The Bottom-Up Approach to Developer Experience: Empowering Engineers to Drive Change

[1] Artem Mukhin
[1] Software Engineer Belgrade, Serbia

Received: 24th Nov 2025 | Received Revised Version: 29th Nov 2025 | Accepted: 24th Dec 2025 | Published: 09th Jan 2026

Volume 08 Issue 01 2026 | Crossref DOI: 10.37547/tajet/Volume08Issue01-07

Abstract

This article examines the issue of friction in software development and proposes, as an alternative and complement to top-down initiatives, the bottom-up enhancement of Developer Experience (DevEx). Large-scale productivity losses caused by cognitive overload, tool inefficiencies, and organizational misalignment translate into big bucks when it happens on an industrial scale. That is what makes this study relevant. The objective is to provide theoretical and practical justification for empowering engineers as agents of change within DevEx governance. The study's novelty lies in integrating three analytical frames—DevEx, Cognitive Load Theory, and the socio-technical congruence model—into a unified, multi-level model of friction, and in proposing practical instruments (a friction log and the DX-champion role) that transmute local engineer-led initiatives into mechanisms for diagnosing and correcting systemic incongruences. The main takeaways speak to the constraints of solely top-down efforts which depend on accumulated measures and choices that disregard the daily worker's perspective. The suggested bottom-up method demonstrates awareness of small-scale mental blocks and establishes ways to leverage nearby worker insights into company progress. An optimal configuration likely combines both approaches. Top-down initiatives provide empirical grounding and organizational commitment. Bottom-up efforts offer concrete means of change and organic growth. The article will be useful to software engineering researchers, technology organization leaders, and practicing engineers interested in systematically elevating the quality of DevEx.

Keywords: Developer Experience, cognitive load, socio-technical congruence, bottom-up approach, software development productivity, engineering practices.

Cite This Article: Mukhin, A. (2026). The Bottom-Up Approach to Developer Experience: Empowering Engineers to Drive Change. The American Journal of Engineering and Technology, 8(01), 44–52. https://doi.org/10.37547/tajet/Volume08Issue01-07

1. Introduction

The problem of friction in software development is not a mere aggregation of minor inconveniences; it is a significant systemic drag on innovation and value delivery. According to Atlassian's State of Developer Experience Report 2025, 90% of developers lose at least 6 hours per week, and 50% lose more than 10 hours due to factors such as information seeking, incompatible tools, and context switching. For a team of 500 engineers, this equates to an annual productivity loss of 7.9 million dollars (Atlassian, 2025). These figures foreground the economic imperative to address the issue.

Developer Experience (DevEx) is a formal discipline dedicated to understanding and mitigating this friction. It encompasses all facets of an engineer's work, from toolchain performance to cognitive and emotional states. Historically, DevEx emerged as an extension of User

Experience (UX) principles applied to the developer as a user of their working environment (Fagerholm & Münch, 2013).

The central thesis of this article is that the contemporary industry approach to DevEx is predominantly top-down in nature. Leadership-originated initiatives, organization-wide platform deployments, and a focus on aggregated metrics characterize it. Although valuable, this approach often fails to apprehend the lived realities of developers' workflows. Atlassian reports a widening perception gap: 63% of developers believe leadership does not understand their real problems, up sharply from 44% the previous year (Atlassian, 2025). This work argues that meaningful and durable improvement necessitates a complementary bottom-up approach that empowers individual engineers to identify and rectify friction.

Atlassian's data reveal a notable paradox: while AI-based tools save developers substantial time (68% save more than 10 hours per week), the sense that leadership does not understand developers' problems rises sharply (Atlassian, 2025). This suggests that top-down technological solutions (e.g., provisioning AI tools) do not resolve the fundamental, systemic sources of friction that developers face. The problem is not the absence of powerful tools, but the lack of alignment between engineers' everyday experiences and leadership's strategic priorities. This paradox powerfully motivates the article's thesis: without an upward channel to articulate and resolve systemic problems, top-down solutions will fail to improve overall developer experience.

This paper aims to: (1) propose a multi-layer model of developer friction grounded in cognitive load theory and socio-technical congruence; (2) articulate how bottom-up DevEx initiatives can operate within this model; and (3) discuss implications for combining top-down and bottom-up approaches.

