Evaluating the Efficacy of Artificial Intelligence in Software Engineering: A Post-February 2026 Analysis

Stephane H. Maes[1]

March 13, 2026

Abstract

The foundations of software engineering have undergone great transformations, especially following the release of frontier Large Language Models in the first quarter of 2026. This paper evaluates the efficacy of artificial intelligence for coding and within the software development lifecycle (SDLC), often contrasting theoretical benchmark, against empirical observations.. While frontier architectures, notably Anthropic Claude 4.6, OpenAI GPT 5.4, and DeepSeek V4, have definitively surpassed human baselines, in isolated synthetic benchmarks, their outcome within enterprise production environments reveals severe problems, confirming our past concerns and predictions. The initial perception of hyper accelerated code generation velocity, at this stage, widely publicly believed, is significantly counterbalanced by the Great Toil Shift, a phenomenon wherein the temporal savings of algorithmic syntax authoring are entirely consumed by the downstream burdens of architectural review, security auditing, code understanding/documentation, and continuous support and maintenance. Efficiency gains are not what they seem.

This paper identifies unprecedented surges in cyclomatic complexity, dynamic security vulnerabilities, and cognitive debt. Furthermore, the analysis identify the severe human toll associated with unrestricted artificial intelligence adoption. Driven by the relentless need to audit stochastic algorithmic outputs, human operators are increasingly suffering from AI Brain Fry, defined as acute mental fatigue resulting from the cognitive overload of continuous algorithmic oversight. This psychological degradation directly catalyzes the proliferation of coding Work Slop, wherein low quality, verbose, and structurally deficient code masquerades as competent engineering, actively destroying the structural integrity of the enterprise application architecture. It seems that this problem will only grow as LLMs evolve.

Ultimately, this paper concludes that while algorithmic systems have altered the velocity and division of technical labor, long term codebase viability remains strictly dependent of senior engineering oversight. Senior developers, QA can't just be replaced by junior developers and AI.

Or, to mitigate these systemic regressions, this paper posits that traditional human and artificial intelligence collaborative paradigms, including unconstrained vibe coding, are fundamentally unsustainable. Instead, the industry must transition toward application aware agentic artificial intelligence platforms. By leveraging dynamic temporal graph memory, and rigorous threat modeling frameworks, these deterministic platforms constrain stochastic generation, enforcing strict SDLC governance autonomously.

———

[1] shmaes.physics@gmail.com

Cite as: Stephane H. Maes, (2026), "Evaluating the Efficacy of Artificial Intelligence in Software Engineering: A Post-February 2026 Analysis", https://doi.org/10.5281/zenodo.19103818, https://shmaes.wordpress.com/2026/03/16/evaluating-the-efficacy-of-artificial-intelligence-in-software-engineering-a-post-february-2026-analysis/, March 13, 2026. (https://osf.io/vz6jt/files/gj4z2).

1. Introduction: The Post-February 2026 Paradigm Shift in Software Engineering

The practical foundations of software engineering have undergone a profound change following the frontier Large Language Model (LLM) releases in early 2026. The integration of artificial intelligence into the software development lifecycle (SDLC) has decisively transitioned from localized, syntax-level autocomplete utilities to autonomous, agentic systems capable of executing complex, multi-step engineering tasks. The rapid succession of model deployments between late 2025 and March 2026, e.g., most notably Anthropic's Claude 4.6, OpenAI's GPT-5.4, Google's Gemini 3.1 Pro, DeepSeek's V4, Mistral's Large 3, and Meta's Llama 4, warrants a rigorous, reevaluation of whether artificial intelligence currently surpasses human capability in the domain of software engineering [1].

While frontier models often exceed human baselines in isolated, synthetic coding benchmarks, their integration into continuous, collaborative production environments reveals severe dichotomies between just code generation velocity, and long-term codebase viability [8,28,44,77]. The proliferation of agentic coding tools has forced the industry to confront an emerging crisis, characterized by escalating technical debt, rendering support and maintenance very hard, the introduction of latent security vulnerabilities, and the erosion of algorithmic explainability [9, 28,44,77]. At organizations operating at the frontier of this technology, such as Anthropic, internal telemetry indicates that approximately 90% of the codebase for their Claude Code engineering tool is now synthesized autonomously by the model itself [13]. However, senior engineers uniformly caution that interacting with these models is a difficult, unintuitive process that demands rigorous human oversight, structured specification design, and continuous validation [13,28,44,77].

Historically, analyses of this paradigm shift have focused almost exclusively on the mechanical and structural implications of artificial intelligence integration, fundamentally overlooking two critical variables that define the empirical reality of the 2026 enterprise ecosystem. The first missing element in contemporary discourse is the psychological and cognitive toll, exacted upon the human operators tasked with managing these systems. The relentless requirement to review, debug, and curate massive and volumes of algorithmically generated verbose syntax has precipitated a widespread phenomenon categorized as AI Brain Fry [75,76]. This acute cognitive fatigue radically diminishes decision making quality, and directly fuels the proliferation of Work Slop, defined as (verbose) algorithmic output that mimics competent engineering, but fundamentally lacks architectural substance [28,44,61,75,76], and describe some function with way more code than needed; a direct consequence of being LLM-based. Unfortunately, as LLM evolve this verbosity, and effects, are growing.

This paper provides an assessment of the comparative efficacy of human, and AI, software engineering paradigms post-February 2026. The analysis evaluates the state of the art across two primary axes. The first axis examines coding productivity, critically contrasting the saturation of synthetic capabilities benchmarks against empirical organizational throughput, the redistribution of engineering toil, and the cognitive limitations of AI in competitive heuristic environments. The second axis analyzes the Quality Triad of software engineering: the long-term maintainability, and supportability, of AI-generated code, the robustness of these systems against dynamic security vulnerabilities, and the critical transition towards Code Cognition, and explainable artificial intelligence (XAI). Through the paper, this analysis aims to articulate the nuanced reality of human-AI collaboration, identifying where

algorithmic systems have eclipsed human capacity, and where human cognitive architecture remains ahead, even if it may[2] become replaceable in the future, or if some approaches may fill related gaps [77].

The assertion that artificial intelligence is better than human engineers may be a fallacy that fails to capture the complexities inherent to modern enterprise software development [7,28,44,77,80].

Finally, we review the approach of constrained vibe coding [82], and SDLC managed through a same agentic AI platform to address the concerns above [28,44,77], in an autonomous way, with application-aware platform agentic AI [62-70,77], à la Zenera are a good example [62,68,69]. Evolving vibe coding to such patterns seems a natural evolution, as what is happening right now will only get worse, and it is not sustainable.

2. Coding Productivity: The Illusion of Velocity Versus Empirical Organizational Throughput

The evaluation of coding productivity in the age of advanced LLMs requires a combination of methodologies: the theoretical capabilities demonstrated in controlled, synthetic benchmarks, and the tangible, impact of these models on enterprise software delivery lifecycles (SDLC) [28,44,77,78].

2.1 The Saturation of Synthetic Benchmarks and the Achievement of Superhuman Baselines

Throughout the first quarter of 2026, artificial intelligence industry witnessed a systematic saturation of historical coding benchmarks[3], rendering traditional evaluation metrics increasingly obsolete for distinguishing the capabilities of newly released (frontier) models [15]. It was predicted [78]. Note that a benchmark is considered saturated when leading models achieve near-ceiling scores, thereby failing to provide statistically significant differentiation between competing architectures [15]. This phenomenon is compounded by the persistent issue of data contamination, wherein the test questions of public benchmarks inadvertently[4] appear in the model's pre-training corpus, making the resulting score the result of memorization, rather than generalized reasoning [15,80].

For example, the GSM8K benchmark, originally designed to test mathematical and logical reasoning, saw models like GPT-3 scoring around 35% in 2021; by early 2026, OpenAI's GPT-5.3 Codex achieved a 99% accuracy rate, completely saturating the benchmark, and rendering it quasi useless for comparative analysis [15]. Such rapid cycle of improvement was predictable [78].

---

[2] Some won't be replaceable any time soon with the conventional approaches used Today.
[3] It has gone a really long way since early approaches, with old AI, as in [81].
[4] Or intentional as it seems that many frontier model players try to convince us of immediate apocalypse, or about reach soon singularity or AGI [SM]. Motives seems to be to increase valuations, obtain founding or escape from constraining contractual terms, invoking special clauses for example for when AGI will be reached.

