Towards a Science of Developer eXperience (DevX)

Benoit Combemale*
*Inria & University of Rennes, France

ABSTRACT As software continues to permeate nearly every facet of modern life, the complexity and ubiquity of digital services underscore the need for sustainable, effective, and inclusive software development practices. Although software engineering has made significant progress in technical challenges since its inception, the human experience of those involved in software creation, broadly defined as developers, remains underexplored. This column advocates for the formal recognition of Developer eXperience (DevX) as a distinct research field. We argue that DevX profoundly influences critical development activities and overall productivity, especially as development becomes increasingly collaborative and diverse in terms of application domains. Building on existing efforts to measure and enhance DevX, we identify key rationales, scientific enablers, and interdisciplinary intersections that support this emerging discipline. We also outline the core scientific challenges ahead, aiming to call for actions from the research community and to promote more human-centered approaches to software engineering.

KEYWORDS Software Engineering, Integrated Development Environment, Software Development Life-cycle, User Experience

1. Introduction

In a world increasingly shaped by software, we are witnessing not only the proliferation of software products but the digital transformation of nearly every aspect of our professional and personal life. This transformation is marked by pervasive, adaptive, contextualized, and personalized digital services that have become integral to modern society. While software engineering as a discipline has matured significantly since its formal recognition at the 1968 NATO conference, the field has largely concentrated on technical challenges across the software development lifecycle. In doing so, it has often overlooked a critical dimension: the experience of developers themselves (Fagerholm & Münch 2012; Forsgren et al. 2024). We use the term *developer* broadly, to encompass not only software professionals typically identified as software engineers, but also scientists, domain experts, and even citizens (e.g., end-user programming (Barricelli et al. 2019)) who interact with software development artefacts. Their experience impacts not only the engagement in key development activities such as design (Palomino et al. 2025) and test (Parry et al. 2022), but also the overall productivity (Razzaq et al. 2024; Noda et al. 2023).

Today, as the demand for high-quality, complex software systems expands across diverse domains and involves very heterogeneous stakeholders, there is an urgent need to focus on the human aspects of software creation. While previous research has explored ways to measure (D'Angelo et al. 2024), understand, and improve (Greiler et al. 2022) the experience of developers, a systematic articulation of the underlying rationales and core challenges remains lacking. This column aims to discuss the motivations for recognizing *Developer eXperience* (DevX) as a research field in its own right. We outline the foundational rationales for this emerging discipline, explore its intersections with existing scientific domains and key enablers, and discuss the scientific challenges. Our goal is to prompt the scientific community to acknowledge and address this vital dimension of software engineering, ultimately fostering more sustainable, effective, and inclusive software development practices.

2. Motivations

While addressing the complexity of software systems and meeting the required quality assurance have been the motivations for software engineering, it is important to review the current and new motivations that lead to a better consideration of the experience of engaged developers. In the following, we discuss

JOT reference format:
Benoit Combemale. *Towards a Science of Developer eXperience (DevX)*. Journal of Object Technology. Vol. 24, No. 1, 2025. Licensed under Attribution - NonCommercial - No Derivatives 4.0 International (CC BY-NC-ND 4.0) http://dx.doi.org/10.5381/jot.2025.24.1.a2

the new dimensions of the complexity of software systems, as well as the diversity of developers and the limits of the anthropic principle in the software development lifecycle.

2.1. System Complexity

Modern software systems are more intricate than ever before, in the form of socio-technical eco-systems (Bencomo et al. 2024). They are characterized by distributed architectures, microservices, machine learning models, and large-scale integrations. Developers must reason about layers of abstractions, manage dependencies, and debug interactions across components that are often opaque or poorly documented. This complexity poses significant cognitive challenges, leading to inefficiencies, errors, and frustration.