2. Materials and Methodology

The study is based on an interdisciplinary analysis that combines empirical data from industry reports, academic research, and conceptual models that describe developer friction. The principal quantitative source is the State of Developer Experience Report 2025 (Atlassian, 2025), which quantifies productivity losses and exposes the perception gap between engineers and leadership. This report serves as the statistical basis for the economic case and for calibrating the scale of friction in terms of lost time and resources.

The theoretical substrate comprises three interrelated frames. First, a conceptualization of Developer Experience as a cognitive, affective, and conative phenomenon (Fagerholm & Münch, 2013), offering a micro-level perspective on individual perception. Second, Cognitive Load Theory describes how the scarcity of working-memory resources can be redistributed and elucidates the process by which external friction turns into a barrier to productivity and learning. Third, the socio-technical congruence (STC) model takes the analysis even higher to the macro-level of organizational structures, thereby exposing the systemic misalignments at their very source.

Methodologically, the study applies three triangulated approaches. First, it compares industry models of productivity, including DORA metrics (Harvey, 2025) and SPACE (Forsgren et al., 2021), revealing a structural bias toward top-down models that cannot be inclusive of all the nuanced aspects of engineering experience. Second, a content analysis of industry discourse is conducted based on domain conference talks (KubeCon, n.d.) and analytical materials (Gerdemann et al., 2024; Forsgren, 2024), which reveals an imbalanced articulation where leadership and platform-team voices dominate systematic bottom-up initiatives. Third, a systematic review of bottom-up practices is conducted, inclusive of onboarding cases (Ju et al., 2021) and empirical cognitive effects studies regarding interruptions and context-switching (Mark et al., 2008). This data helps in building a micro-level picture of friction and corresponding practical tools, such as friction logs and the DX-champion role.

To operationalize the triangulated approach, the study draws on a corpus of approximately thirty industry talks and fifteen analytical materials on developer productivity, platform engineering, and DevEx, selected from KubeCon + CloudNativeCon programs and major industry outlets based on three criteria: (1) DevEx, developer productivity, or platform engineering as a primary focus; (2) explicit discussion of concrete practices or governance mechanisms (e.g., metrics programs, platform initiatives, onboarding strategies); and (3) availability of complete recordings or texts in English. These materials were subjected to a mixed deductive–inductive content analysis: an initial codebook derived from the DevEx, CLT, and STC frames (e.g., "top-down initiative," "bottom-up practice," "inner loop," "outer loop," "coordination/ownership") was iteratively refined through open coding on a pilot subset, after which the final codes were applied to the whole corpus to quantify the relative prominence and co-occurrence of top-down versus bottom-up themes. In parallel, a systematic review of bottom-up DevEx practices was conducted in ACM Digital Library, IEEE Xplore, arXiv, and Google Scholar using combinations of keywords such as "developer experience," "onboarding," "context switching," "interruptions," "cognitive load," "socio-technical congruence," "bottom-up," and "developer-led improvement" over the period 2008–2025; empirical and design-oriented studies were included if they described professional software development settings and reported either (a) cognitive or coordination costs (e.g., interruptions, onboarding barriers) or (b) concrete developer-initiated interventions affecting daily workflows, and the extracted practices were then synthesized into thematic clusters (inner-loop acceleration, onboarding support, self-service tooling, coordination artifacts, communities of practice) that directly inform the practical toolkit proposed in this article.

3. Results and Discussion

The academic lineage of DevEx can be traced to the foundational work of Fagerholm and Münch, who defined it as how developers think and feel about their activities in their work environment (Fagerholm & Münch, 2013). Their conceptual model comprises three principal dimensions, providing a multifaceted lens that moves beyond simplistic performance metrics:

- Cognitive dimension (Cognition) - the developer's perception of infrastructure, tools, and processes.
- Affective dimension (Affect) - the emotional state associated with work, including satisfaction and frustration.
- Conative dimension (Conation) - motivation, action orientation, and the sense of value contribution.

The concept has evolved, yielding more practice-oriented models, such as the DX Framework by Greiler, Storey, and Noda, which are intended to narrow the theory–practice gap. Strong DevEx is tightly coupled with achieving a flow state—deep task immersion that enables peak performance. Thus, high-quality DevEx is a necessary precondition for entering flow.