Model Designation	Developing Organization	SWE-bench (Verified/Lite)	OSWorld-Verified Score	Release / Update Date	Defining Architectural Innovations and Context Capabilities
Claude 4.6 Opus	Anthropic	80.8%	N/A	Feb 4, 2026	1M Token Context Window (Beta), Adaptive Interleaved Thinking, Agentic Workflow Optimization [1]
GPT-5.4 High	OpenAI	81.05%	75.0%	Mar 4, 2026	Unified Reasoning and Execution, OS-Level Tool Integration, Lower Effective Token Cost [8]
Gemini 3.1 Pro High	Google	84.17%	N/A	Feb 18, 2026	Native Multi-modal Processing, Flash-Lite Optimization, Custom Tool Prioritization [3]
DeepSeek-V4	DeepSeek	>80.0% (Estimated)	N/A	Feb 2026	Manifold-Constrained Hyper-Connections (mHC), Ultra-low Inference Cost Optimization [6]
Llama 4 (Maverick)	Meta	N/A	N/A	Apr 2025/2026	1M Token Context, Open-weight Deployment, Mixture of Experts Architecture [5]
Mistral Large 3	Mistral AI	N/A	N/A	Feb 2026	41B Active Parameters, NVFP4 Quantization, Sparse Mixture of Experts [4]

Table 1: It summarizes the state-of-the-art benchmark performances and architectural innovations across the primary frontier LLMs available in the post-February 2026 ecosystem.

Despite these methodological limitations, the sheer scale of the benchmark victories recorded in February and March 2026 cannot be dismissed, as they signal a crossing of the Rubicon in algorithmic problem-solving, again to be expected [78]. OpenAI's strategic transition from the specialized GPT-5.3 Codex to the generalized GPT-5.4 mainline model marked a critical inflection point [8]. GPT-5.4 successfully absorbed the specialized execution, and automation, capabilities of its predecessor, pushing its computer-use score on the OSWorld-Verified benchmark to 75% [8]. Such an achievement represents the first documented instance of an AI system surpassing the 72.4% baseline established by human industry professionals [8]. Concurrently, OpenAI reported that GPT-5.4 matched or exceeded the performance of human professionals on 83% of specialized knowledge-work tasks evaluated in the GDPval framework, a multi-dimensional reasoning assessment where specialized coding models previously faltered [2].

Simultaneously, Anthropic's release of the Claude 4 family, specifically the Opus 4.6 variant, in February 2026, established a new paradigm in agentic GitHub issue resolution [1]. Evaluated on the SWE-bench framework, i.e., a

benchmark explicitly designed to measure a model's capacity to autonomously navigate, debug, and resolve complex issues within real-world, multi-file, software repositories, Claude 4.6 Opus achieved an unprecedented 80.8% resolution rate [16]. This performance was heavily facilitated by the introduction of a beta 1-million token context window, allowing the model to ingest, map, and reason over massive enterprise codebases without catastrophic forgetting [1].

See Table 1 for an overview.

The emergence of these capabilities extends beyond the proprietary models of Western technology conglomerates, as expected [78]. The open-weight ecosystem experienced a parallel explosion in capability, fundamentally altering the economics and accessibility of advanced AI coding tools.

DeepSeek-V4, anticipated for wide release in mid-February 2026, introduced a structural rethinking of transformer deep learning through Manifold-Constrained Hyper-Connections (mHC) [6]. Unlike traditional residual connections that treat all neural layers equally, mHC allows the model to dynamically learn which layers contribute most to logic and syntax resolution, applying learnable scaling on residual paths [6]. This innovation yields approximately 30% faster convergence during training and noticeable quality gains on coding benchmarks, while drastically reducing inference costs [6].

Preliminary data suggests DeepSeek-V4 operates at an input cost of roughly $0.14 per million tokens, i.e., potentially 36 times cheaper than Claude Opus 4.6, while maintaining performance parity on the HumanEval and SWE-bench metrics [22].

Similarly, Mistral Large 3, featuring 41 billion active parameters within a 675 billion parameter sparse mixture-of-experts framework, achieved state-of-the-art results for permissive open-weight models, democratizing access to frontier-level code generation [4].

2.2 The Discrepancy Between Perceived Velocity and Empirical Throughput: The "Great Toil Shift"

While the theoretical productivity of AI coding assistants appears staggering, e.g., being capable of generating comprehensive data pipelines, RESTful APIs, and extensive unit test suites in a matter of seconds, the empirical reality of human-AI collaboration reveals pervasive challenges [7,28,44,77]. To understand how these models impact actual developer productivity, the industry has had to reconsider traditional measurement frameworks, such as the SPACE framework (Satisfaction, Performance, Activity, Communication, Efficiency) and DORA metrics (Deployment Frequency, Lead Time for Changes, Mean Time to Recovery, Change Failure Rate) [27].

A mixed-methods longitudinal study analyzed the validity of different approaches to evaluating the productivity impacts of AI coding assistants [27]. The study, which encompassed survey responses from 2,989 software developers and 11 in-depth qualitative interviews, exposed conflicting perspectives, and a massive disconnect between self-reported satisfaction, and objective time savings [27]. The survey results indicated that an overwhelming 86% of developers reported being either "Satisfied" or "Very satisfied" with AI coding tools such as GitHub Copilot and Cursor [27]. However, when quantifying actual temporal efficiency, the reported time savings were paradoxically modest: 29% of the developers reported saving merely 1 to 30 minutes per week, and 23%

reported saving 31 to 60 minutes per week [27]. Only 18% of the engineering workforce reported saving more than two hours per week through the utilization of advanced LLMs [27].

Theoretically, this discrepancy is explained by a phenomenon called the Great Toil Shift (See [7,28,29,44,77] and references therein). According to the SonarSource State of Code Developer Survey [29], which analyzed global coding practices, in conjunction with telemetry, from over 750 billion lines of code, developers consistently report spending nearly a quarter of their workweek (23% to 25%) on toil tasks, e.g., tedious, repetitive work that hinders high-level innovation. Crucially, this percentage remains statically identical for both frequent, aggressive users of AI coding assistants and those who use AI tools infrequently [29]. We will also see the brain fry impact later on.

The integration of artificial intelligence does not eliminate engineering toil; rather, it alters how it is categorized, and redistributes it downstream [29]. For developers who use AI infrequently, toil traditionally manifests as writing repetitive boilerplate syntax, manually formatting data structures, and attempting to decipher poorly documented legacy codebases [28,29,44,77]. For the most frequent and aggressive users of frontier AI models, the toil seems to have shifted entirely into new domains: managing the explosive accumulation of technical debt, conducting exhaustive audits of stochastic algorithmic outputs, and continuously correcting, or rewriting the subtly flawed code generated by generative AI tools [14]. The developer is no longer primarily a Code Author. Instead, they have been forcibly transitioned into the role of a Code Curator, or Code Manager[5] [7,28,44,77]. Because AI models are fundamentally optimized for rapid text completion, rather than long-term systemic maintenance, they tend to generate verbose, overly complex solutions that lack the necessary architectural abstraction [14,28,44,61,77,79,80]. Consequently, the speed gained during the initial drafting phase is entirely consumed by the subsequent human-in-the-loop review, functional testing, and continuous code quality assurance required to prevent catastrophic failures in edge cases [7,28,44,77].

2.3 Systemic Reconfiguration: From Horizontal Layering to Vertical Integration

Recognizing that individual productivity metrics fail to capture the macro-economic impact of LLMs, researchers have shifted their focus to organizational capability, and Total Factor Productivity [27]. A multiple-case comparative study examined the organizational implications of Generative AI adoption by contrasting two distinct development environments: a traditional enterprise operating a massive legacy codebase (brownfield) and an AI-native startup operating without historical constraints (greenfield) [30].

The study concluded that the true efficacy of AI in software engineering is not realized by substituting human typing speed with algorithmic generation, as we predicted [28,44], but by fundamentally restructuring the division of labor [30]. Traditional software engineering organizations rely on Horizontal Layering, i.e., a paradigm characterized by strict functional specialization, where frontend developers, backend engineers, database administrators, and quality assurance testers operate in siloed divisions. This horizontal structure creates inefficient inter-functional coordination overhead, communication latency, and context switching costs.

The adoption of frontier models like Claude 4.6 and GPT-5.4 facilitates a transition toward Vertical Integration, characterized by end-to-end full-stack ownership by individual engineers [30]. Every developer (can) become(s) a

[5] Aligned with what we recommended to be needed with Vibe Coding / AI coding assistants [28,44].

full stack developer. The study reveals that organizations successfully implementing this AI-driven vertical integration achieved 8 to 33-fold reductions in overall resource consumption [30]. These gains are attributed to the emergence of the Super Employee, i.e., an AI-augmented engineer possessing the cognitive bandwidth to span traditional role boundaries, utilizing AI agents to rapidly execute specialized tasks outside their core competency while they maintain high-level architectural oversight [30][6].

