Proofs of Concept as First-Class Architectural Decision Instruments

Fabio Petrillo
École de technologie supérieur
Montréal, Canada
fabio.petrillo@etsmtl.ca

Bruno Antognolli
École de technologie supérieur
Montréal, Canada
bruno-fernando.antognolli.1@ens.etsmtl.ca

Abstract

Proofs of Concept (PoCs) are used in software-intensive systems to evaluate technologies, reduce uncertainty, and support early decision-making. Despite their role in architectural practice, PoCs remain conceptually underdefined and methodologically ad hoc in both research and practice. In this position paper, we argue that PoCs should be a first-class architectural decision instrument rather than informal experiments or disposable prototypes. We highlight the conceptual ambiguity surrounding PoCs and discuss how this ambiguity undermines architectural decision-making. We then articulate a structured, architecture-centric perspective on PoCs as short-lived, risk-driven experiments explicitly designed to inform architectural choices. Drawing on insights synthesized from academic and grey literature and illustrated through industrial scenarios, we outline a lightweight framework that positions PoCs as repeatable, analyzable, and decision-oriented artifacts. We conclude by surfacing open challenges intended to provoke discussion on how architectural decisions are currently made in practice.

CCS Concepts

• **Software and its engineering → Software architectures**; *Software design engineering*; *Software system structures*; Software evolution; Software development process management.

Keywords

Software Architecture, Proof of Concept, Architectural Decision-Making, Architecture under Uncertainty

ACM Reference Format:
Fabio Petrillo and Bruno Antognolli. 2026. Proofs of Concept as First-Class Architectural Decision Instruments. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE-Companion '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3774748.3795875

1 Introduction

Software architecture is increasingly shaped by uncertainty and complexity [3, 6] . Architects are routinely required to make early, high-impact decisions in contexts characterized by incomplete information, rapidly evolving technologies, and stringent non-functional constraints. This situation is particularly pronounced in contemporary settings such as cloud-native systems, DevOps-oriented organizations, AI and machine learning pipelines, and highly regulated domains such as finance and healthcare [1, 4].

In practice, Proofs of Concept (PoCs) are frequently employed to cope with this uncertainty. Architects and development teams build PoCs to explore the feasibility of technologies, to assess architectural strategies under realistic constraints, and to reduce risk before committing to large-scale design or implementation decisions [2]. Typical questions addressed through PoCs include whether a given technology can meet performance or scalability requirements, whether an architectural approach integrates effectively with an existing ecosystem, or whether regulatory or operational constraints can be satisfied.

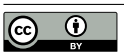
Despite their widespread use, PoCs remain poorly conceptualized in software engineering research and weakly theorized within software architecture. The term "PoC" is often used interchangeably with prototypes, pilots, or minimum viable products, leading to confusion about intent, scope, and expected outcomes [5]. As a result, PoCs are frequently executed in an informal and opportunistic manner, without explicit success criteria, architectural rationale, or systematic linkage to the decisions they are meant to support.

In this paper, we take the position that **PoCs should be treated as first-class architectural decision instruments**. Rather than being viewed as auxiliary or disposable activities, PoCs should be recognized as explicit, short-term, risk-driven experiments whose primary purpose is to inform architectural decision-making. Elevating PoCs to this status has the potential to improve decision quality, enhance traceability, and support more systematic learning in architectural practice. This position paper does not aim to provide a comprehensive empirical evaluation. Instead, it seeks to clarify the architectural role of PoCs, articulate why their current informal treatment is problematic, propose a structured perspective aligned with architectural concerns, and stimulate discussion within the software architecture community.

2 Conceptual Ambiguity of PoCs

One of the central problems surrounding PoCs in software architecture is their conceptual ambiguity. In both academic publications and industrial discourse, the term "PoC" is used to denote a wide range of artifacts and activities. In some cases, a PoC refers to a quick technical experiment designed to test a single hypothesis. In others, it denotes a throwaway prototype, a reduced implementation of a future system, or even a pilot deployed in a production-like environment.

