

ИНФОРМАТИКА И ИНФОРМАЦИОННЫЕ ПРОЦЕССЫ/INFORMATICS AND INFORMATION PROCESSES

DOI: https://doi.org/10.60797/itech.2026.11.10 EDN: SLIKBI

SPARK FOR DEVEX: A MINIMAL BOTTOM-UP METHOD TO TURN DEVELOPER PAIN INTO BUSINESS VALUE

Research article

Mukhin A.[1,] *

[1] ORCID : 0009-0001-6456-5137;

[1] Independent researcher, Belgrade, Serbia

* Corresponding author (tim.mukhin.dev[at]gmail.com)

Suggested: 10.02.2026; Accepted: 14.04.2026; Published: 14.07.2026

Abstract

This article presents SPARK for DevEx, a minimalist and reproducible bottom-up methodology that helps translate individual developer frictions into business-justified initiatives for improving developer experience and productivity. The motivation is the growing organizational cost of invisible workflow friction, which is often normalized and therefore underreported, thereby widening the communication gap between engineers and leadership. In this context, SPARK is proposed as a complementary approach to established top-down programs (e.g., Platform Engineering), enabling developers to collect evidence and build momentum for broader DevEx investments. SPARK operationalizes the translation of subjective pain into measurable indicators by aligning improvement efforts with widely used frameworks such as DORA and SPACE. The method synthesizes findings from academic and applied sources into a practical five-step cycle that supports measurability, transparency, and psychologically safe reporting. The paper discusses cases, such as CI/CD feedback-loop optimization and onboarding acceleration, to show how SPARK can be applied to reduce local bottlenecks and produce data that connects technical changes to delivery and human-factor metrics. The article draws on illustrative scenarios (analytical vignettes) from engineering teams. The intended audience includes DevEx researchers and practitioners, engineering managers, platform team leaders, and organizations seeking to enhance productivity and foster a resilient engineering culture.

Keywords: developer experience, SPARK, DevEx governance, DORA, SPACE.

SPARK ДЛЯ DEVEX: МИНИМАЛИСТИЧНЫЙ ПОДХОД «СНИЗУ ВВЕРХ», ПРЕВРАЩАЮЩИЙ БОЛЬ РАЗРАБОТЧИКОВ В ЦЕННОСТЬ ДЛЯ БИЗНЕСА

Научная статья

Мухин А.[1,] *

[1] ORCID : 0009-0001-6456-5137;

[1] Независимый исследователь, Белград, Сербия

* Корреспондирующий автор (tim.mukhin.dev[at]gmail.com)

Предложена: 10.02.2026; Принята: 14.04.2026; Опубликована: 14.07.2026

Аннотация

В статье представлен SPARK для DevEx — минималистичная и воспроизводимая методология «снизу вверх», которая помогает переводить отдельные затруднения и «трения» в работе разработчиков в инициативы, обоснованные с точки зрения бизнеса, направленные на улучшение опыта разработчиков и повышение продуктивности. Мотивация связана с ростом организационных затрат из-за невидимых помех в рабочих процессах: такие помехи часто нормализуются и потому недооцениваются и не попадают в отчётность, что расширяет коммуникационный разрыв между инженерами и руководством. В этом контексте SPARK предлагается как дополняющий подход к устоявшимся программам «сверху вниз» (например, платформенной инженерии), позволяющий разработчикам собирать доказательства и формировать импульс для более масштабных инвестиций в DevEx. SPARK операционализирует перевод субъективной «боли» в измеримые индикаторы, связывая усилия по улучшениям с широко используемыми фреймворками, такими как DORA и SPACE. Метод объединяет выводы из академических и прикладных источников в практический пятишаговый цикл, поддерживающий измеримость, прозрачность и психологически безопасную отчётность. В статье рассматриваются примеры, такие как оптимизация цикла обратной связи в CI/CD и ускорение онбординга, чтобы показать, как SPARK можно применять для устранения локальных узких мест и получения данных, которые связывают технические изменения с метриками поставки и человеческого фактора. Статья опирается на иллюстративные сценарии (аналитические виньетки) из практики инженерных команд. Целевая аудитория включает исследователей и практиков DevEx, инженерных менеджеров, руководителей платформенных команд и организации, стремящиеся повысить продуктивность и укрепить устойчивую инженерную культуру.

