

Article

Interaction Effects of Team Attributes and System Attributes in Software Maintenance Productivity

Michel Benaroch

Whitman School of Management, Syracuse University, Syracuse, NY 13244, USA; mbenaroc@syr.edu

Abstract

We investigate interaction effects of system attributes—age and volatility—and team attributes—instability and skill-diversity—on software maintenance productivity in the context of lifecycle maintenance involving multiple serial tasks (projects), unlike extant work's focus on single maintenance tasks. Given the knowledge intensity of software maintenance, we apply knowledge creation theory to identify knowledge needs and challenges that system and team attributes create, then develop two theoretical predictions. First, team attributes adversely affect maintenance productivity while system attributes do not exhibit direct negative effects. Second, interactions of system and team attributes have offsetting (substitutive) effects on productivity since their knowledge needs and challenges overlap. We test our predictions using archival data on three years of maintenance work across 426 mission-critical systems at a Fortune 100 company, encompassing over 7500 maintenance tasks executed by thousands of maintainers. Our analysis yields two key insights. First, interactions diminish substantially the strong negative direct effects of team instability and skill-diversity on maintenance productivity—by as much as 20%. System volatility exhibits a small direct effect (0.38% productivity decline), while system age shows no significant direct effect. Second, interaction effects indicate that productivity declines as instable and skill-diverse teams are sharper when working on younger and less volatile systems—the opposite of conventional wisdom. For example, assigning a team with above-average instability to an above-average age system improves productivity by 3.06% through substitutive effects. Our findings demonstrate that congruence between system and team attributes can improve maintenance productivity, with substantial economic implications: organizations should strategically match team configurations to system characteristics rather than attempting to eliminate team instability or diversity universally.

Keywords: software maintenance; maintenance productivity; interaction effects; team instability; team skill-diversity; system age; system volatility



Academic Editor: Giuseppe A. Di Lucca

Received: 1 February 2026

Revised: 19 March 2026

Accepted: 27 March 2026

Published: 31 March 2026

Copyright: © 2026 by the author.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and

conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Software maintenance represents one of the largest controllable expenditures in IT budgets, yet organizations struggle to optimize staffing decisions that directly influence these costs. Maintenance comprises approximately 75% of total software ownership costs, with labor expenses accounting for 60–80% of this figure [1,2]. While managers face constraints from aging systems and volatile business requirements demanding frequent system changes, they retain control over team assignment, training, and staffing strategies [3,4]. The critical question is how to strategically allocate human capital to maximize maintenance productivity, the ratio of product size to total development and support costs [5]. Maintenance productivity depends on: (1) maintainer and team attributes including experience,

stability, and composition; (2) software attributes such as age, complexity, and volatility (change frequency); and (3) congruence between team–system attributes.

We lack empirical guidance on team–system assignment decisions. This gap is acute given contemporary workforce realities: agile and DevOps practices emphasizing fluid team boundaries [6], increasing reliance on distributed teams that naturally increase instability and knowledge diversity [7], and talent mobility trends making stability difficult to maintain [8]. Personnel scarcity from attrition and mobility requires managers to dynamically staff teams from a “bench” of available maintainers [9]. Focusing on ongoing maintenance across multiple projects versus single tasks, these decisions build up over time and determine a system’s lifecycle team—the group of maintainers who complete one or more maintenance tasks based on two key attributes: instability (membership change rate) and skill-diversity (expertise variety). These attributes gain meaning over time—teams change across projects due to turnover and migration, deteriorating system and interpersonal familiarity [10]. While both harm productivity in singular tasks [11,12], their system lifecycle-level effects and interactions with software age and volatility remain understudied. Four studies examined ongoing maintenance work, three of which did not study system–team interaction effects on productivity [13–15], and one focused on interactions of team attributes with only the system attribute of software complexity [16].

We address this void by positing that software age and volatility moderate the adverse effects of lifecycle team instability and skill-diversity on productivity. Specifically, these team attribute effects amplify when age and volatility are high. This proposition can improve maintenance productivity by informing staffing decisions as systems age and volatility increases alongside growing team instability and skill-diversity from offshoring and distributed practices.

We validate our proposition using maintenance data from a Fortune 100 company for a three-year period (January 2009–December 2011). The data cover 426 mission-critical systems maintained over a three-year period, using over 7500 maintenance tasks (projects) executed by thousands of maintainers across the U.S., the U.K., and offshore locations. Results reveal two insights. First, system and team attributes affect productivity through their interactions, not just directly, implying that system age and volatility cannot be looked at independently in maintenance planning. Second, system and team attributes interact in a way that creates a “negative synergy,” where their co-presence compounds productivity deficits. For example, the case of system age and team instability is considered. Our results show that the marginal effect of team instability on productivity depends critically on system age, with profound implications that contradict conventional wisdom. For young systems, team instability’s negative effect approaches its full magnitude: one standard deviation increase in instability on a new operational system reduces productivity by 14.6%. However, as systems age, this effect diminishes substantially—the marginal harm of additional instability becomes progressively smaller.

This paper contributes to the software maintenance literature by advancing our understanding of these interactions and how it offers managers actionable levers for productivity improvement. While system attributes are largely outside managerial control, team composition can be adjusted with each project assignment [17]. For instance, given we found that team instability is more damaging for younger and less volatile systems, managers can implement talent management policies that prioritize stable team assignments for such systems while rotating newer maintainers through older and more volatile systems where learning costs might be lower. Similarly, understanding whether skill-diversity helps or hurts maintenance of volatile systems can inform decisions about when to assign specialists versus generalists, when to invest in cross-training, and how to structure communities of practice around different system portfolios [18]. Given that personnel costs represent

40–70% of maintenance expenditures [19], even modest improvements in team–system alignment could yield substantial savings across an organization’s application portfolio.

2. Background

This section reviews how system and team attributes affect software maintenance, highlighting critical differences from development contexts and identifying research gaps motivating this study.

2.1. Software Maintenance vs. Software Development

Software development and software maintenance differ fundamentally in three ways. First, development is finite while maintenance spans 30–40 years post-development through a series of maintenance tasks (projects) that fix, enhance, and expand systems [20]. Maintenance activities divide into four types that account for different percentages of all maintenance work: corrective (fixing defects, 20%), adaptive (responding to environmental changes, 25%), perfective (enhancing functionality, 50%), and preventive (reducing complexity, 5%).

Second, software maintenance requires more extensive knowledge. Beyond software development’s focus on requirements, domain, and environment [21], maintenance demands understanding legacy code, architecture, documentation, and prior changes [22]. This knowledge is often fragmented or tacit, yet essential for constructing accurate mental models before making changes. Maintainers acquire this knowledge through interactions with previous contributors and extensive use of knowledge artifacts—code, documentation, CASE tools, forums, chats, and emails.

Third, system comprehension dominates software maintenance effort—50–70% of the effort is devoted to “discovery” involving understanding the system deeply before implementing changes [4,22–24]. Maintenance work must conform to aging systems shaped by historical design choices, obsolete platforms, and legacy languages. Personnel turnover exacerbates this challenge by eroding system-specific knowledge. Over time, continuous growth and repeated modifications degrade system structure and supporting artifacts, further increasing the cognitive burden on maintenance teams. Finally, the reactive nature of corrective/adaptive maintenance requires rapid intervention, further amplifies the challenge.

2.2. Main Study Constructs—Lifecycle-Team Attributes

The software engineering literature typically associates a team with a specific development or maintenance task, termed project or modification request [11,12,25]. We call this a *task-team*. Task-teams are typically stable during execution, and task-teams fluctuate in membership for large or long-lived projects [26–28].

Maintaining a system over its lifecycle involves multiple consecutive tasks (projects), each executed by its own task-team. No single task-team maintains a system throughout its lifecycle due to maintainer attrition, migration, and mobility [29,30]. New task-teams form by assigning maintainers to replace unavailable members or add expertise needed for new features or to compensate for lost system knowledge [30,31]. Maintainers are dynamically drawn from a “bench” of available personnel [9], similar to cardiac operating teams [32].

We define a system’s *lifecycle-team* as the union of all task-teams that have completed maintenance tasks on that system. Many task-level studies implicitly recognize the notion of lifecycle-team, expecting some maintainers on a given task to have worked together previously on the same system [11,25,28,30,33,34].

Over time, management assignments of maintainers to teams affect two lifecycle-team attributes: instability and skill-diversity. These attributes constrain or expand the

knowledge base maintainers bring to tasks. (The knowledge base covers the following: how the software system is organized and operates [5,11], what other systems it is connected to [11], business domain and technology components [34], member roles [11,12], and the methods a team uses to coordinate its efforts [35].) *Team instability* is the degree of membership change over time. We focus on instability rather than size because new members primarily replace unavailable ones or provide missing skills [28]. (The effect of team *size*, the number of persons working on a task (project), has not been examined in the lifecycle maintenance context. In software development, team size has a negative effect on productivity. In software maintenance, two studies show mixed results: one shows a negative effect [36,37], and another shows a negative effect for a system release and a positive effect for a group MRs [11]. In lifecycle multi-project maintenance work, as in open-source development, size is a more challenging measure. Knowing who belongs to a team at a given time is not trivial. Personnel turnover, migration and non-availability necessitate adding new maintainers less familiar with the system and with others. For this reason, a more suitable notion is team *stability*, as a function of the duration members stay in a team [38].) *Team skill-diversity* is the variety of functional expertise present. This is conceptually distinct from instability because membership change does not necessarily increase skill-diversity. While teams share common skills (business analyst, application developer, application support analyst, quality assurance analyst), some include additional expertise (IT architecture, database design, information risk, business resiliency).

2.3. Literature on Factors Affecting Maintenance Productivity

Our literature review identifies critical gaps in understanding maintenance productivity drivers. Studying software quality as another maintenance outcome could reveal beneficial tradeoffs; however, it is not without challenges. Some software development studies consider software quality to be a development outcome, measured by the number of software defects found during testing and user acceptance. In this context, [39] found that achieving higher productivity and software quality levels requires diametrically opposed team configurations. Finding similar tradeoffs could certainly complement our results; however, studying such tradeoffs in lifecycle maintenance is challenging—a difficulty that can be illustrated by [5] who measured quality at a “starting” point in the system lifecycle, as a count of defects found during the first seven post-development quarters, and found it to positively impact maintenance productivity shortly thereafter. Extending this approach to the lifecycle context is conceivable, but the challenge lies in defining and measuring quality at an “end” point, specifically how effectively ongoing maintenance fixes past and current defects and delivers error-free changes. Unfortunately, we are not aware of other studies along these lines, particularly regarding unit of analysis and interaction effects between system and team attributes in the lifecycle maintenance context (see Appendix A).

Unit of Analysis Gap: Most studies examine single maintenance tasks rather than ongoing lifecycle maintenance. This focus obscures how effects evolve over time. For instance, personnel turnover distributes system knowledge across more individuals while losing tacit knowledge about the system and team practices, reducing comprehension with potentially substantial cumulative effects [10].

System Attributes: Despite consensus on the negative productivity impacts of system age and volatility [19], empirical evidence remains limited. Relative to *age*, one single-task study found no significant impact on productivity [35], while another found negative effects on team performance with age used as a control variable [36]. Most studies treat age as an explanatory factor for other variables like volatility, size, or complexity [14,15,36,37]. As to system *volatility*, [13] found no effect on lifecycle productivity but negative effects on software quality. Other studies used volatility as a dependent variable [14].

Team Attributes: All studies on team instability and skill-diversity focus on single tasks, defining attributes at the task level. In terms of team *instability*, [38] found that task-level stability benefits maintenance quality and timeliness. Other single-task studies examined team familiarity, implicitly acknowledging lifecycle concepts by examining members' prior collaboration on the same system [36] or different systems [11,12]. Relative to team *skill-diversity*, no studies examine functional skill-diversity in maintenance. Reference [39] recognized its relevance but relied on interview-based perceptions. While the teamwork literature shows that functional knowledge diversity influences performance [40–42], software studies focus on other knowledge dimensions: technical experience improves productivity [5,19], role experience enhances team performance [12], and system and domain knowledge benefits development [34] but not maintenance productivity [35]. Importantly, no study sheds light on the effects of teams' skill-diversity or how it evolves over a system's lifecycle.

Interaction Effects: No studies examine interactions between team attributes and the specific system attributes of age or volatility. A recent lifecycle study examined interactions between team attributes and another system attribute—software complexity [16]. Two other single-task studies examined interactions of system complexity with team familiarity and team geographic diversity [34,36].

In summary, research provides little insight into lifecycle maintenance where evolving teams allocate effort across multiple tasks with the system over time. These gaps motivate us to, first, propose theoretical foundations for empirical patterns beyond mere reporting and, second, examine poorly understood productivity drivers centering on system and team attribute effects and their interactions. Understanding these interactions has significant practical implications for questions like: Should managers seeking to reduce maintenance costs prioritize stabilizing teams or mitigating system aging? Given the inevitability of team instability, how should managers align team and system attributes? Is it better to assign instable teams to younger or older systems, or to more volatile or less volatile systems? Interactions may amplify or offset individual impacts, making these questions essential for maintenance management.