These interpretations differ substantially in scope, cost, lifespan, and intent, yet they are often conflated under the same label. This ambiguity leads to misaligned expectations among stakeholders. Architects, developers, managers, and decision-makers may



This work is licensed under a Creative Commons Attribution 4.0 International License.
ICSE-Companion '26, Rio de Janeiro, Brazil
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2296-7/26/04
https://doi.org/10.1145/3774748.3795875

attribute different meanings and goals to the same PoC. For some, it represents a learning exercise; for others, it serves as evidence of feasibility or as a stepping stone toward production.

From an architectural perspective, this lack of clarity obscures the purpose of the PoC and weakens its value as a decision-support mechanism. The informal treatment of PoCs has concrete consequences for architecture practice. Architectural decisions are frequently justified by PoC outcomes whose assumptions, metrics, and limitations are undocumented or poorly articulated. This undermines decision traceability and makes it difficult to revisit or reassess decisions when system requirements or constraints evolve. Moreover, teams may misallocate resources by investing heavily in PoCs that gradually drift toward production or, conversely, by underinvesting in PoCs that should have been conducted with greater rigor. The reuse of PoC code in production systems can also introduce hidden technical debt, as such code is typically optimized for speed of experimentation rather than architectural quality. Finally, valuable architectural knowledge generated during PoCs is often lost because results are not systematically captured, analyzed, or reused.

Architects and engineering leaders will recognize several recurring patterns. Teams frequently launch PoCs without writing down the architectural decision they intend to inform; consequently, they collect performance numbers that do not speak to the actual trade-off under debate. The measurements themselves often reflect what is easiest to instrument rather than what is architecturally significant; latency is gathered without tail distributions, cost is examined without accounting for image size or cold-start elasticity, and security is discussed without concrete evidence of policy enforcement or auditability. PoCs drift toward production because success criteria and stopping rules were never defined, leading to hidden technical debt when experimental code leaks into operational systems. Valuable knowledge disappears because the outcome is not tied to an Architectural Decision Record or codified as a fitness function that can be checked continuously in CI/CD. These shortcomings are not failures of talent but symptoms of a missing shared discipline: there is no minimal, architecture-centric pattern for conducting fast experiments whose outputs are traceable, comparable, and reusable across teams.

In contrast, established architectural practices such as Architecture Decision Records, risk-driven design, and methods like ATAM emphasize explicit rationale, trade-offs, and structured decision-making. PoCs, despite their central role in early architectural exploration, remain largely absent from these frameworks. This gap suggests a missed opportunity to integrate experimentation and evidence into architectural reasoning better.

3 Reframing PoCs as Architectural Decision Instruments

To address these shortcomings, we propose a reframing of PoCs in software architecture. Rather than treating PoCs as informal or peripheral activities, **we conceptualize them as short-term, scoped experiments explicitly designed to validate architectural hypotheses**. From this perspective, a PoC is primarily a decision-oriented artifact. Its main outcome is not executable code, but evidence that supports or refutes an architectural choice.

The core of our position is that a PoC must be defined specifically by its intent: the reduction of uncertainty regarding a specific architectural hypothesis. Unlike a prototype, which focuses on user experience or functional feasibility, or an MVP, which focuses on market viability, a PoC is an experiment designed to test technical assumptions. Whether the question concerns the latency of a microservices mesh, the reliability of an LLM agent's output, or the performance of a new database versioning strategy, the PoC serves as the empirical evidence required to support a "significant architectural decision."

This reframing highlights several essential characteristics. First, **PoCs are decision-driven in nature**. They are initiated to inform a concrete architectural decision, such as selecting a technology, choosing an architectural style, or validating a deployment strategy. Second, **PoCs are risk-driven**. They focus on the most uncertain, complex, or high-risk aspects of an architecture, where analytical reasoning alone is insufficient. Third, **PoCs are inherently ephemeral**. They are not intended to become production assets, and their code is often disposable. Finally, **PoCs are architecturally scoped**. They target specific architectural concerns, including performance, scalability, security, interoperability, compliance, or operational constraints.

