Reimagining Software Engineering Interviews in the AI Era: Beyond LeetCode Toward Real-World Competency Assessment

Lalitha A R
24f2006078@ds.study.iitm.ac.in

December 19, 2025

Abstract

Traditional technical interviews for software engineering positions have relied heavily on algorithmic problem-solving assessments, commonly referred to as "LeetCode-style" interviews, since their standardization in the 1990s. Recent advances in large language models (LLMs) have disrupted this paradigm by demonstrating substantial capability on algorithmic challenges, prompting industry reconsideration of assessment validity. This paper synthesizes empirical evidence from software engineering research, psychometric studies, learning science, and industry practice to examine whether algorithmic interviews remain effective predictors of job performance in the AI era. We present five converging lines of evidence: (1) LLMs achieve 31.68–43.36% pass rates on hard LeetCode problems, with performance reaching the 63rd percentile of human submissions; (2) empirical data from 131 candidates show zero correlation ($r = 0.0$, $p > 0.05$) between weekly LeetCode preparation hours and interview readiness; (3) historical analysis reveals the format originated from 1980s–1990s practical constraints rather than pedagogical validation; (4) psychometric research demonstrates that isolated cognitive tests exhibit validity coefficients of 0.31–0.42, substantially lower than multimethod approaches combining work samples and structured interviews (0.63–0.65); and (5) production software engineering tasks—legacy code maintenance, collaborative debugging, architectural design—diverge fundamentally from isolated algorithmic puzzles. These findings suggest that the traditional interview format measures pattern-matching capabilities that AI systems increasingly replicate, rather than the complex judgment, collaboration, and contextual reasoning that characterize professional software development. We discuss implications for educators, employers, candidates, and propose evidence-based directions for interview redesign that emphasize authentic task simulation and multimethod assessment.

1 Introduction

The technical interview has become a defining ritual in software engineering hiring, with algorithmic problem-solving assessments serving as the primary mechanism for evaluating candidate competency across major technology companies. Candidates invest substantial resources preparing for these interviews—averaging 6.5 hours per week on platforms like LeetCode [1]—under the assumption that performance on data structure and algorithm challenges predicts job success. However, recent developments in artificial intelligence have exposed fundamental questions about this assessment paradigm. When the design platform Canva announced in June 2025 that it would not only permit but actively encourage AI tool use during technical interviews [7], the decision signaled a broader industry recognition: traditional algorithmic interviews may no longer effectively differentiate human capabilities from machine performance.

This disruption arrives at a moment when the software engineering profession faces significant transformation. Industry reports indicate severe shortages of mid-level to senior engineers capable of system architecture, team leadership, and navigating complex legacy codebases [24], yet interview processes continue to emphasize skills—algorithmic puzzle-solving under time pressure—that bear limited resemblance to professional practice [22, 23]. The disconnect between assessment and job requirements is not novel; practitioners have long questioned whether whiteboard coding predicts real-world performance [10]. What distinguishes the current moment is the convergence of empirical evidence documenting this mismatch across multiple domains: software engineering research, psychometric validity studies, learning science, and now, direct measurement of AI capability on the same problems used to evaluate human candidates.

Recent empirical research provides quantitative evidence of the assessment gap. A survey of 131 software engineering candidates with an average of 9 years industry experience found that weekly hours spent on LeetCode preparation bore no significant relationship to perceived interview readiness ($r = 0.0$, $p > 0.05$), while communication practice and peer mock

interviews yielded approximately twice the readiness improvement [1]. Simultaneously, evaluation of 18 large language models on 2,100 LeetCode problems demonstrates that frontier AI systems achieve pass rates of 31.68–43.36% on first attempts, reaching 67.79% with multiple attempts, and produce code that executes faster than 63% of human submissions [3]. These parallel developments—empirical evidence of weak preparation-to-readiness correlation in humans and substantial AI capability on the same tasks—suggest that algorithmic interviews may measure abilities increasingly replicable by machines rather than distinctively human software engineering competencies.

This paper addresses a critical gap in the literature: while fragmented discussions of interview effectiveness exist across practitioner communities, academic software engineering education research, and psychometric validity studies, no comprehensive synthesis has examined the traditional interview format through the lens of AI capability, learning science, predictive validity, and professional task analysis. We argue that algorithmic interviews persist not because of demonstrated effectiveness but due to historical contingency—the format emerged from 1980s–1990s practical constraints when portable computing was prohibitively expensive [10, 11]—and became institutionalized through cultural momentum rather than empirical validation. The arrival of AI systems capable of solving the same problems exposes this foundation: if the assessment was valid, machine capability should correspond to production software engineering mastery, yet evidence consistently shows AI struggles with design, architectural reasoning, legacy code comprehension, and stakeholder communication [26, 27].