Cognitive Load Theory (CLT) provides the principal mechanism for explaining friction in developers' work (Baxter et al., 2025). CLT distinguishes three types of load on human working memory:

- Intrinsic load - complexity inherent to the task or software system.
- Extraneous load - mental effort imposed by the work environment—poorly designed tools, convoluted processes, fragmented information, and slow feedback loops.
- Germane load - effort directed toward deep learning, sense-making, and the formation of mental schemas (Gkintoni et al., 2025).

The core argument is that poor DevEx manifests as elevated extraneous cognitive load. This load consumes scarce working-memory resources, leaving fewer cognitive capacities for the intrinsic and germane loads required for problem solving and innovation (Mark et al., 2008). Empirical studies on the cost of interruptions and context switching corroborate this theoretical claim. Although interrupted workers may complete tasks faster, they do so at the expense of significantly higher stress, frustration, and exertion (Mark et al., 2008).

The concept of Socio-Technical Congruence (STC), developed by Cataldo, Herbsleb, and Carley, furnishes a macro-level model for understanding systemic friction (Cataldo et al., 2008). STC posits that performance is maximized when patterns of coordination in the development team (the social structure) align with the coordination needs dictated by dependencies in the codebase (the technical structure).

Incongruence is a primary source of friction. It arises when developers who need to collaborate (due to shared code dependencies) do not, or when developers who do not need to collaborate are compelled into superfluous communication. A key implication of this model—that high congruence substantially reduces time-to-resolution for change requests—offers a powerful explanation for why seemingly minor process improvements can exert disproportionate influence on performance.

The following figures present a network visualization of socio-technical congruence (STC) in two states: before (Figure 1) and after (Figure 2) targeted intervention. The visualization links the technical structure (modules and their dependencies) with actual communication channels among responsible developers and teams. According to the multi-level friction model, divergences between these layers (low congruence) are systemic root causes of increased extraneous cognitive load and the attendant decline in development efficiency.

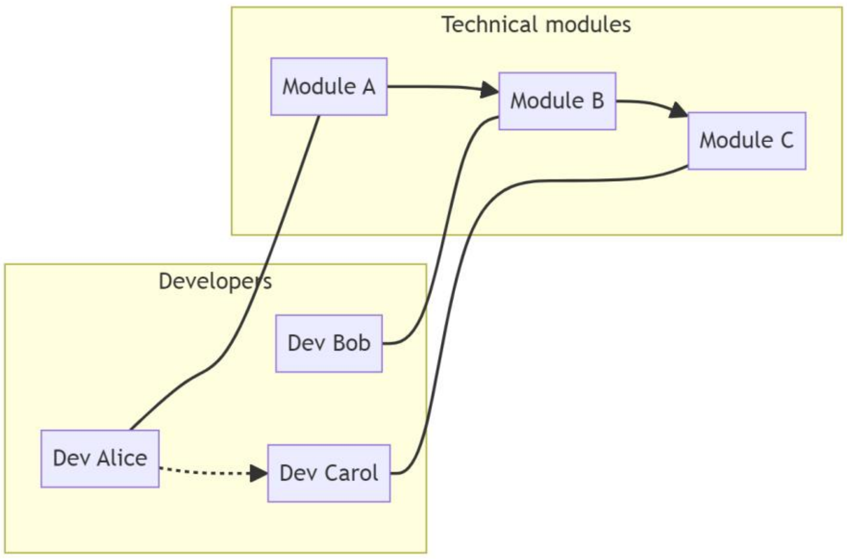

Fig. 1. Socio-technical incongruence, low (compiled by author)

The Before panel highlights key points of mismatch—dependency edges not covered by communication links—which we label gaps and treat as priorities for intervention.

