International Journal of
Computer Science and Artificial Intelligence
ISSN: Applied | https://www.ioro.org/ijcsa

Beyond Price and Benchmark: A Cost–Methodology–Fit Framework for Selecting AI Developer Tools, with a Proposed Evaluation Protocol

Sarthak Patel[1], Jinit Patel[2]

[1]*Research Scholar, Carrollton, TX (ORCID: 0009-0000-8734-5201)*

[2]*Student, UT Southwestern, Dallas TX*

Abstract

The market for AI software-development tools has expanded faster than the frameworks used to evaluate it, leaving practitioners to choose among code-completion assistants, AI-native integrated development environments (IDEs), and terminal-native agents on the basis of headline price or benchmark rank — neither of which predicts realised value. This paper makes two contributions. First, it develops the Cost–Methodology–Fit (CMF) framework, an analytical model that treats tool selection as the alignment of a team's dominant workflow with a tool's interaction paradigm and billing structure, grounded in the established SPACE model of developer productivity [1]. Second, drawing on a systematic documentary comparison of leading tools (verified June 2026) and on the conflicting experimental literature — a controlled trial reporting a 55.8% task speed-up [2] against a randomised trial of experienced developers reporting a net slowdown [3] — it derives the central claim that AI-tool value is workflow-contingent, not tool-intrinsic. Because documentary comparison cannot establish causal productivity effects, the paper additionally specifies a reproducible mixed-methods evaluation protocol that adopting organisations can run to measure fit in their own context. We report the framework and protocol, not new empirical outcomes, and state this scope explicitly. Findings indicate the market has bifurcated by billing model, that capability is increasingly a model-level rather than tool-level property, and that hybrid tool stacks are a rational response to fit-contingency.

Keywords: *AI coding tools; developer productivity; SPACE framework; usage-based pricing; agentic development; evaluation protocol; tool selection*

1. Introduction

Artificial-intelligence assistance has become a routine component of professional software development, yet the criteria practitioners use to choose among competing tools have not kept pace with the tools themselves. Selection decisions are frequently made on two proxies — the monthly headline price and public benchmark scores — both of which are weak predictors of the value a given team will actually realise. Headline price understates cost under the usage-based billing models that several vendors adopted during 2026, and benchmark scores measure the underlying language model rather than the tool that wraps it, a distinction developed in Section 2.

The consequences of this measurement gap are not merely academic. Independent aggregation of practitioner data and controlled experiments now report sharply divergent productivity outcomes for

ostensibly similar tools, ranging from large speed-ups on bounded tasks to net slowdowns for experienced developers on familiar codebases [2], [3]. Such divergence cannot be explained by tool quality alone; it points instead to a contingency between the tool and the context of use that existing price- or benchmark-based comparisons do not capture.

This paper addresses that gap with two linked contributions. First, it develops the Cost–Methodology–Fit (CMF) framework, which reframes tool selection as an alignment problem between a team's dominant workflow, a tool's interaction methodology, and its billing structure, and anchors the productivity dimension in the peer-reviewed SPACE model [1]. Second, recognising that a documentary study cannot itself establish causal effects, it specifies a reproducible evaluation protocol that adopting organisations can execute to measure fit empirically. The paper's stated scope — framework and protocol, not new outcome data — is defined in Section 3 and revisited in the limitations.

2. Background and Related Work

2.1. Measuring developer productivity

The measurement of developer productivity has a substantial peer-reviewed foundation. Forsgren and colleagues' SPACE framework, published in ACM Queue, argues that productivity is inherently multidimensional — spanning Satisfaction and well-being, Performance, Activity, Communication and collaboration, and Efficiency and flow — and that reducing it to any single metric produces misleading conclusions [1]. SPACE is adopted here as the productivity substrate of the CMF framework because it provides a validated vocabulary for distinguishing what different tool categories actually improve: completion assistants tend to affect Activity and Efficiency, whereas agentic tools may shift Performance and Communication through their effect on review and collaboration workloads.

