology

Our overall research design integrates machine learning with econometric analysis and follows an embedded interlayering multimethod "bridge" approach (Sarker et al. 2025). To classify each issue as maintenance or new development, we leverage a machine learning approach that combines GPT-4o–based large language model classification with a fine-tuned, domain-specific model (CodeBERT), supplemented by validations using an alternative model (Gemma-2-2B) and human annotators. The validated innovation classifications enable us to analyze the differential effects of workflow automation mechanisms across distinct types of innovation in the subsequent econometric analyses.

4.1. Research Context

GHA, formally introduced by GitHub in August 2019, serves as the empirical setting for this study.[1] GHA is a powerful and flexible workflow automation system designed to streamline and optimize the diverse processes inherent in the software development lifecycle. It enables developers to create custom automated workflows directly within their GitHub repositories, moving beyond traditional CI and CD to encompass a broader spectrum of development activities.[2] This deep integration within the GitHub ecosystem allows GHA to respond to a wide array of repository events, such as code commits, pull request (PR) creations or updates, issue comments, and release publications.

When GHA was first introduced, it was widely regarded as a novel feature because it enabled the use of reusable code modules (actions) to automate common development tasks. Beyond task execution, GHA also allowed developers to orchestrate workflows by coordinating sequences of activities. Developers define these custom workflows using YAML syntax, specifying sequences of jobs and steps that are triggered by particular events. This capacity for granular workflow automation supports the systematic execution of a multitude of tasks, ensuring consistency, reducing manual overhead, and enabling complex coordinative actions. The versatility of GHA is significantly enhanced by the GitHub Marketplace, which hosts a vast and growing library of prebuilt actions—reusable units of code that perform specific operations. These actions range from traditional CI/CD tasks, such as setting up specific programming language environments, running test suites (e.g., unit, integration, end-to-end tests using frameworks like Jest or PyTest), compiling code, building software artifacts (like Docker images), and deploying applications to various hosting platforms (e.g., Amazon Web Services, Azure, Heroku), to more coordinative and communicative functions. For instance, actions exist to automatically label issues and PRs based on their content or status (e.g., actions/labeler), send notifications to communication platforms like Slack or Microsoft Teams when specific workflow events occur (e.g., a build failure or a successful deployment using actions like slackapi/slack-github-action), manage project boards (e.g., alex-page/github-project-automation-plus), automate the generation of release notes from commit messages, or trigger subsequent workflows based on the outcomes of prior steps, thereby orchestrating a complex chain of events. This rich ecosystem of actions allows developers to not just automate discrete, mechanical tasks but to orchestrate complex sequences involving information handoffs, approvals, and multistage processes as well.

Several key features of GHA make it a particularly suitable context for investigating the nuanced impacts of automation on software development innovation, aligning with our research focus: First, by automating a wide array of tasks through YAML-defined workflows, GHA significantly reduces the need for manual intervention. This feature directly enables the study of mechanization and its impact on innovation speed, such as accelerating the testing and integration of code changes, which is central to understanding how maintenance-oriented innovation is affected.

Second, GHA allows for the creation of customized workflows tailored to project needs. These workflows can comprise multiple jobs, each with numerous steps, executing commands in a precise order and responding to various conditions. This ability to tailor complex automation sequences is crucial for managing both maintenance and new development efforts. It provides the empirical basis for distinguishing between mechanization and orchestration, allowing us to investigate whether and how alternative automation mechanisms support distinct types of OSS innovation—ranging from the incremental nature of maintenance to the substantive demands of new development.

4.2. Data and Measures

4.2.1. Data Sources, Sample Construction. To comprehensively analyze how workflow automation affects software development innovation, we construct data sets spanning January 2019 to December 2020. This period provides a sufficient observational window following GHA's full functionality introduction in August 2019 while avoiding confounding from other automation technologies, such as the launch of GitHub Copilot in June 2021 (Gershgorn 2021).

Our data combines GH Archive's comprehensive GitHub event history with GitHub's Application Programming Interface (API). We construct our sample through a multistage process designed to ensure comparability between treatment and control groups while addressing selection concerns. First, we identify active development projects initiated before 2019 that demonstrate sustained development activity during our observation window and remain active through 2023. Second, we employ a look-ahead strategy to alleviate selection bias (Bapna et al. 2018). We query GitHub's GraphQL API to identify exact GHA enrollment dates by examining commit dates of "github/workflows" directories. Our treatment group comprises repositories enrolling in GHA during 2019–2020, and the control group consists of repositories that did not enroll during this period but eventually enrolled by June 2024. This approach ensures all repositories demonstrate inherent propensity for GHA enrollment, allowing us to focus on differential timing rather than fundamental differences in automation propensity. Third, we implement rolling entry matching (REM) to enhance treatment-control comparability in our staggered enrollment setting. REM addresses the limitations of traditional propensity score matching (PSM) for staggered entry settings (Stuart 2010) by controlling for two dimensions that account for time-varying factors: the propensity to be treated and the relative timing of treatment (Witman et al. 2019). Unlike traditional PSM, which matches at a single point in calendar time, REM aligns treated and control projects based on comparable covariates evaluated at the same relative pretreatment time, ensuring comparability in both characteristics and timing. Online Appendix C provides a detailed comparison of traditional PSM and our LA-REM implementation, along with balance checks using *t*-tests and standardized mean differences (SMD) tests.

4.2.2. Dynamic Workflow Information Extraction and Action Classification. To track the enrollment and evolution of workflow automation, we implement a meticulous data extraction pipeline. This involves traversing all commits associated with the modification of workflow configurations (i.e., changes to the "github/workflows" directory) for each repository in our sample using the GitHub GraphQL API. For each commit, we also extract the complete YAML configuration files to capture the exact automation configuration with GitHub REST API. These YAML files are systematically parsed to identify the actions in use and subsequently classify them according to the mechanization versus orchestration typology we discuss next. To facilitate the empirical analysis, the commit-level workflow data are mapped to the temporal context of each issue to characterize its automation state (e.g., none, mechanization, orchestration, or mechanization and orchestration). Further details of this comprehensive data collection and processing pipeline are available in Online Appendix E.

To empirically evaluate our theoretical distinction between mechanization and orchestration, particularly how they may differentially impact maintenance- and development-oriented innovation, we systematically classify more than 23,600 GitHub Actions. This important first step involves the collection of metadata for all actions available on the GitHub Marketplace. For each action, we gather its functional category tags (e.g., continuous integration, testing, project management, dependency management) as defined by the marketplace. We then map these marketplace category tags into our two-fold theoretical typology. This mapping is guided by an in-depth analysis of each tag's primary functional purpose: whether it primarily serves to automate discrete, mechanical tasks (mechanization) or to facilitate broader information flow, coordination, and collaboration across development processes (orchestration). Actions with two tags that exhibit characteristics of both are classified as belonging to both categories to accurately capture their hybrid nature. This theory-grounded classification ensures that our empirical measures of automation mechanisms align with our conceptual framework. Representative category tags and example actions are presented in Table 1. The detailed classification of all GitHub Marketplace category tags and the theoretical rationale underpinning our mapping process are provided in Online Appendix D.