Ключевые слова: опыт разработчика, SPARK, управление DevEx, DORA, SPACE.

Introduction

Low-quality Developer Experience (DevEx) is not a minor inconvenience but a material economic leakage for organizations. Contemporary studies quantitatively corroborate this problem: according to the Atlassian State of Developer Experience Report 2025, 90% of developers lose at least 6 hours per week due to various workflow frictions, and 50% lose

more than 10 hours. For an organization with a staff of 500 engineers, this translates into annual productivity losses of $7.9 million [1].

A paradox identified in the same report lends the issue particular urgency: AI-based tools save developers significant time (68% save more than 10 hours per week), yet these gains are neutralized mainly by organizational inefficiencies [1]. This speaks to a problem that is far more about process and communication than it is about any lack of technological solutions available.

The normalization of developer pain is implemented as follows. Engineers mostly get used to inefficient workflows, long builds, fragmented tooling, arduous onboarding, acclimating as if this is simply the way things are. The quiet dissatisfaction this state of affairs breeds does not make for a well-articulated case for change, thereby impeding an organization from identifying and addressing systemic inadequacies.

The primary barrier to DevEx improvement is the communication gap between developers and leadership, an empathy gap. Lived experience for many engineers qualitatively confirms these data: requests to allocate resources to DevEx-related work are often rejected in favor of feature delivery. This reinforces the belief that only visible product work is valued, while internal improvements, those that enable and streamline this work, are often overlooked.

One fundamental driver of this gap is the absence of a shared language for discussing DevEx problems. In practice, discovering the term 'Developer Experience' can be a turning point, providing engineers with a vocabulary to articulate the value of optimization initiatives. Without this language, improvement efforts are perceived as minor tweaks rather than strategic investments. Leadership, for its part, sees the outcome (e.g., time saved by AI). Still, not the initial conditions (developer experience), leading to a misdiagnosis: the saved time is credited without removing the extant points of friction [1]. Hence, a causal chain emerges: inarticulate pain to lack of shared language to misrecognition of root causes to misaligned priorities (features vs. friction) to widening empathy gap to economic losses. The SPARK methodology proposed herein is designed to create that shared language.

While centralized, top-down DevEx initiatives, such as Platform Engineering, are valuable, they demand significant investment and organizational sponsorship. A complementary, minimal, bottom-up approach is needed to catalyze change and supply evidence for justifying larger efforts.

This article introduces SPARK as a formalization of bottom-up DevEx. SPARK is a structured, repeatable method that enables any developer to convert individual pain points into a compelling, data-backed business case. This article addresses the question: how can individual developers systematically convert day-to-day friction into business-relevant evidence that improves DevEx?

Materials and Methodology

This study draws on academic literature, industry reports, and applied cases. Three strands form its theoretical background. First is the Atlassian State of Developer Experience Report 2025, which has quantified production losses due to friction. It described a productivity paradox in which internal organizational failures offset the gains from AI tools [1]. Second, research work on production metrics, such as DORA and SPACE, has shown that developer productivity can be operationalized through a balance of delivery speed and human factors [4]. The third includes cognitive psychology and sociotechnical system work, wherein cognitive load, psychological safety, and organizational learning are emphasized [5].

Methodologically, the study relied on three interlinked approaches. The first was a systematic literature review spanning publications on the cognitive cost of interruptions [5], developers' adaptation strategies [7], and productivity measurement methodologies [4]. This established a theoretical frame in which DevEx is treated as a sociotechnical construct rather than merely a set of technical metrics. The second was a comparative analysis of industry reports, including those from [1] and the DORA guidelines [3], which linked local developer initiatives to widely recognized business indicators. The third was a content analysis of applied cases, reconstructed as analytical vignettes: from slow CI/CD builds to onboarding in monorepositories.

