

Open Source Software, Closed Software Factory

James C. Davis*, Wenxin Jiang[†], Nicholas Synovic[‡], George K. Thiruvathukal[‡]

*Purdue University [†]Socket Inc. [‡]Loyola University Chicago

davisjam@purdue.edu wenxin@socket.dev

nsynovic@luc.edu gthiruvathukal@luc.edu

Abstract—Can software vendors disclose source code while retaining a development advantage in their software factories? Using generative AI, competitors are increasingly reverse-engineering or replicating functionality, so source secrecy offers less protection as a “moat”. Offering source access may attract customers seeking assurance or interoperability without necessarily enabling recipients to maintain or extend the software at comparable cost. We conceptualize this as *open source, closed factory*, treating disclosure as a choice of engineering artifacts and evaluating its effects on development cost. We introduce the *factory multiplier*, an end-to-end engineering cost ratio, and propose experiments that selectively remove engineering artifacts from originating factories or provide them to recipients, measuring how these changes affect development costs and whether increasingly capable agents can compensate for missing artifacts.

“A company can outperform rivals only if it can establish a difference that it can preserve”

M. Porter [1]

Index Terms—open source, software supply chain, coding agents, reproducibility, software evolution

I. INTRODUCTION

Generative AI threatens many software companies’ business models, which assume that it is expensive to develop substantial applications (“moat” [2]). In 2026, an individual engineer used coding agents to create open-source alternatives to products from Microsoft and Adobe [3]. Even if these alternatives are not feature complete,¹ they motivate us to examine the value proposition of source code secrecy. An existing product gives imitators a target: they can reproduce its observable functionality [4]–[7]. Maintaining a competitive product, however, requires continually identifying new requirements and delivering acceptable changes [1]. Distributing source code does not necessarily enable recipients to change it at costs comparable to those of the originating organization.

We call the system supporting this ongoing development a *software factory*: the engineering knowledge, tools, validation machinery, and workflows used to turn changing requirements into reliable software updates [8], [9]. Whereas source code embodies implementation decisions, factory resources can preserve the reasons behind those decisions and the constraints that subsequent changes must respect (the “why”

¹We tried the Office ones — they seem adequate for everyday work.

TABLE I

HYPOTHESIZED EFFECTS OF AGENTIC DEVELOPMENT ON FACTORY ADVANTAGE. FOR TASKS DEPENDENT ON ACCUMULATED ENGINEERING KNOWLEDGE, THE FACTORY MAY RETAIN A COST ADVANTAGE.

Task	Engineering problem	Hypothesis
Replication	Reproduce functionality.	Agents reduce replication costs.
Modification	Implement a specified change.	Agents reduce development cost advantage.
Evolution	Resolve ambiguous requirements & changing constraints.	Software factory sustains cost advantage.

behind the “what”). Table I distinguishes three engineering tasks with different dependencies on accumulated engineering knowledge. Replication and modification have relatively well-defined targets, whereas evolution requires determining what changes are needed and what constitutes an acceptable result.

This distinction motivates a posture of *open source, closed factory*, where vendors may consider distributing source code to benefit clients while reserving engineering resources that support their development cost advantage. We treat distribution as a choice of which factory resources to distribute, rather than a binary choice to reveal source. We propose a research program to measure how *factory advantage* varies by engineering task; identify which engineering artifacts contribute to that advantage; and determine whether their benefits transfer to recipients. These relationships may change as recipients reconstruct missing resources or adopt stronger agents.

We introduce the *factory multiplier* (§IV), a general metric for comparing engineering costs between software factories. We develop one experimental operationalization that accounts for requirements interpretation, implementation, and validation. Then we propose controlled experiments (§V) to identify the sources and transferability of factory advantage. In §VI we interpret their outcomes.

Hypothesis. *As coding agents reduce the cost of replicating and modifying software, vendors can retain a development cost advantage through factory resources that support software evolution even if source is open. This advantage increases with specification ambiguity and environmental complexity. Its commercial value depends on the need for continued software evolution.*

This hypothesis reflects Schumpeter’s notion of *creative destruction* [10]: agentic development may erode the competitive value of software artifacts while increasing the relative importance of the factories that produce and evolve them.

TABLE II
CUSTOMER OBJECTIVES AND BENEFITS OF SOURCE ACCESS.