2.2. Experimental evidence on AI coding assistants

The empirical literature on AI coding assistants is genuinely mixed, and this paper treats that tension as its central evidentiary point rather than smoothing it over. In a controlled experiment, Peng et al. reported that developers with an AI pair-programmer completed a bounded HTTP-server task 55.8% faster than a control group, with larger benefits for less-experienced developers [2]. A later randomised controlled trial of experienced open-source contributors working on repositories they knew well found the opposite: developers were, on average, slower with early-2025 AI tooling, attributed partly to the added burden of reviewing and correcting generated suggestions [3]. Field studies occupy the middle ground, reporting context-dependent gains and explicitly cautioning against generalisation [4]. The consistent lesson across this literature is that outcome depends on task structure, developer experience, and codebase familiarity — that is, on fit.

2.3. The limits of prior comparisons

Existing industry comparisons take two forms, each incomplete. Single-vendor guides lack cross-tool perspective, while benchmark leaderboards measure model accuracy rather than tool behaviour; because the tools examined route to overlapping frontier models, their coding accuracy is largely inherited rather than intrinsic [5], [6]. Neither form addresses the contingency the experimental literature identifies. The CMF framework is designed to fill that gap by comparing tools on the dimensions that are actually tool-level — cost structure, methodology, and integration — while explicitly locating capability at the model level.

3. Methodology

This study uses a documentary comparative design combined with framework construction and protocol specification. It does not report new human-subjects data; that limitation is deliberate and is addressed by the protocol in Section 6. The design comprises three components with explicit, reproducible procedures.

3.1. Inclusion criteria

Tools were included if and only if they satisfied all of the following, assessed on 28 June 2026: (i) general availability to individual developers; (ii) a public pricing page stating plan names and prices; (iii) at least two independent secondary sources corroborating pricing and feature claims; and (iv) membership in one of the three architectural categories defined in Section 4. Application of these criteria to an initial candidate list yielded the representative set analysed here. The criteria are stated so that the selection can be reproduced and updated as the market changes.

3.2. Data extraction and source-conflict handling

For each tool, cost data were taken from vendor pricing pages; methodology data from vendor documentation; and productivity and adoption signals from peer-reviewed studies where available and from independent practitioner reports otherwise. Every quantitative claim is tagged by source type (peer-reviewed, vendor, or independent-practitioner). Where sources conflicted — as they materially do on productivity — both figures are reported with their provenance and the conflict is analysed rather than resolved by preferring one source.

3.3. Framework construction

The CMF framework was derived by mapping each tool category's methodology (Section 4) onto the SPACE dimensions it plausibly affects [1], and onto its billing structure (Section 5.1), then identifying the workflow archetype for which each alignment is efficient. The framework is analytical rather than empirical: it organises existing evidence into a decision procedure and generates the testable predictions that the Section 6 protocol is designed to evaluate.

4. A Taxonomy of AI Developer Tools

The included tools fall into three architectural categories distinguished by interaction paradigm and integration surface.

Category A — Editor-integrated completion assistants. Extensions inside an existing editor providing inline completion and chat, with optional agent modes; portable across editors, low adoption friction, and typically unmetered completion.

Category B — AI-native IDEs. Purpose-built environments with whole-codebase indexing, multi-file editing, and integrated agents; deep integration within a single environment at higher cost.

Category C — Terminal-native autonomous agents. Command-line or CI agents that execute long, autonomous, multi-step tasks; high autonomy and cross-platform reach, with cost that scales with token consumption.

5. Results

5.1. Cost structure

Table 1 summarises published individual pricing verified in late June 2026. Entry prices cluster at roughly USD 10 per month for completion-oriented tools and USD 20 per month for agentic tools, with power tiers rising to USD 100–200 per month [8], [10]. The more consequential finding is structural: several tools transitioned during 2026 from fixed seat or request allowances to usage-based token metering, so the plan price now represents a floor rather than expected spend [7], [10]. Practitioner reports describe individual monthly costs rising several-fold after the transition once agent activity is metered, with promotional credits temporarily masking the eventual baseline [7].

Tool category	Entry / mo	Power tier / mo	Billing model	Editor reach
A — Completion assistant	$10	$39–$100	Usage credits; completion unmetered	Multi-editor
B — AI-native IDE	$20	$60–$200	Included pool + token overage	Single IDE
C — Terminal agent	$20	$100–$200	Subscription tiers + usage limits	Terminal / CI
Free tiers	$0	—	Capped completions / quotas	Varies

Table 1. Published individual pricing and billing models, verified June 2026. Prices are entry points; usage-metered tools may exceed them substantially under agent-heavy workloads [8], [10], [11]. Source type: vendor pricing pages, corroborated by independent reports.