As a result, [30] proposes that Human-AI Collaboration Efficacy must supplant individual productivity metrics as the primary optimization target for engineering management [30]. However, [30] also identified a dangerous AI Distortion Effect, which diminishes the returns. If engineering organizations blindly expand their agentic AI workforce scale without investing in the number and training of their senior engineers, the resulting coordination chaos of managing, something absolutely required per [28,44,77], thousands of autonomous AI agents will inevitably collapse the software delivery pipeline. Successful developers, in 2026, are required to master AI orchestration, bridging business logic with technical specifications, and operating with a security-first mindset, rendering the traditional skill of memorizing API syntax largely obsolete [31]. And, as explained in [28,44,77][7], they better be Senior enough or they will be overwhelmed, or will by-pass the SLDC steps required to ensure that the code will be maintainable, supportable, secure, and explainable.

2.4 The Limits Of Algorithmic Heuristics

The assertion that artificial intelligence has entirely superseded human coding capability encounters its most obvious falsification when considering extreme competitive programming. While LLMs excel at synthesizing known patterns, and optimizing localized algorithms, based on their training data, their capacity for maintaining intense, complex logical reasoning over an extended period without losing accuracy or becoming overwhelmed, for novel heuristic discovery, and for continuous strategic abstraction remains demonstrably inferior to top-tier human cognition [32,80]. Not a surprise, per [80].

This dichotomy was illustrated during the AtCoder World Tour Finals 2025 Heuristic contest, which served as a landmark exhibition match pitting elite human intellect against state-of-the-art artificial intelligence [33]. In this 10-hour marathon contest, twelve of the highest-ranked human competitive programmers were matched against OpenAIAHC, an advanced reinforcement-learning AI model, explicitly developed and optimized by OpenAI to push the boundaries of heuristic code generation and algorithmic optimization. The challenge involved an complex optimization problem, designed to aggressively test creativity, logic, and problem-solving stamina, under severe temporal and cognitive constraints.

The results of the competition provided a definitive, albeit narrow, victory for human cognition [34]. While the OpenAIAHC system successfully defeated eleven of the world's top human competitors, i.e., demonstrating its overwhelming superiority over the vast majority of the programming population, it was ultimately defeated by a human programmer, Przemysław Dębiak [33], who outscored the OpenAI system by a 9.5% margin in the final evaluation [34].

---

[6] But that does not imply at all that, the gap between senior or junior developer is being resorbed. But this manifest itself way more, when considering the management of the full SDLC that must simultaneously take place [28,44,61,77].

[7] And comments on the web page versions.

The AtCoder results empirically demonstrate that at the absolute frontiers of computational logic, human cognitive architecture retains a distinct structural advantage. Dębiak noted the psychological and physical toll of the competition, stating he was completely exhausted, and barely alive, after out-maneuvering the AI's relentless, unyielding execution pace [33].

This highlights a fundamental distinction in performance paradigms: artificial intelligence systems exhibit flawless stamina and instantaneous execution of known heuristic patterns, but they lack the capacity for intuitive leaps, lateral reasoning across disparate abstract domains [80], and the continuous recalibration of a global architectural strategy, when faced with entirely novel optimization constraints [32,80]. They are not close to AGI [80], and will never be, with the current approaches. It has been our argument all along. This was confirmed by researchers participating in the ARC-AGI-2 Grand Prize competition: as competent as AI reasoning systems have become, they still exhibit profound flaws, and inefficiencies, in separating inherent knowledge from active reasoning [32]. Until AI systems can mimic the learning efficiencies, and action efficiencies, of human operators in highly novel environments, human experts will remain essential for the vanguard of algorithmic innovation [32,80]. We argue that this will remain the case until new approaches are introduced beyond just GenAI and LLMs [80].

3. The Quality Triad: Maintainability, Security, And The Erosion of Codebase Viability

If the analysis of productivity yields a nuanced perspective of AI as a powerful, but stochastic force multiplier, the analysis of software quality exposes severe, systemic problems that threaten the long-term viability of enterprise software. The true, hidden cost of AI-augmented software engineering in 2026 is measured not in lines of code produced, but in the degradation of code maintainability, the uncontrolled proliferation of security vulnerabilities, and the accumulation of technical and cognitive debt [9,28,44,61,77]. It also justify the value of new schemas as [64,77].

3.1 Structural Degradation: Technical Debt And Algorithmic Verbosity

Artificial intelligence coding tools are fundamentally optimized for functional completion, and immediate execution, rather than for the long-term systemic maintainability of the software [14,28,44,77]. The core training corpus for these massive models is the internet's aggregate of historical code, which is statistically verbose, frequently outdated, structurally inconsistent, and often insecure [14]. As a result, LLMs lack the contextual awareness necessary to abstract logic effectively [28,44,79,80], frequently generating hyper-verbose solutions that maximize surface area for subsequent bugs [14,28,44,77]. As the engineering adage says, "Code is a liability, not an asset", and AI models are currently hyper-accelerating the accumulation of this liability [14,28,44,61,77].

A 2026 quantitative study, evaluating the impact of AI-assisted code generation across multiple enterprise environments, reveals alarming trends regarding codebase health [9], aligned without analysis and prediction [28,44,77]. It finds that pull requests (PRs) co-authored by AI coding assistants contained 1.7 times more structural

and logical issues than PRs authored exclusively by human engineers. When observing the 90th percentile of worst-performing submissions, AI-assisted PRs contained twice as many issues. Specifically, the AI-generated code suffered measurable regressions across all critical quality axes compared to human baselines: Correctness issues were 1.75 times higher, Maintainability issues were 1.64 times higher, and Security issues were 1.57 times higher. The concerns from [28,44] are corroborated, and widely observed, one year later. In repositories, which leaned heavily on autonomous agentic assistance, overall cognitive complexity increased by 39%, while technical debt surged between 30% and 41% following the adoption of AI tools [9]. Furthermore, the study corroborated the Great Toil Shift, by demonstrating that the initial velocity gains experienced by teams adopting AI disappeared entirely within the first few months of use, counteracted by a 30% increase in the change failure rate, and a 23.5% increase in production incidents per PR [9]. And then some, no in fact it's actually many, leaders in companies think it's a good way forward to let go their Senior engineer, and, an even more a sign of total incompetence, to let go their QA teams [28,44,77].

The mechanics of this structural degradation are explicitly illuminated by this 2026 LLM Leaderboard: [10]. Leveraging their static analysis engine, [10] has analyzed over 750 billion lines of code, and tested the leading frontier models against 4,444 distinct Java programming assignments to evaluate their coding personalities. The data exposes a problematic correlation between a model's advanced reasoning capabilities, and its propensity to generate unnecessarily complex, stateful code [10], i.e., more code slop. As models become more "performant" in passing functional tests, their output consistently become more verbose and architecturally complex, imposing massive downstream burdens on the human engineers tasked with reviewing and maintaining the software [10]. We will come back to this.

LLM Model	Total Lines of Code (LOC) Generated	Cyclomatic Complexity Score	Cognitive Complexity Score
OpenCoder-8B	120,288	18,850	13,965
Llama 3.2 90B	196,927	37,948	20,811
GPT-4o	209,994	44,387	26,450
Claude Sonnet 4	370,816	81,667	47,649
GPT-5-minimal	490,010	145,099	111,133

Table 2 illustrates the Cyclomatic and Cognitive Complexity metrics of code generated by various LLMs, clearly demonstrating how models like GPT-5-minimal and Claude Sonnet 4 produce disproportionately dense, and convoluted, logic structures compared to smaller, less capable models [37]. Data is derived from the Analysis of 4,442 distinct Java programming assignments. High cyclomatic and cognitive complexity scores directly correlate with decreased human readability and increased maintenance costs [37].

As Table 2 illustrates, the GPT-5-minimal model generated more than four times the volume of code (490,010 LOC) compared to the smaller OpenCoder-8B model (120,288 LOC), to solve the exact same set of programming assignments [37]. Furthermore, its cyclomatic complexity, i.e., a quantitative measure of the number of linearly independent paths through a program's source code, skyrocketed to 145,099, indicating highly convoluted, heavily branched control flows, which are exponentially more difficult for human engineers to unit test and debug [37].