Our contribution is threefold. First, we synthesize evidence from diverse research domains—software engineering, psychometrics, learning science, AI capability assessment—to provide a unified analysis of interview validity in the AI era. Second, we employ both statistical evidence and logical argument (reductio ad absurdum) to demonstrate that algorithmic puzzle-solving is neither necessary nor sufficient for software engineering competency. Third, we provide practical guidance for stakeholders: educators seeking to align curricula with

professional practice, employers designing more predictive assessments, candidates allocating preparation resources, and platforms developing interview preparation tools. While we focus on algorithmic interviews due to their prevalence, our analysis applies broadly to any assessment format that prioritizes isolated technical performance over authentic simulation of professional contexts.

The remainder of this paper proceeds as follows. Section 2 examines the historical evolution of technical interviews, documenting how the current format arose from contingent circumstances rather than deliberate design. Section 3 analyzes LLM performance on algorithmic challenges and the implications for assessment validity. Section 4 presents empirical evidence on the relationship between preparation methods and interview outcomes. Section 5 reviews psychometric research on predictive validity of selection methods. Section 6 compares interview tasks with actual software engineering work. Section 7 surveys emerging industry practices. Section 8 synthesizes these findings and discusses implications for assessment redesign.

2 Historical Evolution and Contingency of Interview Formats

Understanding why algorithmic interviews dominate current practice requires examining their historical origins. The standard narrative positions these assessments as deliberate designs for evaluating technical competency, yet evidence suggests the format emerged from practical constraints that have long since disappeared.

2.1 Origins in Pre-Computing Era Constraints

Technical interviews in software engineering evolved from earlier practices in mathematics and theoretical computer science, where candidates demonstrated problem-solving ability on chalkboards—the primary collaborative tool in academic settings [11]. When the software

industry began formalizing hiring practices in the 1980s, portable computing technology remained expensive and logistically complex [10]. As one industry retrospective notes: "Back in the 1980s and earlier, portable computers were large and expensive. Bringing a computer along to a candidate interview wasn't practical. Instead, testing software developers using pen and paper was the norm" [10]. The transition to whiteboards in the 1990s represented a convenience improvement—they were "more convenient than paper; easy collaboration; already in classrooms/boardrooms" [10]—rather than a pedagogically motivated innovation.

Microsoft's interview process during the 1980s–1990s played a particularly influential role in standardizing algorithmic problem-solving as the primary assessment mechanism [11]. The company developed what contemporaries described as a "rigorous" process emphasizing coding ability and algorithmic thinking, establishing patterns that subsequent technology companies adopted. However, this standardization occurred before substantial research on the predictive validity of such methods existed. The format's adoption reflected pragmatic considerations—whiteboards were ubiquitous in office environments—and institutional isomorphism as other companies emulated practices of successful firms, not empirical evidence of superior predictive power.

2.2 Persistence Despite Technological Change

A critical observation is that whiteboard-based algorithmic interviews persisted well into the 2000s and beyond, despite fundamental changes in technological availability and professional practice. By the early 2000s, laptop computers were commonplace, integrated development environments had matured, and software engineering had evolved from individual programming to complex collaborative development using version control, continuous integration, and distributed systems [12]. Yet interview formats remained largely unchanged. Practitioner discussions attribute this persistence to several factors: ease of setup compared to provisioning development environments, symbolic value as a collaborative problem-solving exercise, and simple institutional inertia [13]. One retrospective notes that the format con-

tinued "despite compute availability; never formally redesigned" [10].

This historical contingency is consequential. If the algorithmic interview format had been designed based on systematic job analysis and predictive validity research, its persistence might be justified by evidence. Instead, the format represents a frozen accident—a solution to 1980s–1990s logistical constraints that became institutionalized through cultural momentum. Understanding this origin is essential for contemporary debates: resistance to format changes often appeals to tradition ("this is how technical hiring has always been done"), but tradition itself arose from circumstances that no longer apply.