Viewed in this way, PoCs become a mechanism for acquiring architectural knowledge. They complement existing architectural practices by providing empirical input to decision records, supporting continuous architecture evolution, and leveraging DevOps practices such as automation, monitoring, and observability to generate evidence. This perspective aligns with broader efforts to make software architecture more empirical, iterative, and evidence-based.

Based on these perspectives, we propose a more refined definition of PoC within the context of software architecture. This new definition emphasizes the limited scope of a PoC and its inherent nature as an experimental tool. The proposed definition is as follows:

> **Definition**
>
> *Proof of Concept (PoC) is a short-term experiment that validates hypotheses within a limited scope by assessing feasibility, mitigating risks, supporting learning and decision-making.*

We are not advocating a new method or heavier governance for Proofs of Concept. Rather, we argue that when PoCs are used to justify architectural decisions, they should make their assumptions and evidence explicit, instead of influencing the architecture implicitly.

4 A Structured Perspective on PoCs

While PoCs should remain lightweight and flexible, we argue that a minimal level of structure is necessary to ensure their architectural value. Drawing on a synthesis of academic work and industrial practice, we outline a simple, repeatable structure that frames PoCs around three interconnected phases: planning, execution, and decision-making.

The planning phase serves to make assumptions explicit. During this phase, the architectural objective of the PoC is clearly articulated, including the decision it is intended to inform. Relevant

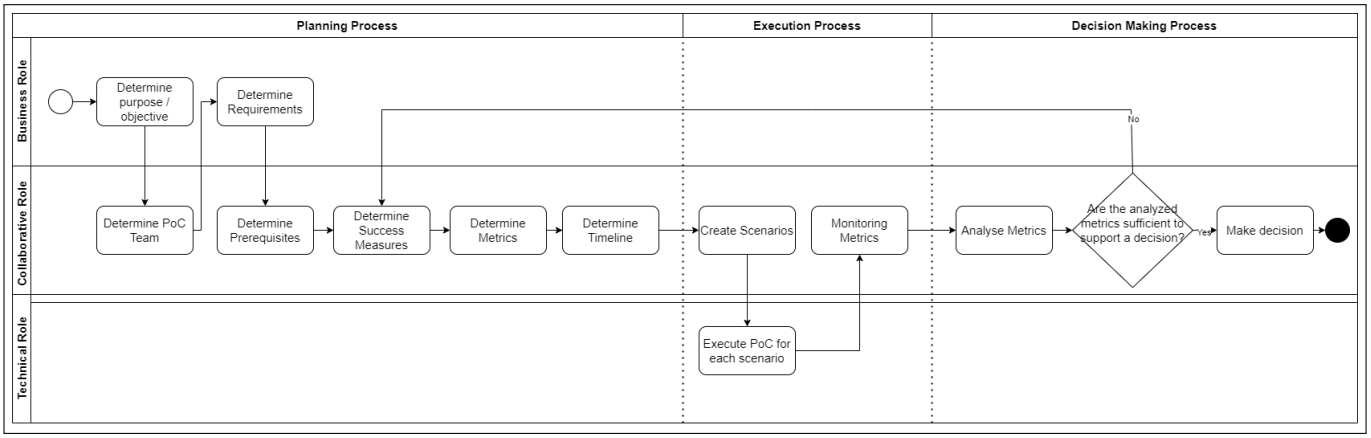


Figure 1: An illustrative view of Proofs of Concept as decision-oriented architectural activities, emphasizing role involvement, metric-driven experimentation, and explicit decision points rather than a prescriptive process.

stakeholders are identified, and the architectural concerns under investigation are specified. Success criteria and evaluation metrics are defined upfront, along with constraints related to time, budget, environment, and tooling. By explicitly linking the PoC to an architectural question, the planning phase helps prevent scope drift and misaligned expectations.

The execution phase focuses on controlled experimentation. The PoC is implemented in a constrained and representative environment, designed to exercise predefined scenarios that reflect the architectural concerns of interest. During execution, both quantitative and qualitative data are collected using metrics aligned with the initial success criteria. Importantly, the goal of execution is not to approximate a full system, but to test specific hypotheses under realistic conditions.

