Post-Velocity Software Engineering: When Change Throughput Outruns Assurance Capacity

Alessio Bucaioni
Mälardalen University
Sweden
alessio.bucaioni@mdu.se

Abstract

Large language models and automation can increase software delivery throughput by lowering the effort required to propose, implement, and integrate changes. However, the capacity to assure those changes, via evidence generation, substantive review, shared understanding, and accountable approval, scales more slowly and can saturate under limited expert attention, validation budgets, and governance obligations. We hypothesize a post-velocity regime in which the rate of change approaches or exceeds assurance capacity, so justified confidence per change declines and risk shifts downstream into operations, yielding diminishing or negative returns from further acceleration. To make this hypothesis empirically testable, we define a construct-level model relating rate of change, assurance capacity, justified confidence, and context-dependent safe velocity, and we propose trace-derived diagnostics for (i) validation lag, (ii) procedural oversight, and (iii) downstream defect shifting. We state falsifiable propositions and outline study designs (e.g., interrupted time series, difference-in-differences) that address confounds such as product maturity and reporting policy changes. We conclude by outlining an intervention space for accountability, preserving acceleration, including risk-stratified gating and differential assurance budgeting in AI-assisted workflows.

CCS Concepts

• **Software and its engineering → Extra-functional properties**; **Software functional properties**;

Keywords

Software Engineering, Post-Velocity, Assurance

ACM Reference Format:
Alessio Bucaioni. 2026. Post-Velocity Software Engineering: When Change Throughput Outruns Assurance Capacity. In *Proceedings of (ICSE Journal Ahead Workshop (JAWs) 26).* ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

ICSE Journal Ahead Workshop (JAWs) 26, Rio de Janeiro, Brazil

ACM ISBN 978-1-4503-XXXX-X/2026/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

In transport optimization, Rory Sutherland noted that once high-speed rail reached a certain speed, further time savings added little value and required disproportionate investment[1] . Operators therefore shifted optimization from speed to journey quality (comfort, reliability, experience). The broader principle is that, beyond a threshold, optimizing a single dimension yields diminishing returns, so progress requires redefining what the system should optimize.

We argue that contemporary software engineering is approaching an analogous threshold. Over the past decade, continuous integration and deployment, pervasive automation, and, more recently, Large Language Models (LLMs) have reduced the marginal cost of producing software changes and increased the feasible rate of change [5]. Many organizations operationalize success through velocity-oriented measures (shorter cycle times, higher throughput, and more frequent releases) implicitly treating additional speed as inherently beneficial. This framing becomes fragile when the rate of change approaches the limits of socio-technical assurance [3]. Tooling can scale change production, but the capacity to validate and govern change (review depth, evidence generation, shared understanding, and accountable approval) scales more slowly[2]. When throughput outpaces this capacity, oversight can become procedural rather than evidentiary, justified confidence per change declines, and defects and risks are more likely to shift downstream into operations and incident response [9, 13].

We therefore hypothesize that AI-accelerated development can induce a *post-velocity* regime: a threshold beyond which validation and accountability, rather than code production, become the binding constraints, and additional acceleration yields diminishing or negative returns. Post-velocity is not a generic speed–quality trade-off. It hypothesizes a threshold: once change production outpaces an organization's capacity to validate, explain, and approve changes responsibly, assurance becomes procedural, evidence per change thins, and operational risk may rise even as throughput increases. This shift is amplified by governance and accountability expectations. Many teams must justify why a change was made, what risks were assessed, what evidence supports it, and who approved it, explicitly in regulated and safety-critical contexts and implicitly in security-sensitive or high-availability services. LLM-assisted workflows can increase the supply of plausible changes faster than teams can evaluate and justify them, increasing the risk that assurance devolves into box-checking rather than sustaining shared understanding and warranted confidence [3].