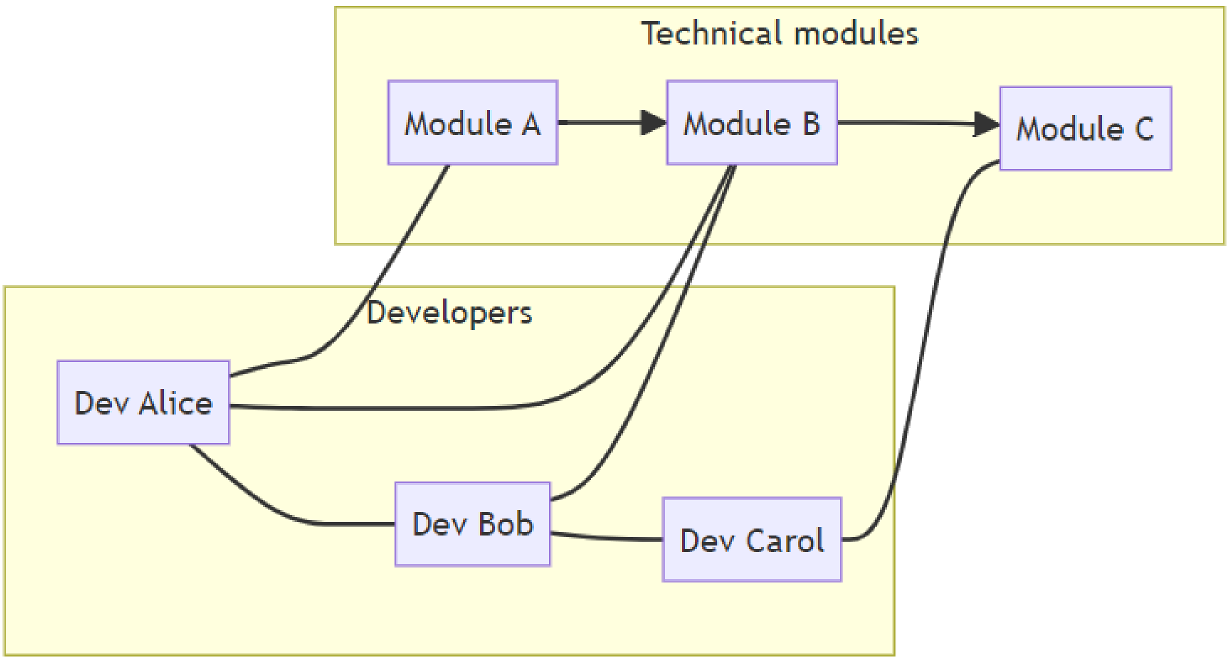

Fig. 2. Socio-technical congruence, high (compiled by author)

The After panel illustrates a scenario in which communication is aligned with technical dependencies; expected effects include an increase in the local STC index, a reduction in median time-to-resolve PRs, and fewer cross-team escalations.

These three theories—DevEx, CLT, and STC—are not independent; together they constitute a multi-level causal model of friction in development. At the macro-level, socio-technical incongruence (e.g., a developer needs information from another team but lacks a clear channel) creates conditions for friction: a systemic root cause. At the meso-level, this systemic friction materializes as concrete DevEx problems: slow feedback loops (waiting on another team's response), fragmented tooling (using disparate systems), and inefficient processes (manual handoffs). Finally, at the micro-level, these DevEx problems directly result in a high extraneous cognitive load for the individual developer. The mental effort spent navigating an incongruent system (e.g., searching for documentation, context switching while awaiting code review) taxes cognitive resources, impedes entry into flow, and yields negative affective consequences (frustration, stress) described in the original DevEx definition. This unified model explains why bottom-up improvements are practical: they are initiated by those most sensitive to the micro-level cognitive load induced by macro-level systemic incongruence. A manager sees a process diagram; a developer feels the mental friction of its deficits.

Consider two influential performance models: DORA and SPACE. DORA metrics (deployment frequency, lead time for changes, change failure rate, mean time to restore) primarily measure outcomes of the outer loop of development—the delivery pipeline. While important, these metrics are lagging indicators of system performance and do not directly reflect the experience of the developer's inner loop. There is concern that DORA metrics can be misused to compare teams or individuals, encouraging metric gaming (Harvey, 2025).

The SPACE model (Satisfaction, Performance, Activity, Communication, and Efficiency/Flow) is presented as a more holistic model that includes developer well-being. But usually, it is a top-down strategy that starts with leadership choosing which metrics to "track" across the organization. While this can be sold as empowerment at the developer level, in reality, it happens at the organizational level (Forsgren et al., 2021). This proves that while such models may be extremely valuable, they institutionalize a top-down view of developers as objects of measurement rather than agents of improvement.