With this, we can conclude that the concerns, and problems identified in [28,44,77], were justified and have now been widely observed, vindicating the need for solutions, as for example the solutions proposed in [28,44,77] are well motivated and justified. Furthermore, we see that improving the performances of the LLMs, worsens some of these issues[8]. In other word, until a better approach is found, the situation will remain the same, or probably worsen. As we will discuss later, we argue that application-aware agentic AI control of vibe coding and the full SDLC, proposed in [77], based on [62-69], can provide a viable way forward. Enterprises should consider this path, instead of pursuing blind adoption of AI, coding / vibe coding, along with ill guided heavy lays off of developers, who are still needed to ensure maintainability of the code. In other words, don't fire your QA and Senior developers, you will regret it [77][9].

3.2 The Insidious Threat Of Cognitive Debt

While technical debt represents the physical, structural degradation of the codebase itself, a far more insidious organizational threat has emerged in 2026: Cognitive Debt [36]. Expanding upon computer scientist Peter Naur's classical theory that programming is not merely the production of text, but fundamentally an exercise in theory building within the developer's mind, modern researchers argue that generative AI actively breaks the links between the engineer and the systems, for which they are responsible [36].

Technical debt lives in the code, but cognitive debt lives in the developers' minds [36]. When a human engineer painstakingly writes a feature, they organically construct a mental model of how their intentions are algorithmically implemented, allowing them to instantly identify, where to intervene when future modifications are required [36], and hopefully document it well in the code. When an AI agent generates thousands of lines of syntactically correct code in seconds, this vital mental theory is never constructed [36,61]. Even if the AI agents produce code that is theoretically easy to understand, it typically does not though, the human operators have lost the plot: they do not understand what the program is supposed to do, how the edge cases are handled, or how to safely alter the logic, when needed say for support, maintenance or to add a new feature [36]. Consequently, software development teams are hitting catastrophic operational walls where they can no longer make even simple changes to their own products without triggering unexpected, cascading failures [36]. It is worse that heavy classical technical debt. It was a key part of the arguments in [28]. As mentioned in a recent footnote, hoping that AI can do it simply by passing unconstrained vibe coded code to it for fix, e.g., support, explain after the fact, update for a new release, or

[8] Let us also put to rest the urban legend that the supportability/maintainability challenges (and explainability) could be solved by using unconstrained vibe/AD coding to iterate / maintain / Support, and explain the messy output. It isn't for many reasons, but let us just argue that this would, at the minimum, be a great recipe for losing any control on the resulting software, and somebody said, software will eat the world… In other words, are we sure that this is a good idea, even if we were to look at it purely from a business efficiency, and ROI point of view? The short term profit will most certainly be traded for long term huge, and unmitigable losses.

[9] Maybe, instead, fire the management who have proposed to do so, as for example discussed in [77], and may give better salary savings.

to add features, is rarely a good solution, even if possible at times. There are exceptions though, as we will also discuss later, constrained vide coding and associated SDLC as described in [77], is a way to systematic address these challenges, which so far has been proven to work very well without all the issues we reported [62-69,77].

This dynamic also poses a severe existential threat to the long-term technical expertise of the software engineering workforce. The qualitative interviews, conducted for [27], highlighted profound managerial anxieties regarding widespread skill decay [27]. Over-reliance on AI models for routine implementation, algorithm design, and debugging actively erodes the training processes necessary for junior software engineers to develop fundamental architectural, and enterprise-grade coding skills [7]. When AI contributes significantly to the authoring process, it fundamentally weakens the team's collective code ownership, perceived accountability, and professional identity [27]. Engineers view their creative contributions as marginalized, leading to a pervasive unwillingness to maintain, or take responsibility for code that they did not author [27]. This is going the wrong way.

To mitigate the rapid accumulation of technical and cognitive debt, elite industry practitioners have begun adopting highly structured, human-in-the-loop governance frameworks, most notably the Plan-Do-Check-Act (PDCA) cycle [39]. This disciplined approach restricts human-AI collaboration to tightly scoped tasks of one to three hours, i.e., aligning with both human attention spans, and the reliable context windows of the models [40]. The PDCA framework mandates spec-driven development, requiring engineers to comprehensively brainstorm specifications, architectural decisions, and data models with the AI, before permitting any code generation [13]. By enforcing atomic commits, red-green test cycles, and micro-retrospectives, teams can utilize AI aggressively while maintaining strict architectural governance and mitigating the explosive proliferation of cognitive debt [13]. It is aligned with our recommendations to combine the tool with rigorous SDLV practices, manual, automated or autonomous [28,44,77], even if our proposal may not directly address the skill decay concerns.

3.3 Dynamic Security Vulnerabilities And Autonomous Auditing

The rapid acceleration of algorithmic code generation directly exacerbates the introduction of sophisticated security vulnerabilities into enterprise systems. While LLMs are highly proficient at producing syntax that passes continuous integration (CI) unit tests, this functional correctness frequently masks fundamental security flaws, that only manifest under adversarial, real-world conditions. Quantitative research from 2026 indicates that between 68% and 73% of AI-generated code contains latent security vulnerabilities that pass standard functional testing [12]. Consequently, enterprise environments operating with unrestricted AI coding assistants have documented a 23.7% absolute increase in security vulnerabilities integrated into their main production branches [31]. Again aligned with our predictions in [28,44,77].

Evaluating the true cybersecurity posture of these advanced models requires moving beyond static evaluation frameworks. Early benchmarks like SecEval and CyberBench evaluated LLMs on natural language processing tasks related to cybersecurity, such as Named Entity Recognition (NER) on threat intelligence reports, multiple-choice knowledge retrieval, and text classification [41]. While useful for testing academic knowledge, these static methodologies fail to capture the dynamic, interactive nature of real-world vulnerability exploitation and patching [43].

To address this gap, researchers introduced the CyberGym framework, a large-scale, dynamic benchmark featuring 1,507 historically accurate vulnerabilities sourced from 188 real-world software projects [43]. CyberGym forces AI agents to act autonomously, within isolated, i.e., containerized, environments, tasking them with generating

functional proof-of-concept exploits, that successfully reproduce complex vulnerabilities based solely on a high-level text description, and access to the codebase [43]. This framework differentiates an agent's true cybersecurity capabilities from its mere theoretical knowledge [43].

The empirical results from the CyberGym benchmark reveal a stark reality: even the most advanced frontier models struggle significantly with dynamic, multi-step security analysis. We knew that it would be the case [28,44,77,80]. See Table 3.

Global Rank	AI Agent Framework	Base LLM Architecture	CyberGym Success Rate (%)	Evaluation Date
1	Anthropic Agent	**Claude Opus 4.6**	66.6%	Feb 5, 2026 [43]
2	SageAgent	**GPT-5**	60.2%	Feb 9, 2026 [43]
3	Anthropic Agent	**Claude Opus 4.5**	50.6%	Feb 5, 2026 [43]
4	Claude Code	**GLM-5**	43.2%	Feb 12, 2026 [43]
5	Kimi Agent	**Kimi K2.5**	41.3%	Feb 2, 2026 [43]
10	OpenHands	**Claude Sonnet 4**	17.9%	May 23, 2025 [43]
17	OpenHands	**Gemini 2.5 Flash**	4.8%	May 15, 2025 [43]

Table 3 presents the targeted vulnerability reproduction success rates of leading models, and agent frameworks, evaluated on the CyberGym platform. Data sourced from the CyberGym Leaderboard, representing success rates for 1-trial executions of targeted vulnerability reproduction tasks [43].

Anthropic's Claude Opus 4.6 demonstrated superior performance in this adversarial domain, achieving a 66.6% success rate, largely due to its integration of an adaptive think tool. that allows for interleaved, multi-turn reasoning and self-correction during the execution of complex exploit chains [1]. However, the fact that the absolute state-of-the-art system fails to identify, or reproduce, vulnerabilities in 33.4% of cases dictates that AI models cannot be trusted to self-regulate, or autonomously audit, their own security outputs in high-stakes environments [43].

The security crisis is further complicated by the emergence of "mixed-origin errors" [45]. As software development workflows now routinely interweave human-written and LLM-generated code within the same files, and functions, developers are encountering hybrid contexts where human and AI bugs interact in highly unpredictable ways [45]. The Tricky^2 benchmark dataset, which explicitly models error co-occurrence, demonstrates that human attempts at fixing code frequently mask latent LLM faults, while automated LLM repair agents frequently reintroduce human defects that were previously patched [45]. In this highly volatile environment, the necessity for robust, human-in-

the-loop (HITL) checkpoints, mandatory AI security auditing tools, and unyielding architectural governance remains absolute [12]. Again fully aligned with [28,44,77].