A targeted search was conducted across Google Scholar and major computing venues' indices (ACM Digital Library and IEEE Xplore), complemented by arXiv for onboarding-related case studies, covering the period 2008–2025. Search strings combined terms such as "developer experience/DevEx", "developer productivity", "DORA metrics", "SPACE framework", "cognitive load", "interruptions", "flow", "psychological safety", "onboarding" and "value stream mapping". The initial search yielded ~140 records; after title/abstract screening and removal of duplicates, ~55 sources were reviewed in full text. 9 sources were retained that either

(a) operationalize delivery performance and productivity measurement (e.g., DORA/SPACE);

(b) provide empirical evidence on cognitive cost, interruptions, or sociotechnical factors in engineering work;

(c) document applied improvement practices relevant to CI/CD and onboarding.

These sources were coded into recurring themes (feedback loops, cognitive load/flow, measurement and governance, sociotechnical enablers), which informed the five-stage SPARK cycle. Industry guidance and reporting were then compared with the academic findings to ensure that SPARK outputs can be communicated in terms that are legible to both engineering and business stakeholders.

Results and Discussion

The industry standard for measuring software delivery performance comprises the DORA (DevOps Research and Assessment) metrics. These four key metrics, Deployment Frequency, Lead Time for Changes, Change Failure Rate, and Mean Time to Recovery, focus on system-level outcomes, providing an objective readout of a team's delivery capability. Table 1 exhibits the classic DORA differentiation: Elite teams achieve very frequent deploys, minimal lead time, and rapid recovery with a very low percentage of failed changes, whereas other levels lag substantially in both speed and reliability [2].

Table 1 - Software Delivery Performance Metrics (DORA)

DOI: https://doi.org/10.60797/itech.2026.11.10.1

Software Delivery Performance Metric	Elite	High	Medium	Low
Deployment Frequency	On-demand (multiple deploys per day)	Between once per week and once per month	Between once per month and once every 6 months	Fewer than once per 6 months
Lead Time for Changes	Less than one hour	Between one day and one week	Between one month and six months	More than six months
Time to Restore Service	Less than one hour	Less than one day	Between one day and six months	More than six months
Change Failure Rate	0%-15%	16%-30%	16%-30%	16%-30%

Note: source [2]

The key inference is that speed and stability are not mutually exclusive; elite teams excel along both axes [3]. This is decisive, as it refutes a common managerial concern that investing in internal improvements (which increases speed) could jeopardize system stability.

It is essential to note that DORA metrics are lagging indicators, as they reveal what is happening (e.g., high lead time) but not why. This limitation creates a need for a methodology that can diagnose the root causes of poor DORA performance.

A more nuanced and multidimensional model is offered by the SPACE framework [4], which decomposes productivity into five dimensions: Satisfaction & Well-being, Performance, Activity, Communication & Collaboration, and Efficiency & Flow.

SPACE's essential contribution is the explicit inclusion of human factors. It recognizes that productivity is not only product output but also developer satisfaction, cognitive load, and the capacity to achieve flow [4]. This directly bridges academic theory and the practical notion of developer pain. The authors intentionally debunk prevalent myths, such as the notion that productivity equals developer activity or that a single metric can tell us everything, thereby providing a theoretical basis for rejecting simplistic and often harmful measurement practices.

When utilized conjointly, DORA and SPACE constitute an effective diagnostic framework. DORA identifies the issue (what): elevated lead time for modifications. SPACE facilitates the identification of underlying causes (why): reduced efficiency and flow as a result of recurrent interruptions, accompanied by diminished stakeholder satisfaction. Yet neither framework prescribes a structured, repeatable process for an individual developer to act upon this information from the bottom up. There is a clear gap for a lightweight, action-oriented method that supplies data to these measurement systems. SPARK is designed to fill that gap by operationalizing an improvement cycle whose outcomes can then be measured via DORA and SPACE.