4.2.3. Innovation Type Classification (Maintenance vs. New Development). To empirically distinguish between incremental innovation (maintenance) and substantive innovation (new development) for the issues in our data set, we develop and execute a multistage classification methodology combining large language model (LLM)-based classification with a fine-tuned domain-specific model. The initial phase involves classifying a random sample of 10,000 GitHub issues (drawn from the full data set of 700,000+ issues before applying LA-REM to ensure diversity) using OpenAI's GPT-4o model. We choose this model for its advanced natural language understanding capabilities for interpreting the nuanced content of software development issues. Following the practice of Yeverechyahu et al. (2024), we prompt the GPT-4o model to classify each issue into one of five operational categories: (1) code development (e.g., feature additions, new capacities, radical improvement), (2) maintenance (e.g., bug fixes, refactoring, dependency updates), (3) documentation and style (e.g., creating or updating documentation, code style changes), (4) testing and quality assurance (e.g., adding or updating tests, version updates), and (5) other (e.g., license changes). We systematically map these five operational categories to our theoretical constructs of maintenance- and development-oriented innovation in software development. Specifically, issues related to maintenance, documentation and style, and testing and quality assurance, which generally focus on incrementally refining or verifying existing aspects of the software, are merged together and classified as `Maintenance`. Issues related to code development, which typically involve creating new functionalities or significantly altering existing ones, are mapped to `New Development`. Issues falling into the other category are classified as `Neither`, representing issues that do not align with a primary focus on neither maintenance nor new development.

To scale our classification approach to more than 700,000 issues, we fine-tune Microsoft's CodeBERT model, a bimodal deep learning model pretrained on both natural language and source code (Feng et al. 2020). Using 10,000 GPT-4o–labeled issues, the model is adapted to our classification task and applied to the full data set. This approach allows for scalable identification of innovation types in software development. To assess robustness, we also apply Google's Gemma-2-2B model and observe highly consistent classifications (Google DeepMind 2024). Table 2 presents the classification results and examples. Online Appendix F provides further details on prompting, theoretical mapping, fine-tuning, and human validation.

4.3. Data Sets and Dependent Variables

We here report detailed descriptions of our primary issue-level data set and our supplementary project-month-level data set. We report summary statistics for the key variables (both raw and log-transformed values) across the two data sets in Table 3.

4.3.1. Issue-Level Repeated Cross-Sectional Data Set. Our primary analysis investigates the impact of workflow automation on OSS innovation speed. Based on our discussion of user-driven innovation in crowd-based OSS platforms (Section 3.1), we capture innovation speed in OSS innovation using issue resolution

Table 1. Classification of Workflow Automation Mechanisms

	Mechanization	Orchestration
Example action tags	`code-quality`, `continuous-integration`, `testing`, `deployment`, `container-ci`, `code-review`, `mobile-ci`	`project-management`, `chat`, `api-management`, `dependency-management`, `open-source-management`, `support`, `deployment-protection-rules`
Example actions	`golangci/golangci-lint-action` `uraimo/run-on-arch-action` `actions/upload-artifact`	`rtCamp/action-slack-notify` `actions/labeler` `kentaro-m/auto-assign-action`

Table 2. Classification of the Nature of Innovation in OSS

	Maintenance	New development
Issue types	Maintenance, documentation and style, testing and quality assurance	Code development
Example issues	- `Fix memory leak causing crashes` - `Resolve database connection timeout errors` - `Refactor duplicate code in utils library` - `Fix CSS styling inconsistencies`	- `Create new GraphQL API architecture` - `Implement Docker containerization` - `Add support for PostgreSQL database` - `Implement OAuth authentication framework`

time, defined as the duration between an issue's submission and its resolution. Each issue and its corresponding resolution time encapsulate the innovation lifecycle for a unit of OSS innovation: from the initial feature request or bug report by a community member, through community discussion and solution development, to the implementation and resolution. Thus, a reduction in issue resolution time serves as a direct and observable measure of accelerated OSS innovation. Accordingly, this issue-level repeated cross-sectional data set, with project and time fixed effects, represents the most appropriate level of analysis for examining innovation speed in OSS development. Specifically, we construct a granular data set of all issues opened within our matched sample repositories during the 2019–2020 study period. In the data set, each issue is uniquely indexed by project p and its sequence i in that project. For each issue, we capture its characteristics (e.g., title, content, labels) and its opening and closing timestamps. We classify each issue as either maintenance or new development, and the procedure is presented in Section 4.2.3.

Another key feature of this data set is our longitudinal tracking of each project's automation configuration. As a workflow is composed of one or more actions, for projects that are enrolled in GHA, we identify the specific action configuration in the workflow when each issue is opened, and further classify the GHA configuration type, that is, none, mechanization, orchestration, or mechanization and orchestration. This allows us to observe within-project temporal variations in GHA configuration, further allowing us to empirically evaluate the effectiveness of alternative workflow automation mechanisms.

Our primary construct of interest, innovation speed, is operationalized as issue resolution time, measured in hours and log-transformed. This dependent variable captures the duration between issue opening and completion. For robustness, we construct alternative measures to account for unclosed issues, including capping unclosed issues at six months, as well as binary indicators for whether an issue is resolved within 30, 60, or 90 days. These robustness check results are reported in Online Appendix G.4. Upon comprehensive issue

Table 3. Descriptive Statistics

	Observations	Mean	Standard deviation	Minimum	Maximum
Project-issue level data set					
Issue resolution time (hours)	280,282	1,014.6	2,132.6	0.0	18,189.8
Issue resolution time (hours, log)	280,282	4.6	2.6	0.0	9.8
Category (issue type)	Observations	Percentage			
Maintenance	208,573	74.4%	–	–	–
New development	71,709	25.6%	–	–	–
Project-month level data set					
Issues closed	109,152	3.8	15.9	0.0	3,317.0
Stars	109,152	15.9	59.1	0.0	6,472.0
Forks	109,152	3.4	13.4	0.0	727.0
Releases	109,152	1.0	4.8	0.0	256.0
Issues closed (log)	109,152	0.8	1.0	0.0	8.1
Stars (log)	109,152	1.5	1.4	0.0	8.8
Forks (log)	109,152	0.8	0.9	0.0	6.6
Releases (log)	109,152	0.4	0.6	0.0	5.5

Notes. Resolution times may appear as zero in summary statistics due to rounding; all values are positive, calculated from timestamps precise to the second level, and then converted to duration in hours, with no true zero-duration resolutions in the data set. In Online Appendix G.4, we report sensitivity tests to exclude possibly trivial issues that were quickly resolved.

classification with LLM, fine-tuning, and human validation, our data set includes issues related to maintenance or new development, which allows us to examine how alternative automation mechanisms differentially affect these distinct types of OSS innovation.

4.3.2. Project-Month-Level Panel Data Set. We begin with the contribution volume, aggregating the number of issues closed at the project-month level. We also include project-month-level measures of innovation outcomes that reflect quality, such as repository stars, forks, and new releases. Stars capture user appreciation, forks reflect developer engagement and collaboration potential, and releases represent tangible innovation milestones. With these project-month-level measures, we seek to comprehensively analyze both contribution patterns and downstream innovation quality outcomes.

4.4. Empirical Strategies

Our empirical identification strategies are tailored to the distinct characteristics of our two data sets, following our LA-REM procedure. For the analysis of issue-level repeated cross-sections with project and time fixed effects, we employ DiD estimations. For the analysis of the project-month-level panel, we use matrix completion (MC) counterfactual estimations.