4. Explainability And The Paradigm Shift Toward Code Cognition

The final pillar of the quality triad is explainability (XAI), i.e., the capacity of an artificial intelligence system to articulate the internal logic, contextual awareness, and mechanistic rationale behind its generated output [46], and typically also document it accurately, and usefully, in the code. As modern enterprise software architectures grow increasingly complicated to understand, and the cognitive debt associated with autonomous AI generation compounds at an exponential rate, explainability has decisively transitioned from an theoretical luxury to a mandatory operational necessity [47][10]. In highly regulated sectors such as global finance, healthcare informatics, and telecommunications, deploying opaque, multi-billion parameter black box neural networks is functionally untenable, due to severe compliance risks, auditability requirements, and operational liability [11], or the need to correct decisions. Ability to maintain and support software also requires documentation, and ability to understand the code [28,44,61,77].

By early 2026, the artificial intelligence industry has finally recognized this imperative, driving a fundamental paradigm shift from mere code generation, i.e., predicting the next statistically likely token based on syntax patterns, to code cognition, i.e., the semantic understanding of logic flows, the interaction of abstract functions, and the capacity to predict system-wide ripple effects [48]. This cognitive evolution requires models to not only write code rapidly, but to reason through their own logic independently, minimizing silent system failures, and supporting robust, trustworthy deployment [46].

The frontier (LLM) models released in February and March 2026 aggressively targeted the explainability domain. OpenAI's GPT-5.4 introduced deep, integrated Thinking modes, explicitly designed to handle complex reasoning tasks with high transparency, allowing operators to review the chain-of-thought steps and data sources utilized during generation [49][11]. Anthropic's Claude 4.6 architecture integrated highly sophisticated, built-in capabilities for articulating its internal decision-making processes [11]. Claude 4.6 is specifically engineered to identify the limitations in its own knowledge base, express appropriate statistical confidence levels regarding its outputs, and flag uncertainty, mechanisms that are vital for supporting responsible deployment in high-stakes applications [11]. Furthermore, open-weight models such as Mistral Large 3 and DeepSeek-V4 enhanced their instruction-tuning protocols to support transparent chain-of-thought outputs, allowing developers to trace the origin of logical deductions [4]. These open source / open weight models keep up and at time win over the proprietary ones [72,78].

[10] It is also mandated by numerous emerging regulations, for use cases where Ai is involved, and that would tentatively everything as AI start coding everything and software keeps on eating the world. It is acutely required in high stakes and highly regulated industries.

[11] It is also a key feature of application-aware AI [62-69,77].

To empirically quantify and enforce these capabilities, the industry has universally adopted specialized Explainability Assessment Benchmarks [53]. These multifaceted evaluation frameworks, such as the HELM architecture, prioritize testing the ethical, logical, and practical transparency of models, moving beyond the mere measurement of functional syntax execution [53]. See Table 4.

Explainability Evaluation Area	**Description of the Metric Evaluated**	**Primary Industry Benchmark Utilized**
Chain-of-thought Ability	Assessing the logical coherence, consistency, and step-by-step transparency of the model's internal reasoning pathways.	Reveal [54]
Effectiveness of Explanations	Measuring overall user comprehension, pedagogical clarity, and the absence of obfuscating jargon in the AI's provided rationale.	e-SNLI [54]
Tendency of Sycophancy	Evaluating the model's dangerous propensity to falsely agree with the user's input, bypassing objective truth in order to please the human prompt author.	SycophancyEval [54]
Mixed-Origin Error Resolution	Assessing the model's ability to transparently explain and repair environments containing deeply interwoven human and AI-generated bugs.	Tricky^2 2 [45]

Table 4 outlines the primary explainability benchmarks utilized to audit LLMs in the 2026 ecosystem.

The drive for structural explainability has also catalyzed revolutionary advancements in Automated Program Repair (APR) [55]. Early APR research relied heavily on rigid, predefined rule templates, constraint-based analysis, and brute-force genetic programming, which often generated brittle patches that lacked semantic understanding [56]. However, the integration of advanced LLM code cognition has yielded sophisticated methodologies like VibeRepair, i.e., an approach that centers patch generation on explicit behavioral intent rather than localized syntax modification [55][12]. Operating on the philosophical premise of "make the behavior sing, and the code will follow", VibeRepair, running on the GPT-4o architecture, successfully repaired 178 critical bugs on the Defects4J v2.0 benchmark, representing a massive 23% improvement over prior state-of-the-art approaches [55]. Similarly, multimodal APR frameworks such as SVRepair are leveraging advanced visual segmentation techniques to iteratively narrow complex user interface artifacts into bug-centered regions, achieving a 36.47% accuracy rate on

[12] Reminiscent of how the meta-agent is interacted with through higher level intent conversations in Application-aware AI [62-69], and its associated constrained and managed vibe coding and SLDC [77]

the SWE-Bench M evaluation, and proving that AI can reason through complex visual-spatial code relationships [58].

Despite these advancements in automated repair and cognitive reasoning, LLMs still suffer from a fundamental lack of domain knowledge [59]. They require continuous, highly structured prompting, and complex Retrieval-Augmented Generation (RAG) pipelines to maintain an accurate context of the enterprise architecture [59][13].

The future of safe, highly productive, AI software engineering relies not on models, that simply author syntax at inhuman speeds, but on AI agents acting as dedicated cognitive engineers [48]. These agents must be capable of generating robust, human-readable documentation, explaining complex architectural transformations, and interacting safely within volatile, mixed-origin codebases [48,77]. Recent empirical studies on AI-generated documentation indicate that while traditional information retrieval metrics (such as BLEU and ROUGE-L) fail to capture the nuance of software explanations, independent qualitative assessments by human experts found that up to 27.7% of AI-generated code comments might be actually superior, in pedagogical quality, to the original human comments [60]. Crucially, this superiority was only achieved when the LLM was provided with rich contextual information regarding the codebase's cyclomatic complexity, module coupling, and overarching business logic. This confirms that the quality of AI output is closely linked to the quality of human architectural oversight [28,44,60].

The evidence aggregated post-February 2026 conclusively dictates that artificial intelligence is not uniformly better than human engineers at coding. Instead, it has emerged as an extraordinarily powerful, highly volatile force multiplier that fundamentally reorganizes the nature, velocity, and risks of software engineering labor. As the industry advances, the definitive skill of the modern software engineer will no longer be the memorization of syntax, but the mastery of AI orchestration, the management of cognitive debt, and the rigorous enforcement of security heuristics, and a managed SDLC with deep integrated human-machine collaborative systems [28,44]. Or it can be constrained and managed vibe coding and SDLC by an (application-aware AI) agentic platform [62-69,77].

5. Vibe Coding and AI Toll on People

Besides the challenges posed by AI/vibe coding with support, maintenance, security, and explainability of the output code, as discussed so far in this paper, and in [28,44,77], [75,76] report also a toll on the operators of these AI tools, characterized as brain fry, where even expert activities now trend towards work slop, i.e., poor quality or mediocre performances and output. These are additional factors contributing to the failure rates of AI, and agentic AI projects, observed in enterprises [73,74].

Besides the human considerations, it is a major issue, as we just discussed that ensuring viable software produced by unconstrained vibe coding, requires strict oversight by senior developers, who have to implement all the traditional steps of SDLC on the output. With AI interactions, their ability to do so degrade due to brain fry, as does their cognitive control of the production. With Junior developers all these tasks can easily overwhelm them, in addition to prevent them to correctly train towards higher seniority. This does not contribute to increased efficiency of the software development process, but instead, it leads to the frustration of developers, and

[13] Application-aware agentic AI [62-69,77], provides this context is a way that does not have the RAG pipelines limitations and implementation efforts associated them in an enterprise context.

managers. The nonsensical AI-generated memos, pitch decks, presentations, and overly verbose and nebulous code, ends up creating more work for colleagues who have to fix what AI got wrong. This explains and adds to our repeated recommendations of caution against strategies aimed at replacing developers, especially Senior developers[14] with unconstrained vibe coding, without suitably taking care of the full SDLC: it simply does not (yet) work that way.

This really justifies trying to automate better the SDLC, à la VIBE-M [44], or better ways, i.e., where the agentic platform controls and constrains the vibe coding and all the other steps of the SDLC [77], so that interactions by developers or operators, are shielded from work slop, handled by the meta-agent / platform in [62-69,77].