The SPARK methodology formalizes concepts from practice into a five-stage iterative cycle, shown in Figure 1.

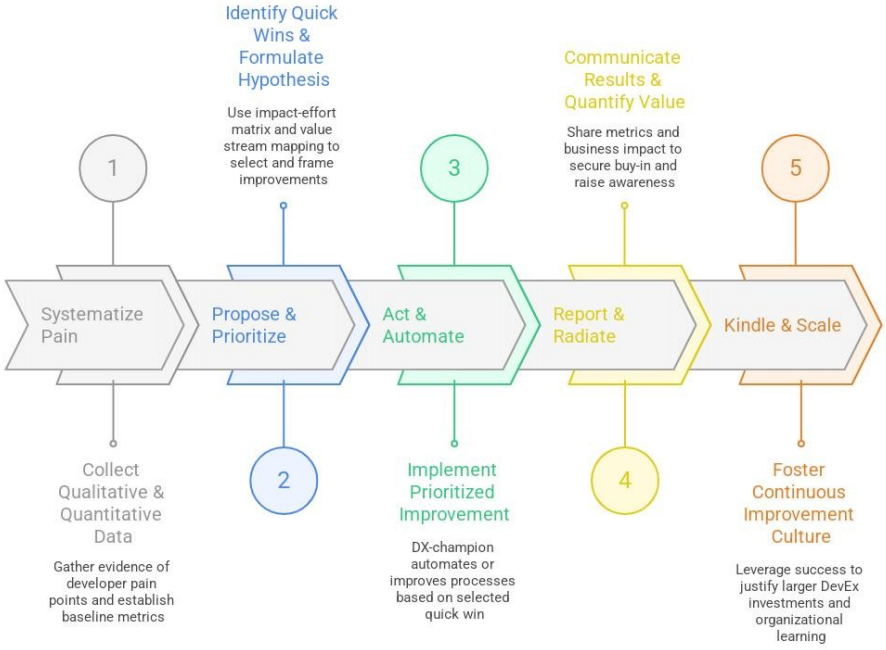

Figure 1 - The SPARK Methodology for DevEx Improvement

DOI: https://doi.org/10.60797/itech.2026.11.10.2

Note: compiled by author

It offers a structured pathway to transform informal observations into a governed improvement process. Each stage is detailed below.

The first stage, Systematize Pain, aims to transition from scattered, anecdotal complaints, such as 'builds are slow,' to structured qualitative and quantitative data. This creates a foundation for a disciplined improvement of internal development processes. Developers are encouraged to keep a friction log, systematically recording interruptions, delays, and other frustrating factors.

Both qualitative and quantitative methods are employed. Qualitative data arise from informal interviews or surveys among peers to surface shared pain points. These conversations focus on three salient aspects of developer experience: feedback cycles, cognitive load, and flow. In parallel, a quantitative baseline is established to measure pain in concrete units, such as timing builds, counting context switches, or measuring the time spent searching for information. Such baselines are essential for objective evaluation of future improvements.

This approach is theoretically grounded in Gloria Mark's research on the psychological cost of interrupted work [5], which demonstrates that persistent interruptions increase stress, frustration, and time pressure, even if overall task duration ultimately decreases. Systematizing pain, therefore, becomes the necessary first step toward mitigating cognitive costs and creating a more productive, humane work environment.

Propose and Prioritize seek the minimum viable improvement, which is a change that has high potential impact with relatively low implementation costs. Technically, getting this quick win secures the momentum for further transformation.

Several methods can be used to guide the selection. A key instrument among these is the Impact–Effort matrix, as it can visually separate quick wins from other tasks, specifically those being high-impact, low-effort tasks. Rational prioritization beyond the loudest problems then becomes possible. For more complex workflows, Value Stream Mapping (VSM) is applied to visualize the end-to-end process, reveal bottlenecks, wait states, and non-value-adding activities, fully aligned with lean principles [6].

Each proposed improvement is cast as a clear, testable hypothesis. For example, if we automate the local environment setup script, the time it takes for a new developer to produce their first commit will decrease by X%. This shift changes from ad hoc tinkering to empirical validation.