The decision-making phase translates experimental results into architectural action. Outcomes are analyzed against predefined criteria, trade-offs and limitations are identified, and a concrete architectural decision is made, such as adopting, rejecting, or revising a proposed approach. At this stage, PoC artifacts can be discarded, while the architectural knowledge gained is retained and documented. This separation between disposable code and persistent knowledge reinforces the role of PoCs as decision instruments rather than implementation precursors.

Figure 1 intentionally exposes aspects of Proofs of Concept that are often left implicit or treated as incidental in architectural practice. Rather than proposing a standardized process, the figure makes visible the architectural work that frequently occurs informally, including the negotiation of objectives across roles, the explicit framing of success criteria and metrics, and the moment at which experimentation should—or should not—justify an architectural commitment. By foregrounding these elements, the figure challenges the common tendency to treat PoCs as purely technical exercises or preliminary implementations, and instead frames them as deliberate acts of architectural reasoning under uncertainty. We include this representation not as a model to be followed, but as a provocation: if PoCs routinely influence high-impact architectural decisions, should their assumptions, evidence, and decision points remain implicit and largely undocumented?

5 Open challenges and Community Discussion

5.1 PoCs Across Contemporary Architectural Contexts

The proposed perspective on PoCs applies across a wide range of architectural contexts commonly discussed in contemporary software architecture. In cloud-native systems, PoCs are used to compare frameworks, runtimes, or deployment strategies under realistic load conditions.

In DevOps environments, they support the evaluation of CI/CD pipelines, observability stacks, and automation tools. In regulated domains, PoCs help assess compliance, auditability, and traceability constraints before large-scale adoption. In AI and machine learning architectures, PoCs are indispensable for testing feasibility, performance, cost, and integration of models and data pipelines.

5.2 Evidence Versus Process: Structure Without Bureaucracy

Thus, treating PoCs as first-class architectural decision instruments raises several open challenges that merit discussion within the software architecture community. One such challenge concerns the balance between standardization and flexibility. While structure is necessary to ensure decision quality, excessive formalization risks turning PoCs into heavyweight processes that undermine their exploratory nature.

From a tooling perspective, there is an opportunity to better leverage DevOps and observability platforms to support PoC-driven decision-making. Finally, the relative absence of PoCs from software architecture research highlights a broader gap between architectural theory and practice that warrants further investigation.

5.3 Integrating PoCs with Architectural Knowledge Artifacts

From an architectural knowledge perspective, Proofs of Concept naturally align with the intent of Architecture Decision Records. While ADRs are designed to document significant architectural decisions and their rationale, PoCs often generate the empirical evidence on which those decisions are implicitly based.

Yet, in practice, the relationship between PoCs and ADRs is rarely made explicit. PoCs are executed, conclusions are drawn, and architectural choices are made, but the assumptions tested, the metrics observed, and the trade-offs identified during experimentation are frequently decoupled from the architectural record. This disconnect contributes to the erosion of decision traceability and makes it difficult to revisit architectural choices as systems evolve or as new constraints emerge.

5.4 The Undocumented PoC as an Architectural Anti-Pattern

From our vision, the habitual separation between Proofs of Concept and Architecture Decision Records constitutes an architectural anti-pattern. ADRs are meant to capture the rationale behind significant architectural choices, yet PoCs are routinely executed precisely to generate the evidence that determines those choices. When PoCs conclude without producing, refining, or invalidating an ADR, architectural decisions are effectively made off the record. The result is a form of "ghost architecture," where high-impact choices are justified by experiments that leave no durable trace of their assumptions, metrics, or trade-offs. This anti-pattern does not stem from negligence or lack of skill, but from a tacit acceptance that PoCs exist outside the architectural knowledge lifecycle. Thus, We refer to this recurring situation as the **Undocumented Architectural Experiment anti-pattern**, in which PoCs effectively determine architectural decisions without producing durable architectural knowledge.