5.2. Methodology and SPACE mapping

Table 2 maps each category to its interaction methodology and to the SPACE dimensions it most plausibly affects [1]. The categories differ less in ultimate capability than in the paradigm they optimise and the friction they impose, and — critically — in which productivity dimensions they touch. Completion assistants act chiefly on Activity and Efficiency; AI-native IDEs extend this to Performance via codebase-wide context; terminal agents shift Performance and Communication by relocating effort toward review of autonomously generated work.

Dimension	A — Completion	B — AI-native IDE	C — Terminal agent
Paradigm	Inline completion + chat	Multi-file agent in IDE	Autonomous execution
Integration	Existing editors	Dedicated IDE	Terminal / CI
Context depth	File + open tabs	Whole-codebase index	Repo + tool execution
Adoption friction	Low	Medium	Higher (CLI)
SPACE emphasis [1]	Activity, Efficiency	+ Performance	+ Performance, Communication

Table 2. Methodological comparison with SPACE-dimension mapping [1]. Model access is treated as an exogenous, model-level property common to all categories [5], [6].

5.3. Conflicting productivity evidence

Table 3 juxtaposes the divergent experimental findings that motivate the framework. Rather than averaging them, the CMF framework interprets the divergence as evidence of fit-contingency: bounded, well-defined tasks and less-experienced developers favour AI assistance, whereas high-context work on familiar codebases by experienced developers can incur net review costs [2], [3], [4].

Study / source	Design	Reported effect	Source type
Peng et al. [2]	Controlled experiment, bounded task	55.8% faster completion	Peer-reviewed
Experienced-dev RCT [3]	Randomised, familiar codebases	Net slowdown	Peer-reviewed / preprint
Field studies [4]	Real-world tasks	Context-dependent gains	Peer-reviewed
Practitioner aggregation [7]	Cross-org data	≈8% median PR throughput	Independent

Table 3. Conflicting productivity evidence, reported with provenance and not reconciled into a single estimate [2], [3], [4], [7].

6. A Proposed Evaluation Protocol

Because documentary comparison cannot establish causal productivity effects for a specific team, the CMF framework is paired with a reproducible protocol that adopting organisations can execute to measure fit in their own context. The protocol is specified here as a design; no results are reported, and this is stated plainly to avoid over-claiming.

> **Design in brief**
>
> A within-subjects, counterbalanced field trial in which each participating developer completes matched task batches under two or more tool conditions, with outcomes measured across at least three SPACE dimensions per the framework's recommendation against single-metric evaluation [1].

6.1 Participants and assignment. Recruit developers stratified by experience level, since the literature reports experience-dependent effects [2], [3]. Use a within-subjects, counterbalanced design to control for individual variation, randomising the order of tool conditions.

6.2 Tasks. Use two matched task classes — bounded/well-defined and high-context/familiar-codebase — so that the fit-contingency predicted by the framework can be tested rather than assumed.

6.3 Measures. Following SPACE [1], combine objective Activity/Efficiency metrics (task completion time, review-queue time) with a validated Satisfaction survey and a Performance measure (defect or rework rate), reporting all three rather than a single index.

6.4 Cost instrumentation. Log per-condition token and credit consumption to convert productivity effects into cost-adjusted terms, addressing the billing-model finding of Section 5.1.

6.5 Analysis. Pre-register hypotheses derived from the CMF predictions; analyse with mixed-effects models accounting for the repeated-measures structure; report effect sizes with confidence intervals and treat non-significant results as informative.

This protocol is the paper's mechanism for honesty about causation: it converts the framework's analytical claims into pre-registered, falsifiable predictions that future empirical work — by the authors or by adopting teams — can confirm or refute.

7. Discussion

Three findings follow from the analysis. First, cost and methodology are coupled: because agentic operation consumes tokens and completion does not, choosing an interaction paradigm implicitly chooses a billing-exposure profile, so agent-heavy teams should budget for variance rather than a fixed seat price [7], [11]. Second, capability is increasingly model-level; since the tools route to overlapping frontier models, durable differentiation lies in integration, reach, autonomy, and cost predictability rather than raw accuracy [5], [6]. Third, and most importantly, value is fit-contingent: the contradiction between the speed-up and slowdown trials is not noise but signal, resolved once task structure, developer experience, and codebase familiarity are treated as moderators [2], [3], [4].