3. Theoretical Lens and Hypotheses Development

3.1. Knowledge Creation Theory

Software maintenance involves two core cognitive activities: code comprehension and code modification [43]. Maintainers must integrate mental models of the existing code with models of the required changes. *Cognitive fit* exists when these models align without gaps that impede task completion. Because maintenance tasks are cognitively demanding and knowledge intensive, the availability, distribution, and use of knowledge critically shape maintenance productivity [19].

Following prior works that apply knowledge creation theory to software maintenance [44,45], we adopt this theoretical lens to explain how system and team attributes affect productivity through their influence on knowledge needs and knowledge-related challenges. *Knowledge creation theory* distinguishes between explicit and tacit knowledge, and specifies modes through which knowledge is created, shared, and transformed [46]. Table 1 offers a summary of these theory constructs. Explicit knowledge is codified, documented, and readily communicated, for example, source code, design documents, data models, and written communications. Tacit knowledge is subjective, context-dependent, and difficult to formalize. It is acquired through experience and shaped by individuals' beliefs and interpretations [47]. In software maintenance, tacit knowledge includes deep understanding of system structure, debugging strategies, and workarounds developed over time. Tacit knowledge also exists at the group level. Through repeated interactions, collaborators develop the shared understanding of tasks, goals, and *who knows what* [48,49].

Table 1. Knowledge types and transfer modes (adapted from [46]).

Knowledge Types		Examples in Maintenance
Explicit	Codified, documentable, and communicable information	Code, design docs, data models, forums
Tacit	Experiential, context-sensitive; includes group meta-knowledge	System structure insights; debugging tactics; knowing who has specific expertise
Knowledge transfer (sharing) modes		
Socialization (tacit→tacit)	Sharing through interaction, observation, imitation, practice	Mentoring novices on system structure
Externalization (tacit→explicit)	Articulating tacit knowledge via documents or models	Re-documentation; explaining behavior via sketches and code samples
Combination (explicit→explicit)	Connecting existing explicit sources to create new knowledge	Synthesizing references for specific maintenance tasks
Internalization (explicit→tacit)	Building tacit understanding by acting on explicit knowledge	Studying data models or code comments to grasp system functioning
Other Knowledge Activities		
Search	Recognizing knowledge needs and locating sources	Finding who last worked on a module; locating documentation
Integration & Coordination	Combining knowledge across maintainers; sequencing interdependent tasks	Leveraging diverse expertise; synchronizing tasks to avoid redundancy

Knowledge creation theory further specifies four knowledge transfer modes. *Socialization* transfers tacit knowledge through shared experience (e.g., mentoring and observation). *Externalization* converts tacit knowledge into explicit forms, such as documentation or diagrams. *Combination* integrates multiple explicit knowledge sources into new explicit knowledge, while *internalization* transforms explicit knowledge into tacit knowledge through use and practice [46]. In addition to these transfer modes, maintenance work relies on complementary knowledge activities, including searching for relevant knowledge, and integrating and coordinating knowledge across maintainers [17].

Because maintenance tasks typically involve multiple maintainers working on interdependent system components, we extend [43] individual-level conceptualization to the team level using shared team cognition research [50]. Shared team cognition comprises knowledge pools about task components and coordination schemes, software architecture and past changes, and the team members' skills and processes [51]. These pools enable members to "form accurate explanations and expectations for the task and, in turn, to coordinate their actions and adapt their behavior" [52]. Like individual cognitive fit, team cognition is shared when members' mental models overlap and they reason similarly about relationships, enabling coordinated collective execution.

Maintainers obtain task-relevant knowledge through exchanges involving different transfer modes. Explicit knowledge communicates readily but requires internalization by receivers, constrained by knowledge quality—how error-free and well structured the code, documentation, and models are. Tacit knowledge is harder to share due to its informal, context-specific nature, requiring socialization (face-to-face interaction) or externalization (articulation via text, diagrams, metaphors), with effectiveness depending on the holder's recall ability.

Knowledge exchanges involve not just transfer but also search (recognize need), integration, and coordination of knowledge. Searching for explicit knowledge is straightforward. Searching tacit knowledge requires knowing who possesses it, formulating reliable requests, and proper recall by the providers. All this hinges on shared vocabulary and meta-knowledge about expertise location. After knowledge is shared, coordination and integration bring multiple maintainers' knowledge to bear timely, enabling members to anticipate each other's actions [17].

We proceed to use knowledge creation theory and shared team cognition to examine how system attributes—age and volatility—and lifecycle-team attributes—instability and skill-diversity—shape maintenance productivity—through both their direct effects (on knowledge availability and transfer) and indirect interaction effects.

3.2. Research Model and Hypotheses

Building on our theoretical framing, Figure 1 presents our research model positing that lifecycle-team instability and skill-diversity directly harm lifecycle maintenance productivity, with system age and volatility moderating these effects. While direct effects are confirmed by task-level studies (Appendix A), we extend them to the lifecycle context and add the moderation effect of direct team attribute effects. Table 2 summarizes the logic of our proceeding research hypotheses development.

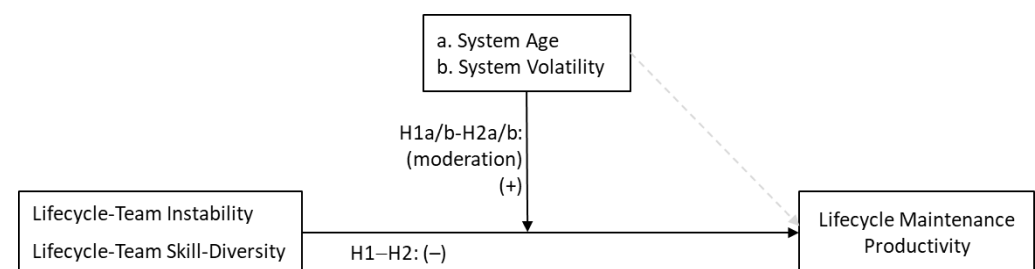


Figure 1. Research model.

Table 2. Knowledge challenges of team and system attributes and their interactions. (underlined words indicate the direction of interaction effects).

Team Instability: H1 (–)	Team Skill-Diversity: H2 (–)
Loss of tacit knowledge (–)	Improved system comprehension (+)
More and less efficient knowledge search & sharing by new members (–)	• Offset difficulty in knowledge recall (+) • Less need for knowledge search (+)
Difficult team coordination & knowledge integration (–)	Weaker shared team cognition (–)
Degraded shared team cognition (–)	• Greater coordination demands (–)
	Difficult knowledge sharing & integration (–)
	• Less k. overlaps, or effective exchanges (–) • More task conflicts, less cohesion (–)

Table 2. Cont.

System age (a)	H1a ↓ (+)	H2a ↓ (+)
Lower quality explicit knowledge (–) Need more knowledge search & sharing (–) • Lower comprehension (–) • Weaker knowledge recall (–) But, age alone need not have an inherent effect on productivity.	Age <u>exacerbates</u> team instability-related knowledge challenges: • Knowledge integration difficulty • Tacit knowledge decay • Team coordination difficulties	Age <u>offsets</u> team diversity-related knowledge challenges: • Knowledge deterioration and tacit loss • Difficulty on knowledge search but <u>exacerbates</u> others: • Team coordination & knowledge integration
System Volatility (b)	H1b ↓ (+)	H2b ↓ (+)
Lower quality explicit knowledge (–) • Need more knowledge search & sharing Need less knowledge search and sharing (+) • Improved system comprehension (+) • Better recall (+) • More reliance on tacit (+)	Volatility <u>exacerbates</u> team instability-related knowledge challenges: • Team coordination difficulties • Tacit knowledge loss but <u>offsets</u> others: • Knowledge recall, tacit reliance ease • Knowledge search & sharing difficulties	Volatility <u>enhances</u> positive team instability-related benefits: • Better system comprehension • Less need for knowledge sharing, exchanges but <u>enhances</u> other negatives: • Lower quality explicit knowledge • Difficult team coordination

3.2.1. Direct Effects of System Attributes

System age and volatility are widely recognized as determinants of maintainability and, indirectly, productivity [19,53–56]. We proceed to delineate their implications for maintenance-related knowledge requirements and, in turn, maintenance productivity [16].

System age is the time since the system was deployed and has unclear productivity impacts. Older systems undergo more maintenance [37,57], deteriorating explicit knowledge through code errors and artifact inconsistencies [14,37]. Age may also weaken maintainers' recall and comprehension, increasing knowledge search and sharing needs. With this said, age per se may not directly reduce productivity. Instead, productivity effects may arise from the conditions that accompany aging, such as the need for increased change frequency (volatility), rather than from time alone. Consistent with mixed findings reviewed earlier [35,36,54], we find insufficient evidence to hypothesize a direct negative effect of age on maintenance productivity, notwithstanding Lehman's law linking system age to entropy and maintenance costs [58]. We therefore treat system age as a moderator rather than a direct driver of productivity.

System volatility is the number of software enhancements per unit time [13,59] and is understudied. While not directly increasing maintenance costs per unit work [13], volatility increases errors and lowers code quality [13,59,60]. Each modification introduces error opportunities and technical debt into code and knowledge artifacts [61]. It lowers explicit knowledge quality and requiring more knowledge recreation, search, and sharing. Conversely, volatility may improve maintainers' comprehension and knowledge recall [13]. Frequent modifications let maintainers learn system structure better and refresh tacit knowledge, making it less prone to degradation and more accurately recalled [62]. This increases tacit knowledge reliance and lowers search/sharing needs. On balance, insufficient reason exists to expect system volatility alone to hurt productivity.

3.2.2. Direct Effects of Team Attributes

Lifecycle-team instability is an artifact of personnel turnover, migration, and so on, and it undermines performance [9,30] through the loss of system-specific knowledge [10] and hard-to-replicate tacit knowledge about the system and teamwork [63]. Specifically, instability has four consequences:

- *Tacit knowledge loss*: Departing members carry non-replicable personal and group tacit knowledge [63,64], including about team linkages [65], creating knowledge gaps.
- *New member learning burdens*: New members must build system-specific mental models to achieve cognitive fit, expending significant effort for comprehending code organization, style, architecture, and inter-module linkages [10]. They rely on knowledge exchanges and shared team cognition to understand code interactions.
- *Degraded shared team cognition*: New members create gaps in shared cognition, weakening knowledge exchanges. They have weaker social ties, less shared system and task knowledge, more uncertainty about others' skills [36], less trust in other members [66], and less common tacit knowledge [67]. Also, less team familiarity means less knowledge overlap, fewer shared experiences, and less agreement on conventions. This adds errors, gaps, and inconsistencies in explicit knowledge assets. This also makes explicit knowledge harder to locate, search, share, and reuse.
- *Coordination difficulties*: Instable teams lack common team knowledge crucial for communication, mutual understanding, trust, and coordination [68,69]. Members face more uncertainty about each other's skills, spending more time on knowledge search and being less likely to voluntarily engage in socialization and tacit knowledge sharing [70]. This escalates the number of knowledge exchanges and information overload [51], slowing learning and mental model adaptation while weakening coordinative capacity [17].

These consequences of team instability translate to knowledge needs and challenges that lower maintenance productivity:

Hypothesis 1 (H1). *Lifecycle-team instability has a negative effect on lifecycle maintenance productivity.*

Lifecycle-team skill-diversity is the variety of functional skills maintainers in a team bring, and it has competing effects [71]. Diverse skills provide more expertise to mitigate knowledge loss, handle accumulated software changes, and address complex tasks involving novel system features. Specialized maintainers adapt mental models to unfamiliar tasks faster with less cognitive load, improving comprehension when knowledge must be recreated. Higher skill-diversity also increases perspective variation [72], broadening expertise pools and allowing easier recognition and solving of complex issues.

However, skill-diversity weakens shared team cognition. This leads to less information sharing across members with non-overlapping backgrounds [40], less effective knowledge exchanges due to specialized vocabularies and knowledge boundaries [73], greater coordination demands causing delayed or inaccurate decisions [74], and reduced coordinative capacity from less agreement among team members on conventions [40]. Low knowledge overlap also means less communication, more cohesion problems, and greater difficulty sharing and assimilating knowledge [72,75]. Finally, functionally diverse teams tend to act as narrow specialists with fixed perspectives [40] and disagree on process [72]. This increases coordination overhead [17], requiring more knowledge exchanges, causing information overload, and reducing productivity [51].