4.4.1. Issue-Level Analysis with DiD. For our analysis of innovation speed, we employ DiD as our primary identification strategy.[3] We utilize the temporal variation between issue creation times and the project-level enrollment timing of GHA to estimate its impact. The DiD indicator *AfterGHA* equals one for issues opened after its associated project enrolls in GHA; *AfterGHA* equals zero for issues opened before its associated project enrolls in GHA or its associated project did not enroll in GHA during the observational window.[4] Therefore, our baseline specification is represented in Equation (1):

$$\log(\text{Resolution Time}_{pi}) = \mathit{AfterGHA}_{pi} \cdot \beta + \alpha_p + \gamma_{t(pi)} + \epsilon_{pi}. \tag{1}$$

Here, each observation is indexed by pi, where p denotes the project and i indexes each issue within project p. This resembles a repeated cross-sectional data structure. $t(pi)$ denotes the year-month when issue i in project p was opened. The indicator $\mathit{AfterGHA}_{pi}$ equals one if project p had enrolled in GHA at the time $t(pi)$ when issue i was opened in project p and zero otherwise; α_p represents project fixed effects, controlling for time-invariant characteristics of each project, whereas $\gamma_{t(pi)}$ captures time fixed effects, absorbing any period-specific shocks or trends that uniformly affect all issues opened during a given time interval. The coefficient β on the DiD indicator $\mathit{AfterGHA}_{pi}$ estimates the average treatment effect on the treated (ATT), our estimand of interest.

4.4.2. Project-Month-Level Analysis with MC. For our project-month-level analysis, we employ the MC estimator to examine contribution volume and project quality. The MC estimator belongs to a class of counterfactual estimation methods designed for panel data, where treatment timing varies across units. It addresses challenges posed by treatment effect heterogeneity and time-varying confounding by approximating the structure of untreated potential outcomes through a low-rank matrix (Athey et al. 2021). This method is particularly appropriate for our setting, which features staggered treatment timing and heterogeneous treatment effects that could differ by project and evolve over time.

The estimation procedure involves four steps: (a) fitting the model using only untreated observations (project-months before GHA enrollment), (b) using the fitted model to impute counterfactual outcomes for treated observations (project-months after enrollment), (c) calculating individual treatment effects as differences between observed and imputed outcomes for each treated project-month, and (d) aggregating individual effects to estimate ATT. The key feature of this approach is that it builds counterfactuals at the observation level, which in turn allows treatment heterogeneity across different dimensions (Liu et al. 2024). The MC method employs regularized optimization to estimate the low-rank matrix L, balancing model fit with simplicity to avoid overfitting. We use cross-validation to select appropriate regularization parameters and employ block bootstrap methods with clustering at the project level for inference.

5. Empirical Results

This section reports the empirical findings, specifically (a) the impact of workflow automation (i.e., GHA enrollment) on OSS innovation speed (issue resolution time); (b) the two automation mechanisms (mechanization and orchestration) respectively driving the speed lift for different types of software innovation (maintenance and new development); (c) the downstream outcomes of community contribution and innovation quality; and (d) a series of robustness checks.

5.1. OSS Innovation Speed

5.1.1. Parallel Pre-trend. We first report the parallel trends analysis for the issue-level DiD estimation. Given that our matched sample contains more than 4,500 projects with an average of only three issues opened per project each month, as a standard practice, we estimate the relative time units to treatment time in quarters. Quarterly relative time units provide more stable and smooth coefficient estimates, reducing noise from monthly volatility in issue activity and ensuring a sufficient number of issues for each project within each period in the event study. Figure 1 visualizes the estimated coefficients and their 95% confidence intervals at each relative time period. Visual inspection of the figure suggests that issue resolution times for the treated and control projects were parallel before GHA enrollment, showing no discernible pretrend. The associated regression coefficients and a formal pretrend test are reported in Online Appendix G.1.

5.1.2. Main DiD Results. Table 4 reports our main DiD results examining the impact of workflow automation on innovation speed. In addition to the project and time fixed effects, column (2) further includes treatment cohort-specific linear time trends to allow treatment cohorts to follow different pre- and posttreatment trajectories. The results show that workflow automation (GHA) significantly speeds up software development innovation across both specifications, and the two columns show almost identical estimates. Specifically, the coefficient of −0.107 in column (2) indicates that workflow automation reduces issue resolution time by approximately 10.1% ($e^{-0.107} - 1$). Given the average resolution time of 42.3 days (1,014.6 hours) for the OSS projects in our sample, this translates to a reduction of approximately 4.3 days (102.5 hours) per issue. This effect remains robust with our baseline DiD specification (column (1)), with a negligible difference in the effect sizes. This consistent pattern across specifications provides robust evidence that workflow automation improves innovation speed in OSS development, supporting Hypothesis 1.

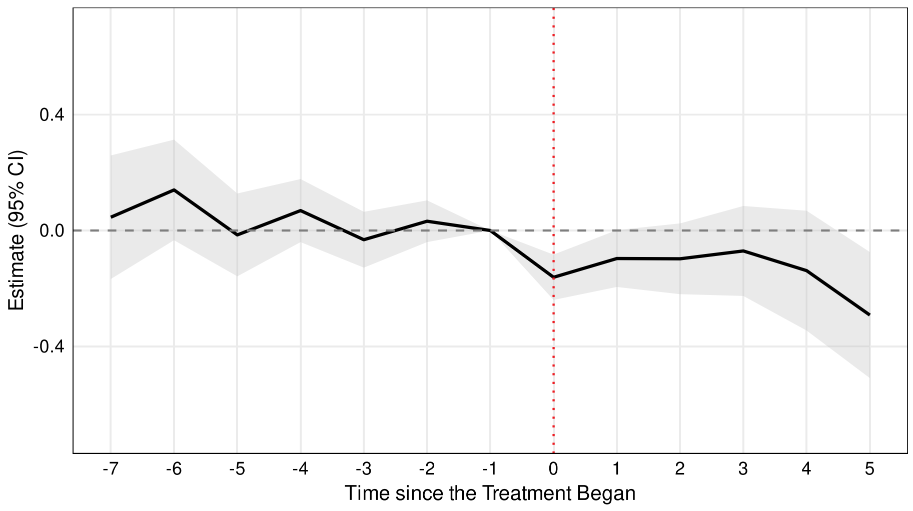
Table 4. Impact of Workflow Automation on Innovation Speed

	Issue resolution time (h, log)	
	(1)	(2)
AfterGHA (β)	−0.106***	−0.107***
	(0.041)	(0.041)
No. of observations	280,282	280,282
R^2	0.177	0.178
Project fixed effects	Yes	Yes
Time fixed effects	Yes	Yes
Treatment cohort linear time trend	No	Yes

Note. Robust standard errors reported in parentheses, clustered at the project level.
***$p < 0.01$.

An average acceleration of 4.3 days in issue resolution represents a meaningful improvement across the open source ecosystem. Although most OSS contributions are voluntary, this time savings translates into real economic value when benchmarked against market labor rates. Based on 1.5 million active projects and a 30% uptake of GHA, we estimate that GHA yields approximately $254.2 million in labor cost savings per month across the GitHub platform (see Online Appendix H for details). Beyond cost savings, faster resolution improves project sustainability by reducing contributor abandonment, maintaining momentum, and enabling quicker iteration—factors especially important in volunteer-driven communities. In fast-moving technological environments, innovation speed can also determine competitive advantage. A 4.3-day acceleration may be the difference between delivering a critical security patch in time or losing users to alternative solutions, particularly for mission-critical OSS projects.

Figure 1. (Color online) Parallel Trends and Dynamic Treatment Effects



Notes. Period −1 is the reference period. Treatment starts from period 0.