6. Vibe-Coding And SDLC, Managed And Constrained By An Agentic Platform

In [77], we explain how to design, and implement, Vibe-Coding and SDLC Managed and Constrained by An Agentic Platform, with a particular implementation done with Zenera application-aware agentic AI platform [62-69,77]. We also discuss how this ensures the ability to reach 100% correctness of the code, and how explainability, security, supportability and maintainability is intrinsic to the approach and ensured.

It provides a great way forward to addressing the issues we identified in this paper, and in [28,44,77], and references therein. Because the interactions are at the higher level, i.e., intent level, and autonomous, we expect that it can reduce brain fry, minimize work slop, and therefore lead to better efficiency gains, with less negative consequences, for using AI for enterprise intelligence and application development. In fact, as discussed in [67,71], we also predict that it can be a catalysts for the disruption of the entire enterprise application market. It can also address the reasons why AI and agentic AI projects fail so often in enterprises [61-69,73,74,77].

Zenera provides application-aware agentic AI [62-69,77]. For example, [77] provides other criteria to fulfill the pattern, e.g., without the RTDC (Real-Time Discovery and Coding) component. Despite tiny steps in that direction taken by vendors like Anthropic, and Cursor, they are not there yet ([77], and online comments on web page), and we are not aware of other qualifying platform out there.

7. A Note on LLM Evolutions

It is important to understand that, in the long term, LLM are or should become commodities [78-80]. It is already the case for many GenAI use cases. In the case of AI coding assistants, and vibe coding, we still see performance dependencies improvement with each release. Updates are also released rapidly within 6 months, or less; while

[14] Or QA.

the gap between frontier proprietary models and open source/open weight ones, is narrowing, and in general caught-up within 6 less than months [72]. Derived SML also catch at similar or faster rates.

This means that any LLM dependent investment is theoretically outdated within 6 months, although often, it does not matter, or the gap can be filled with improved derived or open source models.

AI-based systems should strive to be LLM agnostic, or at least

1) support immediate switch from one to another

2) control the performances of the system so that minimum degradation results.

Otherwise AI investment would be depreciated every 6 months [72,78]. Zenera implementation achieves that, with a separation of concerns, where of concerns/abstraction of the LLM, of course, but also its own control mechanisms, e.g., through correct models of constraints and ability to coach the AI/agents, which can bring 100% correctness, no matter the LLM, assuming starting from good performances of course [62-69,77].

8. Conclusions

The integration of generative artificial intelligence into the software development lifecycle has impacted the velocity, economics, and underlying mechanics of enterprise engineering. Following the deployment of frontier models in early 2026, the industry has achieved impressive theoretical productivity milestones, saturating historical coding benchmarks, and definitively proving that algorithmic systems possess superhuman capabilities in isolated, synthetic syntax generation. However, this profound illusion of velocity is actively counterbalanced by severe challenges, in terms of codebase maintainability, dynamic cybersecurity, and organizational cognitive health.

One year after the start of the vibe coding craze, and [28], the present review and analysis, confirms the concerns with AI assistants, and vibe coding, in terms of security, support maintenance and explainability, as discussed in [28,44], and the need to associate to the activity, rigorous SDLC processes, that they be manual [28,44], or automated/agentic as in [44,77]. It confirms that unconstrained artificial intelligence coding assistants hyper accelerate the accumulation of technical debt, generating highly verbose, cyclomatically complex logic structures that are exponentially more difficult for human teams to test, debug, and support. Furthermore, the reliance on rapid stochastic generation of code actively destroys the mental theories developers hold regarding their own systems, resulting in a rapid surge of cognitive debt that threatens the long term viability of enterprise software infrastructure; and everything else as software eats everything. AI systems are fundamentally optimized for rapid text completion, ignoring the critical imperatives of architectural abstraction, and explainable code cognition.

Equally critical is the profound human toll exacted by these systems. The continuous requirement to audit and filter massive volumes of algorithmically generated syntax has subjected the engineering workforce to severe AI Brain Fry. This acute cognitive fatigue radically diminishes judgment, altering the availability heuristic and directly catalyzing the mass production of Work Slop. When exhausted operators lack the cognitive stamina to thoroughly inspect verbose algorithmic output, low quality code masquerading as competent engineering infects the main production branches, degrading the code quality itself, and actively undermining enterprise security and reliability. The transition of the developer from a creative Code Author to an exhausted Code Curator represents an unsustainable redistribution of engineering toil that provides no genuine macroeconomic advantage.

To navigate this highly volatile paradigm, enterprises must aggressively reject the naive assumption that AI can simply replace senior engineering staff, or QAs. Terminating quality assurance teams and senior architects to achieve immediate financial returns based on the theoretical speed of vibe coding is a catastrophic strategy. Until fully autonomous algorithms are perfected, elite human cognition remains strictly necessary to orchestrate complex architectural strategies, navigate dynamic cybersecurity threats, and meticulously govern the algorithmic output.

Application-aware-AI agentic management and constrain of vibe coding and of the associated full SDLC, ends up being the best recommendation for enterprises to take advantage of AI to increase software development efficiencies [62-69,77]. It works Today, and it may be the way forward. Zenera offer such a platform [68,69].

____

References

[1]: Anthropic, (2026), “Introducing Claude Opus 4.6 – Anthropic”, https://www.anthropic.com/news/claude-opus-4-6, February 5, 2026.

[2]: David Gewirtz, (2026), “OpenAI's new GPT-5.4 clobbers humans on pro-level work in tests - by 83%”, ZDNET, https://www.zdnet.com/article/openai-gpt-5-4/, March 5, 2026.

[3]: Google, (2026), “Release notes”, Gemini API - Google AI for Developers, https://ai.google.dev/gemini-api/docs/changelog, March 10, 2026.

[4]: Mistral, (2026), “Introducing Mistral 3”, Mistral AI, https://mistral.ai/news/mistral-3. Retrieved on Marh 7, 2026.

[5]: Wikipedia, “Llama (language model)”, https://en.wikipedia.org/wiki/Llama_(language_model). Retrieved on March 7, 2026.

[6]: Greek Ai, (2026), “Stop Everything — DeepSeek V4 Might Be the Smartest Coding AI of 2026”, GoPenAI, Medium, https://blog.gopenai.com/stop-everything-deepseek-v4-might-be-the-smartest-coding-ai-of-2026-c8abda20b7b8, February 10, 2026.

[7]: Nihal Kumar Kadri, (2026), “AI Is Not Replacing Software Engineers: It Is Redefining Them”, Towards AI, https://pub.towardsai.net/ai-is-not-replacing-software-engineers-it-is-redefining-them-0929e96c6a2c, March 1, 2026.

[8]: Turing College, (2026), “GPT-5.4 Review: Is It Worth Leaving GPT-5.3 Codex Behind?”, https://www.turingcollege.com/blog/gpt-5-4-review-vs-gpt-5-3-codex, March 7, 2026.

[9]: Mark Levison, (2026), “AI-Generated Code Quality and the Challenges we all face”, Agile Pain Relief, https://agilepainrelief.com/blog/ai-generated-code-quality-problems/, February 2026.

[10]: Prasenjit Sarkar, (2025), “New data on code quality: GPT-5.2 high, Opus 4.5, Gemini 3, and more”, Sonar, https://www.sonarsource.com/blog/new-data-on-code-quality-gpt-5-2-high-opus-4-5-gemini-3-and-more/, December 15, 2025.

[11]: Satyadhar Joshi, (2026), “Architectural Advances and Performance Benchmarks of Large Language Models in Light of Anthropic's Claude Opus 4.6”, Preprints.org, https://www.preprints.org/manuscript/202602.0537, February 6, 2026.

[12]: Konstantin Karpushin, (2026), “The Hidden Costs of AI-Generated Software: Why “It Works” Isn't Enough”, Codebridge, https://www.codebridge.tech/articles/the-hidden-costs-of-ai-generated-software-why-it-works-isnt-enough, February 3, 2026.

[13]: Addy Osmani, (2025), “My LLM coding workflow going into 2026”, Medium, https://medium.com/@addyosmani/my-llm-coding-workflow-going-into-2026-52fe1681325e, December 19, 2025.

[14]: Saqib Shah, (2026), “AI Technical Debt: The Hidden Cost of “Fast” Code in 2026”, Medium, https://medium.com/@saqibshahdev/ai-technical-debt-the-hidden-cost-of-fast-code-in-2026-75c1d85eb3b4, March 5.