On balance, we expect the detrimental effects to overshadow the benefits of team skill-diversity:

Hypothesis 2 (H2). *Lifecycle-team skill-diversity has a negative effect on lifecycle maintenance productivity.*

3.2.3. Interaction Effects: Team Attributes and System Age

Although system age alone may not directly affect productivity, it interacts with team attributes by shaping knowledge challenges. Team instability exacerbates age-related problems such as tacit knowledge decay and coordination difficulties. However, these overlapping challenges are more harmful in *younger systems*, where explicit knowledge quality and tacit knowledge recall are otherwise relatively high. Introducing instability into such settings imposes unnecessary knowledge and coordination burdens, leading to sharper productivity declines than in older systems, where knowledge challenges are already present.

As to team skill-diversity, it offsets aging systems' knowledge challenges—particularly knowledge deterioration and tacit knowledge loss—but adds integration and coordination difficulties. Benefits emerge on older systems, whereas only drawbacks emerge on younger systems. Since comprehension and knowledge recall are higher in younger systems, skill-diverse teams working on younger systems add challenges without offering offsetting benefits.

In sum, we expect productivity decline from instable and skill-diverse teams to be larger on younger systems.

Hypothesis 1a (H1a). *Team instability and system age are substitutive in their effects on productivity, such that the negative effect of team instability is stronger for younger systems.*

Hypothesis 2a (H2a). *Team skill-diversity and system age are substitutive in their effects on productivity, such that the negative effect of skill-diversity is stronger for younger systems.*

3.2.4. Interaction Effects: Team Attributes and System Volatility

While system volatility alone may not directly affect productivity, it interacts with team attributes. Volatility offsets some team instability knowledge challenges but the latter adds tacit knowledge loss and coordination requirements. Volatility improves knowledge recall and tacit knowledge reliance, buffering instability's constraint on socialization and tacit knowledge transfer. It also lowers knowledge search/sharing needs, buffering team instability's adverse sharing effects. These offsetting effects mean instability's negative impacts are smaller in more volatile systems. Additionally, volatility's improved comprehension overlaps with skill-diverse teams' improved comprehension. However, reduced knowledge search/sharing needs in volatile systems can offset skill-diverse teams' greater knowledge search/sharing difficulties.

On balance, we expect the maintenance productivity decline from instable and skill-diverse teams to be sharper in less volatile systems.

Hypothesis 1b (H1b). *Team instability and system volatility are substitutive in their effects on productivity, such that the negative effect of team instability is stronger for less volatile systems.*

Hypothesis 2b (H2b). *Team skill-diversity and system volatility are substitutive in their effects on productivity, such that the negative effect of skill-diversity is stronger for less volatile systems.*

4. Research Method and Setting

To validate the hypothesized research model, we conducted a field study using secondary data. Access to a unique data set facilitated a research design that allowed us to infer the moderation effects of system age and volatility on the direct effects of lifecycle-team

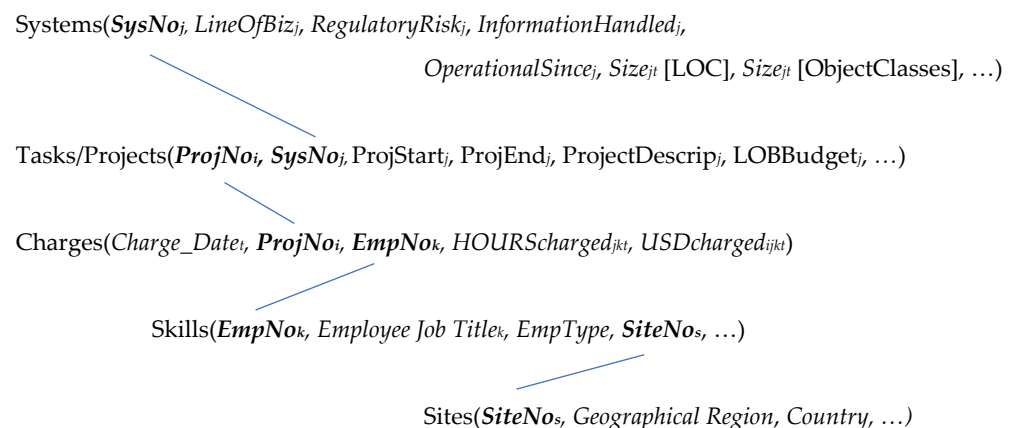
attributes. The data reflect how (1) multiple lifecycle-teams allocate their effort (2) across multiple maintenance tasks (3) performed within the same system and different systems of varying age and volatility.

4.1. Sample, Data, and Descriptive Statistics

We had unique access to a large archival data set on software maintenance work performed on *all* mission critical software systems of a Fortune 100 financial services firm during a three-year period (January 2009–December 2011). Four other studies cover an equal time span of three years [5,13–15], but none examined the effects of team attributes on system lifecycle maintenance productivity.

Details describing the data set are available in Table 3. The sample covers 426 systems that are highly homogeneous. This reduces potential confounds due to differences in programming languages, implementation environment, development process and methodology, or training and incentives. Over 7500 maintenance tasks (projects) were completed on the systems over a three-year period, with an average of over 20 tasks per system over the period. About 2,000,000 charges were posted over the period to the accounting charge-back system, totaling over 15,000,000 person hours. Each charge indicates the number of *daily* hours logged (and number of US dollars or equivalent billed) by a maintainer working on a task on a specific system. Maintenance tasks (projects) were planned annually around budget cycles, independent of team assignment considerations, except for urgent bug and defect fixes. Tasks were completed by more than 3000 maintainers with different skill (job) titles and working in different site locations. Site locations include the United States, the United Kingdom, and over 20 other firm-run offshore centers.

Table 3. Data set description.



Systems	• 426 operational, mission-critical systems • systems are Java-based • 390+ fully developed and supported in-house following a company-wide, object-oriented iterative development and documentation methodology • remaining systems developed and supported partially or fully by a third-party
Maintenance Tasks (Projects)	• Over 7500 tasks (projects) completed on the 426 systems over the three-year period • Tasks recorded as “Requests for Change” tickets on specific systems • Tasks are planned annually around the budget cycle, except for bug and defect fixes which account for a small portion of all maintenance work

Table 3. Cont.

Charges	• 2,000,000 charges posted over the three years to the accounting chargeback system, reflecting the maintenance history and related costs for the 426 systems • A charge indicates the number of <i>daily</i> hours logged and billed (and number of US dollars or equivalent) per maintainer working on a task (ticket) on a specific system. • Charges over the three-year period totaled about 15,000,000 person-hours; as expected, the 30% largest systems accounted for about 60% of all person-hour charges
Maintainers	• Over 3000 maintainers with different skill (job) titles and working in different site locations • Over 95% worked full-time at the firm during the 3-year period or previously employed and hired as part-time consultants
Sites	• Located in the United States, United Kingdom, and over 20 other firm-run offshore centers.

The lifecycle-team of a system was measured by the number of distinct maintainers who posted person-hour charges to tasks (projects) performed on that system over the three-year period. Lifecycle-team **instability** for teams working on systems in the sample was generally high, as is common in most maintenance shops, with a median of 98 maintainers. To put this in perspective, the average task-team size was 14 in Huckman et al.'s [12] study and 9.2 in Kang et al.'s [34] study. The lifecycle-team skill-diversity is equally high, with a median of 23 distinct job titles held by lifecycle-teams in a three-year period.

4.2. Construct Definition, Operationalization and Descriptive Statistics

Table 4 summarizes the raw data items, study variables, descriptive statistics, and transformations applied to data. Data items and variables exhibiting asymmetric distributions and power-type relationships with maintenance effort and cost were log-transformed, consistent with the diseconomies of scale documented in prior studies [10,37,76,77]. We zero-centered interaction terms to avoid multicollinearity [78].

The **dependent** variable is *effort-productivity* (**ProdHrs**). It is the ratio of system size (number of object classes) to the total effort (hours) of maintaining a system over the three-year period [5,56]. Productivity is a function of output units produced and resource units consumed [5,56,79].

The **independent** variables are system *age*, system *volatility*, lifecycle-team *instability*, and lifecycle-team *skill-diversity*. Because our unit of analysis is maintenance effort by a lifecycle team working on a system over a three-year period rather than a single maintenance task (project) in the same system, we measured age and volatility as follows. System age (**Age**) is the number of years since the system's go-live date [14,57]. System volatility (**SoftVolatility**) is the ratio of the number of maintenance tasks in a three-year period to system size (number of classes) [13,14]; this reflects the *Change-Evolution* frequency. Lifecycle-team *instability* (**TeamInstability**) is a count of maintainers who worked on a system in a three-year period; this is a more conservative measure than those of [80] and [81] in that it includes only members who have joined the team and not those who left the team. Lifecycle-team *skill-diversity* (**TeamDiversity**) is a count of distinct job titles of team members who maintained the system in a three-year period [72]. The interaction terms are **Age-TeamInstability**, **Age-TeamDiversity**, **Volatility-TeamInstability**, and **Volatility-TeamDiversity**.

Table 4. Study variables and descriptive statistics ($n = 426$). (dashed lines are for two separate measures for the same variable).

Variable	Measurement	Mean	Min	Max	StDev	Trans-Formed
Raw data items						
Size	Number of object classes	1132	6	17,117	1807	Log *
	Number of non-comment lines of code (LOC)	171,297	511	4,198,640	317,146	
Number of Tasks	Number of maintenance tasks on a system (in 3 years)	23.86	1.00	154.00	27.48	Log
Total Maintenance-Effort	Sum of daily hours charged by maintainers of a system (in 3 years)	52K	11	890K	91K	Log
Total Maintenance-Cost	Sum of daily dollars charged by maintainers of a system (in 3 years)	1152K	29	18,737K	2016K	Log
Dependent variables						
Effort Productivity (ProdHrs)	ln(Size) to ln(Total Maintenance-Effort) ratio	0.65	0.18	2.99	0.22	
Cost Productivity (ProdUSD)	ln(Size) to ln(Total Maintenance-Cost) ratio	0.49	0.15	2.13	0.14	
Independent Variables & Interaction Terms						
Age (Age)	Number of years from a system's go-live date till Dec. 2011	6.95	1.00	42.00	4.49	Log
Software Volatility (SoftVolatility)	ln(Number of Tasks) to ln(Size) ratio	0.43	0.00	1.43	0.21	
Lifecycle-Team Instability (TeamInstability)	Count of distinct maintainers who worked on a system (in 3 years)	164.0	1	1231.0	129.42	Log & Zero-centered
Lifecycle-Team Skill-Diversity (TeamDiversity)	Count of distinct job titles (functional skills) of maintainers who worked on a system (in a 3-year period)	25.16	1	67	14.51	Log & Zero-centered
Interaction Terms (Age-TeamIns/TeamDiv)	$Age \times Team\ Instability$ $Age \times Team\ Diversity$	0.2 0.22	-3.07 -0.15	7.887 2.705	0.91 0.33	
(Volatility-TeamIns/TeamDiv)	$Volatility \times Team\ Instability$ $Volatility \times Team\ Diversity$	0.12 0.12	-1.86 -0.17	5.248 1.8	0.57 0.22	
Control variables						
Complexity (Complexity)	System cyclomatic complexity	27,203	84	764,016	54,904	Log & Zero-centered **
Customer Info (CustInfo)	System handles sensitive customer (financial, personal) data? (binary)	0.462	0.00	0.499	1.00	
Line of Business (LOB)						
LOB-1 (51%)	Line of business a system serves (LOB names omitted for confidentiality)	0.512	0.00	1.00	0.500	1-of-N
LOB-2 (28%)		0.277	0.00	1.00	0.448	
LOB-3 (12%)		0.122	0.00	1.00	0.328	
LOB-others (others)		0.089	0.00	1.00	0.285	
Risk regulatory (Risk)	System's regulatory risk exposure (low = 2, medium = 3, high = 4)	3.052	2.00	4.00	0.771	
Information Type (infoType)	Sensitivity of system data (public = 2, internal = 3, private = 4, confidential = 5)	4.617	2.00	5.00	0.801	

* Log-transformed(x_i) = $\ln(x_i)$. ** Zero-centered(x_i) = $(x_i - \text{mean}(x_i)) / (\text{max}(x_i) - \text{min}(x_i))$.

Control variables include software *complexity* (**Complexity**), measured by each system's cyclomatic complexity [82]. Four other variables available in the data were contextual and reflect system traits specific to our research site. These variables are defined in Table 4 (*customer info, risk, LOB, and information type*). They helped us address endogeneity concerns, as explained shortly. In any event, when we included them as control variables in our econometric model, neither had a significant effect on maintenance productivity.

Table 5 shows descriptive statistics and pairwise correlations. There are very high correlations between the team attributes (team instability and skill-diversity). Because of potential collinearity concerns, we run the forthcoming regression models for each team attribute separately. The remaining correlations are relatively low and raise no major concern.

Table 5. Correlation matrix on included measures.