5.1.3. Instrumental Variables Approach. To complement the DiD approach for estimating the average effect of workflow automation on innovation speed, we conduct an IV analysis that offers an alternative identification strategy.[5] This IV leverages variation in a project's exposure to GHA that is driven by the external activities of its core development team. Specifically, we construct an instrument at the project-month level that captures the extent to which a project's core team members have participated in GHA-enrolled external projects on GitHub up to each specific month. We weight this exposure measure by each contributor's importance (i.e., proxied by their contribution) to the focal project. The core logic behind this weighted exposure variable is that contributors with greater stake and influence are more likely to successfully advocate for and effectuate the project's enrollment in GHA. This measure is designed to reflect the evolving awareness that core developers bring from their external projects, providing a source of variation in GHA enrollment that is plausibly independent of unobserved shocks to the focal project's development trajectory. For a detailed description of the construction of this instrument, please refer to Online Appendix B.

The validity of our IV strategy rests on two key conditions. First, the relevance condition requires that the instrument is strongly correlated with the likelihood that a project enrolls in GHA. This condition is satisfied by design: As core contributors gain exposure to and experience with GHA in external projects, they become more likely to introduce it into their focal projects, advocating for and facilitating its enrollment. Second, the exclusion restriction demands that the instrument influences innovation speed only through its effect on GHA enrollment. By leveraging variation from contributors' activities in external projects, we minimize the risk that the instrument captures unobserved project-specific shocks that could confound our estimates. Overall, the IV approach provides an alternative identification of the effect of workflow automation on innovation speed.

We report the results of the IV regressions in Table 5. Our analysis follows the same specification as the main DiD approach, including project and time fixed effects. The endogenous variable *AfterGHA* (workflow automation enrollment) is instrumented by core members' weighted exposure to external GHA-enabled projects. The first-stage estimation demonstrates that the instrument is highly relevant, with an *F*-statistic far exceeding the conventional threshold, indicating strong predictive power for GHA enrollment and alleviating concerns about a weak instrument. In the second stage, the coefficient on the instrumented *AfterGHA* variable is negative and statistically significant, which translates to a significant local average treatment effect (LATE) of a lift in innovation speed through the channel of our instrument.[6]

Table 5. Impact of Workflow Automation on Innovation Speed (IV Approach)

	AfterGHA First-stage	Issue resolution time (h, log) Second-stage
IV: GHA Exposure	0.011*** (0.002)	
$\widehat{AfterGHA}$ *(Instrumented)*		−1.123*** (0.205)
First-stage *F*-statistic	6,429.2	
Observations	280,282	280,282
Project fixed effects	Yes	Yes
Time fixed effects	Yes	Yes

Notes. Robust standard errors reported in parentheses, clustered at the project level. Endogenous variable: *AfterGHA*. Instrument: Core members' weighted exposure to external GHA-enabled projects.
***$p < 0.01$.

The IV approach provides complementary identification and further evidence for the impact of workflow automation on innovation speed in OSS development.

5.2. Mechanization vs. Orchestration

To further unpack the nuances and mechanisms underlying our main findings, we examine how different types of GHA configurations influence the speed of incremental and substantive innovation. Our theoretical framework predicts that mechanization should be most effective for maintenance-oriented innovation, which involves well-defined, self-contained, and repeatable tasks, whereas orchestration should particularly benefit development-oriented innovation, which requires coordination across diverse actors and tacit knowledge integration.

As mentioned earlier, our data allow us to observe how the GHA systems are implemented, beyond staggered GHA enrollment. In particular, we observe the evolution of GHA configurations over time for each project, with different issues created under different automation configurations. For example, we may observe that there was no GHA configuration (i.e., no actions) when the issue #10 of the project was created; and for issue #15, the GHA configuration may include actions that are only related to mechanization; and for issue #20, the GHA configuration may consist of actions related to both mechanization and orchestration. In practice, projects rarely implement orchestration in isolation; it typically builds on mechanization to coordinate more complex interactions. Consistent with this observation, we exclude issues from projects that were configured only with orchestration actions, which account for just 0.09% of the total.

We use the same issue-level repeated cross-sectional data set to examine how different mechanisms of GHA affect the resolution time of two types of issues. To guide our analysis, we construct a categorical variable, automation type, with three levels: no automation (reference), mechanization only, and mechanization combined with orchestration. We then split the sample by issue type, distinguishing between maintenance and new development issues. For each issue type, we estimate the following model specification:

$$\begin{aligned}&\log(\text{Resolution Time}_{pi})\\&= \beta_1 \cdot \text{Mechanization}_{pi} + \beta_2 \cdot \text{Mechanization\&Orchestration}_{pi}\\&\quad + \alpha_p + \gamma_{t(pi)} + \epsilon_{pi}.\end{aligned} \qquad (2)$$

In Equation (2), most of the notations are the same as Equation (1), except that we interpret β_1 as the impact of mechanization, and β_2 as the joint impact of mechanization and orchestration. The difference $\beta_2 - \beta_1$ represents the effect of orchestration. It is worth noting that whereas the treatment variation in the main analysis (GHA enrollment) occurs at the project level, in this analysis, it operates at the issue level by design (GHA configuration). Following Abadie et al. (2023) and Angrist and Pischke (2009), who consider standard errors as a design issue and should be constructed at the level of treatment variation, we report the robust standard errors at the issue level. We also report results with alternative standard error clustering for all the analyses in Online Appendix G.

Table 6 presents the results of this mechanism analysis using our earlier specification with project and time fixed effects. For maintenance-oriented incremental innovation, mechanization speeds up issue resolution by 7.8% ($e^{-0.081} - 1$). Given the average resolution time of 37.9 days (908.8 hours) for maintenance-oriented innovation, this represents approximately 3.0 days (71 hours) saved per issue. This result provides empirical support for Hypothesis 2a, attesting to the pronounced benefits of mechanization in the predictable and standardized context of maintenance work, where automation can effectively streamline repetitive tasks like unit testing. When we look further at the role of orchestration in maintenance-oriented innovation, we observe that the difference between these coefficients is insignificant ($\beta_2 - \beta_1 = -0.052$, $t = 1.44$, $p > 0.1$), indicating that orchestration provides a limited impact for maintenance-oriented innovation beyond what mechanization alone can achieve. This finding is consistent with our theoretical expectations, as the coordination and knowledge-sharing benefits of orchestration are less critical for tasks that are self-contained and already well defined.

Table 6. Different Workflow Automation Mechanisms and Innovation Speed

	Issue resolution time (h, log)	
	Maintenance	New development
Mechanization (β_1)	−0.081***	−0.037
	(0.023)	(0.039)
Mechanization	−0.133***	−0.218***
+ *Orchestration* (β_2)	(0.028)	(0.050)
No. of observations	208,079	71,480
R^2	0.174	0.261
Project fixed effects	Yes	Yes
Time fixed effects	Yes	Yes

Notes. Robust standard errors are reported in parentheses. No automation is the reference group.
***$p < 0.01$.