To succeed with the development of modern software, organizations must have the agility to adapt faster to constantly evolving environments to deliver more reliable and optimized solutions that can be adapted to the needs and environments of their stakeholders including users, customers, business, development, and IT. However, stakeholders are missing tool support for global decision-making, considering the increasing variability of the solution space, the frequent lack of explicit representation of its associated variability and decision points, and the uncertainty of the impact of decisions on stakeholders and the solution space (Kienzle et al. 2022). This leads to an ad-hoc decision making process that is slow, error-prone, and often favors local knowledge over global, organization-wide objectives. It urges to provide automation and tool support in aid of a multi-criteria decision making process involving different stakeholders within a feedback-driven software development process where feedback cycles aim to reduce uncertainty.

While software is revolutionizing the modern world, software systems evolve under frequently changing environments, and are expected to handle ever-increasing uncertainty. This blurs not only the line between engineering-time and execution-time (Baresi & Ghezzi 2010), but also between software and the real world as both are fusing into a single fabric. These dynamics require accelerated levels of adaptability—indeed, a *temporal* adaptability, i.e., the ability to adapt not only to a fixed space of variable requirements, but also to an emerging chain of changing requirements, often driven by incoming input data.

2.2. Developer Diversity

The developer community is no longer limited to traditional software engineers. Scientists, domain experts, and even non-programmers—often referred to as "citizen developers"—increasingly engage in programming to advance their fields or solve domain-specific problems. This heterogeneity creates challenges in designing tools and processes that cater to diverse skill levels, mental models, and goals. For example, while experienced software engineers may seek extensibility and fine-grained control, citizen developers prioritize simplicity and intuitive interfaces (Bucaioni et al. 2022) with possibly visual programming (e.g., no/low-code development (Luo et al. 2021)), domain experts expect to manipulate relevant abstractions of the domain to bridge the gap between the problem space and the solution space (Madni & Sievers 2018), and scientists relies on complex processes and various software languages to move from continuous mathematical models to efficient implementation on HPC infrastructures (Leroy et al. 2021).

2.3. Complexity of Software Development Practices

Modern software development has evolved into a highly intricate discipline characterized by a multiplication of processes, tools, and responsibilities. Far from being limited to coding alone, contemporary development practices encompass various tasks, including requirement engineering, architectural design, implementation, testing, deployment, infrastructure management, monitoring, and continuous maintenance and evolution. Each of these dimensions introduces its own set of artifacts and tools, leading to increasing cognitive and operational overhead for developers.

A major contributor to this complexity is the proliferation of development artifacts—ranging from source code and documentation to configuration files, test scripts, CI/CD pipelines, and telemetry data. These artifacts are often managed across heterogeneous tools such as version control systems, issue trackers, integrated development environments, build servers, and cloud platforms. The resulting fragmentation demands frequent context switching, which can negatively impact developer productivity and mental well-being (Forsgren et al. 2021; Meyer et al. 2014).

Moreover, developers are now expected to assume responsibilities traditionally handled by specialized roles. Practices like DevOps and Site Reliability Engineering blur the lines between development, and operations. This convergence, while promoting agility and collaboration, also necessitates continuous learning and adaptation to new frameworks, paradigms, and toolchains.

The pace of technological change further exacerbates the complexity. The rapid evolution of programming languages, libraries, cloud-native technologies, and architectural styles (e.g., microservices, serverless computing) demands that developers not only build software but also stay perpetually current with trends and best practices.

Hence, the complexity of software development today arises not just from the systems being built but also from the ecosystem in which they are developed, thus degrading the developer experience.

3. Rationales for DevX

In an era where software systems underpin critical infrastructures and drive global innovation, the experience of developers —the creators of these systems— has become a central concern. DevX transcends traditional software engineering metrics by emphasizing the holistic impact of tools, workflows, and environments on developers' creativity, engagement, and overall effectiveness (Forsgren et al. 2024). DevX is not merely a productivity enhancer; it is an enabler of accessible, pleasurable, and accountable development practices that align with the growing complexity and diversity of modern software ecosystems. In the following, we review the rationales of DevX from the point of view of software engineering.