Variables	Mean	Min	Max	StdDev	(1)	(2)	(3)	(4)	(5)	(6)	(7)
(1) ProdHrs	0.18	0.65	2.99	0.22	1.00						
(2) ProdUSD	0.15	0.49	2.13	0.14	0.94 *	1.00					
(3) ln(Age)	0.00	1.76	3.74	0.63	−0.10 *	−0.06	1.00				
(4) SoftVolatility	0.00	0.43	1.43	0.21	−0.31 *	−0.17 *	0.30 *	1.00			
(5) ln(Complexity)	4.43	9.16	13.55	1.56	−0.41 *	−0.57 *	0.18 *	0.38 *	1.00		
(6) ln(TeamInstability)	0.00	4.49	7.12	1.23	−0.41 *	−0.25 *	0.26 *	0.24 *	0.34 *	1.00	
(7) ln(TeamDiversity)	0.00	2.99	4.20	0.78	−0.40 *	−0.22 *	0.24 *	0.33 *	0.33 *	0.92 *	1.00

N = 426. * shows significance at the 0.05 level.

4.3. Model Specification

We use a linear regression estimation model to test our hypotheses, consistent with the research model shown in Figure 1 and the log-transformed nature of our data:

$$\begin{aligned}
 \ln(\mathbf{ProdHrs}) = & \alpha_0 + \alpha_1 \ln(\mathbf{Age}) + \alpha_2 \mathbf{SoftVolatility} + \alpha_3 \ln(\mathbf{Complexity}) \\
 & + \alpha_4 \ln([\mathbf{TeamInstability}/\mathbf{TeamDiversity}]) \\
 & + \alpha_5 (\ln(\mathbf{Age}) \times \ln[\mathbf{TeamInstability}/\mathbf{TeamDiversity}]) \\
 & + \alpha_5 (\ln(\mathbf{Volatility}) \times \ln[\mathbf{TeamInstability}/\mathbf{TeamDiversity}]) + \varepsilon
 \end{aligned} \quad (1)$$

The dependent variable is **ProdHrs**, the independent variables of interest are **TeamInstability** or **TeamDiversity** and **Age** or **SoftVolatility**, the moderation effects are the interaction terms, and the control variable is **Complexity**.

We tested the model for potential violation of standard Ordinary Least Squares (OLS) assumptions [83]. We found no evidence of non-normality of error terms (residuals) per Shapiro–Wilk's test, no evidence of heteroskedasticity per White's test, no influential outliers per Cook's test, and no multicollinearity problems based on VIF < 2.5 for all independent variables.

We also tested for endogeneity problems, particularly whether explanatory variables are uncorrelated with the regression error term. Of three common sources of endogeneity, we are worried about omitted variables bias [84], that is, the exclusion of a relevant variable that affects the dependent variable and is correlated with other independent variables. This concern could arise if unobserved management practices actively and systematically influence team. (The same concern does not hold for system volatility. System volatility could be sensitive to management decisions to undertake more tasks (projects) or join/split some tasks. However, in our research site, tasks are planned annually around a budget cycle, not based on ongoing management choices, and preventative tasks generally account

for only 5% of all maintenance work.) For example, sending work to low-wage offshore sites or assigning extra “monitoring” maintainers to certain projects would influence team attributes like instability and skill-diversity, beyond just maintainer attrition, turnover, promotion, relocation, and so on. These variables would not be fully determined exogenously in a random fashion. Our endogeneity testing found no endogeneity problem with either team attributes. See Appendix B for details.

5. Results

5.1. Regression Analysis

We use the OLS regression estimates for the interpretation of our empirical results (see Table 6). Column (1) includes the system attributes. All coefficients are negative. The coefficient of age is not statistically significant. The coefficient of system volatility is significant, deviating from our expectation (or lack thereof). Columns (2) and (4) add one team attribute at a time (without interaction terms). Consistent with our expectations, both have negative and significant relationships with productivity. Compared to column (1), the coefficients for software age are relatively unchanged across columns (2) and (4), and the coefficient for system volatility become non-significant in column (1).

Table 6. OLS regression for maintenance productivity.

Dependent Variable	Exp. Sign	Effort Productivity (ProdHrs) ($n = 426$)				
		(1)	(2)	(3)	(4)	(5)
Complexity	(−)	−0.617 *** (−15.37)	−0.633 *** (−17.04)	−0.660 *** (−19.81)	−0.638 *** (−17.12)	−0.677 *** (−20.09)
Age		−0.060 (−1.54)	−0.056 (−1.55)	−0.008 (−0.23)	−0.056 (−1.55)	−0.021 (−0.65)
Volatility		−0.528 *** (−12.75)	−0.083 (−1.30)	−0.110 * (−1.86)	−0.188 *** (−3.39)	−0.254 *** (−5.05)
Team Instability	(−)		−0.539 *** (−8.58)	−0.408 *** (−7.10)		
Age × Instability	(+)			0.153 *** (4.59)		
Volatility × Instability	(+)			0.305 *** (8.12)		
Team Skill-Diversity	(−)				−0.454 *** (−8.46)	−0.276 *** (−5.33)
Age × Skill-Diversity	(+)					0.156 *** (4.74)
Volatility × Skill-Diversity	(+)					0.319 *** (7.62)
R^2		0.421	0.507	0.613	0.505	0.608
adj. R^2		0.417	0.503	0.607	0.501	0.602

Standardized beta coefficients; t statistics in parentheses. * $p < 0.10$, *** $p < 0.01$.

We can offer an economic interpretation of the standardized beta coefficients based on the marginal effects calculated in Table 7. For example, consider the results in column (2) showing the effects of system age and volatility without considering the interaction effect. For system *age* or *volatility* on their own, one-unit increase in these variables is associated with a -0.056 ($p > 0.1$) or -0.083 ($p > 0.1$) decrease in log(productivity), meaning that maintaining a system that is one standard deviation older or more volatile than the average system lowers maintenance productivity by 0.78% or 0.38%, respectively. Both

effects are practically negligible and statistically insignificant, suggesting age or volatility alone have virtually no impact on maintenance productivity. For team *instability*, one-unit increase in this variable is associated with a -0.538 ($p < 0.01$) decrease in $\log(\text{productivity})$, meaning that maintaining a system using a team one standard deviation less stable than the average team lowers maintenance productivity by -14.6% , holding other variables constant. This is a substantial negative effect—team instability significantly harms productivity when considered at average system age or volatility. A similar interpretation applies with the coefficients for team skill-diversity in column (4). These **results support Hypotheses H1 and H2**.

Table 7. Economic marginal effects of main and interaction terms.

Percent change in productivity for a <i>one standard deviation increase</i> in attribute		Team Instability	Team Skill-Diversity
		H1 (—)	H2 (—)
		-14.60%	-7.77%
System Age	-0.78%	3.06% ⁽²⁾	3.83%
System Volatility	-0.38% ⁽¹⁾	1.15%	1.51%

⁽¹⁾ The marginal effect of variable x on outcome variable y is $\beta(x) \cdot \text{SD}(x) \cdot \text{SD}(y)$, where β is the regression coefficient and SD denotes standard deviation. For example, the -0.38% marginal effect of system volatility on its own is $0.22 \times 0.21 \times -0.008$, where 0.22 and 0.21 are the SDs of effort productivity and volatility, respectively (Table 4), and -0.008 is the coefficient for volatility in Table 6, column 2. ⁽²⁾ The marginal effect of interaction term $z (=x_1 \cdot x_2)$ on variable y is $\beta(z) \cdot \text{SD}(z) \cdot \text{SD}(z) \cdot \text{SD}(y)$. For example, the -3.06% marginal effect of the interaction term age $\times$ team instability is computed as $0.22 \times 0.908 \times -0.153$, where 0.22 and 0.908 are the SDs of effort productivity and age $\times$ instability, respectively (Table 4), and -0.153 is the regression coefficient for volatility in Table 6, column 3.

Moving on to interpret the interaction of age and team instability, the interaction term has a significant positive coefficient. This indicates the effect of team instability is conditional on system age, where the positive sign indicates a substitutive (offsetting) effect between the two variables. As both age and instability increase together, their combined negative effect on productivity is less severe than the sum of their individual effects. The interaction dampens the productivity loss—software age attenuates the negative effect of team instability. This means the productivity loss due to team instability is stronger in younger systems and weaker in older systems. For example, in Table 6, column (2), the sum of coefficients for age and team instability is -0.595 ($-0.056 + -0.539$), and in column (3), the interaction term lowers the sum to -0.263 ($-0.008 + -0.408 + 0.153$). This is apparent also from the economic interpretation of marginal effects shown in Table 7. Compared to a team with average instability maintaining a system with average age, and ignoring the interaction term for a moment, the maintenance productivity of a team with one standard deviation higher instability working on a system one standard deviation older is 15.38% lower ($-0.78 + -14.60$). Adding the interaction effect to the picture, maintenance productivity drops by only 12.32% ($-0.78 + -14.60 + 3.06$). This represents almost a 20% improvement in productivity, from -15.38% to -12.32% . By the same token, if system age and team instability are each one standard deviation below the average, productivity drops by an extra 2.98% , from -15.38% to -18.36% ($-0.78 + -14.60 - 3.06$). Overall, these results **strongly support hypotheses H1a/b and H2a/b**.

The two-way interaction graph in Figure 2 demonstrates the interaction effects. For team instability and skill-diversity, as either grows larger, maintenance productivity drops more sharply for young and low-volatility systems than for old and high-volatility systems. The interaction pattern shows a “crossing” effect. Both team attributes and system attributes have detrimental effects on productivity, but their effects are substitutive—the stronger is the effect of one, the weaker is the effect of the other. For example, using instable teams on young or low-volatility systems lowers maintenance productivity more than on old and

high-volatility systems. The same applies to interactions with team skill-diversity, except that productivity appears to increase slightly when high skill-diversity teams work on high-volatility systems.

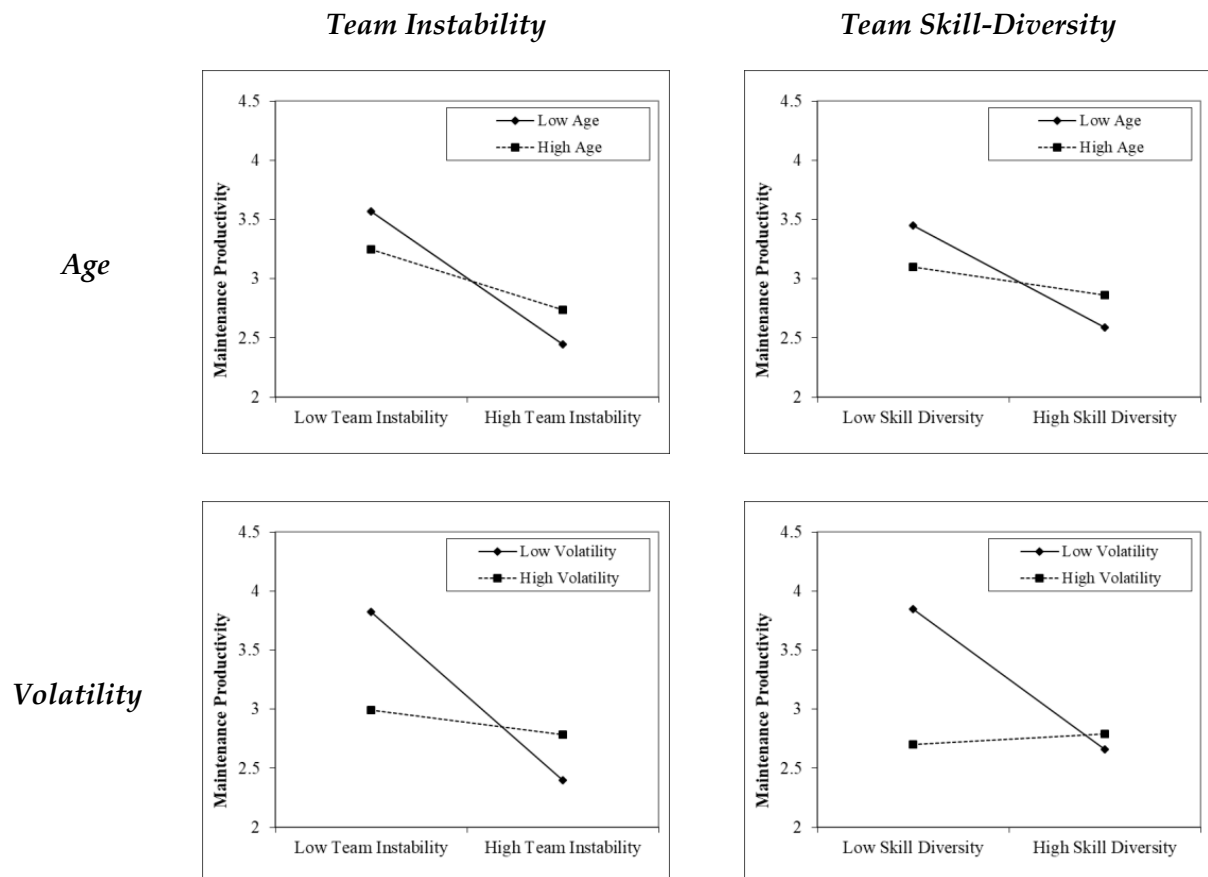


Figure 2. Interaction affect between team attributes and system attributes (dependent variable is maintenance productivity).