Objective	How source access helps
Security audit	Inspect security-sensitive behavior against stated requirements [15], [19].
Contractual acceptance	Supplement acceptance tests by inspecting the implementation [20].
Formal verification	Check code against properties and proof assumptions.
Interoperability	Resolve gaps in protocol or format documentation by inspecting the implementation [14].
Regulatory assessment	Inspect code for evidence of regulatory conformance [21].

II. WHY DISTRIBUTE SOURCE?

If competitors can reproduce a product without its source, withholding that code offers little competitive protection. Investment firms are already evaluating software companies’ technical “moats” in light of inexpensive replication [11], [12], while agent-assisted reverse engineering expands the means of reproducing software without source access [13].

Meanwhile, customers may benefit from source access without intending to maintain or extend the software themselves (Table II). The appropriate disclosure depends on their objectives. For example, specification programs document protocols, file formats, and other interoperability mechanisms [14], though they do not establish implementation conformance. Source inspection can provide additional evidence of how requirements are implemented and support security assurance [15]–[18]. Thus, distributing source may provide customers with assurance or interoperability benefits even when they intend to continue to pay for the software.

III. COMMERCIAL ADVANTAGE: CLOSED FACTORY?

A retained development advantage requires resources that improve engineering outcomes and are costly for recipients to reconstruct or replace [1]. Divergent implementations may also duplicate maintenance effort when fixes must be adapted across versions, particularly if recipients lack the resources needed to preserve compatibility.

A. Resources a Source Release May Leave Out

Software factories and product lines have long organized reusable architectures, models, and processes into production systems [8], [22], [23]. Agentic approaches make some of this knowledge executable as instructions, tools, and checks [9], [24]. Following [25], we distinguish four overlapping categories of factory resources whose contribution and transferability can be tested. These categories identify candidate resources for comparison rather than assume that every resource provides an advantage. The factory multiplier evaluates the cost difference (§IV), while artifact comparisons investigate which resources contribute to it (§V-C).

Engineering Knowledge: Requirements, documentation, architectural models, and dependency graphs describe the system

TABLE III
REPORTED IMPROVEMENTS IN INDUSTRIAL AGENTIC DEVELOPMENT ENVIRONMENTS.

Company	Reported measurement	Change
Spotify	Pull-request frequency [33]	76% increase
Uber	Cost per agent session [34]	52% decrease
Shopify	Agent merge rate, increased 36% to 77% [35]	2.1×
Amazon	Median deployment velocity, 50+ teams [36]	4.5×

and its obligations; structured repository representations capture control and data dependencies for context retrieval [26]. Cloudflare reports consolidating engineering guidance into a governed corpus retrieved by agents during code and design review [27]. Such resources may remain in internal documentation systems or contain proprietary knowledge.

Agent Guidance and Tools: Repository instructions, skills, retrieval mechanisms, and interfaces guide agents’ code navigation, editing, and execution; specialized interfaces, structural search, and fault localization support task performance and code localization [28], [29]. Uber reports more than 3,600 engineer-created skills and over 30,000 daily executions [30]. Such resources may be maintained separately or depend on internal tools and services.

Validation and Enforcement: Tests, static analyses, conformance checks, policy validators, and admission gates assess and admit changes: models express obligations, validators check conformance, and gates enforce results [24]. GitLab describes a harness combining automated constraints, independent review, and recurring maintenance [31]. Such resources may depend on private test data, internal policies, or separately managed review and admission systems.

Production Infrastructure: Build and deployment pipelines, orchestration, execution environments, and workflows support repeatable production and evaluation through operational resources as well as repository configuration. Anthropic reports replacing its source control infrastructure with Scion for growing agent workloads, while cautioning that its specialized architecture is not a general replacement for existing systems [32]. Such resources may depend on hosted services, credentials, or organization-specific infrastructure unavailable in a source distribution.

B. Reported Industrial Improvements

Table III summarizes reported improvements in agent acceptance, engineering activity, deployment velocity, and operating cost associated with investments in agentic development environments. These heterogeneous measurements motivate studying factory resources but are neither directly comparable nor evidence of a lower cost per acceptable change.

Spotify reports better agent performance in standardized codebases and describes how internal tooling provides component context and automated feedback on engineering standards [33]. Uber and Shopify report improvements without changing models, suggesting a contribution from the surrounding environment without isolating its causal effect [34], [35].

Amazon reports greater gains from changing both tools and engineering practices than from adopting tools alone [36].