3.1. Fostering Creativity and Engagement

Creativity is a cornerstone of effective software development. Developers constantly devise novel solutions to technical challenges, a process that thrives in environments fostering engagement and flow. Poorly designed tools, fragmented workflows, and opaque systems disrupt this creative process, eroding engagement and increasing cognitive load. DevX research highlights the importance of affordances—features that intuitively guide developers toward productive interactions. By enabling seamless exploration, experimentation, and debugging, a strong DevX cultivates environments where developers can fully leverage their creative potential.

3.2. Enhancing Accessibility and Confidence

The democratization of software development has brought diverse actors into the field, including scientists, domain experts, and citizen developers. Accessibility is a critical aspect of DevX, ensuring that tools and processes cater to varying levels of expertise and technical backgrounds. Equally vital is fostering confidence, as intuitive and transparent systems help developers trust their tools and outputs. By prioritizing accessibility and confidence, DevX bridges the gap between novice and expert developers, empowering all to contribute effectively.

3.3. Prioritizing Social Translucence and Accountability

Modern software development is inherently collaborative, often involving distributed teams working across different time zones and roles. Social translucence—the ability to make others' actions and intentions visible in a respectful and comprehensible way—is a key attribute of effective development environments. Tools like forges and practices like DevOps promote accountability by offering clear records of contributions while facilitating open communication and feedback. DevX research emphasizes the role of such systems in building trust, reducing misunderstandings, and ensuring that collaborative processes are as seamless as individual workflows.

3.4. Balancing Pleasure with Productivity and Efficiency

A compelling DevX is not merely functional; it is also pleasurable. Developers derive satisfaction from smooth workflows, elegant tools, and systems that anticipate their needs. This pleasure, in turn, feeds motivation and engagement, fostering sustainable productivity. For instance, tools that offer visual feedback, gamified elements, or aesthetic design can transform routine tasks into enjoyable experiences. However, pleasure must align with productivity and efficiency. Tools should minimize repetitive or redundant work and allow developers to achieve their goals swiftly without sacrificing the joy of problem-solving. Striking this balance is a central challenge in DevX research.

3.5. Addressing Overwhelming Complexity and Artifact Management

As software systems grow in complexity, developers must navigate a large set of software development artefacts—source code, documentation, tests, logs, configurations—across numerous tools and platforms. This fragmentation often leads to inefficiencies, as developers are forced to context-switch or manually integrate disparate workflows. DevX research seeks to streamline these processes, offering integrated environments that provide developers with a clear mental model of their work and reduce friction. By improving efficiency and productivity, such environments enable developers to spend more time on creative and high-impact tasks.

4. Key Enablers and Scientific Challenges

The experience of software developers is a multifaceted concern that extends beyond the technical concerns related to artefacts, tools, and methodologies. It also encompasses human and social dimensions, thereby demanding a comprehensive perspective. In this section, we identify key enablers and discuss the remaining open scientific challenges, with attention to both technical and social aspects.

4.1. Technical Dimension

From a technical standpoint, enhancing the developer experience hinges on the evolution of programming environments, languages, and tools. One fundamental concern is program comprehension. As systems grow in complexity, understanding their structure and behavior becomes increasingly difficult. This necessitates improved static and dynamic analysis techniques, better visualization methods, and deeper insights into the cognitive processes involved in reading and modifying code.

The interfaces through which developers interact with code also play a critical role. Programming notebooks, such as Jupyter, support exploratory and data-centric workflows, yet still present usability and reproducibility challenges. Similarly, diverse programming styles—ranging from live and exploratory programming to literate programming—demand flexible and responsive environments that support rapid feedback, inline documentation, and iterative development. Methods to facilitate the understanding of the overall behavior—such as example-based programming (Niephaus et al. 2020), omniscient debugging (Pothier & Tanter 2009) and moldable development (Nierstrasz & Gîrba 2024)—offer promising avenues but require further refinement and integration with the various programming styles and interfaces.

Another important area involves the design and implementation of program representations that are both abstract and manipulable. These representations should enable intuitive interaction, particularly for end-users. Visual programming environments (incl., block-based programming environment (Weintrop & Wilensky 2017)) attempt to address this by accelerating the learning curve, though issues related to scalability, expressiveness, and debugging remain open.