5.2. Robustness

We tested the robustness of our econometric results to several changes in our research design. First, we used seemingly unrelated regression (SURE). Since we estimated Model (1) using two OLS regression runs for team instability and skill-diversity separately, there is a concern that the error terms of the two regressions are correlated [85]. To rule out this possibility, we rerun the model using SURE. The OLS and SURE coefficients are nearly identical, indicating no cross-correlation of the error terms.

Second, we rerun the regressions with two alternative dependent variables. One is *cost-productivity* (**ProdUSD**), i.e., the ratio of system size to the total cost (US dollars) of maintaining a system over the three-year period. The results are similar, as seen in Table 8. Two-way interaction graphs show same patterns as well. Another dependent variable we tried is *effort-productivity* (**ProdHrs**) calculated using KLOC as an alternative measure of size. Again, the results were similar (not tabulated here).

Third, we tried alternative measures of team-related independent variables (not tabulated here).

- **Team Instability:** To account for potential effects of attrition rates, we applied a more intricate alternative measure than a simple count of maintainers that worked on a system over the three-year period. We used the share of work attributed to each maintainer and produced similar results to the ones in Table 6 (not tabulated here). The share of work attributed to a person was calculated as the average duration-share

of a person working on a system for over three years, or $\sum_k C_{i,k,j} / \sum_k C_{i,k}$, where $C_{i,k,j}$ is an hour-charge maintainer j makes for task k on system i .

- **Team Skill-Diversity:** We also used an entropy-based measure of team skill-diversity that yielded similar results to the ones in Table 6 (not tabulated here). We defined the entropy-based measure of team diversity as $\sum_{i=1}^T P_i (\ln(P_i))$, where P_i is the fraction of the team members holding a distinct functional title i , $\sum P_i = 1$ and T is the number of distinct functional titles in the team [72].

Finally, as noted in Section 4.3, we added control variables representing contextual system-specific attributes that could explain the variation in lifecycle maintenance productivity. The variables specified whether a system was developed fully in-house (**APP_Developer_D**), the line of business it serves (**LOB_D1**, **LOB_D2**, **LOB_D3**, or others), its objective recovery time (**RTO**), its recovery time capability (**RTC**), its regulatory risk (**RISK**), and its SOX compliance (**SOX_Rating**). Again, the regression results are similar to those in Table 6, where none of the added variables came out statistically significant.

Table 8. OLS regression for cost-based maintenance productivity.

Dependent Variable	Exp. Sign	Effort Productivity (ProdUSD) ($n = 426$)				
		(1)	(2)	(3)	(4)	(5)
Complexity	(−)	−0.749 *** (−20.21)	−0.760 *** (−21.61)	−0.785 *** (−24.97)	−0.761 *** (−21.22)	−0.798 *** (−24.75)
Age		−0.071 ** (−1.97)	−0.068 ** (−1.99)	−0.010 (−0.33)	−0.068 * (−1.97)	−0.025 (−0.80)
Volatility		−0.438 *** (−11.47)	−0.099 (−1.62)	−0.149 *** (−2.67)	−0.223 *** (−4.17)	−0.296 *** (−6.16)
Team Instability	(−)		−0.411 *** (−6.91)	−0.283 *** (−5.20)		
Age × Instability	(+)			0.204 *** (6.49)		
Volatility × Instability	(+)			0.228 *** (6.42)		
Team Skill-Diversity	(−)				−0.287 *** (−5.55)	−0.138 *** (−2.79)
Age × Skill-Diversity	(+)					0.207 *** (6.56)
Volatility × Skill-Diversity	(+)					0.234 *** (5.84)
R^2		0.507	0.558	0.655	0.541	0.641
adj. R^2		0.504	0.553	0.649	0.537	0.635

Standardized beta coefficients; t statistics in parentheses. * $p < 0.10$, ** $p < 0.05$, *** $p < 0.01$.

6. Discussion

6.1. Main Findings

We investigated how two system attributes—age and volatility—and two team attributes—instability and skill-diversity—affect maintenance productivity directly and indirectly in ongoing lifecycle maintenance work. Our premise is that these system attributes and team attributes interact to jointly influence maintenance productivity, and that managerial team assignment decisions achieving congruence between these attributes could improve maintenance productivity.

Recognizing knowledge as a crucial input to maintenance work, we applied knowledge creation theory to software maintenance and developed two proposition sets. The first concerned direct effects of team instability and skill-diversity, but not system age and volatility, on lifecycle maintenance productivity. We found empirical support indicating these team attributes show strong negative effects. In economic terms, compared to the average team, a team with one standard deviation higher instability and higher skill-diversity has 14.6% and 7.77% lower productivity, respectively; both figures are statistically significant at the $p < 0.01$ level. For system attributes, we expected age and volatility not to negatively affect productivity but found that volatility has a relatively negligible effect—the productivity of maintaining a system one standard deviation more volatile than the average system is 0.38% lower; this figure is not statistically significant. These results extend prior findings to the context of system lifecycle maintenance.

The second proposition set posits system and team attributes impact productivity indirectly through substitutive interactions. We found empirical support for these propositions as well. These attribute types have offsetting interaction effects: co-presence of certain system and team attributes dampens overall productivity deficits most attributes create individually. For example, compared to the average system and maintenance team, the productivity of a team with one standard deviation higher (lower) instability assigned to work on a system one standard deviation older (younger) is 3.06% higher (lower); this figure is significant at the $p < 0.01$ level. Results similarly show team skill-diversity impacts productivity via interactions with system attributes.

These results have a meaningful interpretation based on the theoretical lens we used. For example, consider the case of system age and team instability. The interaction between age and team instability reveals a substitutive relationship between two sources of knowledge disruption. In younger systems, explicit knowledge artifacts are relatively coherent and maintainers' tacit knowledge and shared mental models are still relatively aligned. Accordingly, team instability destroys valuable team shared cognition, increases new maintainers' learning burdens, and prematurely erodes tacit system and coordination knowledge. This results in a sharp productivity loss. Conversely, older systems already suffer from degraded documentation, accumulated technical debt, and weakened tacit knowledge recall. Accordingly, the marginal damage due to additional team instability is smaller, as knowledge gaps and coordination challenges are already present even in stable teams. Moreover, adding new maintainers may refresh mental models and inject expertise better aligned with the system's current state, thus partially counteracting age-related knowledge decay. Thus, from a knowledge creation perspective, the interaction underscores that the value of team stability depends critically on the quality and freshness of the underlying knowledge it preserves.

6.2. Implications for Research

Our findings offer several important theoretical and methodological contributions. First, we extend task-level software maintenance research to the system lifecycle level, demonstrating that relationships established in single-task contexts do not fully translate to ongoing maintenance environments. The lifecycle perspective reveals that direct effects of system and team attributes diminish substantially when interaction effects are considered. This finding is invisible in task-level maintenance research. This suggests prior research may have overestimated the independent influence of individual factors while underestimating the importance of attribute congruence.

Second, our use of knowledge creation theory provides a powerful lens for understanding the dynamics of software maintenance work. We demonstrated the theory's richness in conceptualizing how specific attributes and their interactions affect productivity through

knowledge search, sharing, integration, and coordination mechanisms. This theoretical framing can be extended to examine how different software development methodologies (agile, waterfall, DevOps) shape knowledge creation patterns that subsequently affect maintenance, how architectural choices (monolithic, microservices, serverless) influence the knowledge challenges teams face, and how emerging practices like documentation-as-code, knowledge graphs, and AI-assisted code comprehension tools moderate the relationships we identified.

Third, our focus on ongoing lifecycle maintenance, not just task-level work, reveals the importance of studying cumulative effects of longitudinal changes in lifecycle team attributes. Future research should examine how temporal trajectories of team stability (gradual vs. sudden changes) differentially affect productivity, whether there are threshold effects where team instability crosses from manageable to damaging, and how organizations can use predictive analytics to forecast productivity impacts of planned personnel changes. Longitudinal studies tracking the same systems and teams over 5–10 years would offer particularly compelling evidence on how ongoing personnel dynamics interact temporally with evolving system attributes.

Fourth, our findings raise questions about optimal maintenance strategies across system portfolios. Organizations maintain hundreds of systems simultaneously with constrained personnel resources. Portfolio-level research could examine, for example, whether organizations should concentrate stable, homogeneous teams on high-volatility mission-critical systems while rotating diverse teams through low-volatility systems, how to dynamically rebalance team assignments as systems age and maintainer expertise evolve, and whether portfolio optimization algorithms could improve aggregate productivity by intelligently matching team and system attributes across entire application estates.

6.3. Implications for Practice

Our findings have several actionable implications for how organizations manage maintenance teams over the system lifecycle. Broadly speaking, our findings highlight the importance of adopting a lifecycle-sensitive approach to maintenance management. Decisions about staffing, rotation, and team composition should be aligned with system attributes and knowledge conditions rather than governed by uniform organizational norms. By matching team strategies to the evolving knowledge needs of systems, organizations can better manage maintenance productivity over the long run. We offer specifics below aimed at addressing a concern expressed by [29]:

“Organizations unwittingly forgo opportunities to leverage team knowledge with practices that disrupt team structures, such as assigning members based on individual availability rather than prior association with other team members, or failing to control member turnover in standing teams.”

Staffing policies should not treat team stability and skill-diversity as uniformly beneficial. For example, stability is most valuable in the early stages of a system’s life, when shared cognition and high-quality knowledge artifacts can still be preserved. For younger systems, minimizing team instability can substantially improve maintenance productivity by protecting tacit knowledge and reducing coordination costs. Conversely, enforcing high stability in older systems may yield diminishing returns, as the knowledge environment has already degraded and much of the original shared understanding has been lost. Organizations should identify systems where team stability and skill-diversity matter most (younger, less volatile systems based on our findings) and implement policies ensuring these systems receive stable and less diverse team assignments. This might mean establishing “core teams” for such systems while using more fluid staffing for older, volatile systems where instability’s negative effects are dampened.

In addition, the stronger effects of team instability and skill-diversity indicate management has substantial leverage to improve productivity through workforce management strategies. First, given that a one standard deviation increase in instability reduces productivity by 14.6%, retention initiatives targeting key maintainers could yield substantial ROI. Second, since new team members create productivity losses, organizations should invest in structured onboarding programs, mentorship pairings, and knowledge transfer protocols. Tools like code comprehension assistants, annotated architecture diagrams, and recorded “system walkthroughs” can accelerate new maintainer ramp-up. Last, while team diversity offers benefits in complex problem-solving, our findings suggest it imposes coordination costs. Organizations might establish “communities of practice” where diverse specialists can develop shared vocabulary and understanding, reducing communication barriers when they collaborate on maintenance tasks.

More generally, our interaction effect findings reveal substantial opportunities to improve productivity through team–system congruence through strategies such as the following. First, organizations can develop staffing tools that recommend team–system pairings based on attribute compatibility. For example, when assigning work to a young, low-volatility system, prioritize stable, homogeneous teams. When assigning work to an old, high-volatility system, management should tolerate higher team instability and diversity since substitution effects dampen their negative impacts. Second, rather than randomly rotating maintainers across systems, managers can strategically rotate personnel through systems in sequences that minimize productivity losses. For instance, new maintainers could start on older, volatile systems where instability effects are smaller, then transition to younger, stable systems once they have gained experience and can contribute to stable teams. Finally, our findings suggest that reliance on offshore and distributed maintainers to make assignments based on labor cost savings alone may be counterproductive. The cost savings from offshore labor could be partially or entirely offset by productivity losses if teams are assigned to younger, less volatile systems. If offshore arrangements increase team instability and diversity, these teams should be preferentially assigned to older, more volatile systems where negative synergies are smaller.

One last implication follows from our finding that weak direct effects of system age and volatility suggest management should give substantially less weight to these attributes in formulating system retirement policies and maintenance budget allocations. For example, age-based retirement criteria lack empirical support, considering we show productivity depends more on team–system congruence than age alone. Budget models inflating maintenance costs simply because systems are old or frequently modified may misallocate resources. Instead, managers should assess whether current team configurations align appropriately with system characteristics.

6.4. Limitations

Our findings are subject to limitations. First, using data from a single firm raises the possibility of bias toward organization-specific practices. Interviews with technology executives at our research site could not rule out that the company’s conservative system-retirement policy may affect results. However, single-site data effectively addresses potential heterogeneity and rules out confounding factors likely to muddy causal relationships (e.g., environment, incentives, training, methods).

