From Assistance to Agency: Rethinking Autonomy and Control in CI/CD Pipelines

Anonymous Author(s)

Abstract

AI agents are assuming active roles in Continuous Integration and Continuous Deployment (CI/CD) workflows, yet the research community lacks a shared vocabulary for describing what it means for CI/CD to be *agentic*, how much decision authority is delegated, and where control should reside. This paper argues that the central challenge in agentic CI/CD is not improving task performance but designing *authority transfer*: determining which decisions agents may make, under what constraints, and with what recourse.

To structure this argument, we introduce a distinction between *data-plane authority* (localized interventions such as patch generation and test reruns) and *control-plane authority* (modifications to pipeline configuration, deployment policies, and approval gates). Drawing on research prototypes and industrial platforms, we show that current systems operate mainly at the data plane under *bounded autonomy*, with safety achieved through surrounding governance infrastructure rather than intrinsic agent guarantees. We identify three patterns: constrained autonomy as the dominant design, external governance as the primary safety mechanism, and a widening gap between deployment momentum and evaluation methodology. We propose a research agenda arguing that control-plane safety and governance mechanisms represent the most urgent open problem, followed by formalization of autonomy boundaries, evaluation frameworks, and modeling human-agent coordination.

CCS Concepts

• **Software and its engineering** → **Software development process management**; • **Computing methodologies** → *Intelligent agents*.

Keywords

CI/CD pipelines, DevOps automation, Agentic systems, Autonomous software systems, AI-assisted software engineering

ACM Reference Format:

Anonymous Author(s). . From Assistance to Agency: Rethinking Autonomy and Control in CI/CD Pipelines. In . ACM, New York, NY, USA, 5 pages.

1 Introduction

Continuous Integration and Continuous Deployment (CI/CD) is central to modern software delivery [11, 29]. AI techniques have long supported tasks such as failure diagnosis and operational monitoring [24, 27], typically as predictive or advisory components embedded within specific pipeline stages. In these systems, decision authority remains with human operators or fixed pipeline logic.

Recent advances in large language model-based systems and software engineering agents [17, 23, 31] have introduced components capable of planning, tool use, and action execution. Industrial platforms now describe agents that observe pipeline state, generate patches or configuration changes, and submit modifications through repository workflows [2, 14–16, 28]. Despite this rapid adoption, the research community lacks a shared vocabulary for characterizing how autonomous these systems are and where decision authority resides.

This paper argues that the central challenge in agentic CI/CD is not improving remediation performance but designing *authority transfer*: determining which decisions agents may make, under what constraints, and with what recourse. To structure this problem, we distinguish between *data-plane authority*—localized interventions within a pipeline execution (e.g., patch generation, test reruns)—and *control-plane authority*—modifications to pipeline configuration, deployment policies, and approval gates. Delegating control-plane authority alters organizational risk boundaries, yet current systems overwhelmingly confine agents to the data plane.

Drawing on research prototypes and industrial platforms, we characterize the current state as *bounded autonomy* and identify three patterns. First, constrained autonomy dominates: agents operate within predefined action spaces, with human approval mediating integration. Second, safety is achieved primarily through surrounding governance infrastructure (e.g., pull requests and policy gates) rather than intrinsic agent guarantees. Third, deployment momentum is outpacing evaluation methodology, and the field lacks benchmarks suited to decision-making agents.

We conclude by outlining a research agenda that prioritizes control-plane safety and governance mechanisms, followed by formalization of autonomy boundaries, evaluation frameworks, and human–agent coordination. Treating authority transfer as a first-class design problem enables more precise reasoning about agentic behavior in CI/CD pipelines.

2 Agentic CI/CD in Practice: Examples from Research and Industry

To ground the discussion, we draw on peer-reviewed research and publicly available industrial documentation. Because many recent developments originate from commercial platforms rather than archival publications, both sources help clarify how agentic capabilities are framed and constrained in practice. Our goal is not exhaustive coverage, but a capability-focused set of exemplars that motivates the observations and research agenda that follow.

2.1 Research Prototypes

Early examples of agentic behavior in CI/CD appear in automated repair systems. Repairnator continuously monitors CI failures, synthesizes candidate patches, and submits pull requests for human review [26, 30]. R-HERO extends this pattern by incorporating continual learning into patch generation [4]. These systems observe pipeline state, make decisions about remediation strategies, and generate concrete changes. However, integration authority remains mediated by pull request workflows, limiting their autonomy.