Industry discourse shows a pronounced skew toward top-down perspectives. Most of them hold roles with 'Manager', 'Director' or 'Platform Lead' in the title, discussing large-scale, top-down initiatives such as developer platform toolchain standardization and team-level metric programs (KubeCon, n.d.). There are very few rank-and-file engineers who give talks on grassroots efforts. While academic work heavily leans toward bottom-up phenomena, developer behavior, and onboarding barriers, this does not seem to be reflected in the practical industry discourse (Ju et al., 2021).

This imbalance engenders a self-reinforcing cycle. Conference organizers seek speakers who can narrate large-scale, high-impact initiatives, as these are perceived as more valuable to an audience of leaders and budget decision-makers. Managers and directors are precisely those who lead such top-down projects and have full visibility into them. Individual developers spearheading smaller bottom-up initiatives often lack a formal platform or mandate to present at major conferences. Consequently, conference discourse becomes dominated by managers speaking about top-down solutions. Attendees (usually other managers) infer that the correct way to pursue DevEx is via large, top-down platform initiatives. They return to their companies and champion similar projects, perpetuating the cycle. This creates an industry-wide blind spot: the most effective, context-dependent, and high-leverage improvements—those surfaced bottom-up—are systematically underrepresented and undervalued in public discourse, amplifying the very top-down bias this article seeks to contest.

The trend toward Platform Engineering is the principal technological response to DevEx challenges. Platforms aim to improve DevEx by providing a standardized, self-service layer that abstracts infrastructure complexity (Gerdemann et al., 2024). The advantages are evident: increased coherence, reduced cognitive load associated with operations, and accelerated onboarding.

Yet this top-down approach has potential drawbacks. If a platform team becomes decoupled from its developer-customers, it risks paving a road that leads where developers do not need to go. The platform itself can become a fresh source of friction and extraneous cognitive load if its interfaces are unintuitive or its capabilities misaligned with real workflows (Forsgren, 2024).

The approach is summarized as a practical toolkit that any engineer can employ and illustrated in Figure 3.

Method 1: Friction Log. A systematic, low-overhead means to record problematic moments in a developer's work. Borrowed from product management, the friction log is a document that captures difficulties or frustrations encountered during the workflow. For developers, these may include slow builds, tangled documentation, flaky tests, or non-intuitive APIs. Benefits include making the invisible visible, providing data-based grounds for prioritization, and cultivating empathy.

Method 2: DX-Champion. The DX-champion is formalized as the social analogue of the friction log. Not necessarily a formal role, it is a responsibility voluntarily assumed by an engineer to:

1. Collect and synthesize friction logs and peer feedback.
2. Identify patterns and prioritize the most consequential problems.
3. Initiate minor improvements (e.g., write a script, enhance documentation).
4. Report outcomes and the value of improvements to leadership, acting as a bridge between the development team and leaders.