Second, we used maintenance productivity as the only outcome variable, leaving software quality as another maintenance outcome measure. Including quality in ongoing maintenance studies has challenges but could reveal important tradeoffs. Ramasubbu et al. [86] found that achieving higher productivity and quality in global software teams

requires diametrically opposed configurational choices. Unearthing similar tradeoffs in ongoing maintenance contexts could complement our findings.

Third, our analyses combined all maintenance work independent of subtypes (corrective, adaptive, perfective, preventive). Sensitivity to maintenance types might better explain system age impacts. Evidence suggests younger systems receive more corrective maintenance [87], while older systems receive more perfective maintenance aimed at dampening age's negative effect [54]. We plan to classify maintenance projects by type and expand theory development and empirical investigation accordingly.

Fourth, our three-year observation window, while substantial, may not capture long-term dynamics. Some effects—such as the gradual erosion of system comprehension as original developers depart or the cumulative impact of technical debt—may manifest over longer periods. Extending this research to 5–10-year windows would strengthen confidence in findings.

Fifth, we did not examine potential moderators of our interaction effects, such as maintenance methodology (waterfall vs. agile), architectural style (monolithic vs. microservices), or organizational culture (knowledge-sharing vs. siloed). These factors might amplify or attenuate the relationships we identified, suggesting boundary conditions for our findings.

Last, some might raise concerns about the age of our data set and the possibility that emerging AI tools may soon alter software maintenance practices. However, while generative AI has improved support for modest-size coding tasks, maintaining mission-critical enterprise systems requires deep understanding of complex legacy systems, organizational contexts where they reside, reliable verification, safe integration with legacy infrastructure, compliance with strict operational constraints, and so on. Current AI tools still lack such capabilities (e.g., [88–91]). For instance, recent studies found that AI tools can recognize code syntax but struggle with deeper semantic understanding and may hallucinate incorrect interpretations of program structures [92], and they often misjudge whether code satisfies requirements and raise reliability concerns about AI maintenance assistants [93].

More specific areas where AI tools fall short can be summarized as follows. First, software maintenance demands deep contextual understanding of decades of architectural evolution, historical design decisions, and system dependencies. These tasks remain extremely difficult for AI. Second, mission-critical enterprise systems typically feature proprietary formats, siloed data, and tightly coupled architectures. These present structural barriers to AI adoption. Third, in safety- or mission-critical domains, trustworthiness, verification, and regulatory compliance are essential. AI assistants are not yet qualified to independently maintain systems where failures are highly consequential. Finally, AI integration into large enterprise environments is constrained by cost, operational disruption, and infrastructure requirements. AI tools are not yet “plug-and-run” for most organizations.

7. Conclusions

This study makes three principal contributions to software maintenance research and practice. *Theoretically*, we demonstrate that knowledge creation theory provides a powerful lens for understanding how system and team attributes jointly shape maintenance productivity through their effects on knowledge search, sharing, integration, and coordination. Our lifecycle perspective reveals that attribute interactions substantially alter the magnitude and even direction of effects observed in task-level studies, highlighting the importance of studying ongoing maintenance as a distinct phenomenon.

Empirically, we provide the first large-scale evidence of substitutive interactions between system and team attributes in lifecycle maintenance. While team instability and skill-diversity strongly reduce maintenance productivity in isolation, their effects are sig-

nificantly dampened when systems are older and more volatile through a substitution effect invisible at the task level and in current software cost models; for example, relative to age, assigning a more unstable and skill-diverse team improves maintenance productivity by 3.06% and 3.83%, respectively ($p < 0.01$). These findings challenge conventional wisdom that older, volatile systems inherently demand stable, homogeneous teams; in fact, such systems may tolerate greater team flux precisely because their characteristics offset instability's downsides.

Practically, we demonstrate that management can substantially improve maintenance productivity not by reducing team instability and diversity per se—which may be infeasible given contemporary workforce realities—but rather by strategically aligning team configurations with system characteristics. For example, given an average age system, assigning it a maintenance team one standard deviation more (less) unstable improves (worsens) maintenance productivity by 3.06% (−3.06%) ($p < 0.01$). This shifts the maintenance management conversation from “how do we prevent team instability?” to “given inevitable instability, which systems should receive which teams?”

Looking forward, we advance several research directions. First, extending this work to emerging contexts, such as cloud-native systems and AI-augmented maintenance, will test whether our findings generalize beyond traditional enterprise systems. Second, developing prescriptive, evidence-based models and decision support tools that operationalize team–system congruence could improve staffing practices. Third, investigating temporal dynamics, such as how attribute trajectories unfold over time and how organizations can anticipate and adapt, would provide richer understanding of lifecycle maintenance as a strategic capability.

In conclusion, software maintenance remains a critical yet understudied aspect of IT management. We hope our demonstration of the importance of attribute interactions will inspire researchers and practitioners to adopt more nuanced, context-sensitive approaches to understanding and managing this essential activity. The path to improved maintenance productivity lies not in eliminating challenging team or system attributes, but in understanding how they interact and making intelligent decisions that harness these interactions to organizational advantage.

Funding: This research received no external funding.

Data Availability Statement: The data is unavailable for sharing due to the non-disclosure confidentiality agreement the authors signed with the research site providing the data.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Relevant Prior Studies

Table A1. System attributes examined in software maintenance.

Operational Measure	Work Unit	Context	Finding	Dependent Variable	Study
System Age					
Application age	Task	Maint.	n.s.	Maintenance productivity	[35]
Age of the portfolio in months	Project	Maint.	(−)	Percentage growth in module count	[37]
	“	“	(+)	Number of maintenance activities (number of corrections, adaptations, enhancements, etc.)	[37]
	“	“	(+)	Module count	[37]
	“	“	(+)	Number of corrections per module	[37]
	“	“	(+)	Number of activities per developer	[37]
Age from go-live date	Period (3 years)	Maint.	n.s.	Software volatility (no. of software changes per system over a set time frame divided by SLOC)	[14]
Old vs. new system	Period (3 years)	Maint.	(+)	System volatility (stability index)	[15]
			n.s.	System complexity (cyclomatics per module)	[37]
Time elapsed between start of the very first MR project on a system and start of the current MR	MR	Inc-Dev.	(−)	Team efficiency (MR completion time)	[36]
System Volatility					
Tot no. of enhancements performed in 3 years	Period (3 years)	Maint.	n.s.	Maintenance cost (total cost in dollars)	[13]
			(+)	Quality (number of errors)	[13]
Cyclomatic complexity	Period (3 years)	Maint.	(−)	Software volatility (no. of software changes per system over a set time frame divided by SLOC)	[14]
Time span between software changes	Period (3 years)	Maint.	(−)	Software volatility (no. software changes per system over a set time frame divided by SLOC)	[14]

n.s.—non-significant relationship.

Table A2. Team attributes examined in software maintenance.

Operational Measure	Work Scope	Context	Finding	Dependent Variable	Study
Team Knowledge-Diversity					
Team General Technical Experience					
Maintainers' individual skill level (rated by management)	Project	Maint.	(−)	Effort (tot. hours)	[56]
Personnel capability (subjective) aggregated for team	LC	“	n.s.	Productivity (size/tot. hours)	[5]
Team capability (% members rated high by others in team)	Project	“	(+)/n.s.	Productivity/Effort (tot. hours)	[10]
Technical capability of members of product team	LC	“	(+)	Productivity (size/tot. hours)	[37]
Team Experience with Maintenance					
Individual/ Avg. unrelated-systems experience	MR/Group of MRs		(−)/(−)	Productivity	[11]
Maintained vs. developed MR	MR	“	(−)	Team efficiency (in MR time completion)	[37]
	Release	“	(+)	Maintenance effort (no. of hours)	[11]
MR with more than 38% repair deltas	MR	“	(−)	Team efficiency (in MR time completion)	[37]
Team (Managerial) Role Experience					
Avg. no. of years in a given role in the team	Project	Dev	(+)	Team performance (software quality)	[12]
Team Experience with System/System Domain					
Same-system experience	MR/Group of MRs	Inc-Dev	(−)/(−)	Productivity on one MR/group MRs/release	[11]
Related-systems experience	MR/Group of MRs	“	(−)/(−)	Productivity on one MR/group MRs/release	[11]
Unrelated-systems experience	MR/Group of MRs	“	(−)/(−)	Productivity on one MR/group MRs/release	[11]
No. of past deltas completed per member	MR	“	(+)	Team efficiency (in MR time completion)	[37]
Avg. no. of similar-domain projects per member	Project	Dev	(+)	Performance (reduced effort)	[34]
3+ years exp. on maintained system, 3+ years maintenance exp., or other combination	“	Maint.	n.s.	Productivity (size/effort)	[35]
Team Stability					
Stability (duration of stay of developers in team)	Project	ut-source	(+)	Project performance (quality and timeliness of meeting expected deliverables)	[38]
Team additions and attrition over time	“		n.s.		[38]

Table A2. Cont.

Operational Measure	Work Scope	Context	Finding	Dependent Variable	Study
Team Size					
No. of developers who completed deltas in an MR	MR (delta)	Inc-Dev	(−)	Team efficiency (in MR time completion)	[37]
No. of developers in group MR	Group of MRs	“	(+)	Productivity (effort/no. of hours per group MR)	[11]
Avg. no. of developers per MR	Release	“	(−)	Productivity (effort/no. of hours per release)	[11]
Avg. role exp. across individuals in team	“	Dev	(−)	Team Perform. (schedule adherence)	[12]
Full-time equiv. headcount of no. of people in project	“	“	(−)	Quality (size/no. of code errors)	[7]
No. of people involved in project	“	“	(−)	Productivity (size/tot. effort/hours)	[86]
No. of persons involved in change task	MR	“	(−)	Performance (neg. of MR completion time)	[77]
Number of persons involved	“	“	n.s.	Quality (no. of soft. failures)	[77]
Number of persons involved	Project	“	(−)/n.s.	Productivity/Product Quality	[7]
Number of persons involved	Project	“	(−)/(−)	Productivity/Product Quality	[86]
Team Familiarity					
Individual/Avg. experience working w/others	MR/Group MRs	Inc-Dev	(−)/(−)	Productivity	[11]
Avg. no. of times a member worked w/every other member	Dev.	“	(+)	Team performance (software quality)	[12]
Avg. no. of MRs in which every pair developed code in the past	MR (delta)	“	(+)	Team efficiency (in MR time completion)	[37]
Interaction between Team Attributes					
Team variety diversity × team size	Project	Dev.	(+)	Performance	[34]
Team variety diversity × task complexity	“	“	(+)	Performance	[34]
Team familiarity (stability) × team size	MR	Inc-Dev	(+)	Team efficiency (in MR time completion)	[37]
Team familiarity (stability) × geog. dispersion	“	“	(+)	Team efficiency (in MR time completion)	[37]
Task familiarity × complexity	“	“	(−) op.	Team efficiency (in MR time completion)	[37]
Task familiarity × team familiarity (stability)	“	“	(−) op.	Team efficiency (in MR time completion)	[37]
Team instability × software complexity	Period (3 years)	Maint.	(−)	Maintenance productivity	[16]
Team skill-diversity × software complexity	Period (3 years)	Maint.	(−)	Maintenance productivity	[16]

n.s.—non-significant relationship; op.—relationship opposite to what was expected.

Appendix B. Testing for Endogeneity of Lifecycle-Team Attributes

We test for endogeneity to ensure the robustness of our empirical results. As explained in Section 4.2, the endogeneity concern applies to both lifecycle-team attributes, namely, **TeamInstability** and **TeamDiversity**. Because these lifecycle-team variables are highly correlated and used one at a time in Equation (1), we test for endogeneity of each variable separately. If each of the lifecycle-team variables is found not to be endogenous, there is no need to worry about endogeneity of the interaction term [88]. (It is worth pointing out a well-established scenario in the econometric literature [88]. When the goal is to estimate the parameter of an interaction term ($X_1 \times X_2$) of an endogenous variable (X_1) and an exogenous variable (X_2), where all other regressors are jointly independent of X_2 , “the OLS estimate of the coefficient on this interaction term is consistent” ([88], p. 72). This means that an omitted variable causes no real trouble when the primary interest is in the interaction term, similar to our case.) Hence, we can check for endogeneity of one team attribute at a time, using Equation (1) with the interaction terms excluded.