[15]: LXT, (2025), “LLM Benchmarks Explained: What Each One Measures and How to Choose for Your Use Case, (2026)”, https://www.lxt.ai/blog/llm-benchmarks/, November 06, 2025.

[16]: Chizaram Ken, (2026), “AI dev tool power rankings & comparison [Feb. 2026]”, LogRocket Blog, https://blog.logrocket.com/ai-dev-tool-power-rankings/, March 12, 2026.

[17]: Nicolas Zeeb, “Claude Opus 4.6 vs 4.5 Benchmarks (Explained)”, Vellum, https://www.vellum.ai/blog/claude-opus-4-6-benchmarks, February 6, 2026 (no more available online, copy at: https://web.archive.org/web/20260206221312/https://www.vellum.ai/blog/claude-opus-4-6-benchmarks).

[18]: Sonar, “LLM Leaderboard for Code Quality & Security – Sonar”, https://www.sonarsource.com/the-coding-personalities-of-leading-llms/leaderboard/. Retrieved on March 7, 2026.

[19]: Anthropic, (2026), “Claude Opus 4.6 System Card”, https://www-cdn.anthropic.com/14e4fb01875d2a69f646fa5e574dea2b1c0ff7b5.pdf, February 6, 2026.

[20]: Koray Kavukcuoglu, (2026), “Gemini 2.0 is now available to everyone”, Google Blog, https://blog.google/innovation-and-ai/models-and-research/google-deepmind/gemini-model-updates-february-2025/, February 5, 2026.

[21]: Blake Crosley , (2026), “DeepSeek V4 Targets Coding Dominance with Mid-February Launch”, Introl, https://introl.com/blog/deepseek-v4-february-2026-coding-model-release, January 14, 2026.

[22]: BarbaraSchwarz, (2026), “Deepseek V4 - All Leaks and Infos for the Release Day - Not Verified!”, Reddit, https://www.reddit.com/r/DeepSeek/comments/1ridmnm/deepseek_v4_all_leaks_and_infos_for_the_release/, March 1, 2026.

[23]: Venusverse, (2025), “Llama 4 Maverick”, AI Model Guide, https://venusverse.in/ai-models/llama-4. Retrieved on March 7, 2026.

[24]: Maxim Saplin, (2025), “Llama 4 - 10M Context? Coding? Decent Follow-up?”, DEV Community, https://dev.to/maximsaplin/llama-4-10m-context-coding-decent-follow-up-426n, April 8, 2025.

[25]: Vertu, (2026), “DeepSeek V4 Technical Preview: The Next Milestone in Code Intelligence / DeepSeek V4 Guide: 90% HumanEval & 1M+ Token Code Context”, https://vertu.com/lifestyle/deepseek-v4-technical-preview-the-next-milestone-in-code-intelligence/, January 26, 2026.

[26]: WaveSpeed, (2026), “DeepSeek V4: Everything We Know About the Upcoming Coding AI Model”, WaveSpeed.ai, https://wavespeed.ai/blog/posts/deepseek-v4-everything-we-know-about-the-upcoming-coding-ai-model/, January 19, 2026.

[27]: Valerie Chen, Jasmyn He, Behnjamin Williams, Jason Valentino, Ameet Talwalkar, (2026), “Beyond the Commit: Developer Perspectives on Productivity with AI Coding Assistants”, arXiv:2602.03593v1.

[28]: Stephane H. Maes, (2025), “The Gotchas of AI Coding and Vibe Coding. It’s All About Support And Maintenance”, https://doi.org/10.5281/zenodo.15343349, https://shmaes.wordpress.com/2025/04/28/the-gotchas-of-ai-coding-and-vibe-coding-its-all-about-support-and-maintenance/, April 28, 2025, (https://osf.io/kjz9t/download/).

[29]: Prasenjit Sarkar, (2026). “The great toil shift: How AI is redefining technical debt”, Sonar, https://www.sonarsource.com/blog/how-ai-is-redefining-technical-debt/, February 12, 2026.

[30]: Chi Zhang, Zehan Li, Ziqian Zhong, Haibing Ma, Dan Xiao, Chen Lin, Ming Dong, (2026), “From Horizontal Layering to Vertical Integration: A Comparative Study of the AI-Driven Software Development Paradigm”, arXiv:2601.22667v1.

[31]: Michael Tridi, (2026), “The Impact of AI Coding in 2026: Developer Productivity Revolution with 90% AI-Generated Code”, Trigi Digital, https://trigidigital.com/blog/ai-coding-impact-2026, ARC Prize 2025 Results and Analysis, January 28, 2026.

[32]: Mike Knoop, (2025), “ARC Prize 2025 Results and Analysis. Year of the Refinement Loop”, Arc Prize, https://arcprize.org/blog/arc-prize-2025-results-analysis, December 5, 2025.

[33]: Jose Enrico, (2025), “Human Just Barely Beats OpenAI's AI in Historic Coding Contest. The next time you use ChatGPT, always be proud that AI can't still replace you.”, Tech Times, https://www.techtimes.com/articles/311429/20250721/human-just-barely-beats-openais-ai-historic-coding-contest.htm, July 21, 2025.

[34]: Winged IT, (2025), “Polish Programmer Wins Over AI in AtCoder World Tour Finals 2025”, https://wingedit.pl/events-2/polish-programmer-wins-over-ai-in-atcoder-world-tour-finals-2025/, July 18, 2025.

[35]: Sue Gee, (2025), “Human Programmer Outwits OpenAI's o3”, i-programmer.info, https://www.i-programmer.info/news/105-artificial-intelligence/18195-human-programmer-outwits-openais-o3.html, July 23, 2025.

[36]: Margaret-Anne Storey, (2026), “How Generative and Agentic AI Shift Concern from Technical Debt to Cognitive Debt”, https://margaretstorey.com/blog/2026/02/09/cognitive-debt/, February 9, 2026.

[37]: Sonar, “LLM Leaderboard for Code Complexity”, https://www.sonarsource.com/the-coding-personalities-of-leading-llms/leaderboard/complexity, Retrieved on March 5, 2026.

[38]: Prasenjit Sarkar, (2025), “The Coding Personalities of Leading LLMs—GPT-5 Update”, Sonar, https://www.sonarsource.com/blog/the-coding-personalities-of-leading-llms-gpt-5-update/, August 27, 2025.

[39]: Ken Judy, (2025), Reducing AI Code Debt: A Human-Supervised PDCA Framework for Sustainable Development”, Agile Alliance, https://agilealliance.org/reducing-ai-code-debt/, August 21, 2025.

[40]: Ken Judy, Ben Linders, (2025), “A Plan-Do-Check-Act Framework for AI Code Generation”, InfoQ, https://www.infoq.com/articles/PDCA-AI-code-generation/, October 20, 2025

[41]: GitHub, “CyberBench: A Multi-Task Cyber LLM Benchmark”, https://github.com/jpmorganchase/CyberBench. Retrieved on March 7, 2026.

[42]: Jie Zhang, Haoyu Bu, Hui Wen, Yu Chen, Lun Li, Hongsong Zhu, (2024), “When LLMs Meet Cybersecurity: A Systematic Literature Review”, arXiv:2405.03644v1.

[43]: Zhun Wang, Tianneng Shi, Jingxuan He, Matthew Cai, Jialin Zhang, Dawn Song, “Evaluating AI Agents' Real-World Cybersecurity Capabilities at Scale”, CyberGym, https://www.cybergym.io/. Retrieved on March 2, 2026.

[44]: Stephane H. Maes, (2025), “Ensuring the Maintainability and Supportability of “Vibe-Coded” Software Systems: A Framework for Bridging Intuition and Engineering Rigor”, https://doi.org/10.5281/zenodo.15354102, https://shmaes.wordpress.com/2025/05/06/ensuring-the-maintainability-and-supportability-of-vibe-coded-software-systems-a-framework-for-bridging-intuition-and-engineering-rigor/, May 6, 2025, (https://osf.io/2nu8r/download/).

[45]: Cole Granger, Dipin Khati, Daniel Rodriguez-Cardenas, Denys Poshyvanyk, (2026), “Tricky²: Towards a Benchmark for Evaluating Human and LLM Error Interactions”, arXiv:2601.18949v1.

[46]: Markus Borg, (2025). “Human, What Must I Tell You?”, CodeScene, https://codescene.com/blog/human-what-must-i-tell-you, September 23, 2025.