These accounts neither separate general agent capabilities from organization specific resources nor establish what recipients could reproduce from disclosed source. They also omit some costs of constructing and maintaining the reported factories. We therefore propose comparing the costs of acquiring, reconstructing, or replacing those resources rather than inferring a retained advantage from the reports alone.

IV. MEASURING THE FACTORY MULTIPLIER

We propose the *factory multiplier*, M , to quantify the engineering cost advantage a vendor retains after distributing source code and selected supporting artifacts. It compares the recipient's cost of achieving acceptable outcomes with the vendor's cost using its originating software factory. Different operationalizations accommodate different engineering tasks and available measurements.

A. Abstract Metric

Factory multiplier. The factory multiplier M measures the relative engineering cost of achieving acceptable software changes in two factories. For a software factory F , let $K(F)$ denote the engineering cost of achieving acceptable outcomes on comparable tasks under independent acceptance criteria (smaller is better). Then:

$$M = \frac{K(F_{\text{recipient}})}{K(F_{\text{vendor}})}. \quad (1)$$

Values $M \gg 1$ indicate a substantial vendor advantage, $M \approx 1$ indicates comparable costs, and $M \ll 1$ indicates a substantial recipient advantage.

This definition is independent of how engineering costs are measured, *e.g.*, the distinct metrics in Table III. For experimental design, one operationalization follows.

B. Operationalization for Controlled Experiment

We operationalize the multiplier using matched modification and evolution tasks T (Table I), comparing vendor and recipient costs under common budgets and acceptance criteria. Replication is evaluated separately.

Task budgets and scores Each task $t \in T$ receives a budget $B_t > 0$ and importance weight $w_t > 0$, fixed before evaluation and shared across environments; unused budgets cannot transfer between tasks. Its score $s_E(t)$ is expenditure through acceptable completion, or the full budget B_t if it does not complete within budget. The latter is a penalty score regardless of actual expenditure, not an estimate of eventual completion cost.

Factory multiplier For $s_F(t) > 0$, the paired ratio $m_t(F, D) = s_D(t)/s_F(t)$ compares task scores between environments. The factory multiplier is their weighted average:

$$M(F, D) = \frac{\sum_{t \in T} w_t m_t(F, D)}{\sum_{t \in T} w_t}. \quad (2)$$

Values above one indicate higher weighted task scores in the recipient environment. The multiplier averages task-level ratios rather than comparing total expenditures. A zero originating score leaves the multiplier undefined.

Artifact contribution To measure the contribution of artifact family k , we compare the original factory against its ablated counterpart F_{-k} :

$$A_k = M(F, F_{-k}), \quad (3)$$

Values above one indicate higher weighted task scores after removing the artifact family.

V. EXPERIMENTS

We propose four complementary experiments to evaluate replication, modification, evolution, and the contributions of factory artifacts. Recall Table I distinguishes these task classes.

A. Experimental Setup

Task selection and budgets Tasks, budgets, and importance weights would be specified before trials using originating factory valuations, external criteria, or independent evaluation. Budgets and weights could vary independently across tasks but would match between environments, with the same total budget $\sum_{t \in T} B_t$ and no transfers between tasks.

Controlled comparison Environments F and D would share the software snapshot, tasks, weights, and acceptance obligations, differing only in engineering resources. Modification tasks would specify changes, whereas evolution tasks would introduce specification ambiguity through unresolved requirements or constraints. Both environments would receive identical initial information and a common clarification procedure, with discovery and clarification included in evaluated work. Independent acceptance would preserve obligations when internal validators are removed.

Cost accounting Expenditure includes human labor (*e.g.*, engineering hours), agent execution (*e.g.*, token usage), infrastructure, and attributable setup, reconstruction, and upkeep. Costs are allocated over a stated horizon and charged against task budgets. Elapsed time and resource consumption are reported separately.

B. Task-Based Experiments

We propose three experiments corresponding to the task classes in Table I.

Replication experiments test whether coding agents reduce the cost of reproducing specified observable behavior without source, using common behavioral targets and agent baselines. Replication is evaluated separately from the factory multiplier.

Modification experiments compare factories on identical, specified changes under independent acceptance criteria. The factory multiplier (Eq. 2) measures relative engineering costs.