We test for endogeneity using Two-Stage Least Squares (2SLS) estimation, **whose results suggest no endogeneity problem with either lifecycle-team attributes**. To identify instrumental variables (IV), we rest on the plausibility that certain contextual or system-specific factors available in the data may impose greater maintenance demands and induce management to “preferentially” assign certain maintenance tasks to specific work sites and/or maintainer teams. The contextual factors (IVs) we use: **InfoType** indicates the type of information a system handles (confidential, internal, public, etc.), **SOX_Rating** indicates a system’s level of SOX compliance (none, low, high), **APP_Developer_D** indicates if a system was developed fully in-house (or by a third-party), **RTO** indicates a system’s objective recovery time, and **LOB_C** indicates if a system serves a central line of business (as opposed to less central lines of business). These IVs meet the essential identification requirement [84]. They are associated with the variables tested for endogeneity (IV relevance) but not directly related with the outcome variable (IV validity). As seen in Table A3, the IVs do not impact the outcome variable based on their low correlations with lifecycle maintenance productivity (**ProdHrs**); also, they have statistically insignificant coefficients when regressed against **ProdHrs** ($p = 0.267$; $R\text{-sq} = 0.001$). At the same time, the contextual variables have significant and relatively high correlations with the lifecycle-team variables (Table A3). More importantly, as seen in Table A4, when IVs are regressed against each lifecycle-team attribute, their coefficients are significant and the goodness-of-fit F-statistics exceed the threshold of 10 for detecting weak instruments [84].

Table A3. Pairwise correlation matrix with IVs.

Variables	Mean	Min	Max	Std. Dev.	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
(1) ProdHrs	0.18	0.65	2.99	0.22	1.00							
(2) ln(TeamInstability)	0.00	4.49	7.12	1.23	−0.41 *	1.00						
(3) ln(TeamDiversity)	0.00	2.99	4.20	0.78	−0.40 *	0.92 *	1.00					
(4) InfoType	4.62	2	5	0.80	0.02	0.26 *	0.26 *	1.00				
(5) SOX_Rating	0.13	0	2	0.48	0.02	0.20 *	0.17 *	0.06	1.00			
(6) APP_Developer_D	0.93	0	1	0.24	−0.02	−0.05	−0.10 *	0.06	0.11 *	1.00		
(7) RTO	15.2	1	100	24.42	−0.09	−0.30 *	−0.26 *	−0.10 *	−0.11 *	0.04	1.00	
(8) LOB_C	0.12	0	1	0.33	−0.09	−0.28 *	−0.24 *	−0.18 *	−0.04	0.04	0.47 *	1.00

N = 426. * shows significance at the 0.05 level.

Table A4. Regressing IVs on the endogenous variables.

	TeamInstability	TeamDiversity
InfoType	0.194 *** (4.32)	0.205 *** (4.50)
SOX_Rating	0.166 *** (3.73)	0.147 *** (3.23)
APP_Developer_D	−0.082 * (−1.84)	−0.130 *** (−2.87)
LOB_C	−0.155 *** (−3.07)	−0.121 ** (−2.36)
RTO	−0.189 *** (−3.75)	−0.161 *** (−3.16)
R^2	0.188	0.162
adj. R^2	0.179	0.152
F-stat	29.37	37.68

Standardized beta coefficients; t statistics in parentheses. * $p < 0.10$, ** $p < 0.05$, *** $p < 0.01$. $N = 426$.

We run 2SLS estimation with the IV method [84] for each team attribute separately. The results are summarized in Table A5. Specifically, we run Stata's *ivreg 2sls* with a model that instruments one lifecycle-team attribute at a time. This concurrently estimates two equations: an inner one regresses the team attribute on IVs (**SOX_Rating**, **APP_Developer_D**, **RTO**, and **LOB_C**), and an outer one regresses the explanatory model with the team attribute replaced by its inner regression estimate. As seen in Table A5, the results are similar qualitatively to those of standard OLS estimation, in sign and significance. The conclusion, again, is that endogeneity of team attributes is not an issue.

Table A5. Regression results with endogeneity tests in the high-complexity subsample.

Dependent Variable		Effort Productivity (ProdHrs) ($n = 146$)			
		(1) OLS	(2) 2SLS	(3) OLS	(4) 2SLS
Age		−0.149 ** (−2.45)	0.043 (0.56)	−0.142 ** (−2.46)	0.064 (0.96)
SoftVolatility		0.063 (0.55)	0.363 (1.12)	0.022 (0.24)	−0.004 (−0.02)
Complexity	(−)	0.154 ** (2.50)	0.306 *** (4.40)	0.147 ** (2.50)	0.299 *** (4.78)
TeamInstability	(−)	−0.736 *** (−6.51)	−1.216 *** (−3.21)		
TeamDiversity	(−)			−0.735 *** (−7.90)	−0.836 *** (−3.44)
R^2		0.486	0.359	0.536	0.480
adj. R^2		0.472	0.341	0.523	0.465

OLS = Ordinary Least Square method, 2SLS = 2-stage least square method. Standardized beta coefficients; t statistics in parentheses. ** $p < 0.05$, *** $p < 0.01$.

References

1. Erlikh, L. Leveraging legacy system dollars for e-business. *IT Prof.* **2000**, *2*, 17–23. [CrossRef]
2. Seacord, R.C.; Plakosh, D.; Lewis, G.A. *Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices*; Addison-Wesley Professional: Boston, MA, USA, 2003.
3. Dekleva, S.; Drehmer, D. Measuring software engineering evolution: A Rasch calibration. *Inf. Syst. Res.* **1997**, *8*, 95–104. [CrossRef]
4. Pigoski, T.M. *Practical Software Maintenance: Best Practices for Managing Your Software Investment*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 1996.

5. Krishnan, M.S.; Kriebel, C.H.; Kekre, S.; Mukhopadhyay, T. An Empirical Analysis of Productivity and Quality in Software Products. *Manage. Sci.* **2000**, *46*, 745–759. [\[CrossRef\]](#)
6. Hoda, R.; Salleh, N.; Grundy, J.; Tee, H.M. Systematic literature reviews in agile software development: A tertiary study. *Inf. Softw. Technol.* **2017**, *85*, 60–70. [\[CrossRef\]](#)
7. Ramasubbu, N.; Balan, R.K. Globally Distributed Software Development Project Performance: An Empirical Analysis. In Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Dubrovnik, Croatia, 3–7 September 2007; pp. 125–134.
8. Sull, D.; Sull, C.; Zweig, B. Toxic culture is driving the great resignation. *MIT Sloan Manag. Rev.* **2022**, *63*, 1–9.
9. Dibble, R.; Gibson, C.B. Crossing team boundaries: A theoretical model of team boundary permeability and a discussion of why it matters. *Hum. Relat.* **2018**, *71*, 925–950. [\[CrossRef\]](#)
10. Banker, R.D.; Datar, S.; Kemerer, C.F. Factors Affecting Software Maintenance Productivity: An Explanatory Study. In *Proceedings of the Eighth International Conference on Information Systems*; ACM: New York, NY, USA, 1987; pp. 160–175.
11. Boh, W.F.; Slaughter, S.A.; Espinosa, J.A. Learning from experience in software development: A multilevel analysis. *Manag. Sci.* **2007**, *53*, 1315–1331. [\[CrossRef\]](#)
12. Huckman, R.S.; Staats, B.R.; Upton, D.M. Team Familiarity, Role Experience, and Performance: Evidence from Indian Software Services. *Manag. Sci.* **2009**, *55*, 85–100. [\[CrossRef\]](#)
13. Banker, R.D.; Slaughter, S.A. The moderating affects of structure on volatility and complexity in software enhancement. *Inf. Syst. Res.* **2000**, *11*, 219–240. [\[CrossRef\]](#)
14. Zhang, X.; Windsor, J.; Pavur, R. Determinants of software volatility: A field study. *J. Softw. Maint. Evol. Res. Pract.* **2003**, *15*, 191–204. [\[CrossRef\]](#)
15. Heales, J. Factors affecting information system volatility. In *Proceedings of 20th International Conference on Information Systems, Atlanta, GA, USA, 12–15 December 1999*; Association for Information Systems: Atlanta, GA, USA, 2000; pp. 70–83.
16. Benaroch, M.; Lyytinen, K. How Much Does Software Complexity Matter for Maintenance Productivity? The Link Between Team Instability and Diversity. *IEEE Trans. Softw. Eng.* **2023**, *49*, 2459–2475. [\[CrossRef\]](#)
17. Faraj, S.; Sproull, L. Coordinating expertise in software development teams. *Manag. Sci.* **2000**, *46*, 1554–1568. [\[CrossRef\]](#)
18. Espinosa, J.A.; Nan, N.; Carmel, E. Do Gradations of Time Zone Separation Make a Difference in Performance? A First Laboratory Study. In Proceedings of the ICGSE, Munich, Germany, 27–30 August 2007; pp. 12–22.
19. Banker, R.D.; Davis, G.B.; Slaughter, S.A. Software development practices, software complexity, and software maintenance performance: A field study. *Manag. Sci.* **1998**, *44*, 433–450. [\[CrossRef\]](#)
20. Nah, F.F.; Faja, S.; Cata, T. Characteristics of ERP software maintenance: A multiple case study. *J. Softw. Maint. Evol. Res. Pract.* **2001**, *13*, 399–414. [\[CrossRef\]](#)
21. Hansen, S.W.; Robinson, W.N.; Lyytinen, K.J. Computing Requirements: Cognitive Approaches to Distributed Requirements Engineering. In Proceedings of the 45th Hawaii International Conference on System Sciences (HICSS), Maui, HI, USA, 4–7 January 2012; pp. 5224–5233.
22. Davison, J.W.; Mancl, D.M.; Opdyke, W.F. Understanding and Addressing the Essential Costs of Evolving Systems. *Bell Labs Tech. J.* **2000**, *5*, 44–54. [\[CrossRef\]](#)
23. Fraser, S.; Beck, K.; Booch, G.; Coplien, J.; Johnson, R.; Opdyke, W. Beyond the Hype: Do Patterns and Frameworks Reduce Discovery Costs? In Proceedings of the 12th Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '97), Atlanta, GA, USA, 5–9 October 1997; pp. 342–344.
24. Pfleeger, S.L. What software engineering can learn from soccer. *IEEE Softw.* **2002**, *19*, 64–65. [\[CrossRef\]](#)
25. Espinosa, J.A.; Slaughter, S.A.; Kraut, R.E.; Herbsleb, J.D. Team knowledge and coordination in geographically distributed software development. *J. Manag. Inf. Syst.* **2007**, *24*, 135–169. [\[CrossRef\]](#)
26. Narayanan, S.; Balasubramanian, S.; Swaminathan, J.M. A matter of balance: Specialization, task variety, and individual learning in a software maintenance environment. *Manag. Sci.* **2009**, *55*, 1861–1876. [\[CrossRef\]](#)
27. Brooks, F.P. The mythical man-month. *ACM SIGPLAN Not.* **1975**, *10*, 193. [\[CrossRef\]](#)
28. Britto, R.; Smite, D.; Damm, L.-O.; Börstler, J. Evaluating and strategizing the onboarding of software developers in large-scale globally distributed projects. *J. Syst. Softw.* **2020**, *169*, 110699. [\[CrossRef\]](#)
29. Lewis, K. Knowledge and performance in knowledge-worker teams: A longitudinal study of transactive memory systems. *Manag. Sci.* **2004**, *50*, 1519–1533. [\[CrossRef\]](#)
30. Brenner, R. Balancing resources and load: Eleven nontechnical phenomena that contribute to formation or persistence of technical debt. In Proceedings of the 2019 IEEE/ACM International Conference on Technical Debt, TechDebt 2019, Montreal, QC, Canada, 26–27 May 2019; pp. 38–47. [\[CrossRef\]](#)
31. Paletz, S.B.F.; Schunn, C.D. Assessing group-level participation in fluid teams: Testing a new metric. *Behav. Res. Methods* **2011**, *43*, 522–536. [\[CrossRef\]](#)