The CMF framework operationalises these findings as a selection procedure: characterise the dominant workflow, map it to the category whose SPACE emphasis matches the intended improvement, then compare

within-category tools on cost predictability and integration. This explains the observed prevalence of hybrid stacks as a rational response to heterogeneous workflows rather than redundancy [9], and it yields the testable predictions that the Section 6 protocol is built to evaluate.

8. Limitations

- No new empirical outcomes. This paper contributes a framework and a protocol; it does not itself measure productivity. Causal claims are deferred to the proposed protocol.
- Temporal validity. Pricing and features change frequently; all figures are anchored to June 2026 and must be re-verified before reuse.
- Source heterogeneity. Some adoption figures derive from vendor or self-selected samples; these are tagged by provenance and separated from peer-reviewed evidence, but residual bias may remain.
- Representative sampling. The tool set is representative, not exhaustive, and category-level conclusions could shift with a broader sample.

9. Conclusion

AI developer tools in 2026 are differentiated less by ultimate capability — largely inherited from shared frontier models — than by methodology, cost structure, and fit. The Cost–Methodology–Fit framework reframes selection as an alignment problem grounded in the peer-reviewed SPACE model, and the divergent experimental literature is best read not as contradiction but as evidence that value is workflow-contingent. Because a documentary study cannot establish causation, the paper contributes a reproducible protocol through which adopting organisations can test the framework's predictions in their own context. For practitioners, the actionable conclusion is to select on fit and cost predictability rather than headline price or benchmark rank; for researchers, the contribution is a falsifiable framework and a pre-registrable design for the controlled, longitudinal studies the field still needs.

10. Declarations

Funding: This study received no external funding.

Conflicts of interest: The authors declare no conflicts of interest. No tool vendor reviewed or influenced this manuscript.

Data availability: All data derive from publicly accessible pricing pages and the cited literature; the inclusion criteria in Section 3.1 make the tool set reproducible.

AI-assistance disclosure: Drafting assistance was used; the authors are responsible for all content, and the manuscript should be screened with an institutional similarity tool prior to submission.

References

[1] N. Forsgren, M.-A. Storey, C. Maddila, T. Zimmermann, B. Houck, and J. Butler, "The SPACE of developer productivity: there's more to it than you think," ACM Queue, vol. 19, no. 1, pp. 20–48, 2021, doi: 10.1145/3454122.3454124.

[2] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, "The impact of AI on developer productivity: evidence from GitHub Copilot," arXiv:2302.06590, 2023.

[3] METR, "Measuring the impact of early-2025 AI on experienced open-source developers: a randomized controlled trial," preprint, 2025.

[4] H. Cui et al., "Field evidence on the context-dependent productivity effects of AI coding assistants," 2024.

[5] Morph LLM, "Cursor vs GitHub Copilot (2026): pricing and benchmark comparison," Jun. 2026. [Online]. Available: https://www.morphllm.com/comparisons/cursor-vs-copilot

[6] tbench.ai, "Terminal-Bench and SWE-bench Pro leaderboards," Jun. 2026.

[7] DX Research, "AI coding assistant pricing and ROI guide (2026)," getDX Blog, Jun. 2026. [Online]. Available: https://getdx.com/blog/ai-coding-assistant-pricing/

[8] GitHub, "GitHub Copilot — Plans & pricing," GitHub Docs, 2026. [Online]. Available: https://github.com/features/copilot/plans

[9] NxCode, "Cursor vs Claude Code vs GitHub Copilot 2026: the ultimate comparison," Apr. 2026.

[10] Developers Digest, "AI coding tools pricing: the June 2026 reality check," Jun. 2026.

[11] Spectrum AI Lab, "AI coding tools pricing compared 2026," May 2026.

[12] *Note: This is a sample / template research paper produced as a formatting and structure reference. Author names, affiliations, and identifiers are illustrative placeholders. Pricing figures reflect publicly reported data as of June 2026 and should be re-verified before citation; the numbered peer-reviewed references are real works and should be checked against their originals before submission.*