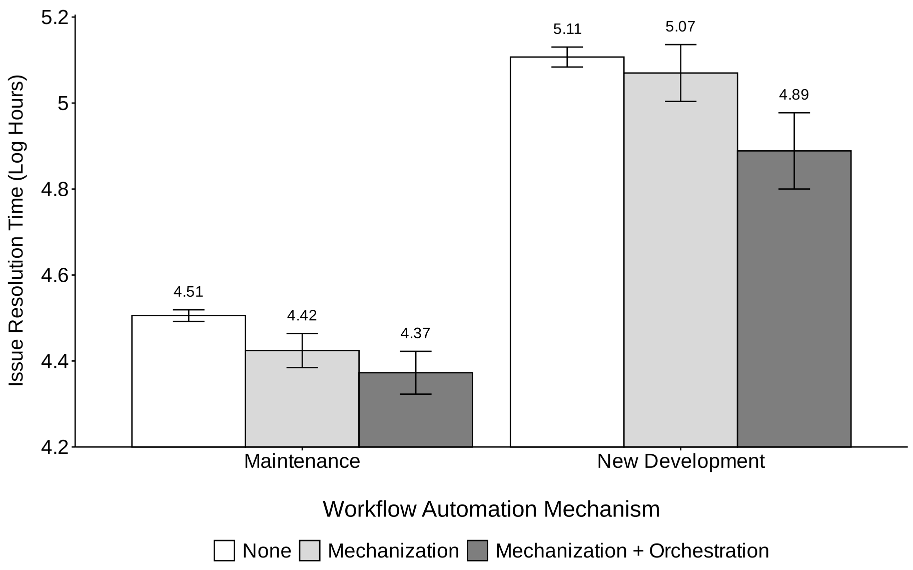
The results for development-oriented substantive innovation reveal a different pattern. Mechanization alone shows no significant effect on development issues (−3.6%, $p > 0.1$), suggesting that mechanization alone provides limited value for work that requires novelty and creativity. This supports the premise of Hypothesis 2b: For development-oriented innovation, the core challenge is not the execution of discrete, mechanizable steps but the management of complex interdependencies and knowledge flows, which mechanization alone does not address. However, when mechanization is combined with orchestration, the effect becomes dramatically larger: a 19.6% ($e^{-0.218} - 1$) reduction in resolution time ($p < 0.01$). The difference between these coefficients is statistically significant ($\beta_2 - \beta_1 = -0.181,\ t = 2.85,\ p < 0.01$), indicating that workflow orchestration is the key driver of speed lift for these tasks. We further compute the economic effects of the contribution of orchestration for accelerating development-oriented innovation, which translates to an innovation speed lift of 16.6% ($e^{-0.181} - 1$), or an average reduction of approximately 9.1 days (219.5 hours) in issue resolution.

Bearing the above discussion in mind, we summarize the differential impact of automation mechanisms on maintenance- and development-oriented innovation here. Figure 2 compares the three groups of issues with different workflow automation configurations, visualizing the predictive margins and their 95% confidence intervals. The respective effects are further broken down by two types of OSS innovation. The height of the bars represents predictive margins in log(hours) from Equation (2), with standard errors calculated using the delta method. Whereas the primary driver of the speed lift for incremental innovation of maintenance is mechanization, substantive innovation of new development benefits significantly more from orchestration. This pattern aligns with our Hypothesis 2a and Hypothesis 2b: Maintenance benefits from task mechanization that removes repetitive bottlenecks, whereas new development requires workflow orchestration that facilitates communication and coordination.

5.3. Downstream Outcomes

A natural question arises as to whether the observed increases in innovation speed occur at the expense of

Figure 2. Issue Resolution Time by Innovation Type and Automation Mechanism



other project outcomes, such as the contribution volume or project quality. To examine the existence or absence of such potential tradeoffs associated with innovation speed gains, we employ the matrix completion approach to empirically estimate the effects of GHA enrollment on a set of downstream project outcomes, with the estimation results reported in Table 7.

The findings reveal positive effects on the volume of community contributions and the overall quality of projects. Starting with community contribution, GHA enrollment significantly increases the volume of closed issues by 18.2%, indicating enhanced capacity to resolve community requests. For project quality outcomes, GHA enrollment increases project stars by approximately 16.5%, indicating enhanced community recognition and appreciation. Similarly, project forks increase by 9.9%, suggesting greater developer interest in building on automated projects. Most notably, project releases increase by 15.6%, indicating shorter development cycles.

These results provide evidence that workflow automation accelerates innovation speed without compromising project quality. Projects with GHA can handle larger volumes of community issues without proportionally increasing the workload on developers. The scaling capability of workflow automation systems is particularly valuable in OSS environments where coordination and communication are often the bottlenecks limiting project growth. The improvements in project quality—more stars, forks, and releases—indicate that these operational enhancements translate into tangible project success. Projects that can resolve issues faster are better positioned to maintain momentum, attract users and contributors, and achieve their developmental goals. The increased release frequency suggests that workflow automation enables more ambitious development cycles, while the growth in stars and forks reflects enhanced project visibility and recognition within the OSS community.

5.4. Robustness Checks

To assess the robustness of our findings, we conduct an extensive set of additional analyses, summarized in Table 8 and detailed in online corresponding appendices. All robustness checks, except the final two, relate to the innovation speed analyses, which use a repeated

Table 7. Impact of Workflow Automation on Downstream Outcomes (MC Estimation)

	Issues closed (log)	Stars (log)	Forks (log)	Releases (log)
AfterGHA	0.167***	0.153***	0.094***	0.145***
	(0.014)	(0.016)	(0.010)	(0.011)
Method	MC	MC	MC	MC
Treated observations	18,504	18,504	18,504	18,504
No. of observations	109,152	109,152	109,152	109,152
Project fixed effects	Yes	Yes	Yes	Yes
Time fixed effects	Yes	Yes	Yes	Yes

Note. Robust standard errors reported in parentheses, clustered at the project level.
***$p < 0.01$.

Table 8. Summary of Robustness Checks

Concern	Analysis	Details
DiD assumption	Event study plot, relative time model, pretrend tests	Section 5.1, Online Appendix G.1
Sample composition	Weighted versus unweighted specifications	Online Appendix A
Endogeneity	Instrumental variables with external GHA exposure	Online Appendix B, Section 5.1.3
Issue composition	Checks on issue compositional changes	Online Appendix G.2
Model specification	Alternative specifications (negative binomial, Cox hazards models)	Online Appendix G.3
Outcome measurement	Resolution time with capping, closure probability	Online Appendix G.4
Sample outliers	Excluding issues with quick resolution times and bots	Online Appendix G.5
DiD modeling	Alternative DiD estimators (CSDiD, 2SDiD, BJS estimators)	Online Appendix G.6
Spurious correlation	Permutation test with random reassignments	Online Appendix G.7
Sample sensitivity	Leave-out sensitivity tests (random and 10-fold)	Online Appendix G.8
Treatment granularity	Timestamp-based versus project-month level	Online Appendix G.9
Classification scheme	Gemma-2-2B issue categorization, human validation	Online Appendix F, Online Appendix G.10
Standard errors	Alternative standard error clustering at project-month level	Online Appendix G.11
MC assumption	Event study plot, pretrend tests	Online Appendix G.12
DiD modeling	Alternative DiD estimators for downstream outcomes (TWFE DiD, 2SDiD, CSDiD)	Online Appendix G.13

cross-sectional data set at the issue level. The final two checks pertain to the downstream outcomes analyses, based on a panel data set structured at the project-month level. Collectively, these robustness checks affirm the robustness of our findings across identification assumptions, estimation methods, and sample configurations.

6. Discussion

6.1. Key Findings

This study conceptually proposes and empirically evaluates whether workflow automation accelerates OSS innovation and its associated mechanisms (mechanization and orchestration). Using a multimethod approach integrating machine learning and econometrics, we analyze a large-scale data set of more than 4,500 GitHub repositories and 280,000 issues. Leveraging LA-REM with DiD, our study shows that workflow automation accelerates OSS innovation, and this impact is nuanced, varying with the automation mechanism and the type of OSS innovation.