If PoCs are to be treated as first-class architectural decision instruments, this anti-pattern must be confronted explicitly. A PoC that neither produces an Architecture Decision Record nor invalidates a candidate ADR should be considered architecturally incomplete, regardless of how technically successful the experiment may appear. Framed this way, PoCs become accountable architectural acts rather than informal preludes whose influence is absorbed implicitly. This stance is intentionally confrontational: it challenges the normalization of undocumented experimentation and reframes PoCs as evidence-producing activities whose primary responsibility is to shape durable architectural knowledge, even when their implementation artifacts are deliberately discarded. Seen this way, **PoCs act as empirical gates in the architectural decision lifecycle**, determining whether a candidate decision is promoted, revised, or rejected.

5.5 Scope and Intent

It is important to clarify the scope of the position advanced in this paper. We are not proposing a new architectural method, nor advocating for additional layers of governance or heavyweight process around Proofs of Concept. We also do not suggest that PoCs must always be formalized or documented in the same way across contexts. Rather, our claim is narrower and deliberately pragmatic: when PoCs are used to inform significant architectural decisions, their assumptions, evidence, and outcomes should be made explicit enough to support architectural reasoning, traceability, and learning. The intent is not to constrain experimentation, but to prevent architectural decisions from being driven by experiments whose influence remains implicit and undocumented.

6 Conclusion

Proofs of Concept occupy a paradoxical position in contemporary software architecture practice: they are routinely used to justify high-impact architectural decisions, yet they remain conceptually underdefined and weakly integrated into architectural knowledge management. In this paper, we have argued that this disconnect is not merely terminological or procedural, but architectural in nature. By treating PoCs as informal or disposable activities, organizations allow critical architectural decisions to be shaped by experiments whose assumptions, evidence, and limitations are rarely made explicit.

Reframing PoCs as first-class architectural decision instruments provides a way to confront this tension. Viewed as short-lived, risk-driven experiments whose primary outcome is evidence rather than code, PoCs can be more deliberately aligned with architectural reasoning, documentation, and learning. In particular, making explicit the relationship between PoCs and architectural decisions helps prevent architectural knowledge from remaining implicit and ephemeral.

We do not advocate new methods or heavier governance, but rather greater intentionality and visibility around how experimentation informs architectural commitments. We offer this position as a provocation to the community: if PoCs continue to play a decisive role in shaping software architectures, can we afford to let their influence remain undocumented and largely invisible? We believe that acknowledging PoCs as explicit architectural decision instruments is a necessary step toward more empirical, accountable, and continuously evolving software architectures, and a fruitful topic for discussion within the software architecture community.

References

[1] David Ameller, Claudia Ayala, Jordi Cabot, and Xavier Franch. 2012. How do software architects consider non-functional requirements: An exploratory study. In *2012 20th IEEE International Requirements Engineering Conference (RE)*. 41–50. doi:10.1109/RE.2012.6345838

[2] G.A. Baptista and F. Abbruzzese. 2024. *Software Architecture with C# 12 and .NET 8: Build Enterprise Applications Using Microservices, DevOps, EF Core, and Design Patterns for Azure*. Packt Publishing Limited. https://books.google.ca/books?id=wJJQ0AEACAAJ

[3] Naeem Esfahani, Kaveh Razavi, and Sam Malek. 2012. Dealing with uncertainty in early software architecture. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering* (Cary, North Carolina) *(FSE '12)*. Association for Computing Machinery, New York, NY, USA, Article 21, 4 pages. doi:10.1145/2393596.2393621

[4] P. Lakkarasu. 2025. *Designing Scalable and Intelligent Cloud Architectures: An End-to-End Guide to AI Driven Platforms, MLOps Pipelines, and Data Engineering for Digital Transformation*. Deep Science Publishing.

[5] I. Mistrik. 2012. *Aligning Enterprise, System, and Software Architectures*. Business Science Reference. https://books.google.ca/books?id=lv3EswzXZpsC

[6] Dalia Sobhy, Rami Bahsoon, Leandro Minku, and Rick Kazman. 2021. Evaluation of Software Architectures under Uncertainty: A Systematic Literature Review. *ACM Trans. Softw. Eng. Methodol.* 30, 4, Article 51 (Aug. 2021), 50 pages. doi:10.1145/3464305