The Act and Automate stage implements the prioritized improvement. The emphasis is on action over protracted planning, reflecting the agile, bottom-up character of the method. Typically, a developer acting as a DX champion writes the script, improves documentation, or automates a recurring team task.

Illustrative improvements vary. For onboarding, a one-command bootstrap script can standardize and simplify local environment setup, a common pain in monorepos and complex systems. For CI/CD, build caching or selective test execution strategies can reduce build times and accelerate feedback loops. Documentation can be improved via a centralized, searchable knowledge base, directly addressing the chronic problem of inefficient information retrieval.

Next comes Report and Radiate. The goal is to measure the realized impact and communicate it to business leadership and engineering management in intelligible terms, translating technical achievements into business outcomes. This begins by comparing new measurements with the established baseline, e.g., Build time decreased from 20 to 5 minutes.

To increase persuasiveness, results are framed within established industry models such as DORA and SPACE. For example: A 75% reduction in build time directly improves our lead time for changes (a key DORA metric) while elevating efficiency and developer flow (SPACE). Likewise, onboarding automation can be framed as: Our new script reduces the time to first commit for newcomers from 3 days to 2 hours, positively affecting newcomer satisfaction and well-being (SPACE).

A pivotal element is quantifying business value by converting time savings into financial terms. A calculation might read: Saving 15 minutes per day on builds for each of 50 developers frees 62.5 hours per week. This equates to more than one and a half FTEs or an annual saving of $X. Such framing provides a direct business justification. To consolidate success and win support for future initiatives, broad Radiate, through demos, team meetings, and internal blogs, is essential.

The final stage is Kindle and Scale. Here, the authority earned from a quick win is leveraged to nurture a culture of continuous improvement. The initial success evidences value and supports larger investment in DevEx. Each completed cycle builds trust and makes resource acquisition for subsequent cycles easier, thereby seeding a virtuous loop that is emblematic of continuous improvement.

As data accumulate across cycles, an individual champion's initiative can evolve into more. These data become a powerful argument for formalizing DevEx improvement efforts, potentially culminating in a dedicated team such as Platform Engineering. Thus, evidence generated by bottom-up, local initiatives furnishes a robust business case for strategic, top-down investment decisions.

This process can be viewed through the lens of double-loop organizational learning. Single-loop learning fixes the immediate problem, e.g., speeding up a slow build (the how). Double-loop learning asks why the builds were sluggish in the first place, prompting a reassessment of processes or resource allocation principles. This shifts organizations from reactive patching to proactive, systemic enhancement of engineering culture.

Consider examples illustrating SPARK. As a first case, depicted in Figure 2, take a team facing long CI/CD builds:

– S (Systematize Pain): the team observes CI builds often exceed 30 minutes, forcing context switches and degrading concentration. They record build times for a week, establishing a median wait of 32 minutes. This directly impairs developer efficiency and flow (SPACE);

– P (Propose and Prioritize): using the impact–effort matrix, they identify dependency caching as a low-effort, high-impact remedy. The hypothesis is that there will be a ≥50% reduction in build time;

– A (Act and Automate): the team's DX champion spends half a day adjusting the CI/CD pipeline to cache dependencies in cloud storage;

– R (Report and Radiate): after a week, the new median build time is 8 minutes, a 75% reduction. The champion briefs leadership: We improved our lead time for changes (DORA) by an average of 24 minutes per commit. This saves ~2 hours per developer per week, freeing $X in productivity and enabling more frequent deployments;

– K (Kindle and Scale): success induces other teams to adopt the caching strategy. Aggregated wins then justify broader investment in a more capable CI/CD platform.