2.3 Implications for Current Practice

The historical analysis establishes an important baseline: the algorithmic interview format lacks a foundation in empirical validation of its predictive validity. This is not to claim the format is necessarily ineffective—effectiveness is an empirical question we address in subsequent sections—but rather that its widespread adoption cannot be justified by appeal to evidence-based design. As software engineering has evolved to encompass collaborative development, complex system integration, legacy code maintenance, and increasingly, AI-augmented workflows, the gap between assessment format (designed for 1980s–1990s constraints) and job requirements has widened. The question is no longer whether the historical format was reasonable given its context, but whether it remains valid given contemporary realities.

3 Large Language Model Capability and the Trivialization Thesis

The emergence of large language models with substantial coding capabilities represents the most visible disruption to traditional technical interviews. Understanding the scope and limitations of this capability is essential for evaluating whether algorithmic assessments remain

differentiating.

3.1 Quantitative Performance on Algorithm Benchmarks

Recent systematic evaluation of LLM performance on LeetCode problems provides concrete data on AI capabilities. Wang et al. [3] evaluated 18 language models on a dataset of 2,100 LeetCode problems spanning Easy, Medium, and Hard difficulty levels. Table 1 summarizes key results.

Table 1: LLM Performance on LeetCode Benchmark (2,100 Problems)

Model	**Pass@1 (%)**	**Pass@10 (%)**	**Notes**
Canonical Solutions	97.94	98.04	Expert-written reference
GPT-4-omni	43.36	61.95	Frontier model
GPT-4	31.68	67.79	Widely used
GPT-4-turbo	31.25	50.00	Speed-optimized
Copilot	8.09	19.12	Integrated in IDEs
CodeLlama-13B	4.17	14.71	Open-source

Several patterns emerge from these data. First, frontier models achieve substantial success rates: GPT-4 solves approximately one-third of problems on first attempt, increasing to two-thirds with multiple attempts. This performance substantially exceeds random chance and demonstrates genuine problem-solving capability. Second, performance varies significantly by model class: widely-deployed tools like GitHub Copilot show much lower success rates (8.09% pass@1), suggesting that capability is not uniform across AI systems. Third, when LLMs do produce correct solutions, their code executes faster than 63% of human submissions [3], indicating not just correctness but competitive efficiency.

However, these data also reveal limitations. Even the best-performing models fail on approximately 32–56% of problems (calculating failure rate from pass@1 and pass@10 bounds), and performance on "Hard" difficulty problems is substantially lower than on "Easy" or "Medium" problems. Additionally, contemporary research on benchmark contamination suggests that reported performance may overestimate generalization. Studies comparing

model performance on problems released before versus after training cutoff dates document significant performance drops on temporally held-out problems, suggesting that apparent mastery may partially reflect training data overlap rather than genuine reasoning capability [29, 30].

3.2 Context-Dependency and Production Software Engineering

While LLM capability on isolated algorithmic challenges is substantial, evidence suggests this capability does not transfer uniformly to production software engineering tasks. Analysis of AI limitations in real-world development contexts identifies several systematic gaps [26–28]. AI code generation tools are described as "built to produce plausible code, not production-grade software" [26], with particular weaknesses in requirements analysis, architectural design, and documentation—phases that constitute substantial portions of professional software development yet are absent from algorithmic interviews.

Empirical studies of AI impact on developer productivity provide further context. A randomized controlled trial with 96 Google engineers found that AI-enhanced development tools improved completion speed by approximately 21% on isolated coding tasks (95% CI: -0.51 to 0.03 on log scale; $\beta = -0.24$, $p = 0.038$) [5]. However, a separate RCT examining frontier AI tools (Claude, GPT-4o) in real-world project contexts during February–June 2025 found that AI access actually *increased* completion time by 19% compared to baselines [6]. This discrepancy—21% speedup on controlled tasks versus 19% slowdown in production—highlights the context-dependency of AI benefit. Production software engineering involves coordination overhead, context-switching between tasks, integration with existing systems, and communication with stakeholders, none of which are present in isolated algorithmic challenges.

The challenges AI faces in production contexts mirror the skills absent from algorithmic interviews: "Data quality and availability: Incomplete or biased datasets reduce AI reliability and increase costs. Integration complexity: Legacy systems and multiple toolchains create compatibility bottlenecks and delay deployment. Scalability issues: Models perform well in

controlled tests but fail at production scale with high-volume systems" [27]. This suggests a structural mismatch: if AI trivializes algorithmic interviews yet struggles with production tasks, then algorithmic interviews measure capabilities that are increasingly automatable while omitting capabilities that remain distinctively human.