[1] https://www.youtube.com/shorts/7-4C1AwkIXg
[2] Practitioner commentary: https://techtrenches.substack.com/p/the-great-software-quality-collapse

Although prior work spans productivity, DevOps performance, human factors, and governance, it rarely provides a construct-level, falsifiable account of (i) when acceleration stops helping due to assurance saturation, (ii) how this saturation appears in observable traces, and (iii) which interventions preserve accountability while still benefiting from AI-enabled throughput. Without such a model, organizations may optimize for shipping more changes while under-investing in the work that makes those changes safe, comprehensible, and governable.

This paper makes three contributions: (i) a construct-level framing that defines rate of change (R), assurance capacity (A), justified confidence per change (J), and safe velocity (V_s), and clarifies their interactions under saturation; (ii) trace-based diagnostics and falsifiable propositions that capture validation lag, procedural oversight, and downstream defect shifting while ruling out alternative explanations (e.g., product maturity and reporting shifts); and (iii) a research agenda with study designs, measurement protocols, and an intervention space that ties acceleration to assurance evidence and accountability constraints.

2 Background and related work

We position our vision relative to work on development velocity and software quality, the cognitive/evaluative burden introduced by AI-assisted development, and oversight and accountability constraints. We select representative studies for conceptual coverage, with the goal of surfacing velocity-centric assumptions and motivating the gap addressed by a post-velocity perspective.

Prior work suggests that prioritizing speed can degrade internal quality and increase defects when assurance does not scale with throughput [1, 9, 12]. Case-study and RAD-oriented evidence reports worsening post-adoption indicators and incentive-driven deferral of maintainability under time pressure [1, 12], while economic analyses argue that organizations monetize speed while under-measuring downstream costs, allowing deferred validation to compound operational risk [9]. This motivates our concern, but the literature typically frames assurance scaling as adding automation and process, rather than analysing saturation effects and accountable approval as binding constraints.

Studies suggest that fast-generating AI can raise cognitive burden by shifting human work from authoring to time-pressured evaluation under uncertainty [7, 8, 11]. Empirical evidence links GenAI use to strain and fatigue via mechanisms such as automation complacency and increased evaluative workload [8], and high-stakes review is often framed as intensive proofreading of unauthored, opaque outputs that invites automation bias [11]. At a governance level, scale and partial autonomy can outpace human capability, making continuous supervision infeasible and increasing missed errors [7]. These results motivate bounded validation capacity, but they do not yet provide a software-engineering measurement model for saturation and threshold effects.

Meaningful human oversight is often treated as non-removable in safety- and mission-critical systems, where accountability and foreseeability constrain decisions [2, 4, 6, 10]. Work against fully autonomous agents highlights risks when systems act faster than humans can intervene and when errors cascade across interconnected actions [10], while self-adaptive systems research shows that autonomy can erode situational awareness until intervention becomes ineffective [4]. Assurance frameworks therefore emphasize human agency and iterative evidence accumulation, relaxing constraints only as evidence grows [6]. This supports our claim that, under accountability constraints, safe velocity is bounded by justified approval rather than generation speed.

Positioning and gap summary

Prior work links delivery acceleration to quality erosion when assurance lags, shows that GenAI can shift work from authoring to high-stakes evaluation, and argues that accountability constraints limit autonomy. What is missing is an empirically testable account of when acceleration stops paying off due to assurance saturation, and how this appears in traces in ways separable from staffing changes, product maturity, or reporting shifts. We address this gap with a construct model relating rate of change (R), assurance capacity (A), justified confidence per change (J), and context-dependent safe velocity (V_s), and with three computable diagnostics for distinct breakdown modes: delayed validation (D1), reduced evidence-of-understanding per unit change (D2), and downstream concentration of failures (D3). This framing supports falsifiable propositions and study designs to estimate saturation thresholds and evaluate accountability-preserving interventions, rather than assuming assurance can always scale alongside throughput.