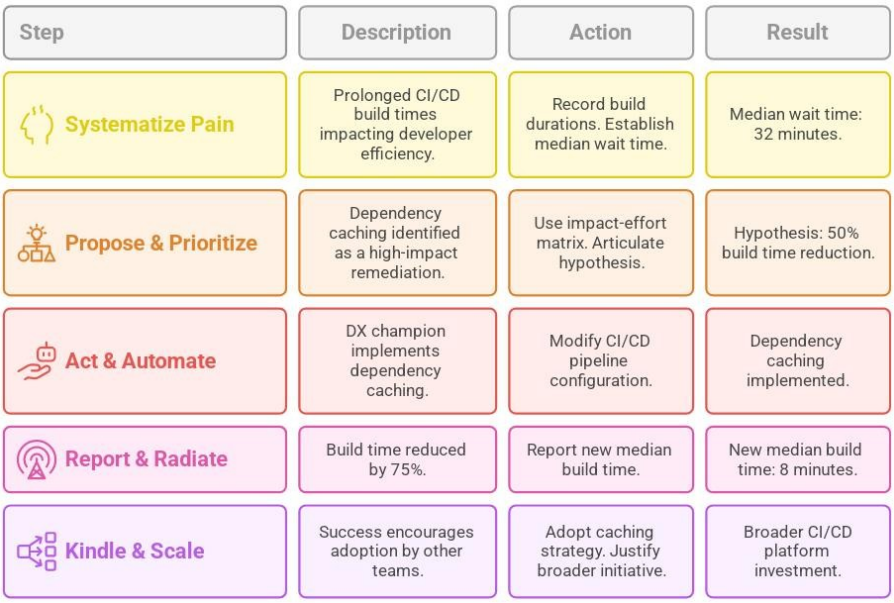

Figure 2 - SPARK Framework Applied to CI/CD Build Times
DOI: https://doi.org/10.60797/itech.2026.11.10.3

Note: compiled by author

As a second example, consider improving onboarding in a monorepository:

– S (Systematize Pain): new hires report that achieving a successful local build takes over a week due to complex setup and outdated documentation, a common monorepo problem. This triggers strong dissatisfaction and harms well-being [7];

– P (Propose and Prioritize): the team opts to create a unified automated setup script, targeting a time to first build of under one hour;

– A (Act and Automate): two developers pair-program a one-command bootstrap in a day to install dependencies, configure the environment, and run initial tests. They update the root README to make this script the official entry point;

– R (Report and Radiate): the next newcomer builds and runs tests in 45 minutes. The team reports that their new onboarding script has cut preparation time by 95%, improving the newcomer experience and accelerating time-to-productivity. This directly enhances our ability to scale the team effectively;

– K (Kindle and Scale): the script becomes the golden path and a core component of the internal developer platform. This success underwrites dedicating more time to other paved roads for everyday developer tasks.

Accordingly, SPARK is not merely a tool for technical process improvement; it is an intervention in a sociotechnical system [8]. The methodology acknowledges deep couplings between people (the social system) and their tools, processes, and workflows (the technical system). The Systematize Pain, Report and Radiate, and Kindle and Scale stages are intrinsically social acts that reshape communication patterns and build a shared understanding, key to sociotechnical change.

Launching the SPARK cycle requires a baseline of psychological safety. Developers must feel secure enough to express dissatisfaction and report issues without fear of being labeled complainers or underperformers.

In turn, successful SPARK execution can raise psychological safety. When leadership responds positively to a data-backed improvement proposal, it signals that such contributions are valued, encouraging further bottom-up innovation [9]. A positive feedback loop emerges: initiating SPARK requires some psychological safety; completing the cycle, especially Report and Radiate, where work is recognized, reinforces confidence that such risks are safe and valued. Managerial endorsement then strengthens psychological safety across the team, making it more likely that others will initiate their own SPARK cycles. Thus, a self-sustaining mechanism of continuous improvement takes shape.

This dynamic also situates SPARK within the broader continuous-improvement tradition associated with PDCA and Kaizen. Similar to PDCA, SPARK follows an iterative logic: identify a problem, implement a bounded intervention, assess outcomes, and consolidate learning for subsequent cycles. Recent literature has further shown that contemporary PDCA-based approaches place increasing emphasis on structured measurement, sustainment mechanisms, and the disciplined embedding of improvement routines, rather than treating improvement as a one-off corrective action [10], [11].