To better support a wide range of users and tasks, programming languages and environments are increasingly incorporating mixed notations that combine text, graphics, and symbolic elements. Creating coherent semantics and robust tooling around these hybrid notations presents a significant research challenge. Closely related is the field of abstraction engineering, which focuses on developing, managing, and evolving meaningful abstractions.

As software development processes shift toward continuous practices, such as continuous integration and deployment, new demands are placed on tools and infrastructures. Environments must support real-time feedback and ongoing changes while maintaining stability. This, in turn, drives the need for more adaptable input and output modalities—including touch, voice, and gesture-based interactions—that can make programming more accessible and intuitive.

Finally, there remains a strong need for infrastructures, meta-tools, and frameworks that streamline the creation and extension of programming environments. These tools are essential not only for researchers and educators but also for practitioners who wish to customize or prototype new workflows.

4.2. Social Dimension

Beyond the technical realm, the social dimension of developer experience introduces a range of human-centric and interdisciplinary challenges. Addressing these effectively requires the engagement of researchers from the social sciences and humanities (SHS), alongside computer scientists and software engineers.

At the individual level, it is important to consider how concepts such as affordance, cognitive load, motivation, and emotional engagement influence the way users interact with programming environments. Tools should be designed with these psychological and ergonomic factors in mind, to foster inclusive and empowering experiences for diverse populations.

At the collective level, social dynamics such as collaboration, communication, and shared understanding come to the forefront (Booch & Brown 2003). Concepts like social translucence—the visibility of others' activities and intentions—are vital for supporting teamwork and community practices. Designing programming environments that promote awareness, coordination, and mutual support among users remains an open challenge.

Research communities such as Human–Computer Interaction (HCI)—including Computer-Supported Cooperative Work (CSCW)—and Visual Languages and Human-Centric Computing (VL/HCC) have developed longstanding bodies of work on issues highly relevant to understanding and improving DevX. HCI and CSCW provide rich insights into collaboration, coordination, and socio-technical aspects of software development (Grudin 1994; Herbsleb & Mockus 2003; Stol & Fitzgerald 2018), while VL/HCC has focused on making programming more accessible and comprehensible through visual notations, end-user programming, and human-centered design of tools (Burnett & Myers 2014; Cao et al. 2013). Leveraging this prior work can provide theoretical foundations and practical strategies for advancing DevX research and practice.

In summary, the development of future programming environments must be informed by both technical innovation and human-centered design. Progress in this area will depend on a sustained commitment to cross-disciplinary research and on tools that reflect the complexity of both the artefacts and the communities that produce and maintain them.

5. Conclusion

DevX is a multidimensional construct that encompasses creativity, engagement, accessibility, confidence, and collaboration. It emphasizes affordance, social translucence, accountability, and the balance between pleasure, productivity, and efficiency. As software systems grow increasingly complex and developer populations become more diverse, systematically investing in DevX is essential to empower developers and to ensure the sustainable evolution of software engineering practices.

To foster such progress, we conceptualize DevX along measurable dimensions such as creativity, emotional well-being, flow, and collaboration efficiency. Each dimension can be studied through complementary methods, including surveys, telemetry, and cognitive load analysis. While these dimensions are not exhaustive, they provide a concrete starting point for future empirical research and a shared vocabulary for the community.

A critical challenge, however, lies in the fragmented nature of existing research. Relevant insights originate from diverse subdomains—cognitive psychology (e.g., cognitive load, flow), organizational science (e.g., collaboration models, team structures), and usability research (e.g., developer tools and interfaces)—yet they often remain siloed. DevX offers a unifying perspective by articulating their interdependencies: for instance, an unsuitable organizational model may elevate cognitive load and diminish well-being, while poor tool usability can disrupt flow and undermine collaboration efficiency.