3 Post-velocity software engineering

What should we optimize once throughput no longer yields proportional gains in justified confidence and outcomes? Post-velocity posits threshold effects: assurance scales only up to a point, after which acceleration increases change volume more than justified confidence. To keep this framing testable, we define the core constructs and diagnostics and summarize their operationalization in Table 1.

Definition 3.1 (Post-velocity software engineering). Post-velocity software engineering denotes a regime in which the rate of change R meets or exceeds the socio-technical assurance capacity A to govern that change. Beyond this saturation point, additional acceleration yields diminishing or negative returns because validation, shared understanding, and accountable approval become binding constraints, which lowers justified confidence per change J and increases outcome risk. Optimisation therefore shifts from maximising throughput to managing the R–A relationship by prioritising assurance effectiveness and foreseeability under bounded human evaluation capacity. Operationally, we expect non-linear effects: once $R > A$, marginal increases in R correlate with a disproportionate decline in J and/or a disproportionate increase in downstream failures.

Post-velocity goes beyond a generic speed–quality trade-off by positing a regime shift: once assurance capacity saturates, added throughput increases change volume more than warranted confidence, and accountable approval becomes the bottleneck. This motivates diagnostics, falsifiers, and interventions that make acceleration conditional on evidence rather than assumed beneficial.

3.1 Core constructs

We use four constructs throughout the paper. Table 1 summarizes candidate proxies and data sources for these constructs, and the failure modes we expect once $R \gtrsim A$.

Table 1: Operationalization summary for post-velocity.

Concept	Operational proxies (examples)	Data sources	Expected failure mode when $R \gtrsim A$
R Rate of change	PRs/week; churn; diff-size distribution; deploy frequency; lead time	VCS and PR platform; CI/CD logs; release logs	Change volume grows faster than validation
A Assurance capacity	Review minutes/change; reviewer load; CI minutes/PR; test evidence/change; triage capacity	PR traces; CI logs; on-call/incident systems	Saturation of reviewers/tests; increasing queues
J Justified confidence	Review depth score; evidence completeness; post-change stability	PR text; test artifacts; static analysis; incident linkage	Evidence thins; approvals become procedural
V_s Safe velocity	Risk-stratified max R under acceptable defect/incident bounds	Governance policy; risk labels; component criticality	Throughput exceeds accountable approval bandwidth
D1 Validation lag	Time-to-first-review; pending PRs; test queue backlog; postmortem backlog	PR platform; CI system; incident tools	Growing validation backlog
D2 Procedural oversight	Rationale density vs diff size; "LGTM" rate on large diffs; missing evidence links	PR reviews; templates; policy checks	Symbolic approvals replace evidentiary review
D3 Downstream defect shifting	Rollbacks; change failure rate; incident rate; production hotfixes	Deployment logs; incident reports; postmortems	Pre-release issues surface in operations

Rate of change (R). The volume of change introduced per unit time, including code, configuration, tests, and deployment actions. Operational proxies include PRs/week, churn, diff-size distributions, deployment frequency, and change lead time.

Assurance capacity (A). The socio-technical resources available to validate and govern change per unit time. This includes reviewer time and expertise, test execution budget, triage and postmortem capacity, and governance bandwidth. Operational proxies include review minutes per change, reviewer load, CI minutes per PR, test evidence produced per change, and incident follow-up capacity.

Justified confidence per change (J). The extent to which a given change is supported by assurance evidence and accountable understanding. J is not self-reported confidence; it is evidence-based confidence. Operational proxies can combine review depth indicators (rationale quality, comment substance relative to diff size), test evidence (new tests, coverage deltas, validation artifacts), and post-change outcomes (rollback/incident association).

Safe velocity (V_s). The maximum rate of change such that accountability requirements are met and outcome risks remain within acceptable bounds for a given context. V_s depends on risk class, architecture criticality, and available assurance mechanisms, and is constrained by accountable approval and shared understanding.