At the same time, SPARK is closer to Kaizen in its bottom-up and participatory character. Recent studies of Kaizen and lean implementation stress that local, employee-driven improvement depends on repeated participatory routines, knowledge-

sharing processes, and the ability of frontline actors to surface problems and propose changes [12], [13]. In this respect, SPARK should not be read as an alternative to PDCA or Kaizen, but as a DevEx-specific adaptation of the continuous-improvement tradition. Its distinctive contribution lies in making developer friction explicitly measurable and communicable in terms legible to engineering leadership through DORA- and SPACE-aligned evidence. Thus, SPARK combines the disciplined cyclical logic of PDCA with the incremental, bottom-up ethos of Kaizen, while specializing both for the sociotechnical realities of software development work [10], [11], [12], [13].

This positioning also clarifies SPARK's relationship to larger organizational DevEx programs. Research on lean implementation suggests that bottom-up improvement activities are valuable for generating local learning and momentum, but that durable, organization-wide gains depend on managerial sponsorship and cross-level coordination [13], [14]. Accordingly, SPARK is best understood not as a substitute for Platform Engineering or other top-down DevEx investments, but as an evidence-generating precursor and complement to them: it helps surface validated local bottlenecks, demonstrate their business relevance, and create a stronger basis for subsequent platform-level decisions [11], [14].

SPARK should be situated within the broader landscape of DevEx strategies, particularly in contrast to Platform Engineering's top-down approach. Platform Engineering aims to reduce cognitive load and improve DevEx by providing a centralized, self-service Internal Developer Platform (IDP).

SPARK is not an alternative to Platform Engineering, but its precursor and complement. The data and small-scale wins generated by multiple independent SPARK cycles provide the concrete business justification required to warrant significant investment in a dedicated platform team. The paved roads built by the platform team ought to be grounded in the paths first charted by SPARK's bottom-up initiatives.

The present work is presented as a method proposal, supported by a synthesis of prior research and illustrative scenarios, intended to clarify its application rather than serve as an empirical evaluation. As such, the quantitative effects discussed in the examples should be interpreted as context-dependent and sensitive to organizational factors, such as team topology, codebase scale, tooling maturity, and baseline psychological safety. Future validation would benefit from a longitudinal, multi-team study that applies SPARK across several cycles, with pre-registered success criteria and consistent measurement using a small, stable set of DORA- and SPACE-aligned indicators, complemented by qualitative feedback. Comparative evaluation against a baseline (or matched teams not using SPARK) and reporting of adoption costs, maintenance overhead, and sustainability of improvements would further strengthen external validity and clarify where SPARK is most effective.

Conclusion

This article presents SPARK — a novel, minimal, and repeatable methodology that empowers developers to drive bottom-up DevEx improvements. It demonstrated how SPARK furnishes a key mechanism for translating qualitative developer pain into quantitative data aligned with established measurement systems (DORA, SPACE) and resonant with business stakeholders. SPARK should be viewed as a sociotechnical catalyst that enables a culture of continuous improvement while providing the empirical foundation for strategic, top-down investments in developer productivity. The examples that have been presented were illustrative syntheses grounded in industry reports and academic literature. A suggested next step would be an actual longitudinal empirical study of organizations that formally adopt SPARK, allowing its actual impact over time to be measured. Further research is needed into the anti-patterns of bottom-up DevEx initiatives that SPARK's structured character can mitigate, and how its emphasis on measurement and communication prevents cowboy coding. A valuable next step would be to investigate specific leadership and cultural preconditions that enable SPARK to thrive, building on discussions of psychological safety and organizational learning.

Конфликт интересов

Не указан.

Conflict of Interest

None declared.

Рецензия

Все статьи проходят рецензирование. Но рецензент или автор статьи предпочли не публиковать рецензию к этой статье в открытом доступе. Рецензия может быть предоставлена компетентным органам по запросу.