Our first key finding is that workflow automation (GHA) leads to a notable acceleration in overall innovation speed, with a 10.1% reduction in issue resolution time, translating to an average saving of approximately 4.3 days per issue. Second, following our conceptualization that distinguishes between *mechanization* and *orchestration*, we empirically demonstrate these two distinct mechanisms of workflow automation to differentially impact incremental and substantive innovation. Specifically, mechanization primarily benefits maintenance-oriented innovation, reducing associated issues' resolution time by an average of 7.8% (3.0 days per issue). In contrast, for development-oriented innovation, mechanization shows a negative yet not significant effect (−3.6%), indicating its limited value for novel, creative work. However, orchestration demonstrates a significant impact for accelerating development-oriented innovation, contributing to a 16.6% (9.1-day) reduction in average issue resolution time. Third, the lift in innovation speed does not compromise key project-wise outcomes, such as overall output and quality. Instead, workflow automation delivers benefits that extend beyond innovation speed improvement. Specifically, our panel-data counterfactual estimations indicate workflow automation also leads to substantial growth in community contribution capacity: Issues closed increase by 18.2%, demonstrating that GHA enhances projects' ability to process community requests efficiently. Additionally, workflow automation leads to marked improvements in key quality indicators: Community stars increase by 16.5%, developer forks grow by 9.9%, and software releases rise by 15.6%. These outcomes reveal that workflow automation generates tangible outcomes beyond innovation speed, such that projects with GHA see greater participation volume and generate higher-quality outputs.

6.2. Implications for Research and Practice

This research makes several notable contributions to the literature on software development automation. First, we contribute a process automation perspective that complements the emerging focus on content automation in software engineering (Song et al. 2024, Yeverechyahu et al. 2024). Rather than treating automation as a monolithic or purely substitutional construct that replaces human input (Raisch and Krakowski 2021, Gong and Png 2024), we conceptualize workflow automation in OSS as comprising two distinct but complementary mechanisms: mechanization and orchestration. This decomposition helps clarify how automation shapes innovation dynamics beyond task execution alone. We also move beyond level-based automation frameworks, such as the continuum from low to high levels of automation (Parasuraman et al. 2000, Kaber 2018), by emphasizing conceptual differentiation between mechanisms rather than degrees of automation.

Second, our research contributes to the innovation literature by examining how an emerging technology—workflow automation—serves as a novel antecedent of software innovation. Although prior work has established IT's enabling role in innovation broadly (Swanson 1994, Fichman et al. 2014, Guo et al. 2023) and highlighted the coordination needs that shape digital innovation outcomes (Sherif et al. 2006, Nambisan et al. 2017, Hwang et al. 2019), we focus specifically on how different workflow automation mechanisms affect the speed of innovation within OSS projects. We introduce a new typology—mechanization and orchestration—to conceptually differentiate automation mechanisms, and we contextualize the distinction between incremental and substantive innovation by mapping it to the software setting as maintenance-focused versus development-focused innovation (Dewar and Dutton 1986, Henderson and Clark 1990). Our empirical analysis demonstrates that mechanization aligns with incremental, maintenance-oriented innovation involving discrete and repeatable tasks, whereas orchestration supports more substantive, development-oriented innovation requiring creativity and coordination. In emphasizing innovation speed, our study also uncovers the temporal dimension of innovation, which remains underexplored in software development research (Kessler and Chakrabarti 1996). Although automation is often framed in terms of operational efficiency, our findings reveal its broader potential to expand innovation capacity by enabling organizations to manage routine operations while simultaneously advancing complex development work.

Third, our study enriches the OSS development literature. Our findings provide implications for how process automation technologies can resolve the fundamental tension between innovation potential and coordination challenges in crowd-based, distributed development environments. The literature has extensively documented how open, decentralized collaboration structures create fertile ground for innovation through characteristics such as openness, autonomy, and loose coupling (Raymond 1999, Levine and Prietula 2014). However, these same characteristics that enable creativity also generate significant coordination and communication challenges that can constrain innovative potential (Mockus et al. 2002, Howison and Crowston 2014, Lindberg et al. 2016), particularly when OSS projects involve heterogeneous contributors (Setia et al. 2012, Safadi et al. 2021). Through our distinction between mechanization and orchestration, we demonstrate how different automation mechanisms can address specific execution overload and coordination challenges. Most notably, our findings suggest that workflow orchestration has the potential to reduce coordination failures and communication bottlenecks in loosely coupled development environments without constraining the flexibility and openness that make OSS conducive to innovation. The downstream effects we observe, including more community engagement and enhanced project quality, provide further evidence that workflow automation may create a virtuous cycle that enhances both innovation speed and quality.

Practically, our findings offer useful guidance for aligning automation strategies with innovation goals. For OSS projects, maintainers and developers aiming for stability and incremental improvements can benefit from mechanizing tasks like testing, formatting, and deployment. In contrast, when OSS projects move toward more exploratory or product development phases—such as building new features or restructuring architecture—orchestration becomes more valuable, helping to streamline coordination. For organizations focused on rapid innovation—such as start-ups, research and development teams, or agile units—our results underscore the strategic role of orchestration-based automation. These tools can facilitate complex, creative work by enabling tighter feedback loops and more adaptive coordination during uncertain, high-velocity development cycles. More broadly, the mechanization–orchestration distinction extends beyond OSS. Although mechanization maps to the automation of routine operational work (e.g., data entry, file processing, standardized calculations), orchestration captures the automation of coordination and managerial functions (e.g., approvals, scheduling, routing, cascading updates). Recognizing this functional differentiation is critical to understanding the nuanced impact of automation on knowledge work and carries important implications for the future of work design.

6.3. Limitations and Future Research

We acknowledge several limitations of this study, which also point to fruitful avenues for future research. First, our findings are based on econometric analyses of archival data. Given the observational nature of our research design, we are unable to manipulate GHA enrollment or configuration, which limits the causal interpretation of the findings. Whereas we leverage complementary identification strategies (DiD and IV) to mitigate some of the common endogeneity concerns, we cannot completely rule out potential unobserved confounding factors, such as potential self-selection regarding GHA enrollment decisions and issues with the stable unit treatment value assumption. Future research could leverage experiments, where feasible, to validate and extend our findings. Second, our empirical investigation is situated in the unique context of OSS development, and thus, we caution readers against generalizing our findings to other settings without accounting for contextual nuances. It is notable that OSS development features crowd-based loose coupling and operates with minimal organizational hierarchies. Future studies can explore the applicability of our results to other organizational settings. Third, workflow automation holds significant potential to transform traditional business processes beyond software development. With firms such as Zapier and UiPath offering advanced automation solutions tailored to nontechnical domains, future research could leverage data from these types of platforms to systematically assess their broad organizational impacts.

In an increasingly automated world, accelerating innovation requires not only the *mechanization* of task execution but also the *orchestration* of collaborative workflows. This study distinguishes between these two mechanisms, demonstrating their differential impacts on two types of innovation. As automation systems assume more routine operational tasks, the locus of human contribution may shift toward activities that benefit from orchestrated coordination and complex problem solving. Our findings suggest that workflow automation, particularly orchestration, plays a critical role in navigating innovation's operational and collaborative demands. Notably, the boundary between mechanization and orchestration is historically contingent: What once required orchestration may, over time, become mechanizable. As workflow automation becomes increasingly integral to collaborative creation, we hope this study, by distinguishing mechanization and orchestration as two fundamental logics of workflow automation, provides a conceptual foundation for future research on automation in software development and, more broadly, in digitally mediated knowledge work.

Endnotes

[1] GHA was launched in limited public beta in October 2018 (Daigle 2018). Beginning in August 2019, GHA supports CI/CD and is free for all public projects (Friedman 2019).

[2] Although sharing certain functionalities with established CI/CD tools like Travis CI and Circle CI, GHA features native integration within the GitHub platform. Further, GHA not only manages build, test, and deployment pipelines but also automates a wider array of repository events and interactions, such as directly managing issues, pull requests, labels, assignments, comments, and triggering notifications and feedback based on specific events.