More recent LLM-based repair systems exhibit increasing agent-like behavior. RepairAgent [5] autonomously plans and executes

multi-step repair strategies by invoking tools, gathering diagnostic information, and iterating on candidate patches. ChatRepair [33] uses conversational LLM interaction to fix bugs through iterative dialogue with test feedback. These systems demonstrate richer planning and tool-use capabilities than earlier template-based approaches, yet human reviewers retain final say over merging: no generated patch is integrated without explicit approval. BuildSheriff [35] addresses a complementary problem, triaging CI test failures to their root causes, illustrating how diagnostic intelligence can precede and support remediation agents.

Recent work also combines LLM-driven diagnosis with remediation workflows that resemble agent loops. LogSage analyzes CI/CD logs to detect failures and support remediation actions in an industrial setting [34]. Adjacent DevSecOps research such as AutoGuard frames self-healing as a proactive control layer that can adapt pipeline responses using reinforcement learning [3]. Bouzenia and Pradel [6] present an LLM agent capable of autonomously setting up and executing test suites for arbitrary projects, demonstrating that agentic capabilities are extending beyond patch generation toward broader build and test automation.

A notable gap separates agent behavior at the *development layer* from the *delivery layer*. Li et al. present AIDev, a large-scale study of AI agent activity on GitHub repositories, enabling analysis of agent-authored pull requests and interaction patterns with human contributors [22]. While this work offers empirical foundations for understanding how agents contribute code, it does not address integration, deployment, or control-plane authority within CI/CD pipelines. This gap is significant: it means the community has growing empirical evidence about agent behavior in code contribution but almost none about agent behavior in release governance.

2.2 Industrial Systems

Industrial platforms increasingly describe CI/CD functionality using agent-oriented language, and critically, the pattern is consistent across multiple vendors. Major development platforms have each introduced agentic CI/CD capabilities: GitHub integrates coding agents directly into repository and workflow processes, enabling agents to translate human-readable intent into workflow artifacts, perform root-cause analysis, and propose fixes while merge authority remains mediated by review gates [13, 14]; GitLab's Duo agent platform provides a dedicated pipeline-fix flow within merge request workflows [15]; Amazon Q Developer generates code changes triggered by CI signals, routing results through pull request review [2]; and Google's Gemini CLI GitHub Action enables LLM-driven agents to run within GitHub Actions workflows [16].

Specialized vendors exhibit the same pattern. Nx Self-Healing CI, Datadog's Bits AI Dev Agent, and Gitar's autonomous build-fix engine each monitor pipeline outcomes, generate remediation actions, and route proposed changes through pull request mechanisms [12, 28]. Dagger has published architectural blueprints for constructing self-healing pipelines using AI agents within containerized CI environments [9].

These systems exhibit observable properties associated with agency: perception of pipeline signals, decision-making over candidate actions, and execution through artifact generation. However, applying the data-plane and control-plane distinction, we observe that all publicly documented systems concentrate agent behavior at the data plane. Continuous delivery research emphasizes the importance of release governance and deployment discipline [11, 19, 29]. No system we examined delegates control-plane authority (i.e., modification of deployment policies, approval gates, rollback thresholds, or workflow definitions) without human approval. This is a significant finding: despite the marketing language of autonomy, the industry has converged on a conservative authority model in which decision-making and action execution are present but authority remains structurally mediated by repository permissions, approval gates, and governance policies.

2.3 Architectural Implications

The bounded autonomy observed across current systems has architectural consequences. CI/CD pipelines are traditionally deterministic execution graphs governed by explicit approval gates and disciplined deployment strategies [11, 29]. Introducing agentic components embeds adaptive decision loops within these structures.