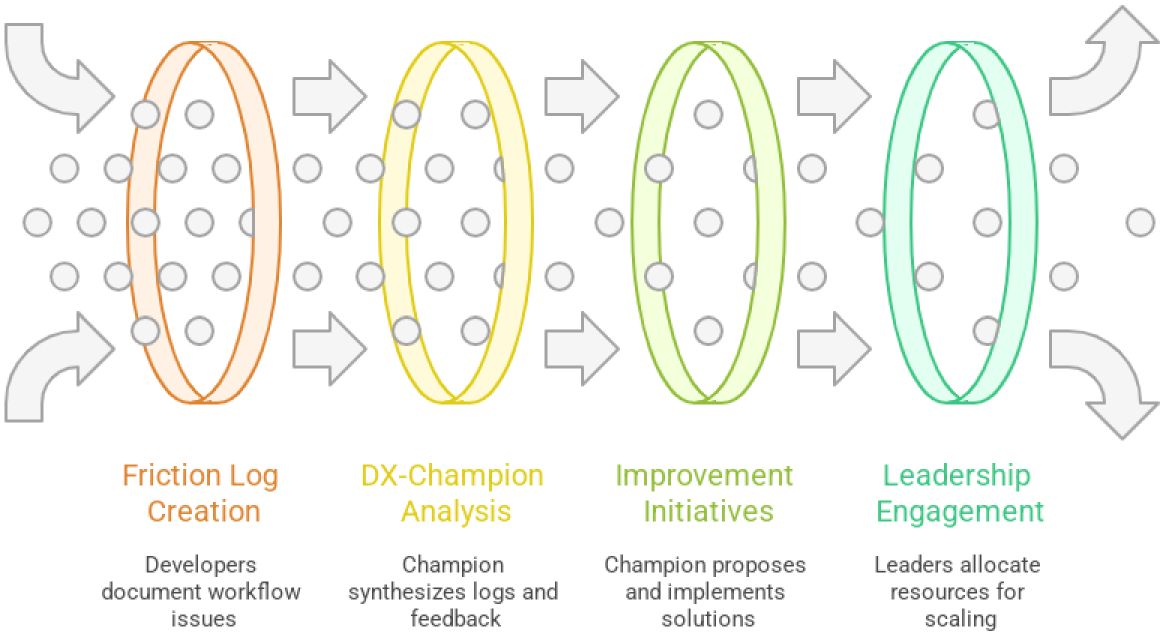
Friction Log Creation
Developers document workflow issues
DX-Champion Analysis
Champion synthesizes logs and feedback
Improvement Initiatives
Champion proposes and implements solutions
Leadership Engagement
Leaders allocate resources for scaling

Fig. 3. Enhancing Developer Experience through Feedback (compiled by author)

These bottom-up tools can be construed as socio-technical probes. A friction log is not merely a list of complaints; it is an instrument for gathering qualitative data to locate points of socio-technical incongruence. An entry such as Had to message three people in Slack to find the required configuration file is a direct symptom of misalignment between a technical need (access to configuration) and the social structure (no clear owner or documentation). The DX-champion functions as the analyst of these data. By synthesizing logs, they do more than find bugs; they conduct a qualitative analysis of the organization's socio-technical system from the perspective of those who bear its costs most acutely. The small improvements they propose (e.g., Let's create a single, versioned configuration guide) are not mere fixes; they are targeted interventions designed to raise socio-technical congruence. This reframes bottom-up tools from simple improvement hacks into sophisticated, developer-driven methods for diagnosing and resolving systemic organizational problems—the practical application of STC that requires no formal mandate or managerial oversight to initiate.

Both approaches are necessary and most effective in tandem. The ideal model is symbiosis. Developers and DX-champions use friction logs to surface real problems, experiment with small-scale solutions, and ensure a continuous stream of high-quality qualitative data about the state of developer experience. A top-down amplifier then engages. Leadership and platform teams heed this feedback, provide resources (time, budget) to promising initiatives, use their strategic vantage to identify organization-wide regularities, and create platforms and standards that scale successful bottom-up solutions across the organization.

Recommendations for leaders follow. Rather than simply deploying measurement systems, leaders should create conditions for bottom-up feedback to emerge: fostering psychological safety, actively collecting and responding to friction logs, and identifying and empowering potential DX-champions within their teams. Their role shifts from director to facilitator.

Recommendations for developer's concern becoming effective agents of change: articulating friction in terms of business impact (e.g., This slow build costs X hours per week, delaying feature delivery), starting with small, visible fixes to earn trust, and forming alliances with other developers to demonstrate that a problem is shared rather than idiosyncratic.

To illustrate how the proposed framework manifests in real-world settings, this section presents several brief case vignettes from large software organizations. Research indicates that bottom-up initiatives in Developer Experience (DevEx) can reshape organizations just as effectively as large top-down programs. Within a large-scale commercial software product, the systematic articulation of the need for sustained DX work led to the creation of a dedicated DevEx function that had not previously existed. This established a permanent upward channel through which day-to-day friction could be translated into the language of business priorities and technical plans. In terms of socio-technical congruence (STC), such a function adds a coordination node between teams and their dependencies; in terms of Cognitive Load Theory (CLT), it directs effort toward reducing extraneous load in the developer's inner loop.