Review

All articles are peer-reviewed. But the reviewer or the author of the article chose not to publish a review of this article in the public domain. The review can be provided to the competent authorities upon request.

Список литературы на английском языке / References in English

1. The State of Developer Experience in 2025 // Atlassian. — 2025. — URL: https://dam-cdn.atl.orangelogic.com/AssetLink/5yt05dl5q8s1xljrs8747h8x6240c32p.pdf (accessed: 01.01.2026).

2. Portman D.G. Using the Four Keys to measure your DevOps performance / D.G Portman // Google Cloud. — 2020. — URL: https://cloud.google.com/blog/products/devops-sre/using-the-four-keys-to-measure-your-devops-performance (accessed: 02.01.2026).

3. Harvey N.D. DORA's software delivery metrics: the four keys / N.D. Harvey // DORA. — 2025. — URL: https://dora.dev/guides/dora-metrics-four-keys/ (accessed: 03.01.2026).

4. Forsgren N. The SPACE of Developer Productivity / N. Forsgren, M.-A. Storey, C. Maddila [et al.] // Queue. — 2021. — Vol. 19, № 1. — P. 20–48. — DOI: 10.1145/3454122.3454124.

5. Mark G. The cost of interrupted work / G. Mark, D. Gudith, U. Klocke // Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI '08. — 2008. — P. 107–110. — DOI: 10.1145/1357054.1357072.

6. Jeong B.K. A Software Development Life Cycle Case Study Of Lean IT With Value Stream Mapping Analysis / B.K. Jeong, S.-Y. Ji, D.H. Jeong // Information Resources Management Journal. — 2022. — Vol. 35, № 1. — P. 1–18. — DOI: 10.4018/irmj.291527.

7. Ju A. A Case Study of Onboarding in Software Teams: Tasks and Strategies / A. Ju, H. Sajnani, S. Kelly [et al.] // Arxiv. — 2021. — DOI: 10.48550/arxiv.2103.05055.

8. Polojarvi D. A systematic literature review of sociotechnical systems in systems engineering / D. Polojarvi, E. Palmer, C. Dunford // Systems Engineering. — 2023. — Vol. 26, № 4. — DOI: 10.1002/sys.21664.

9. Zhang W. Understanding how organizational culture affects innovation performance: A management context perspective / W. Zhang, X. Zeng, H. Liang [et al.] // Sustainability. — 2023. — Vol. 15, № 8. — DOI: 10.3390/su15086644.

10. Pecas P. PDCA 4.0: A New Conceptual Approach for Continuous Improvement in the Industry 4.0 Paradigm / P. Pecas, J. Encarnacao, M. Gamboa [et al.] // Applied Sciences. — 2021. — Vol. 11, № 16. — P. 7671. — DOI: 10.3390/app11167671.

11. Naughton E. A structured model for continuous improvement methodology deployment and sustainment: A case study / E. Naughton, R. Moran, M. Kharub [et al.] // Heliyon. — 2024. — Vol. 10, № 21. — P. e40034. — DOI: 10.1016/j.heliyon.2024.e40034.

12. Jones O.W. Development of a Kaizen series model: abducting a blend of participatory formats to enhance the development of process improvement practices / O.W. Jones, J. Gold, J. Claxton // Total Quality Management & Business Excellence. — 2021. — Vol. 33, № 7–8. — P. 1–27. — DOI: 10.1080/14783363.2021.1911633.

13. van Beers J.C.A.M. Effective hospital-wide lean implementation: top-down, bottom-up or through co-creative role modeling? / J.C.A.M. van Beers, D.H. van Dun, C.P.M. Wilderom // International Journal of Lean Six Sigma. — 2021. — Vol. 13, № 1. — P. 46–66. — DOI: 10.1108/ijlss-02-2021-0024.

14. Vinodh S. Integration of continuous improvement strategies with Industry 4.0: a systematic review and agenda for further research / S. Vinodh, J. Antony, R. Agrawal [et al.] // The TQM Journal. — 2020. — DOI: 10.1108/tqm-07-2020-0157.