These hybrid structures resemble feedback-loop architectures from self-adaptive and autonomic systems research, particularly the MAPE-K (Monitor-Analyze-Plan-Execute over shared Knowledge) reference model [21, 32]. However, the analogy is limited in important ways. MAPE-K assumes a single managed element with a well-defined adaptation policy, and its control loop is typically designed by the system architect with full authority over the feedback mechanism [8, 18]. In CI/CD pipelines, authority is *socially mediated*: multiple human stakeholders (developers, reviewers, release managers) share governance responsibility, and the "managed element" spans organizational processes, not just technical infrastructure. An agent that modifies a deployment threshold is not simply executing a control loop; it is intervening in a socio-technical governance structure. The data-plane/control-plane distinction captures this difference: data-plane actions resemble classical MAPE-K adaptations within a bounded execution context, while control-plane actions alter the governance structure, which MAPE-K does not capture. As capabilities expand, pipeline architectures should make authority boundaries explicit, distinguishing advisory, semi-autonomous, and control-plane roles. Treating these distinctions as design variables can prevent unintentional escalation of operational authority.

An important open question concerns autonomy escalation. As agents move from data-plane interventions to control-plane modifications, small expansions of authority may compound over time. For example, an agent initially permitted to generate patches may later be allowed to modify workflow definitions or deployment thresholds. Each incremental extension appears locally justified, yet collectively these changes may shift effective release governance from human-centered review toward agent-mediated control. Understanding how such escalation occurs, and how it can be bounded without eliminating useful automation, represents a critical design and policy challenge.

A related concern is authority visibility. As agentic capabilities are introduced incrementally, developers and operators may not have clear mental models of which decisions are being made autonomously, which are advisory, and which remain human-controlled. Prior work in self-adaptive systems emphasizes the importance of explicit feedback loops and observability for maintaining trust and

control [8, 21]. In CI/CD contexts, insufficient visibility into agent decision processes may lead to misplaced trust, delayed human intervention, or incorrect assumptions about compliance boundaries. Making autonomy levels and authority scopes explicit within pipeline tooling may therefore be as important as improving underlying model performance.

A further concern is adversarial robustness. Agents that generate workflow configurations or modify pipeline artifacts introduce a new attack surface for supply chain compromise. An agent-proposed change that passes CI checks but introduces a subtle vulnerability may be more difficult to detect than a conventional malicious commit, because reviewers may place unwarranted trust in agent-generated artifacts [25]. Similarly, prompt injection or manipulation of the signals an agent observes (e.g., crafted log messages or error outputs) could steer remediation behavior in unintended directions. These risks are amplified at the control plane: an agent that modifies deployment policies or approval gates under adversarial influence could compromise release governance at an organizational level. Security implications of agentic CI/CD therefore extend beyond the confidentiality and integrity of individual artifacts to the trustworthiness of the delivery process itself.

2.4 Summary

Across the systems we examined, a dominant pattern emerges: semi-autonomous, repair-oriented agents embedded within human-governed workflows. Decision authority is bounded, action spaces are constrained, and final integration or deployment control remains with human stakeholders. We now formalize this and two related patterns as explicit observations.

3 Core Observations

Our analysis of research prototypes and industrial systems reveals three consistent patterns that characterize the current state of agentic CI/CD.

3.1 Observation 1: Constrained Autonomy Dominates

Although many systems are described as agentic, most operate under tightly bounded autonomy. Research prototypes from Repairnator through RepairAgent and ChatRepair generate patches but defer integration decisions to human reviewers [4, 5, 26, 33]. Industrial tools across multiple vendors similarly route generated changes through pull request or merge request workflows [14, 15, 28].

In practice, these systems exercise delegated authority within predefined action spaces rather than independent control over pipeline orchestration or deployment strategy. As noted earlier, no publicly documented system delegates control-plane authority without human approval. The dominant pattern is semi-autonomous assistance embedded within existing governance structures, mirroring earlier transitions from manual release engineering to automated CI/CD workflows, where automation sped execution without immediately displacing human release authority [1, 29]. Agentic CI/CD appears to be following a similar incremental trajectory, which suggests that the current bounded-autonomy equilibrium is not accidental but reflects a recurring pattern in how organizations absorb automation into governance-sensitive processes.

3.2 Observation 2: Governance Carries the Safety Burden

Safety and accountability in current agentic CI/CD systems are achieved primarily through external governance mechanisms (e.g., pull request approval workflows, repository permissions, and policy gates) rather than intrinsic agent guarantees. None of the reviewed systems uses formal verification of agent behavior or built-in safety constraints independent of the surrounding infrastructure.