A first-order effect was the radical acceleration of the local mobile edit–build–run cycle. The typical delay between saving in the IDE and observing changes on a device or emulator had been on the order of one to two minutes, repeatedly breaking flow. Optimizing the pipeline reduced the wait time to a matter of seconds, sharply improving feedback quality and the subjective work experience. In CLT terms, this constitutes a direct reduction of extraneous load by eliminating waiting and context switching; in DevEx terms, it restores the conditions necessary for entering and maintaining a flow state.

In parallel, a DX portal was implemented to aggregate data on dependencies and npm package vulnerabilities across the codebase. During a large-scale security remediation effort, this interface enabled the rapid identification and prioritization of updates, facilitating planning with shared, inspection-friendly data. The portal functions as a coordination artifact: technically dispersed signals become a single source of truth, and the cost of alignment falls. In STC terms, it increases transparency across teams and code artifacts, thereby raising congruence.

Preparation was also undertaken for the removal of a widely used library from a large codebase. Usage sites were inventoried and classified, the potential for automation was estimated, and a migration plan was produced with a Gantt chart, documentation, and scripts/codemods. Execution was paused for organizational reasons, but the methodology itself—discovery, partial automation, and staged rollout—remains reusable and applicable. This approach again aligns social and technical layers: a common plan, clear migration windows, and explicit rollback points reduce unplanned communications and escalations.

Beyond strictly engineering interventions, product-grade recommendations on UI performance and user experience were prepared. The combination of design and engineering sharpened problem framing and produced solutions that reduce cognitive "noise" for both developers and end users. Taken together, such interventions increase the proportion of germane cognitive effort by removing friction in common scenarios.

Practices did not remain local. Results and approaches were disseminated through an internal company community, enabling knowledge diffusion and reuse of successful patterns. Strengthening network ties across teams increased socio-technical congruence without formal reorganization.

A similar bottom-up logic proved effective in another large organization. In an internal developer tools unit, two productivity tools were designed and built end-to-end on the basis of studied team needs. Most of the user experience was planned up front, allowing the tools to address recognizable daily "grit" and eliminate manual glue work. The result was shorter durations for typical workflows and fewer context switches.

In a commercial division, a self-service media-planning tool replaced the manual work of two to three analysts. Managers were able to perform calculations in seconds or minutes instead of waiting more than a day, and to adjust plans in real time during client meetings. Architecturally, the solution evolved from a versioned Excel prototype to a full interface on MS Access, and then to a React-based web application. After being handed off to another team, it was released externally and integrated into the product line. Removing this human gate between request and calculation is a canonical instance of increased congruence: technical capability brought into alignment with the organization's coordination structure.

To support decision-making, OLAP/MDX was mastered to extract the exact aggregates required, and automated reports were configured in SSRS, allowing managers and planners to self-serve standardized data. This reduced reliance on narrow bottlenecks, accelerated the iteration cadence, and minimized "search" communication around metrics. Additionally, an internal communication venue was launched and moderated, reaching more than 800 participants by 2022. A single locus for questions and answers, together with curated FAQs, sped expert discovery and fostered de facto standards.

From these episodes emerges a reproducible set of practices. Assigning a rotating DX champion with sustained, lightweight focus on papercuts and friction logs; running short sprints to accelerate the inner loop with explicit targets; establishing a minimal dependency-hygiene portal with a "fix-next" queue; applying a method for large-scale change based on discovery, automation, and wave-based rollout; preferring self-service over human gates; and cultivating a community of practice—all of these reduce extraneous cognitive load while increasing the alignment of social and technical dependencies. The metrics most suitable for tracking within this approach include edit-to-run loop time, median and percentile review wait times, CI queueing time, and time-to-first-meaningful-contribution for newcomers.

Within the article's model, these observations illustrate how DevEx micro-practices return working memory to the task at hand, how cognitive load is reallocated away from extraneous demands toward intrinsic and germane effort, and how socio-technical congruence increases through coordination artifacts and transparent paths. Bottom-up improvements operate as socio-technical probes, revealing local bottlenecks, generating data for prioritization, and then scaling with the participation of platform and leadership teams. The result is the sought-after symbiosis in DX governance, with reality and speed supplied from below and resilience and scale supplied from above.