Evolution experiments test whether factory advantage increases with specification ambiguity. Tasks vary in initially

TABLE IV
COMMERCIAL SIGNIFICANCE OF FACTORY ADVANTAGE UNDER
INEXPENSIVE FUNCTIONAL REPLICATION, MAPPED TO THE TASK CLASSES
IN TABLE I.

Task	Small advantage	Large advantage
Replication	<i>Replication pressure (outside M)</i>	
Modification	Cost competition	Maintenance advantage
Evolution	Cost competition	<i>Competitive differentiation</i>

unresolved requirements and constraints, while paired factories share initial information and stakeholder clarification procedures. We measure the costs of requirements discovery, uncertainty resolution, and acceptable changes, reporting task scores and completion outcomes separately.

Datasets Some existing benchmarks offer relevant tasks: Behavior2Code provides black-box reconstruction tasks for replication experiments [4], while SWE-INTERACT provides interactive tasks with progressively revealed requirements for evolution experiments [37]. We would adapt these benchmarks to compare factory resources and degrees of specification under common tasks and acceptance criteria.

C. Artifact Ablation Experiments

We propose two complementary interventions to measure the contributions and transferability of factory resources:

- **Factory ablation.** Remove artifact family k while holding tasks, budgets, weights, and acceptance criteria fixed.
- **Recipient addition.** Compare environments before and after receiving an artifact family; test selected combinations to identify interactions among resources.

We would initially hold agent capability fixed, then vary agent configurations and engineering budgets to test whether they compensate for missing factory knowledge. Process-centric trajectory analysis, such as Graphectory [38], could help explain how artifact ablations change agent behavior.

VI. INTERPRETING THE FACTORY MULTIPLIER

The multiplier distinguishes vendor advantage, comparable scores, and recipient advantage. Because it averages weighted task-score ratios rather than total expenditure, interpretation requires completion outcomes. Mutual noncompletion yields a task ratio of one without establishing equivalent capability.

Retained factory advantage ($M > 1$) Higher recipient scores indicate an originating advantage under the evaluation rules, not proof that withheld artifacts caused it or that disclosure is commercially preferable. Where replication is inexpensive, such results motivate investigating whether broader source distribution preserves development advantage. Artifact comparisons would test which resources contribute, while completion outcomes would distinguish expenditure differences from noncompletion.

Comparable scores ($M \approx 1$) Comparability requires a stated tolerance and uncertainty, with successful completion distinguished from equal budget scores. Parity may reflect sufficient distributed resources, factory reverse engineering [39],

or acquired development machinery [9], [40]. Reconstruction, integration, upkeep, and operating costs must be included before interpreting these routes as economical substitutes.

Recipient advantage ($M < 1$) Lower recipient scores may reflect stronger agents, better tools, or removal of obsolete guidance, but these explanations require controlled comparisons. Relaxed obligations, omitted expenditure, and differing completion outcomes must be excluded as explanations before claiming superior development capability.

Across regimes, results may change as recipients learn, factories evolve, and reconstruction costs are amortized. Where replication is inexpensive, Table IV relates these outcomes to demand for modification and evolution. Attributable security analysis and remediation enter expenditure, but licensing rights, other security consequences, and overall commercial value remain separate. No multiplier value alone justifies releasing or withholding source.

VII. FUTURE PLANS

The experiments in §V provide the foundation for a research program addressing three questions. After that:

How does factory advantage vary by task? We will compare modification and evolution across domains with different knowledge and validation requirements, varying the degree of specification as proposed in §V. Separate replication trials will assess the premise that observable functionality can be reproduced economically, distinguishing replication pressure from an advantage in maintaining and extending software.

Which factory artifacts contribute to advantage? Factory ablation will examine documentation, tests, agent instructions, engineering models, and validation controls, including selected interactions. Repository context retrieval and analysis assisted agents motivate these comparisons [26], [29]. The pilot will test whether independent acceptance remains consistent when internal validation resources are removed.

Can recipients acquire equivalent capabilities? Recipient addition will test whether resources valuable within the originating factory also reduce recipient costs. Recipients will then reconstruct missing resources from source, history, and observed behavior [39], or use stronger agents. We will vary models and harnesses separately [28] to distinguish substitution for missing resources from better use of available ones. Repeated matched evaluations will track whether advantages persist as recipients learn and factory snapshots age, accounting for reconstruction and upkeep over a stated horizon.