[47]: Anushree Chatterjee, (2025), “Explainability Techniques for LLMs & AI Agents: Methods, Tools & Best Practices”, testRigor, https://testrigor.com/blog/explainability-techniques-for-llms-ai-agents/, October 9, 2025.

[48]: Anil K Shukla, (2025), “From Code Generation to Code Cognition: The Quiet Evolution of AI Reasoning”, Medium, https://medium.com/@shuklaks/from-code-generation-to-code-cognition-the-quiet-evolution-of-ai-reasoning-75c786a65c50, September 30, 2025.

[49]: SourceForge, “Generate Best GPT-5.4 Thinking Alternatives & Competitors”, https://sourceforge.net/software/product/GPT-5.4-Thinking/alternatives. Retrieved on March 7, 2026.

[50]: Rasiksuhail, (2026), “I Built an AI Content Release Optimizer in 4 Hours With GPT 5.4”, https://pub.towardsai.net/i-built-an-ai-content-release-optimizer-in-4-hours-with-gpt-5-4-d5c20e583fdc, March 7, 2026.

[51]: Andrew Dyuzhov, (2025), “What is The Best AI Model in 2025/2026?”, AI Hub, https://overchat.ai/ai-hub/the-best-ai-model, December 3, 2025.

[52]: Lumenova AI Research Team, (2025), “Cognitive Capability Test: Claude vs GPT-5 vs Gemini (Part III)”, https://www.lumenova.ai/ai-experiments/frontier-ai-cognitive-test-claude-gpt5-gemini-part3/, November 7, 2025.

[53]: Lamatic.ai, (2024), “25 Best LLM Benchmarks to Test AI Models for Reliable Results”, Lamatic Labs, https://labs.lamatic.ai/p/llm-benchmarks/, December 2, 2024.

[54]: Xin Guan, (2024), “Navigating the LLM Benchmark Boom: A Comprehensive Catalogue”, Holistic AI, https://www.holisticai.com/blog/navigating-llm-benchmark, July 1, 2024.

[55]: Taohong Zhu, Lucas C. Cordeiro, Mustafa A. Mustafa, Youcheng Sun, (2026), “Specification Vibing for Automated Program Repair”, arXiv:2602.08263v1.

[56]: Qingxiao Xu, Ze Sheng, Zhicheng Chen, Jeff Huang, (2026), “A Systematic Study of LLM-Based Architectures for Automated Patching”, arXiv:2603.01257v1.

[57]: Quanjun Zhang, Chunrong Fang, Yuxiang Ma, Weisong Sun, Zhenyu Chen, (2023), “A Survey of Learning-based Automated Program Repair”, arXiv:2301.03270v3.

[58]: Xiaoxuan Tang, Jincheng Wang, Liwei Luo, Jingxuan Xu, Sheng Zhou, Dajun Chen, Wei Jiang, Yong Li , (2026), “SVRepair: Structured Visual Reasoning for Automated Program Repair”, arXiv:2602.06090v1.

[59]: Yuzhi Wang, (2025), “A Review of Research on AI-Assisted Code Generation and AI-Driven Code Review”, https://www.researchgate.net/publication/398105387_A_Review_of_Research_on_AI-Assisted_Code_Generation_and_AI-Driven_Code_Review, November 2025.

[60]: Ian Guelman, Arthur Gregório Leal, Laerte Xavier, Marco Tulio Valente, (2024-2025), “On the Quality of AI-Generated Source Code Comments: A Comprehensive Evaluation, arXiv:2408.14007v2.

[61]: Stephane H Maes, Ramu Sunkara, (2026), "Vibe Coding = More Code. Is it good, though?", Zenera AI, March 5, 2026 (+ LinkedIn post).

[62]: Zenera, (2026), “The Enterprise AI Agent Factory. Your enterprise is unique. Your AI agents should be too. The Enterprise AI Problem, Solved”, https://zenera.ai. Retrieved on March 5, 2026.

[63]: Stephane H. Maes, et al. (2026), US Patent Application, "AI System and Method of Meta-Agent and Application-Aware AI, Including Real-Time Discovery and Coding, for Deterministic Constraint Modeling, and Runtime Evolution of Software Applications", 2026.

[64]: Stephane H. Maes, "Agentic Smart ITIL, And The Disruption Of The Market Of Conventional Enterprise Applications", March 1, 2026.

[65]: Stephane H Maes, (2026), “The Era of Application-Aware AI”, Zenera, February 2026.

[66]: Stephane H. Maes, (2026), “Achieving Zero-Effort, Quasi-Zero Cost Integration with Zenera RTDC, and Application-Aware AI”, Zenera, February 2026.

[67]: Stephane H. Maes, (2026), “Agentic AI, The Obsolescence of The Enterprise Application & Zenera, The Catalyst”, February 2026.

[68]: Zenera, (2026), “Zenera: From Conversation to Production in Minutes”, YouTube, https://zenera.ai/introvideo. Retrieved on March 1, 2026.

[69]: Zenera, “Enterprise Analytics Reimagined, Build live reports and dashboards in minutes – no pipelines, no coding, https://zenera.ai/zeneraanalytics. Retrieved on March 5, 2026.

[70]: Ramu Sunkara, Stephane H. Maes, (2026), “Agentic AI Platform with Agent Rewind, and Reliable execution”, Zenera Newsletter, March 2026.

[71]: Stephane H. Maes, Ramu Sunkara, (2026), “The death of SAAS and the enterprise apps: an application of the strangler fig pattern”, LinkedIn, March 4, 2026.

[72]: Ramu Sunkara, Stephane H. Maes, (2026),“The '6-Month Gap': Why your AI strategy is already outdated”, Zenera Newsletter + LinkedIn post, February 11, 2026.

[73]: Aditya Challapally, Chris Pease, Ramesh Raskar, Pradyumna Chari, "The GenAI Divide – State of AI in Business 2025", MIT NANDA, July 2025.

[74]: Gartner, "Gartner Predicts Over 40% of Agentic AI Projects Will Be Canceled by End of 2027", June 25, 2025.

[75]: Allison Morrow, (2026), “AI is exhausting workers so much, researchers have dubbed the condition ‘AI brain fry’”, CNN, https://www.cnn.com/2026/03/13/business/ai-brain-fry-nightcap?cid=android_app, March 13, 2026.

[76]: Matthew Kropp, Megan Hsu, Olivia T. Karaman, Jason Hawes and Gabriella Rosen Kellerman, (2026), “When Using AI Leads to “Brain Fry”. A new study finds that certain patterns of AI use are driving cognitive fatigue, while others can help reduce burnout.”, Harvard Business Review, https://hbr.org/2026/03/when-using-ai-leads-to-brain-fry, March 5, 2026.

[77]: Stephane H. Maes, (2026), “Vibe-Coding and SDLC Constrained And Managed By An Application-Aware AI-Like Agentic Platform”, https://zenodo.org/doi/10.5281/zenodo.18972674, https://shmaes.wordpress.com/2026/03/11/vibe-coding-and-sdlc-constrained-and-managed-by-an-application-aware-ai-like-agentic-platform/, March 11, 2026. (https://osf.io/vz6jt/files/yv2xf).

[78]: Stephane H. Maes, (2025), “The Circle of Life for LLMs. Was the Reaction to DeepSeek Justified?”, https://zenodo.org/doi/10.5281/zenodo.14838733, https://shmaes.wordpress.com/2025/02/15/the-circle-of-life-for-llms-was-the-reaction-to-deepseek-justified/, February 5, 2025. (osf.io/j2vsz_v1, vixra:2503.0009v1).

[79]: Stephane H. Maes, (2024), “Fixing Reference Hallucinations of LLMs”, https://doi.org/10.5281/zenodo.14543939, https://shmaes.wordpress.com/2024/11/29/fixing-reference-hallucinations-of-llms/, November 29, 2024. (osf.io/u38w4/, viXra:2412.0149v1).

[80]: Stephane H. Maes, (2024), “The Trouble with GenAI: LLMs are still not any close to AGI. They will never be”, https://zenodo.org/doi/10.5281/zenodo.14567206, https://shmaes.wordpress.com/2024/12/26/the-trouble-with-genai-llms-are-still-not-any-close-to-agi-they-will-never-be/, December 25, 2024. (osf.io/qdaxm/, viXra:2501.0015v1).

[81]: Stephane H. Maes, Karan Singh Chhina, Guillaume Dubuc, (2021), “Natural language translation-based orchestration workflow generation”, US Patent 11120217.

[82]: Wikipedia, “Vibe coding”, https://en.wikipedia.org/wiki/Vibe_coding. Retrieved on March 2, 2026.