This reliance on external governance has a critical implication: as agent capabilities expand to include broader planning and artifact modification [17, 23], the safety model depends entirely on the strength and coverage of surrounding controls. Explicit reasoning about authority boundaries and escalation paths is largely absent from current research and industrial documentation, leaving the governance layer as an unexamined single point of trust.

This governance-first safety model echoes patterns observed in DevSecOps integration, where security automation is typically embedded within policy-constrained workflows rather than granted unconstrained authority [3, 24]. The extension of similar constraints to agentic CI/CD suggests that autonomy expansion will likely remain mediated by compliance and risk management requirements.

3.3 Observation 3: Evaluation Lags Behind Deployment

Many industrial platforms now offer agentic CI/CD features [14], yet systematic evaluation methods for these systems remain underdeveloped. Traditional CI/CD metrics emphasized in prior surveys, such as build duration, deployment frequency, and failure rate [11, 24, 29], were designed for deterministic automation and do not capture decision quality, constraint adherence, escalation behavior, or long-term governance impact. Existing AIOps and DevOps evaluation frameworks primarily assess anomaly detection accuracy and operational performance gains [24, 27], which are inadequate for agentic systems that autonomously generate and execute actions on repository artifacts or deployments.

The core issue is that evaluation must move beyond task-level correctness toward system-level assessment. An agent may successfully repair a failing build while increasing reviewer burden, triggering unstable rollback cascades, or subtly reshaping governance practices. Such socio-technical effects are not captured in conventional CI/CD benchmarking approaches, and no evaluation framework currently accounts for them in a standardized manner.

These three observations converge on a single conclusion: agentic CI/CD is not only a model-capability question, but a reallocation of decision authority within socio-technical delivery systems. Constrained autonomy persists because governance mechanisms carry the safety burden, and evaluation has not yet caught up to validate alternative designs. Clarifying how authority is bounded, evaluated, and coordinated is therefore the central research challenge.

4 Research Agenda

We outline four research directions, ordered by urgency. Control-plane safety is the most pressing because it addresses the highest-risk frontier of capability expansion. The remaining directions provide the conceptual, empirical, and organizational foundations needed to support safe delegation.

4.1 Control-Plane Safety and Governance Mechanisms

As noted earlier, current systems avoid control-plane delegation. Expansion into control-plane modification (i.e., altering pipeline configuration, deployment policies, or approval gates) introduces qualitatively different risks, because such changes affect organizational governance rather than individual artifacts. The adversarial concerns are particularly critical at the control plane, where a single compromised modification can affect all subsequent releases.

Research is needed on safety mechanisms tailored to CI/CD governance, including policy-aware agents that reason about their own authority, constrained action synthesis that prevents control-plane modifications outside explicit delegation, and automated rollback strategies with formally specified guarantees. The MAPE-K model offers a basis for feedback-loop design [8, 21], but must be extended to account for socially mediated authority and multi-stakeholder governance in CI/CD pipelines.

4.2 Formalizing Autonomy Boundaries

Current systems implement bounded autonomy in an ad hoc manner, typically relying on pull request workflows and repository permissions to constrain agent behavior. A principled framework is needed to specify and verify autonomy boundaries in CI/CD pipelines. Such a framework could build on access control, policy specification, and runtime monitoring [18], extending these ideas to represent graded delegated decision authority rather than binary permission models. Recent work on LLM agent observability [10] suggests that runtime monitoring of agent decisions is feasible and could enable formal boundary enforcement.

One practical direction is to model decision authority, action scopes, and escalation paths explicitly within pipeline architectures. Such models would allow autonomy to be treated as a configurable design parameter rather than an emergent side effect of tooling. The data-plane/control-plane distinction we propose offers a starting vocabulary for such specifications.

4.3 Evaluation Frameworks for Agentic CI/CD

Existing CI/CD metrics overlook decision quality, constraint adherence, and governance impact, and the field lacks shared benchmarks for comparing autonomy levels and their associated trade-offs.

Future work should develop reproducible evaluation environments that capture realistic pipeline workflows, approval processes, and failure scenarios. Related efforts exist in adjacent domains: AIOpsLab evaluates AI agents in cloud operations [7], ITBench offers diverse real-world IT automation tasks for agent evaluation [20], and AIDev analyzes agent-authored pull requests in development workflows [22]. However, no comparable standardized environment exists for CI/CD pipelines that captures approval gates, policy constraints, control-plane changes, and human overrides.