Across these studies, reporting will pair task scores with expenditure, completion coverage, and uncertainty, distinguishing budget exhaustion from failed acceptance. Recipient parity, recipient advantage, and harmful artifact effects will constrain claims about retained resources. The intended result is evidence about when source disclosure preserves development advantage; commercial value remains a separate question.

VIII. CONCLUSION

In *open source, closed factory*, vendors may disclose source code while retaining an advantage in software evolution. To quantify this advantage, we introduce the factory multiplier and an empirical research program. If substantiated, our hypothesis would indicate that competitive advantage in software engineering is shifting from possession of the product to possession of the means of production.

REFERENCES

- [1] M. E. Porter, "What is strategy?" *Harvard Business Review*, vol. 74, no. 6, pp. 61–78, 1996.
- [2] J. Barney, "Firm resources and sustained competitive advantage," *Journal of management*, vol. 17, no. 1, pp. 99–120, 1991.
- [3] Brandom Thomas, "ArtCraft: Controllable AI for Artists," accessed 2026-10-22. [Online]. Available: <https://getartcraft.com/>
- [4] Turing Frontier Research Lab, "Behavior2Code: Evaluating Coding Agents on Black-Box Software Reconstruction," Aug. 2026, accessed 2026-10-23. [Online]. Available: <https://labs.turing.com/benchmarks/behavior2code>
- [5] N. Vasilakis, A. Benetopoulos, S. Handa, A. Schoen, J. Shen, and M. C. Rinard, "Supply-Chain Vulnerability Elimination via Active Learning and Regeneration," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021, pp. 1755–1770.
- [6] E. Lamprou, J. Dai, G. Ntousakis, M. C. Rinard, and N. Vasilakis, "Lexo: Eliminating Stealthy Supply-Chain Attacks via LLM-Assisted Program Regeneration," *arXiv preprint arXiv:2510.14522*, 2025.
- [7] T. Singla and J. C. Davis, "Software Supply Chains Are Dead: Use-Case-Oriented Regeneration," in *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2026, pp. 64:1–64:14.
- [8] J. Greenfield and K. Short, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley, 2004.
- [9] B. Staples, "When Code Is Abundant," GitLab Blog, Aug. 2026, accessed 2026-10-14. [Online]. Available: <https://about.gitlab.com/blog/when-code-is-abundant/>
- [10] J. A. Schumpeter, *Capitalism, Socialism and Democracy*. Harper & Brothers, 1942.
- [11] A. Ahmad, "The Moat Hasn't Disappeared. It Has Been Redefined." 500 Global, May 2026, accessed 2026-10-14. [Online]. Available: <https://programs.500.co/content/the-moat-hasn-t-disappeared-it-has-been-redefined>
- [12] McMorro, Ryan, "Bain Tests Software Takeover Targets by Vibecoding AI Replicas," *Financial Times*, Jun. 2026, 22 June 2026. [Online]. Available: <https://www.ft.com/content/e5bac4d1-b1f8-43a4-bd54-b182d5357af0>
- [13] Morluto, "morluto/rea: Reverse engineer anything with agents, from app behavior down to native binaries." [Online]. Available: <https://github.com/morluto/rea>
- [14] Kangaxx-0. [MS-OPENSPECLP]: Open specifications. [Online]. Available: <https://learn.microsoft.com/en-us/openspecs/main/ms-openspeclp/3589baea-5b22-48f2-9d43-f5bea4960ddb>
- [15] laramillermst. Online access to source code. [Online]. Available: <https://learn.microsoft.com/en-us/security/engineering/online-sources>
- [16] Cisco transparency service center FAQ.
- [17] Kaspersky transparency center | kaspersky | kaspersky. [Online]. Available: <https://www.kaspersky.com/transparency-center>
- [18] Transparency. [Online]. Available: <https://www.huawei.com/en/trust-center/transparency>
- [19] C. Okafor, T. R. Schorlemmer, S. Torres-Arias, and J. C. Davis, "SoK: Analysis of software supply chain security by establishing secure design properties," in *ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED)*, 2022, pp. 15–24.
- [20] T. N. Kudo, R. d. F. Bulcão-Neto, V. V. Graciano Neto, and A. M. R. Vincenzi, "Aligning Requirements and Testing through Metamodeling and Patterns: Design and Evaluation," *Requirements Engineering*, vol. 28, no. 1, pp. 97–115, 2023.