4. Conclusion

The analysis underscores that friction in software development is neither epiphenomenal nor a local inefficiency; it is a fundamental constraint on innovation and organizational dynamics. Classical top-down approaches—platform initiatives and organization-level metrics—are important, yet limited insofar as they primarily operate with aggregated indicators and strategic decisions that do not capture the micro-level of engineers' everyday experiences. Thus, they risk reproducing precisely the mismatches between technical dependencies and social structures that the STC theory identifies as systemic sources of friction.

The cognitive, affective, and conative dimensions of Developer Experience, combined with cognitive load and the STC model, form a holistic, multi-level analytical framework. One direct corollary is that the higher the extraneous cognitive load, the more blocked the path to flow becomes, and therefore the lower the ratings on the affective indicators that point to systemic inefficiency. Thus, any sustainable DevEx improvements must result from mechanisms that minimize extraneous load. Here, bottom-up practices, the friction log, and the DX-champion role assume decisive importance. They not only surface micro-problems but also become instruments for diagnosing socio-technical incongruence, rendering visible the hidden rifts between an organization's social and technical structures.

Accordingly, the bottom-up concept is not reducible to local tweaks or hacks, but constitutes a systemic feedback loop embedded within organizational development. When engineers act as initiators of change, they serve as forces for socio-technical alignment, thereby translating cognitive dissonance at the individual level into macro-level adjustments within the organizational process. In this approach, top-down initiatives do not remain mere directions from above but become means through which validated solutions from below are adopted and subsequently institutionalized.

The best architecture for governing developer experience is a balance of two forces. One supplies feeling and reality from the bottom up, while the other gives support and size from the top down. Their coupling minimizes the cognitive tax on developers, raises socio-technical congruence, and ultimately transforms DevEx from a peripheral discipline into a strategic determinant of organizational competitiveness.

References

1. Atlassian. (2025). *State of Developer Experience*

Report 2025. Atlassian. https://www.atlassian.com/teams/software-development/state-of-developer-experience-2025

2. Baxter, K. A., Sachdeva, N., & Baker, S. (2025). The Application of Cognitive Load Theory to the Design of Health and Behavior Change Programs: Principles and Recommendations. *Health Education & Behavior*, *52*(4), 469–477. https://doi.org/10.1177/10901981251327185
3. Cataldo, M., Herbsleb, J. D., & Carley, K. M. (2008). Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '08,* 2-11. https://doi.org/10.1145/1414004.1414008
4. Fagerholm, F., & Münch, J. (2013). Developer Experience: Concept and Definition. *Arxiv*. https://doi.org/10.48550/arxiv.1312.1452
5. Forsgren, N. (2024, January 23). *Quantifying the impact of developer experience*. Microsoft Azure. https://azure.microsoft.com/en-us/blog/quantifying-the-impact-of-developer-experience/
6. Forsgren, N., Storey, M.-A., & Maddila, C. (2021). *The SPACE of Developer Productivity*. ACM. https://queue.acm.org/detail.cfm?id=3454124
7. Gerdemann, D., McLaren, L., Werner, S., & Hesse, D. (2024, August 19). *Speeding up software development*. Kearney. https://www.kearney.com/service/digital-analytics/article/speeding-up-software-development
8. Gkintoni, E., Antonopoulou, H., Sortwell, A., & Halkiopoulos, C. (2025). Challenging Cognitive Load Theory: The Role of Educational Neuroscience and Artificial Intelligence in Redefining Learning Efficacy. *Brain Sciences*, *15*(2), 203. https://doi.org/10.3390/brainsci15020203
9. Harvey, N. (2025). *DORA's software delivery metrics: the four keys*. DORA. https://dora.dev/guides/dora-metrics-four-keys/
10. Ju, A., Sajnani, H., Kelly, S., & Herzig, K. (2021). A Case Study of Onboarding in Software Teams: Tasks and Strategies. *Arxiv*. https://doi.org/10.48550/arxiv.2103.05055
11. KubeCon. (n.d.). *Keynote Speakers*. LF Events. Retrieved August 11, 2025, from https://events.linuxfoundation.org/archive/2024/kubecon-cloudnativecon-north-america/program/keynote-speakers/
12. Mark, G., Gudith, D., & Klocke, U. (2008). The cost of interrupted work. *Proceedings of the Twenty-Sixth Annual CHI Conference on Human Factors in Computing Systems - CHI '08*, 107–110. https://doi.org/10.1145/1357054.1357072