Evaluation criteria should extend to include stability, transparency, human override frequency, and long-term effects. A practical entry point is a minimal protocol that separates remediation effectiveness from governance and stability costs. Drawing on CI/CD and DevOps perspectives [24, 29] and AIOps failure-management concerns [27], benchmark suites for agentic CI/CD should report: remediation success (fraction resolved, time-to-repair under fixed budgets), governance impact (PR iterations, reviewer interventions, override frequency), stability and regression (recurrence and agent-induced regressions), and policy adherence (violations of scoped action constraints, especially control-plane actions). Such reporting enables comparison of bounded-autonomy designs without assuming full autonomy as the target outcome.

4.4 Human-Agent Coordination Models

Current systems rely heavily on human-in-the-loop approval mechanisms, yet little research examines how developers interpret, trust, or override agent decisions in CI/CD contexts. As agents assume broader responsibilities, coordination complexity will increase.

During incident response, ambiguous authority between an agent and an on-call engineer can delay mitigation or produce conflicting actions. Research is needed on interaction models that clarify responsibility boundaries, support meaningful explanations, and prevent ambiguous authority in such high-pressure situations. Understanding how agent behavior reshapes collaboration patterns in DevOps teams is essential for responsible deployment. Empirical datasets such as AIDev [22] provide a foundation for studying agent-developer interaction patterns, though analogous datasets capturing CI/CD-specific coordination remain absent.

Advancing agentic CI/CD requires progress across all four directions, with control-plane safety as the enabling prerequisite. Without governance mechanisms that can bound control-plane authority, broader autonomy delegation risks undermining the release discipline that CI/CD was designed to enforce [11, 19].

5 Limitations

Our discussion relies partly on publicly available industrial documentation, which may reflect marketing or incomplete disclosure, so we prioritize observable properties over performance claims. The space is evolving quickly and public evidence on large-scale outcomes remains limited, so we emphasize authority and governance structure over long-term operational effectiveness.

6 Conclusion

Agentic CI/CD today is best characterized as *bounded autonomy*: agents operate within human-governed approval structures rather than exercising independent control. Across research prototypes and industrial platforms, we observe three recurring patterns: constrained autonomy as the dominant design, safety achieved through external governance infrastructure, and a growing gap between rapid deployment and systematic evaluation. The distinction between *data-plane authority* and *control-plane authority* clarifies the stakes. Current systems remain comparatively safe because they confine agents to the data plane and avoid delegating control-plane authority. As capabilities expand and competitive pressures intensify, this boundary will be increasingly tested. This paper therefore argues that the central research challenge is the design of *authority transfer*. Control-plane safety and governance mechanisms constitute the most urgent priority, followed by formalizing autonomy boundaries, developing evaluation frameworks, and modeling human–agent coordination. Treating authority transfer as a first-class design problem is essential to ensuring that future agentic CI/CD systems are safe, accountable, and empirically grounded.

References

[1] Bram Adams and Shane McIntosh. 2016. Modern Release Engineering in a Nutshell: Why Researchers Should Care. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. 78–90. doi:10.1109/SANER.2016.18

[2] Amazon Web Services. 2026. Third-party integration with Amazon Q Developer (GitHub). AWS Documentation. https://docs.aws.amazon.com/amazonq/latest/qdeveloper-ug/third-party-integration.html Accessed 2026-02-07.

[3] Praveen Anugula, Avdhesh Kumar Bhardwaj, Navin Chhibber, Rohit Tewari, Sunil Khemka, and Piyush Ranjan. 2025. AutoGuard: A Self-Healing Proactive Security Layer for DevSecOps Pipelines Using Reinforcement Learning. https://arxiv.org/abs/2512.04368

[4] Benoit Baudry, Zimin Chen, Khashayar Etemadi, Han Fu, Davide Ginelli, Steve Kommrusch, Matias Martinez, Martin Monperrus, Javier Ron, He Ye, and Zhongxing Yu. 2021. A Software-Repair Robot Based on Continual Learning. *IEEE Software* 38, 4 (2021), 28–35. doi:10.1109/MS.2021.3070743

[5] Islem Bouzenia, Premkumar Devanbu, and Michael Pradel. 2025. RepairAgent: An Autonomous, LLM-Based Agent for Program Repair. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, Los Alamitos, CA, USA, 2188–2200. doi:10.1109/ICSE55347.2025.00157