3.3 Implications for Interview Validity

The LLM performance data support a more nuanced claim than simple "trivialization." AI systems demonstrate substantial but imperfect capability on algorithmic challenges: frontier models solve 32–68% of problems, not 95–100%. However, this partial capability is sufficient to undermine the discriminant validity of algorithmic interviews—the ability to distinguish between candidates with different levels of competency. Consider a scenario: if a candidate can access an AI system (whether through memorization, pattern recognition, or actual tool use) that achieves 67% success on hard problems, the interview no longer reliably measures the candidate's independent problem-solving capability. This concern motivates emerging industry practices of explicitly permitting AI tool use during interviews [7, 8], effectively conceding that the assessment should evaluate AI-augmented performance rather than attempting to isolate human-only capability.

More fundamentally, the divergence between AI algorithmic capability and AI production software engineering capability suggests that algorithmic interviews measure the wrong construct. A reductio ad absurdum argument clarifies this point: (1) If algorithmic puzzle-solving mastery corresponds to software engineering competency, then (2) AI systems that master algorithmic puzzles should demonstrate production software engineering competency. (3) AI systems achieve 32–68% success on algorithmic puzzles. (4) Yet AI systems struggle systematically with production software engineering tasks (design, legacy integration, stakeholder communication). (5) Therefore, algorithmic puzzle-solving mastery does not correspond to software engineering competency. The premises are well-supported by the evidence reviewed above; the conclusion follows logically. This argument does not claim algorithmic

skills are irrelevant to software engineering—they clearly have some relationship—but rather that they are insufficient and may not even be necessary for many professional contexts.

4 Empirical Evidence on Preparation Methods and Outcomes

If algorithmic interviews validly measured software engineering competency, preparation for such interviews should correlate with professional capability and job readiness. Recent empirical research allows us to test this hypothesis.

4.1 The Bell et al. Study: Design and Findings

The most comprehensive empirical investigation of technical interview preparation examined 131 software engineering candidates, comprising both professionals (with an average of 9 years industry experience, median 7 years) and students [1]. The study design used an online survey to assess preparation methods, resource usage, effectiveness perceptions, and the role of computing education. Of the 131 participants, 82.4% (n=108) had completed at least one technical interview, with participants averaging 3 interviews each, providing substantial relevant experience for evaluating preparation effectiveness.

The study's central findings challenge common assumptions about interview preparation:

1. *Volume of algorithmic practice shows no correlation with readiness.* Participants reported spending an average of 6.5 hours per week (median 4 hours) on LeetCode-style preparation. However, correlation analysis found no significant relationship between weekly preparation hours and self-reported interview readiness ($r = 0.0$, $p > 0.05$) [1]. Similarly, the number of daily LeetCode problems completed showed no significant effect on perceived preparedness.

2. *Communication practice and peer collaboration show strong positive effects.* Candidates

who reported more frequent communication practice and training with peers showed approximately 2× higher perceived preparedness compared to those who did not engage in such activities [1]. This effect size was statistically significant and substantially larger than any effect from increased coding practice volume.

3. *Candidates recognize problem-solving as critical but prepare in ways inconsistent with this recognition.* Over 85% of participants identified problem-solving as a key skill for technical interviews [1], yet the majority allocated preparation time to isolated coding drills rather than collaborative problem-solving in authentic contexts.

The study's qualitative findings provide additional context: "Most candidates feel their preparation methods do not prepare them for technical interviews. However, candidates who reported practicing communication more frequently as well as training with others reported significantly higher perceived preparedness—while those who reported completing more daily LeetCode problems and studying more hours per week did not" [1]. Additionally, participants consistently reported that their computing education was unsupportive of interview preparation, suggesting a disconnect between academic curricula and industry hiring practices.

4.2 Interpretation and Mechanisms

The null correlation between preparation volume and readiness ($r = 0.0$) is a striking finding that requires explanation. Several mechanisms may contribute:

Misalignment between practice format and interview requirements. LeetCode preparation typically involves individual work on isolated problems with immediate correctness feedback. However, technical interviews—even algorithmic ones—require explanation of reasoning, communication under observation, and adaptation to interviewer feedback [1]. Skills developed in solitary practice may not transfer to social performance contexts. The finding that communication practice and peer mock interviews show 2× effectiveness suggests that

authentic simulation of interview conditions is more valuable than volume of problem-solving practice.