Measurement commitments. We treat A and J as latent constructs measured through multiple proxies rather than as subjective judgments. In empirical studies, we will (i) define proxies with units, (ii) normalize them within team or project baselines to reduce cultural and tooling confounds, and (iii) report sensitivity analyses across alternative proxy sets instead of relying on a single metric. For J, we distinguish process evidence (e.g., review substance relative to diff size; linked test or evidence artifacts) from outcome evidence (e.g., association with rollbacks or incidents), and we use outcome signals to validate J rather than to define it.

Conceptual model We hypothesize a saturation regime in which assurance capacity A grows sub-linearly, while automation can increase the rate of change R rapidly. Let J denote justified confidence per change. In a pre-saturation regime ($R \ll A$), J remains approximately stable because teams apply substantive validation and review. In a saturation regime ($R \gtrsim A$), J declines as evidence and review depth per change thin, which can raise downstream operational risk even as throughput increases. We define safe velocity V_s as the context-dependent upper bound on sustainable delivery under accountability constraints, and we bound it by the organisation's ability to maintain justified approval and shared understanding rather than by code generation speed.

3.2 Candidate diagnostics

We propose three candidate diagnostics (D1-D3) computable from engineering traces and operational data (Table 1); Section 4 outlines how we will validate whether they reliably predict post-velocity outcomes. These diagnostics operationalize three observable manifestations of $R \gtrsim A$: delayed evidence (D1), reduced substance of oversight per unit change (D2), and downstream concentration of failures (D3).

D1: Validation lag. Increasing latency between change proposal and substantive validation (e.g., time-to-first-substantive-review, time-to-merge conditioned on review, CI queue delay), and/or growth in review/test queues as R increases.

D2: Procedural oversight. Approvals and governance actions that provide less evidence of substantive understanding per unit change than the team's historical baseline (e.g., declining rationale density relative to diff size, increasing LGTM-style approvals on large diffs). D2 does not claim that short reviews are inherently low-quality; it assumes that under saturation, evidence-of-understanding per unit change declines and oversight becomes increasingly symbolic rather than evidentiary.

D3: Downstream defect shifting. Issues that would plausibly be detected pre-release increasingly surfacing in operations (e.g., increased rollback frequency, incident rate attributable to recent changes), controlling for product maturity and reporting changes.

3.3 Falsifiable propositions

To make the post-velocity hypothesis testable, we formulate propositions with observable implications and falsifiers.

P1: Threshold effects. Holding assurance-capacity proxies roughly constant, outcome risk shows a change-point as R increases: below a context-dependent threshold, defect escape remains stable; above it ($R \gtrsim A$), higher R associates with increased defect escape and operational risk. *Falsifier:* no change-point is detected (or it does not align with A proxies) and outcomes remain stable as R rises.

P2: LLM amplification of mismatch. LLM tools reduce the marginal cost of change and increase change volume and/or size; unless assurance evidence scales proportionally, justified confidence per change J decreases as R increases. *Falsifier:* If LLM adoption raises R while assurance evidence scales so that J remains stable or improves, the mismatch mechanism is not supported.

P3: Accountability-bounded safe velocity. In domains requiring accountable approval, safe velocity V_s is bounded by humans' ability to maintain shared understanding and provide justified approval, even if automation scales generation. *Falsifier:* If such settings sustain higher R without increased procedural oversight and without worse outcomes, accountability may not bind as proposed.

3.4 Boundary conditions and scope

Post-velocity is not expected to apply uniformly. Threshold effects may be weak or absent when (i) changes are small and local, (ii) modularity provides strong isolation, (iii) formal methods or exhaustive automated checks scale assurance evidence, (iv) release cadence is intentionally slow, or (v) risk and accountability constraints are low. Conversely, post-velocity dynamics should be stronger when changes are frequent and cross-cutting, systems are tightly coupled, and approval is constrained by safety, security, regulatory, or availability requirements.