- [21] O. Kosenkov, P. Elahidoost, T. Gorschek, J. Fischbach, D. Méndez, M. Unterkalmsteiner, D. Fucci, and R. Mohanani, "Systematic Mapping Study on Requirements Engineering for Regulatory Compliance of Software Systems," *Information and Software Technology*, vol. 178, p. 107622, 2025.
- [22] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.
- [23] K. Pohl, G. Böckle, and F. van der Linden, *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, 2005.
- [24] J. C. Davis, K. Kalu, H. Peng, and P. V. Patil, "Model-Based Agentic Software Engineering," *arXiv preprint arXiv:2608.25174*, 2026.
- [25] A. Miura-Ko, "The ai-pilled software factory," Floodgate Insights, Sep. 2026, industry article (grey literature). [Online]. Available: <https://floodgate.com/insights/the-ai-pilled-software-factory>
- [26] W. Liu, A. Yu, D. Zan, B. Shen, W. Zhang, H. Zhao, Z. Jin, and Q. Wang, "GraphCoder: Enhancing repository-level code completion via coarse-to-fine retrieval based on code context graph," in *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2024, pp. 570–581.
- [27] T. Reimann, "How Cloudflare enforces engineering standards using AI," Cloudflare Blog, Aug. 2026, accessed 2026-10-13. [Online]. Available: <https://blog.cloudflare.com/engineering-standards-enforcement/>
- [28] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press, "SWE-agent: Agent-Computer interfaces enable automated software engineering," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.
- [29] Y. Zhang, H. Ruan, Z. Fan, and A. Roychoudhury, "AutoCodeRover: Autonomous program improvement," in *ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2024, pp. 1592–1604.
- [30] G. Orosz, "How Uber uses AI for development: Inside look," The Pragmatic Engineer, Mar. 2026, accessed 2026-10-13. [Online]. Available: <https://newsletter.pragmaticengineer.com/p/how-uber-uses-ai-for-development>
- [31] GitLab, "AI-assisted development playbook," GitLab Handbook, 2026, accessed 2026-10-13. [Online]. Available: <https://handbook.gitlab.com/handbook/engineering/workflow/ai-assisted-development/>
- [32] Redesigning source control at anthropic. [Online]. Available: <https://www.anthropic.com/research/redesigning-source-control>
- [33] N. Gustavsson, "Coding is no longer the constraint: Scaling developer experience to teams and agents at Spotify," Spotify Engineering, Jun. 2026, accessed 2026-10-13. [Online]. Available: <https://engineering.atspotify.com/2026/6/code-with-claude-coding-is-no-longer-the-constraint>
- [34] U. K. Medisetty, "Running a Software Factory Efficiently at Uber Scale," Uber Engineering Blog, Aug. 2026, accessed 2026-10-14. [Online]. Available: <https://www.uber.com/us/en/blog/efficient-software-factory/>
- [35] T. Lütke, "Learning on the shop floor," May 2026, accessed 2026-10-13. [Online]. Available: <https://x.com/tobi/article/2053121182044451016>
- [36] S. Sivasubramanian, "How Frontier Teams Are Reinventing AI-Native Development," AWS Machine Learning Blog, Jun. 2026, accessed 2026-10-14. [Online]. Available: <https://aws.amazon.com/blogs/machine-learning/how-frontier-teams-are-reinventing-ai-native-development/>
- [37] M. Raghavendra, A. Gunjal, A. Sabharwal, and Y. He, "SWE-INTERACT: Reimagining SWE Benchmarks as User-Driven Long-Horizon Coding Sessions," *arXiv preprint arXiv:2606.30573*, 2026. [Online]. Available: <https://arxiv.org/abs/2606.30573>
- [38] S. Liu, Y. Chen, R. Krishna, S. Sinha, J. Ganhotra, and R. Jabbarvand, "Process-Centric Analysis of Agentic Software Systems," *Proceedings of the ACM on Programming Languages*, vol. 10, no. OOPSLA1, 2026.
- [39] S. Ducasse and D. Pollet, "Software Architecture Reconstruction: A Process-Oriented Taxonomy," *IEEE Transactions on Software Engineering*, vol. 35, no. 4, pp. 573–591, 2009.
- [40] GitLab, "Customize GitLab Duo Agent Platform," GitLab documentation, 2026, accessed 2026-10-14. [Online]. Available: https://docs.gitlab.com/user/duo_agent_platform/customize/