32. Edmondson, A.C. Speaking up in the operating room: How team leaders promote learning in interdisciplinary action teams. *J. Manag. Stud.* **2003**, *40*, 1419–1452. [CrossRef]
33. Huckman, R.S.; Staats, B.R. Fluid tasks and fluid teams: The impact of diversity in experience and team familiarity on team performance. *Manuf. Serv. Oper. Manag.* **2011**, *13*, 310–328. [CrossRef]
34. Kang, K.; Hahn, J.; De, P. Learning Effects of Domain, Technology, and Customer Knowledge in Information Systems Development: An Empirical Study. *Inf. Syst. Res.* **2017**, *28*, 797–811. [CrossRef]
35. Jorgensen, M. Experience with the accuracy of software maintenance task effort prediction models. *IEEE Trans. Softw. Eng.* **1995**, *21*, 674–681. [CrossRef]
36. Espinosa, J.A.; Slaughter, S.A.; Kraut, R.E.; Herbsleb, J.D. Familiarity, Complexity, and Team Performance in Geographically Distributed Software Development. *Organ. Sci.* **2007**, *18*, 613–630. [CrossRef]
37. Barry, E.J.; Kemerer, C.F.; Slaughter, S.A. How software process automation affects software evolution: A longitudinal empirical analysis. *J. Softw. Maint. Evol. Res. Pract.* **2007**, *19*, 1–31. [CrossRef]
38. Narayanan, S.; Balasubramanian, S.; Swaminathan, J.M. Managing Outsourced Software Projects: An Analysis of Project Performance and Customer Satisfaction. *Prod. Oper. Manag.* **2011**, *20*, 508–521. [CrossRef]
39. Espinosa, J.A.; DeLone, W.; Lee, G. Global boundaries, task processes and IS project success: A field study. *Inf. Technol. People* **2006**, *19*, 345–370. [CrossRef]
40. Bunderson, S.J.; Sutcliffe, K.M. Comparing Alternative Conceptualizations of Functional Diversity in Management Teams: Process and Performance Effects. *Acad. Manag. J.* **2002**, *45*, 875–893. [CrossRef]
41. Jehn, K.A.; Northcraft, G.B.; Neale, M.A. Why differences make a difference: A field study of diversity, conflict, and performance in workgroups. *Adm. Sci. Q.* **1999**, *44*, 741–763. [CrossRef]
42. Pelled, L.H.; Eisenhardt, K.M.; Xin, K.R. Exploring the black box: An analysis of work group diversity, conflict, and performance. *Adm. Sci. Q.* **1999**, *44*, 1–28. [CrossRef]
43. Shaft, T.M.; Vessey, I. The role of cognitive fit in the relationship between software comprehension and modification. *MIS Q.* **2006**, *30*, 29–55. [CrossRef]
44. Rodriguez, O.; Martinez, A.; Vizcaino, A.; Favela, J.; Piattini, M. Identifying Knowledge Management Needs in Software Maintenance Groups: A Qualitative Approach. In *Proceedings of the 5th Mexican International Conference on Computer Science (ENC 2004), Colima, Mexico, 24 September 2004*; IEEE Computer Society Press: Los Alamitos, CA, USA, 2004; pp. 72–79.
45. Mohd, N.M.Z.; Abdullah, R.; Murad, M.A.A.; Selamat, M.H. Managing Knowledge in Collaborative Software Maintenance Environment. In *Knowledge Management*; Virtanen, P., Helander, N., Eds.; Books on Demand: Norderstedt, Germany, 2010.
46. Nonaka, I.; von Krogh, G.; Voelpel, S. Organizational knowledge creation theory: Evolutionary paths and future advances. *Organ. Stud.* **2006**, *27*, 1179–1208. [CrossRef]
47. Agresti, W. Knowledge Management. *Adv. Comput.* **2000**, *53*, 171–283.
48. Liang, D.W.; Moreland, R.; Argote, L. Group versus individual training and group performance: The mediating role of transactive memory. *Pers. Soc. Psychol. Bull.* **1995**, *21*, 384–393. [CrossRef]
49. Wegner, D.M. Transactive memory: A contemporary analysis of the group mind. In *Theories of Group Behavior*; Mullen, B., Goethals, G.R., Eds.; Springer: New York, NY, USA, 1986; pp. 185–205.
50. Bedwell, W.L. Adaptive Team Performance: The Influence of Membership Fluidity on Shared Team Cognition. *Front. Psychol.* **2019**, *10*, 2266. [CrossRef]
51. Ellwart, T.; Antoni, C.H. Shared and Distributed Team Cognition and Information Over-load: Evidence and Approaches for Team Adaptation. In *Information and Communication Overload in the Digital Age*; Marques, R.P.F., Batista, J.C.L., Eds.; IGI Global Scientific Publishing: Hershey, PA, USA, 2017.
52. Cannon-Bowers, J.A.; Salas, E.; Converse, S. Shared mental models in expert team decision making. In *Individual and Group Decision Making: Current Issues*; Castellan, N.J., Jr., Ed.; Lawrence Erlbaum Associates, Inc.: Mahwah, NJ, USA, 1993; pp. 221–246.
53. Barry, E.; Slaughter, S.A. Measuring software volatility: A multi-dimensional approach. In *Proceedings of the 21st International Conference on Information Systems, Brisbane, Australia, 10–13 December 2000*; Association for Information Systems: Atlanta, GA, USA, 2000; pp. 412–413.
54. Jones, C. Geriatric Issues of Aging Software. *CROSSTALK J. Def. Softw. Eng.* **2007**, *20*, 4–8. Available online: <https://api.semanticscholar.org/CorpusID:114226117> (accessed on 26 March 2026).
55. Kemerer, C.F. Reliability of function points measurement: A field experiment. *Commun. ACM* **1993**, *36*, 85–97. [CrossRef]
56. Banker, R.D.; Datar, S.M.; Kemerer, C.F.; Zweig, D. Software complexity and maintenance costs. *Commun. ACM* **1993**, *36*, 81–94. [CrossRef]
57. Kemerer, C.F.; Slaughter, S.A. Determinants of Software Maintenance Profiles: An Empirical Investigation. *J. Softw. Maint. Res. Pract.* **1997**, *9*, 235–251. [CrossRef]
58. Lehman, M.M. Laws of Software Evolution Revisited. In *European Workshop on Software Process Technology*; Springer: Heidelberg, Germany, 1996; pp. 108–124.

59. Hager, J. Software cost reduction methods in practice: A post mortem analysis. *J. Syst. Softw.* **1991**, *14*, 67–77. [\[CrossRef\]](#)
60. Butcher, G.G. *Addressing Software Volatility in the System Life Cycle*; Colorado Technical University: Colorado Springs, CO, USA, 1997.
61. Littlewood, B.; Strigini, L. The risks of software. *Sci. Am.* **1992**, *267*, 62. [\[CrossRef\]](#)
62. Holden, S.J.S.; Vanhuele, M. Know the name, forget the exposure: Brand familiarity versus memory of exposure context. *Psychol. Mark.* **1999**, *16*, 479–496. [\[CrossRef\]](#)
63. Ton, Z.; Huckman, R.S. Managing the impact of employee turnover on performance: The role of process conformance. *Organ. Sci.* **2008**, *19*, 56–68. [\[CrossRef\]](#)
64. Gaimon, C. Planning information technology-knowledge worker systems. *Manag. Sci.* **1997**, *43*, 1308–1328. [\[CrossRef\]](#)
65. Argote, L.; Learning, O. *Retaining and Transferring Knowledge*; Kluwer Academic Publishers: Norwell, MA, USA, 1999.
66. McCarter, M.W.; Sheremeta, R.M. You Can't Put Old Wine in New Bottles: The Effect of Newcomers on Coordination in Groups. *PLoS ONE* **2013**, *8*, e55058. [\[CrossRef\]](#)
67. Argote, L.; Lee, S.; Park, J. Organizational learning processes and outcomes: Major findings and future research directions. *Manag. Sci.* **2021**, *67*, 5399–5429. [\[CrossRef\]](#)
68. Cramton, C.D. The mutual knowledge problem and its consequences for dispersed collaboration. *Organ. Sci.* **2001**, *12*, 346–371. [\[CrossRef\]](#)
69. Aral, S.; Brynjolfsson, E.; Van Alstyne, M.W. Antecedents and Consequences of Mutual Knowledge in Teams. 2008. Available online: <https://ssrn.com/abstract=1299260> (accessed on 26 March 2026).
70. Hinds, P.J.; Mortensen, M. Understanding conflict in geographically distributed teams: The moderating effects of shared identity, shared context, and spontaneous communication. *Organ. Sci.* **2005**, *16*, 290–307. [\[CrossRef\]](#)
71. Przybilla, L.; Wiesche, M.; Thatcher, J.B. Conceptualizing Fluid Team Membership and Its Effects in IT Pro-jects: A Preliminary Model. In Proceedings of the 28th European Conference on Information Systems (ECIS2020), Marrakech, Morocco, 15–17 June 2020.
72. Liang, T.; Liu, C.; Lin, T.; Lin, B. Effect of team diversity on software project performance. *Ind. Manag. Data Syst.* **2007**, *107*, 636–653. [\[CrossRef\]](#)
73. Mark, G.; Lyytinen, K.; Bergman, M. Boundary objects in design: An ecological view of design artifacts. *J. Assoc. Inf. Syst.* **2007**, *8*, 546–568. [\[CrossRef\]](#)
74. Zhou, Y.M. Designing for complexity: Using divisions and hierarchy to manage complex tasks. *Organ. Sci.* **2013**, *24*, 339–355. [\[CrossRef\]](#)
75. Cohen, W.M.; Levinthal, D.A. Absorptive Capacity: A New Perspective on Learning and Innovation. *Adm. Sci. Q.* **1990**, *35*, 128–152. [\[CrossRef\]](#)
76. Boehm, B.W.; Abts, C.; Brown, A.W.; Chulani, S.; Clark, B.K.; Horowitz, E.; Steece, B. *Software Cost Estimation with COCOMO II*, 1st ed.; Prentice Hall PTR: Upper Saddle River, NJ, USA, 2000.
77. Herbsleb, J.; Mockus, A. An empirical study of speed and communication in globally distributed software development. *IEEE Trans. Softw. Eng.* **2003**, *29*, 481–494. [\[CrossRef\]](#)
78. Toothaker, L.E.; Aiken, L.S.; West, S.G. *Multiple Regression: Testing and Interpreting Interactions*; Sage: Newbury Park, CA, USA, 1991.
79. Card, D.N. The Challenge of Productivity Measurement. In Proceedings of the Pacific Northwest Software Quality Conference, Portland, OR, USA, 9–11 October 2006.
80. Mortensen, M.; Haas, M.R. Rethinking teams: From bounded membership to dynamic participation. *Organ. Sci.* **2018**, *29*, 341–355. [\[CrossRef\]](#)
81. Narayanan, S.; Swaminathan, J.M.; Talluri, S. Knowledge diversity, turnover, and organizational-unit productivity: An empirical analysis in a knowledge-intensive context. *Prod. Oper. Manag.* **2014**, *23*, 1332–1351. [\[CrossRef\]](#)
82. Kemerer, C.F. Software complexity and software maintenance: A survey of empirical research. *Ann. Softw. Eng.* **1995**, *1*, 1–22. [\[CrossRef\]](#)
83. Bollinger, G.; Belsley, D.A.; Kuh, E.; Welsch, R.E. *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*; Wiley and Sons: New York, NY, USA, 1980.
84. Wooldridge, J.M. *Introductory Econometrics: A Modern Approach*, 4th ed.; South-Western: Melbourne, Australia, 2009.
85. Greene, W.H. *Econometric Analysis*, 5th ed.; Pearson Education: London, UK, 2005.
86. Ramasubbu, N.; Kemerer, C.F.; Hong, J. Structural Complexity and Programmer Team Strategy: An Experimental Test. *IEEE Trans. Softw. Eng.* **2012**, *38*, 1054–1068. [\[CrossRef\]](#)
87. Gefen, D.; Schneberger, S.L. The Non-Homogeneous Maintenance Periods: A Case Study of Software Modifications. In Proceedings of the Conference Software Maintenance, Monterey, CA, USA, 4–8 November 1996.
88. Nizalova, O.Y.; Murtazashvili, I. Exogenous Treatment and Endogenous Factors: Vanishing of Omitted Variable Bias on the Interaction Term. *J. Econ. Methods* **2014**, *5*, 71–77. [\[CrossRef\]](#)

89. Singh, M. Integrating Artificial Intelligence with Legacy Systems: A Systematic Analysis of Challenges and Strategic Considerations. *Eur. J. Comput. Sci. Inf. Technol.* **2025**, *13*, 38–45. [[CrossRef](#)]
90. Diggs, C.; Doyle, M.; Madan, A.; Scott, S.; Escamilla, E.; Zimmer, J.; Thaker, S. Leveraging LLMs for Legacy Code Modernization: Challenges and Opportunities for LLM-Generated Documentation. Available online: <https://arxiv.org/abs/2411.14971> (accessed on 26 March 2026).
91. Klemmer, J.H.; Horstmann, S.A.; Patnaik, N.; Ludden, C.; Burton, C.; Powers, C.; Massacci, F.; Rahman, A.; Votipka, D.; Lipford, H.R.; et al. Using AI Assistants in Software Development: A Qualitative Study on Security Practices and Concerns: Extended Version. In Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24), Salt Lake City, UT, USA, 14–18 October 2024; Volume 1. [[CrossRef](#)]
92. Ma, W.; Liu, S.; Lin, Z.; Wang, W.; Hu, Q.; Liu, Y.; Liu, Y. LLMs: Understanding Code Syntax and Semantics for Code Analysis. *arXiv* **2017**, arXiv:2305.12138.
93. Jin, H.; Chen, H. Are LLMs Reliable Code Reviewers? Systematic Overcorrection in Requirement Conformance Judgement. *arXiv* **2026**, arXiv:2603.00539. [[CrossRef](#)]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.