[6] Islem Bouzenia and Michael Pradel. 2025. You Name It, I Run It: An LLM Agent to Execute Tests of Arbitrary Projects. *Proc. ACM Softw. Eng.* 2, ISSTA, Article ISSTA047 (June 2025), 23 pages. doi:10.1145/3728922

[7] Yinfang Chen, Manish Shetty, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Jonathan Mace, Chetan Bansal, Rujia Wang, and Saravan Rajmohan. 2025. AIOpsLab: A Holistic Framework to Evaluate AI Agents for Enabling Autonomous Clouds. In *Proceedings of Machine Learning and Systems*, Vol. 7.

[8] Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Di Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. 2009. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In *Software Engineering for Self-Adaptive Systems*, Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee (Eds.). Lecture Notes in Computer Science, Vol. 5525. Springer, 1–26. doi:10.1007/978-3-642-02161-9_1

[9] Dagger. 2025. *Automate Your CI Fixes: Self-Healing Pipelines with AI Agents*. https://dagger.io/blog/automate-your-ci-fixes-self-healing-pipelines-with-ai-agents Blueprint-style description for constructing AI-agent-driven self-healing CI pipelines.

[10] Liming Dong, Qinghua Lu, and Liming Zhu. 2024. AgentOps: Enabling Observability of LLM Agents. *arXiv preprint arXiv:2411.05285* (2024).

[11] Brian Fitzgerald and Klaas-Jan Stol. 2017. Continuous Software Engineering: A Roadmap and Agenda. *Journal of Systems and Software* 123 (2017), 176–189. doi:10.1016/j.jss.2015.06.063

[12] Gitar. 2026. Automated build failure fix solutions (autonomous CI/CD healing engine). Vendor documentation / blog. https://cms.gitar.ai/automated-build-failure-fix-solutions/ Accessed 2026-02-07.

[13] GitHub. 2026. *About GitHub Copilot coding agent*. https://docs.github.com/en/copilot/concepts/agents/coding-agent/about-coding-agent Concept documentation describing Copilot coding agent autonomy and PR-based delegation.

[14] GitHub. 2026. Continuous AI in Practice: What Developers Can Automate Today with Agentic CI. https://github.blog/ai-and-ml/generative-ai/continuous-ai-in-practice-what-developers-can-automate-today-with-agentic-ci/. GitHub Blog. Accessed 2026-02-11.

[15] GitLab. 2026. Fix CI/CD pipeline flow. GitLab Docs (Duo agent platform). https://docs.gitlab.com/user/duo_agent_platform/flows/foundational_flows/fix_pipeline/ Accessed 2026-02-07.

[16] Google GitHub Actions. 2026. run-gemini-cli: A GitHub Action invoking the Gemini CLI. GitHub repository. https://github.com/google-github-actions/run-gemini-cli Accessed 2026-02-07.

[17] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Transactions on Software Engineering and Methodology* 33, 8 (2024), 220:1–220:79. doi:10.1145/3695988

[18] Markus C. Huebscher and Julie A. McCann. 2008. A Survey of Autonomic Computing—Degrees, Models, and Applications. *Comput. Surveys* 40, 3 (2008), 7:1–7:28. doi:10.1145/1380584.1380585

[19] Jez Humble and David Farley. 2010. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional.

[20] Saurabh Jha, Rohan Arora, Yuji Watanabe, Takumi Yanagawa, Yinfang Chen, Jackson Clark, Bhavya Bhavya, Mudit Verma, Harshit Kumar, Hirokuni Kitahara, Noah Zheutlin, Saki Takano, Divya Pathak, Felix George, Xinbo Wu, Bekir O. Turkkan, Gerard Vanloo, Michael Nidd, Ting Dai, Oishik Chatterjee, Pranjal Gupta, Suranjana Samanta, Pooja Aggarwal, Rong Lee, Pavankumar Murali, Jae wook Ahn, Debanjana Kar, Ameet Rahane, Carlos Fonseca, Amit Paradkar, Yu Deng, Pratibha Moogi, Prateeti Mohapatra, Naoki Abe, Chandrasekhar Narayanaswami, Tianyin Xu, Lav R. Varshney, Ruchi Mahindru, Anca Sailer, Laura Shwartz, Daby Sow, Nicholas C. M. Fuller, and Ruchir Puri. 2025. ITBench: Evaluating AI Agents across Diverse Real-World IT Automation Tasks. In *Forty-second International Conference on Machine Learning*. https://arxiv.org/abs/2502.05352