This column seeks to establish DevX as a recognized research field within software engineering and to call the community to action. The immediate next step is to consolidate foundational concepts, establish measurable dimensions, develop benchmarks for empirical studies, and provide a comprehensive research roadmap that integrates the aforementioned dimensions and subdomains, ultimately advancing both the science and practice of software engineering. In the medium term, validated instruments and integrated tools can emerge, enabling evidence-based improvements in practice and industry adoption. In the longer term, a mature science of DevX could establish dedicated venues and curricula, deliver holistic tool ecosystems, and demonstrate tangible impacts on innovation, sustainability, and inclusivity in software development. Such a trajectory illustrates how DevX can evolve from a promising perspective into a transformative discipline at the heart of software engineering.

About the authors

Benoit Combemale is a Research Director at Inria and a Full Professor of Software Engineering at the University of Rennes. His research interests in Software Engineering include Software Language Engineering, Model-Driven Engineering, and Software Validation & Verification. You can contact the author at benoit.combemale@inria.fr or visit http://combemale.fr.

Acknowledgments

The author thanks all the early readers for their valuable feedback. I also kindly thank Valentin Bourcier and Steven Costiou for our insightful discussions.

References

Baresi, L., & Ghezzi, C. (2010). The disappearing boundary between development-time and run-time. In *Proceedings of the fse/sdp workshop on future of software engineering research* (p. 17–22). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/1882362.1882367 doi: 10.1145/1882362.1882367

Barricelli, B. R., Cassano, F., Fogli, D., & Piccinno, A. (2019). End-user development, end-user programming and end-user software engineering: A systematic mapping study. *Journal of Systems and Software*, *149*, 101-137. Retrieved from https://www.sciencedirect.com/science/article/pii/S0164121218302577 doi: https://doi.org/10.1016/j.jss.2018.11.041

Bencomo, N., Cabot, J., Chechik, M., Cheng, B. H. C., Combemale, B., Wąsowski, A., & Zschaler, S. (2024). *Abstraction engineering.* Retrieved from https://arxiv.org/abs/2408.14074

Booch, G., & Brown, A. W. (2003). Collaborative development environments. *Adv. Comput.*, *59*, 1–27. Retrieved from https://doi.org/10.1016/S0065-2458(03)59001-5 doi: 10.1016/S0065-2458(03)59001-5

Bucaioni, A., Cicchetti, A., & Ciccozzi, F. (2022). Modelling in low-code development: a multi-vocal systematic review. *Software and Systems Modeling*, *21*(5), 1959–1981.

Burnett, M. M., & Myers, B. A. (2014). Future of end-user software engineering: beyond the silos. In *Future of software engineering proceedings* (p. 201–211). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2593882.2593896 doi: 10.1145/2593882.2593896

Cao, J., Kwan, I., Bahmani, F., Burnett, M., Fleming, S. D., Jordahl, J., … Yang, S. (2013). End-user programmers in trouble: Can the idea garden help them to help themselves? In *2013 ieee symposium on visual languages and human centric computing* (p. 151-158). doi: 10.1109/VLHCC.2013.6645260

D'Angelo, S., Lin, J., Dicker, J., Egelman, C., Hodges, M., Green, C., & Jaspan, C. (2024). Measuring developer experience with a longitudinal survey. *IEEE Software*, *41*(4), 19–24.

Fagerholm, F., & Münch, J. (2012). Developer experience: Concept and definition. In *2012 international conference on software and system process (icssp)* (pp. 73–77).

Forsgren, N., Kalliamvakou, E., Noda, A., Greiler, M., Houck, B., & Storey, M.-A. (2024). Devex in action. *Communications of the ACM*, *67*(6), 42–51.

Forsgren, N., Storey, M.-A., Maddila, C., Zimmermann, T., Houck, B., & Butler, J. (2021, March). The space of developer productivity: There's more to it than you think. *Queue*, *19*(1), 20–48. Retrieved from https://doi.org/10.1145/3454122.3454124 doi: 10.1145/3454122.3454124

Greiler, M., Storey, M.-A., & Noda, A. (2022). An actionable framework for understanding and improving developer experience. *IEEE Transactions on Software Engineering*, *49*(4), 1411–1425.