4 Research agenda

Our goal is to move post-velocity from a framing to a research program with concrete measurements, designs, and falsifiers.

4.1 Research questions

We propose four Research Questions (RQs) that map to the constructs, diagnostics, and propositions.

RQ1: Detecting post-velocity regimes. Under what conditions does an organization enter post-velocity operation ($R \gtrsim A$)? Which diagnostics (D1–D3) predict subsequent quality and reliability outcomes across contexts, and which remain robust to reporting and process changes?

RQ2: Threshold effects on outcomes. Does the R–outcome relationship exhibit a change-point consistent with assurance saturation (P1)? How do assurance practices, risk class, architectural criticality, and governance shift the threshold's location and sharpness?

RQ3: LLM-driven mismatch. How do LLM tools change the distribution of change volume, size, and novelty, and how does assurance evidence scale in response (P2)? How do review depth and justified confidence per change evolve as R increases?

RQ4: Accountability constraints and safe velocity. In settings with explicit accountability requirements, what constitutes safe velocity V_s (P3)? Which governance and tooling mechanisms preserve foreseeability and meaningful human control under sustained acceleration?

4.2 Study designs

We combine longitudinal trace analysis, controlled studies, and high-assurance case analysis to triangulate mechanisms and outcomes. We treat confounds (e.g., product maturity, staffing, and reporting policy) as first-class threats and design identification strategies accordingly.

S1: Longitudinal field studies with causal structure. We analyze trace and operational data before/after LLM adoption or across matched teams with different AI assistance levels. We measure outcomes (incidents, rollbacks, change failure rate, post-release defects) and mechanisms (D1–D3, J), and estimate effects via interrupted time series and difference-in-differences while controlling for team size, on-call, product maturity, and reporting changes. We test thresholds with segmented regression and change-point models, check whether breakpoints align with A proxies, and report robustness across proxy sets plus sensitivity to seasonality and release-freeze periods.

S2: Controlled studies of validation under throughput pressure. We run review-and-approval tasks with and without LLM assistance while varying time budgets and change volume. We measure defect detection, review depth (e.g., rationale and evidence linkage relative to diff size), confidence calibration, and approval quality. By holding domain and task difficulty constant, the design isolates saturation effects driven by throughput pressure and AI assistance.

S3: High-assurance case analyses. We study settings with explicit accountability and foreseeability constraints (e.g., regulated development, safety cases, security certification). We document how teams operationalize V_s, which governance invariants they enforce, and where acceleration breaks them. We examine how evidence artifacts scale with R and when oversight shifts from evidentiary to procedural.

S4: Measurement construction and replication package. We develop and validate protocols to compute D1–D3 and composite J from common toolchains, covering extraction, normalization, and threat analysis. We release a replication kit for cross-project measurement and document limitations, privacy constraints, and expected failure modes (e.g., missing incident linkage, inconsistent review templates).

4.3 Intervention space

We group interventions (I1-I3) into classes with measurable signatures on D1–D3 and J, and we evaluate them by testing whether they shift the estimated saturation threshold or reduce post-threshold outcome risk.

(I1) Risk-stratified gating. We require stronger evidence and review for high-criticality changes. We expect lower downstream defect shifting (D3) for high-risk work, with any increase in validation lag (D1) concentrated in that risk class rather than across all changes.

(I2) Differential assurance budgeting. We allocate reviewer and CI capacity in proportion to sustained R, change novelty, and component criticality (e.g., reviewer load caps, adaptive CI budgets, triage staffing triggers). We expect D1 and D2 to stabilize as R rises, and we test whether J remains stable rather than thinning.

(I3) Evidence-carrying changes. We require each change to include linked assurance evidence (tests, risk assessment, review rationale) and automate evidence capture where possible. We expect higher J and lower procedural oversight (D2) without necessarily reducing R, because the intervention increases evidence density rather than constraining throughput.