[21] Jeffrey O. Kephart and David M. Chess. 2003. The Vision of Autonomic Computing. *Computer* 36, 1 (2003), 41–50. doi:10.1109/MC.2003.1160055

[22] Hao Li, Haoxiang Zhang, and Ahmed E. Hassan. 2026. AIDev: Studying AI Coding Agents on GitHub. arXiv:2602.09185 https://arxiv.org/abs/2602.09185

[23] Junwei Liu, Kaixin Wang, Yixuan Chen, Xin Peng, Zhenpeng Chen, Lingming Zhang, and Yiling Lou. 2024. Large Language Model-Based Agents for Software Engineering: A Survey. arXiv:2409.02977 https://arxiv.org/abs/2409.02977

[24] Alok Mishra and Ziadoon Otaiwi. 2020. DevOps and Software Quality: A Systematic Mapping. *Computer Science Review* 38 (2020), 100308. doi:10.1016/j.cosrev.2020.100308

[25] Akshay Mittal and Vivek Venkatesan. 2025. Leveraging Generative AI for Proactive Security and Automated Remediation in Cloud-Native CI/CD Pipelines. In *International Conference on Software Engineering and Data Engineering*. Springer, 18–39.

[26] Martin Monperrus, Simon Urli, Thomas Durieux, Matias Martinez, Benoit Baudry, and Lionel Seinturier. 2019. Repairnator Patches Programs Automatically. *Ubiquity* 2019, July (2019), 1–12. doi:10.1145/3349589

[27] Paolo Notaro, Jorge Cardoso, and Michael Gerndt. 2021. A Survey of AIOps Methods for Failure Management. *ACM Transactions on Intelligent Systems and Technology* 12, 6, Article 81 (Nov. 2021), 45 pages. doi:10.1145/3483424

[28] Nx. 2025. *AI-Powered Self-Healing CI*. https://nx.dev/docs/features/ci-features/self-healing-ci Nx Documentation. Accessed 2026-02-12.

[29] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. 2017. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access* 5 (2017), 3909–3943. doi:10.1109/ACCESS.2017.2685629

[30] Simon Urli, Zhongxing Yu, Lionel Seinturier, and Martin Monperrus. 2018. How to design a program repair bot? insights from the repairnator project. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*. ACM, 95–104. doi:10.1145/3183519.3183540

[31] Yanlin Wang, Wanjun Zhong, Yanxian Huang, Ensheng Shi, Min Yang, Jiachi Chen, Hui Li, Yuchi Ma, Qianxiang Wang, and Zibin Zheng. 2025. Agents in software engineering: Survey, landscape, and vision. *Automated Software Engineering* 32, 2 (2025), 70.

[32] Steve R. White, James E. Hanson, Ian Whalley, David M. Chess, and Jeffrey O. Kephart. 2004. An Architectural Approach to Autonomic Computing. In *Proceedings of the 1st International Conference on Autonomic Computing (ICAC 2004)*. 2–9. doi:10.1109/ICAC.2004.8

[33] Chunqiu Steven Xia and Lingming Zhang. 2024. Automated Program Repair via Conversation: Fixing 162 out of 337 Bugs for $0.42 Each using ChatGPT. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Vienna, Austria) *(ISSTA 2024)*. Association for Computing Machinery, New York, NY, USA, 819–831. doi:10.1145/3650212.3680323

[34] Weiyuan Xu, Juntao Luo, Tao Huang, Kaixin Sui, Jie Geng, Qijun Ma, Isami Akasaka, Xiaoxue Shi, Jing Tang, and Peng Cai. 2025. LogSage: An LLM-Based Framework for CI/CD Failure Detection and Remediation with Industrial Validation. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 3742–3753. doi:10.1109/ASE63991.2025.00310

[35] Chen Zhang, Bihuan Chen, Xin Peng, and Wenyun Zhao. 2022. BuildSheriff: change-aware test failure triage for continuous integration builds. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 312–324. doi:10.1145/3510003.3510132