[3] It is worth noting that our DiD data set resembles a data structure with repeated cross sections rather than a conventional unit-time panel. We observe different issues and their associated resolution time within projects. Therefore, certain panel-based methods, such as counterfactual estimators that require unique project-month observational units, are not suitable for this purpose. The DiD method, however, is well suited to leverage the project-level staggered variations in GHA enrollment to estimate the effects of GHA.

[4] *AfterGHA* is equivalent to the DiD expression $Treat_p \cdot After_{p,t(pi)}$, where $Treat_p$ indicates whether project p enrolled in GHA during our observation period, and $After_{p,t(pi)}$ is a dummy variable equal to one if issue pi was opened after project p's GHA enrollment. The issue opening time is uniquely determined by issue pi, which we denote as $t(pi)$.

[5] We thank the associate editor and an anonymous reviewer for suggesting this instrument.

[6] The IV strategy here identifies the LATE for the subset of projects that enabled GHA solely as a result of their core members' prior exposure to the feature. Because these complier projects are likely to be more responsive—having core members likely already acquired the knowledge needed to effectively implement automation—the LATE estimate is expected to be larger than the ATE or ATT obtained from DiD or FE estimates.

References

Abadie A, Athey S, Imbens GW, Wooldridge JM (2023) When should you adjust standard errors for clustering? *Quart. J. Econom.* 138(1):1–35.

Acemoglu D, Restrepo P (2019) Automation and new tasks: How technology displaces and reinstates labor. *J. Econom. Perspective* 33(2):3–30.

Adam M, Roethke K, Benlian A (2023) Human vs. automated sales agents: How and why customer responses shift across sales stages. *Inform. Systems Res.* 34(3):1148–1168.

Adjerid I, Ayvaci MUS, Özer Ö (2023) Value of algorithm-enabled process innovation: The case of sepsis. *Manufacturing Service Oper. Management* 25(4):1545–1566.

AlMarzouq M, Zheng L, Rong G, Grover V (2005) Open source: Concepts, benefits, and challenges. *Comm. Assoc. Inform. Systems* 16(1):756–784.

Amabile TM (1988) A model of creativity and innovation in organizations. *Res. Organ. Behav.* 10(1):123–167.

Angrist JD, Pischke J-S (2009) *Mostly Harmless Econometrics: An Empiricist's Companion*, 1st ed. (Princeton University Press, Princeton, NJ).

Anthony C, Bechky BA, Fayard A-L (2023) "Collaborating" with AI: Taking a system view to explore the future of work. *Organ. Sci.* 34(5):1672–1694.

Athey S, Bayati M, Doudchenko N, Imbens G, Khosravi K (2021) Matrix completion methods for causal panel data models. *J. Amer. Statist. Assoc.* 116(536):1716–1730.

Banker RD, Datar SM, Kemerer CF (1991) A model to evaluate variables impacting the productivity of software maintenance projects. *Management Sci.* 37(1):1–18.

Bapna R, Ramaprasad J, Umyarov A (2018) Monetizing freemium communities: Does paying for premium increase social engagement? *MIS Quart.* 42(3):719–736.

Basu A, Kumar A (2002) Research commentary: Workflow management issues in e-business. *Inform. Systems Res.* 13(1):1–14.

Branstetter LG, Drev M, Kwon N (2019) Get with the program: Software-driven innovation in traditional manufacturing. *Management Sci.* 65(2):541–558.

Brynjolfsson E (2023) The turing trap: The promise & peril of human-like artificial intelligence. *Augmented Education in the Global Age* (Routledge, New York), 103–116.

Carrillo JE (2005) Industry clockspeed and the pace of new product development. *Production Oper. Management* 14(2):125–141.

Chen Z, Chan J (2024) Large language model in creative work: The role of collaboration modality and user expertise. *Management Sci.* 70(12):9101–9117.

Chen W, Jin F, Xue L (2022) Flourish or perish? The impact of technological acquisitions on contributions to open-source software. *Inform. Systems Res.* 33(3):867–886.

Crowston K (1997) A coordination theory approach to organizational process design. *Organ. Sci.* 8(2):157–175.

Cui Z, Demirer M, Jaffe S, Musolff L, Peng S, Salz T (2024) The effects of generative AI on high skilled work: Evidence from three field experiments with software developers. Preprint, submitted September 5, http://dx.doi.org/10.2139/ssrn.4945566.

Daigle K (2018) GitHub Actions: Built by you, run by us. https://github.blog/enterprise-software/automation/action-demos/.

Davenport TH (1993) *Process Innovation: Reengineering Work through Information Technology* (Harvard Business School Press, Boston).

Dewar RD, Dutton JE (1986) The adoption of radical and incremental innovations: An empirical analysis. *Management Sci.* 32(11): 1422–1433.

Dhanaraj C, Parkhe A (2006) Orchestrating innovation networks. *Acad. Management Rev.* 31(3):659–669.

Doshi AR, Hauser OP (2024) Generative AI enhances individual creativity but reduces the collective diversity of novel content. *Sci. Adv.* 10(28):eadn5290.

Feng Z, Guo D, Tang D, Duan N, Feng X, Gong M, Shou L, et al. (2020) CodeBERT: A pre-trained model for programming and natural languages. Cohn T, He Y, Liu Y, eds. *Findings of the Association for Computational Linguistics: EMNLP 2020* (Association for Computational Linguistics, Stroudsburg, PA), 1536–1547.

Fershtman C, Gandal N (2011) Direct and indirect knowledge spillovers: The "social network" of open-source projects. *RAND J. Econom.* 42(1):70–91.

Fichman RG, Dos Santos BL, Zheng ZE (2014) Digital innovation as a fundamental and powerful concept in the information systems curriculum. *MIS Quart.* 38(2):329–354.

Friedman N (2019) GitHub Actions now supports CI/CD, free for public repositories. https://github.blog/news-insights/product-news/github-actions-now-supports-ci-cd/.

Garcia R, Calantone R (2002) A Critical look at technological innovation typology and innovativeness terminology: A literature review. *J. Production Innovation Management* 19(2):110–132.

Georgakopoulos D, Hornick M, Sheth A (1995) An overview of workflow management: From process modeling to workflow automation infrastructure. *Distribution Parallel Databases* 3(2): 119–153.

Gershgorn D (2021) GitHub and OpenAI launch a new AI tool that generates its own code, Accessed December 19, 2025, https://www.theverge.com/2021/6/29/22555777/github-openai-ai-tool-autocomplete-code.

Gong J, Png IPL (2024) Automation enables specialization: Field evidence. *Management Sci.* 70(3):1580–1595.

Google DeepMind (2024) Gemma 2: Improving open language models at a practical size. Preprint, submitted July 31, https://arxiv.org/abs/2408.00118.

Gunn TG (1982) The mechanization of design and manufacturing. *Sci. Amer.* 247(3):114–130.

Guo F, Li Y, Maruping LM, Masli A (2023) Complementarity between investment in information technology (IT) and IT human resources: Implications for different types of firm innovation. *Inform. Systems Res.* 34(3):1259–1275.

Gupta AK, Smith KG, Shalley CE (2006) The interplay between exploration and exploitation. *Acad. Management J.* 49(4):693–706.

Hammer M, Champy J (1993) *Reengineering the Corporation: A Manifesto for Business Revolution*. (Harpercollins Publishers, New York).

Hann IH, Roberts JA, Slaughter SA (2013) All are not equal: An examination of the economic returns to different forms of participation in open source software communities. *Inform. Systems Res.* 24(3):520–538.