Across interventions, we treat evaluation as a socio-technical problem: we measure outcomes and mechanisms, we report trade-offs explicitly (e.g., where D1 increases to prevent D3), and we include governance and accountability constraints as design requirements rather than after-the-fact checks.

5 Conclusion

We hypothesized that AI-accelerated development can induce a post-velocity regime in which the rate of change meets or exceeds socio-technical assurance capacity, reducing justified confidence per change and increasing outcome risk. To make this hypothesis actionable, we introduced a definition of post-velocity software engineering, articulated computable diagnostics and falsifiable propositions, and outlined a research program and intervention space that condition acceleration on evidence and accountability. If supported empirically, the post-velocity framing motivates a shift from optimizing throughput alone to optimizing assurance effectiveness and the human capacity to understand, govern, and responsibly endorse continuous change.

Acknowledgments

This work is supported by: (a) the Swedish Agency for Innovation Systems through the projects “Secure: Developing Predictable and Secure IoT for Autonomous Systems" (2023-01899), (b) the Key Digital Technologies Joint Undertaking through the project “MATISSE: Model-based engineering of digital twins for early verification and validation of industrial systems" (101140216), and (c) the Clean Energy Transition Partnership through the project “FLEXI: Human-centered AI and digital twin powered energy system integration for flexibility markets" (101069750).

References

[1] Ritu Agarwal, Jayesh Prasad, Mohan Tanniru, and John Lynch. 2000. Risks of rapid application development. *Commun. ACM* 43, 11es (2000), 1–es.

[2] Alessio Bucaioni, Antonio Cicchetti, Gordana Dodig-Crnkovic, Romina Spalazzese, Emma Söderberg, and Dániel Varró. 2026. Engineering Future Critical CPSs with Trustworthy GenAI Across the Lifecycle. In *48th IEEE/ACM International Conference on Software Engineering*. http://www.es.mdu.se/publications/7308-

[3] Ward Cunningham. 1992. The WyCash portfolio management system. *ACM Sigplan Oops Messenger* 4, 2 (1992), 29–30.

[4] Rogério De Lemos. 2020. Human in the loop: What is the point of no return?. In *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 165–166.

[5] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M. Zhang. 2023. Large Language Models for Software Engineering: Survey and Open Problems. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*. 31–53. doi:10.1109/ICSE-FoSE59343.2023.00008

[6] Odd Ivar Haugen, Dag McGeorge, Tore Myhrvold, Christian Agrell, and Andreas Hafver. 2024. Assurance of AI-enabled systems. In *Proceedings of the 2024 8th International Conference on Advances in Artificial Intelligence*. 100–106.

[7] Andreas Holzinger, Kurt Zatloukal, and Heimo Müller. 2025. Is human oversight to AI systems still possible? 59–62 pages.

[8] Syed Md Faisal Ali Khan and Salem Suhluli. 2025. Generative AI and Cognitive Challenges in Research: Balancing Cognitive Load, Fatigue, and Human Resilience. *Technologies* 13, 11 (2025), 486.

[9] Herb Krasner. 2021. The cost of poor software quality in the US: A 2020 report. *Proc. Consortium Inf. Softw. QualityTM (CISQTM)* 2 (2021), 3.

[10] Margaret Mitchell, Avijit Ghosh, Alexandra Sasha Luccioni, and Giada Pistilli. 2025. Fully autonomous ai agents should not be developed. *arXiv preprint arXiv:2502.02649* (2025).

[11] Joshua W Ohde, Lauren M Rost, and Joshua D Overgaard. 2025. The burden of reviewing LLM-generated content. AIp2400979 pages.

[12] Maluane Rubert and Kleinner Farias. 2022. On the effects of continuous delivery on code quality: A case study in industry. *Computer Standards & Interfaces* 81 (2022), 103588.

[13] Chang Xiao and Zixiaofan Yang. 2025. Streaming, fast and slow: Cognitive load-aware streaming for efficient llm serving. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology*. 1–13.