Grudin, J. (1994). Computer-supported cooperative work: history and focus. *Computer*, *27*(5), 19-26. doi: 10.1109/2.291294

Herbsleb, J., & Mockus, A. (2003). An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering*, *29*(6), 481-494. doi: 10.1109/TSE.2003.1205177

Kienzle, J., Combemale, B., Mussbacher, G., Alam, O., Bordeleau, F., Burgueño, L., … Syriani, E. (2022, October). Global Decision Making Over Deep Variability in Feedback-Driven Software Development. In *ASE 2022 - 37th IEEE/ACM International Conference on Automated Software Engineering* (p. 1-6). Rochester, MI, United States: IEEE. Retrieved from https://inria.hal.science/hal-03770004 doi: 10.1145/3551349.3559551

Leroy, D., Sallou, J., Bourcier, J., & Combemale, B. (2021). When scientific software meets software engineering. *Computer*, *54*(12), 60-71. doi: 10.1109/MC.2021.3102299

Luo, Y., Liang, P., Wang, C., Shahin, M., & Zhan, J. (2021). Characteristics and challenges of low-code development: The practitioners' perspective. In *Proceedings of the 15th acm / ieee international symposium on empirical software engineering and measurement (esem).* New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3475716.3475782 doi: 10.1145/3475716.3475782

Madni, A. M., & Sievers, M. (2018). Model-based systems engineering: Motivation, current status, and research opportunities. *Systems Engineering*, *21*(3), 172–190.

Meyer, A. N., Fritz, T., Murphy, G. C., & Zimmermann, T. (2014). Software developers' perceptions of productivity. In *Proceedings of the 22nd acm sigsoft international symposium on foundations of software engineering* (pp. 19–29).

Niephaus, F., Rein, P., Edding, J., Hering, J., König, B., Opahle, K., … Hirschfeld, R. (2020). Example-based live programming for everyone: building language-agnostic tools for live programming with lsp and graalvm. In *Proceedings of the 2020 acm sigplan international symposium on new ideas, new paradigms, and reflections on programming and software* (p. 1–17). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3426428.3426919 doi: 10.1145/3426428.3426919

Nierstrasz, O. M., & Gîrba, T. (2024). Moldable development patterns. In *Proceedings of the 29th european conference on pattern languages of programs, people, and practices.* New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3698322.3698327 doi: 10.1145/3698322.3698327

Noda, A., Storey, M.-A., Forsgren, N., & Greiler, M. (2023). Devex: What actually drives productivity? *Communications of the ACM*, *66*(11), 44–49.

Palomino, P., Fonseca, M., Souza, J., Toda, A., Pereira, R. L., Cordeiro, T., … Demerval, D. (2025). Enhancing developers experience (devex) for successful design system implementation. *International Journal of Human–Computer Interaction*, *41*(1), 807–819.

Parry, O., Kapfhammer, G. M., Hilton, M., & McMinn, P.

(2022). Surveying the developer experience of flaky tests. In *Proceedings of the 44th international conference on software engineering: Software engineering in practice* (pp. 253–262).

Pothier, G., & Tanter, E. (2009). Back to the future: Omniscient debugging. *IEEE Software*, *26*(6), 78-85. doi: 10.1109/MS.2009.169

Razzaq, A., Buckley, J., Lai, Q., Yu, T., & Botterweck, G. (2024). A systematic literature review on the influence of enhanced developer experience on developers' productivity: Factors, practices, and recommendations. *ACM Computing Surveys*, *57*(1), 1–46.

Stol, K.-J., & Fitzgerald, B. (2018, September). The abc of software engineering research. *ACM Trans. Softw. Eng. Methodol.*, *27*(3). Retrieved from https://doi.org/10.1145/3241743 doi: 10.1145/3241743

Weintrop, D., & Wilensky, U. (2017). Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education (TOCE)*, *18*(1), 1–25.