Henderson RM, Clark KB (1990) Architectural innovation: The reconfiguration of existing product technologies and the failure of established firms. *Admin. Sci. Quart.* 35(1):9–30.

Howison J, Crowston K (2014) Collaboration through open superposition: A theory of the open source way. *MIS Quart.* 38(1):29–50.

Hwang EH, Singh PV, Argote L (2019) Jack of all, master of some: Information network and innovation in crowdsourcing communities. *Inform. Systems Res.* 30(2):389–410.

Jain H, Padmanabhan B, Pavlou PA, Raghu TS (2021) Editorial for the special section on humans, algorithms, and augmented intelligence: The future of work, organizations, and society. *Inform. Systems Res.* 32(3):675–687.

Kaber DB (2018) Issues in human–automation interaction modeling: Presumptive aspects of frameworks of types and levels of automation. *J. Cognitive Engrg. Decision Making* 12(1):7–24.

Kane GC, Ransbotham S (2016) Research note—Content and collaboration: An affiliation network approach to information quality in online peer production communities. *Inform. Systems Res.* 27(2):424–439.

Kessler EH, Chakrabarti AK (1996) Innovation speed: A conceptual model of context, antecedents, and outcomes. *Acad. Management Rev.* 21(4):1143–1191.

Kriebei CH, Raviv A (1980) An economics approach to modeling the productivity of computer systems. *Management Sci.* 26(3): 297–311.

Leonardi PM (2014) Social media, knowledge sharing, and innovation: Toward a theory of communication visibility. *Inform. Systems Res.* 25(4):796–816.

Lerner J, Tirole J (2002) Some simple economics of open source. *J. Industry Econom.* 50(2):197–234.

Levine SS, Prietula MJ (2014) Open collaboration for innovation: Principles and performance. *Organ. Sci.* 25(5):1414–1433.

Li B, Huang N, Shi W (2025) Forced to change? Media exposure of labor issues and firm artificial intelligence investment. *Inform. Systems Res.*, ePub ahead of print May 27, https://doi.org/10.1287/isre.2022.0402.

Liao G-Y, Huang T-L, Dennis AR, Teng C-I (2024) The influence of media capabilities on knowledge contribution in online communities. *Inform. Systems Res.* 35(1):165–183.

Lindberg A, Berente N, Gaskin J, Lyytinen K (2016) Coordinating interdependencies in online communities: A study of an open source software project. *Inform. Systems Res.* 27(4):751–772.

Liu L, Wang Y, Xu Y (2024) A practical guide to counterfactual estimators for causal inference with time-series cross-sectional data. *Amer. J. Political Sci.* 68(1):160–176.

Markman GD, Gianiodis PT, Phan PH, Balkin DB (2005) Innovation speed: Transferring university technology to market. *Res. Policy* 34(7):1058–1075.

Maruping LM, Matook S (2020) The evolution of software development orchestration: Current state and an agenda for future research. *Eur. J. Inform. Systems* 29(5):443–457.

Medappa PK, Srivastava SC (2019) Does superposition influence the success of FLOSS projects? An examination of open-source software development by organizations and individuals. *Inform. Systems Res.* 30(3):764–786.

Mendelson H, Pillai RR (1998) Clockspeed and informational response: Evidence from the information technology industry. *Inform. Systems Res.* 9(4):415–433.

Mikalef P, Benlian A, Conboy K, Tarafdar M (2025) Responsible AI starts with the artifact: Challenging the concept of responsible AI in IS research. *Eur. J. Inform. Systems* 34(3):407–424.

Millman Z, Hartwick J (1987) The impact of automated office systems on middle managers and their work. *MIS Quart.* 11(4): 479–491.

Mockus A, Fielding RT, Herbsleb JD (2002) Two case studies of open source software development: Apache and Mozilla. *ACM Trans. Software Engrg. Methodology* 11(3):309–346.

Nambisan S, Lyytinen K, Majchrzak A, Song M (2017) Digital innovation management: Reinventing innovation management research in a digital world. *MIS Quart.* 41(1):223–238.

Osterloh M, Rota S (2007) Open source software development—Just another case of collective invention? *Res. Policy* 36(2):157–171.

Parasuraman R, Sheridan T, Wickens C (2000) A model for types and levels of human interaction with automation. *IEEE Trans. Systems Man Cybernetics Part A Systems Humans* 30(3):286–297.

Perry-Smith JE, Shalley CE (2003) The social side of creativity: A static and dynamic social network perspective. *Acad. Management Rev.* 28(1):89–106.

Raisch S, Krakowski S (2021) Artificial intelligence and management: The automation–augmentation paradox. *Acad. Management Rev.* 46(1):192–210.

Ramaraju N, Pant S, Pant G (2025) Sparking innovation: The effect of inventor gender diversity on recombinant innovation. *Inform. Systems Res.*, ePub ahead of print June 5, https://doi.org/10.1287/isre.2023.0343.

Raymond E (1999) The cathedral and the bazaar. *Knowledge Tech. Policy* 12(3):23–49.

Safadi H, Johnson SL, Faraj S (2021) Who contributes knowledge? Core-periphery tension in online innovation communities. *Organ. Sci.* 32(3):752–775.

Sarker S, Bala H, Hong Y, Kankanhalli A, Rossi M, Gu B, Oestreicher-Singer G (2025) Advancing next-generation multimethod research in information systems: A framework and some recommendations for authors and evaluators. *Inform. Systems Res.* 36(2):647–668.

Setia P, Rajagopalan B, Sambamurthy V, Calantone R (2012) How peripheral developers contribute to open-source software development. *Inform. Systems Res.* 23(1):144–163.

Shah SK (2004) Understanding the nature of participation & coordination in open and gated source software development communities. *Acad. Management Proc.* 2004(1):B1–B5.

Sherif K, Zmud RW, Browne GJ (2006) managing peer-to-peer conflicts in disruptive information technology innovations: The case of software reuse. *MIS Quart.* 30(2):339–356.

Singh, Tan, Mookerjee (2011) Network effects: The influence of structural capital on open source project success. *MIS Quart.* 35(4):813.

Smith A (1982) The mechanization of work. *Sci. Amer.* 247(3):66–75.

Song F, Agarwal A, Wen W (2024) The impact of generative AI on collaborative open-source software development: Evidence from GitHub Copilot. Preprint, submitted October 2, https://arxiv.org/abs/2410.02091.

Stuart EA (2010) Matching methods for causal inference: A review and a look forward. *Statist. Sci.* 25(1):1–21.

Swanson EB (1994) Information systems innovation among organizations. *Management Sci.* 40(9):1069–1092.

Wasko MM, Faraj S (2005) Why should I share? Examining social capital and knowledge contribution in electronic networks of practice. *MIS Quart.* 29(1):35–57.

Witman A, Beadles C, Liu Y, Larsen A, Kafali N, Gandhi S, Amico P, et al. (2019) Comparison group selection in the presence of rolling entry for health services research: Rolling entry matching. *Health Services Res.* 54(2):492–501.

Woodman RW, Sawyer JE, Griffin RW (1993) Toward a theory of organizational creativity. *Acad. Management Rev.* 18(2):293–321.

Yeverechyahu D, Mayya R, Oestreicher-Singer G (2024) The impact of large language models on open-source innovation: Evidence from GitHub Copilot. Preprint, submitted September 12, https://arxiv.org/abs/2409.08379.

Zhou E, Lee D (2024) Generative artificial intelligence, human creativity, and art. *PNAS Nexus* 3(3):pgae052.