Diminishing returns and expertise reversal effects. Learning science research on deliberate practice indicates that effectiveness depends on appropriate challenge level and feedback quality [19]. Excessive volume of similar problems may yield diminishing returns, particularly for experienced practitioners (the study sample averaged 9 years experience). What appears as "more preparation" may represent repetition of already-mastered patterns rather than acquisition of new capabilities.

Construct validity issues. If technical interviews measure capabilities beyond algorithmic problem-solving (communication, reasoning explanation, handling ambiguity), then practice focused solely on algorithm optimization would naturally show weak correlation with interview performance. The strong effect of communication practice supports this interpretation.

4.3 Limitations and Implications

Several limitations warrant consideration. The study used self-reported measures of both preparation time and interview readiness, which may be subject to recall bias and social desirability effects. The sample size (n=131), while the largest available for this research question, is modest for drawing strong causal conclusions. The cross-sectional design precludes establishing causality—we cannot definitively conclude that increasing communication practice would cause improved performance, only that the two are associated. Longitudinal or experimental designs would strengthen causal inference.

However, even with these limitations, the findings are consequential. If candidates are investing an average of 6.5 hours per week in preparation activities that show zero correlation with outcomes, while underinvesting in activities (communication, peer practice) that show strong positive associations, then current preparation practices may be highly inefficient. Moreover, the finding challenges the implicit theory underlying algorithmic interviews: if performance on such interviews corresponds to the skills being practiced (algo-

rithmic problem-solving), we would expect practice volume to correlate with performance. The lack of such correlation suggests either that interviews measure something other than algorithmic skill, or that algorithmic skill itself is not the relevant construct for software engineering competency. Either interpretation undermines the validity of the current assessment paradigm.

5 Psychometric Evidence on Predictive Validity

The ultimate criterion for any selection method is predictive validity: how well do assessment results predict subsequent job performance? Industrial-organizational psychology has accumulated substantial evidence on this question across multiple domains, allowing us to contextualize technical interviews within broader selection science.

5.1 Foundational Concepts and Measurement

Predictive validity is typically quantified using correlation coefficients between assessment scores and subsequent job performance ratings. A validity coefficient of 1.0 would indicate perfect prediction, while 0.0 indicates no predictive relationship. In practice, selection methods rarely exceed 0.60 even under optimal conditions, as job performance is influenced by numerous factors beyond those assessed during hiring.

Interpretation of validity coefficients requires understanding their relationship to explained variance. A validity coefficient of r corresponds to explained variance of r^2. Thus, a method with validity 0.40 explains 16% of variance in job performance, leaving 84% unexplained. Higher validity coefficients allow more meaningful differentiation between candidates and reduce hiring errors.

Recent meta-analytic research has revised previously-reported validity estimates downward. A 2021 comprehensive review found that prior research had "erroneously high" validity estimates due to over-correction for statistical artifacts [14]. The revision indicates that "true

validities of the selection methods have not changed over time, but we are now estimating validity more accurately and more conservatively than before" [14]. Table 2 summarizes revised validity coefficients for major selection methods.

Table 2: Predictive Validity of Selection Methods (Revised Estimates)

Method	**Previous (1998)**	**Revised (2021)**	**Change**
Work sample testing	0.54	0.33	−39%
Cognitive ability testing	0.51	0.31	−39%
Structured job interview	0.51	0.42	−18%
Job knowledge testing	0.48	0.40	−17%
Combined methods (2021 estimates):			
GMA + work sample	—	0.63–0.65	—
GMA + structured interview	—	0.63	—

5.2 Application to Technical Interviews

Technical interviews that focus on algorithmic problem-solving resemble cognitive ability tests in their emphasis on abstract reasoning and problem-solving under time pressure. If we categorize such interviews as specialized cognitive ability tests, the relevant validity benchmark is approximately 0.31–0.42 [14,17]. This suggests that isolated algorithmic assessments explain roughly 10–18% of variance in job performance, leaving 82–90% unexplained.

Research in medical education provides an informative parallel. A study of selection methods for clinical training found that situational judgment tests and clinical problem-solving tests each showed validity of 0.50–0.56 ($p < 0.001$) when predicting one-year job performance [15]. Critically, when combined with realistic job simulation (selection center assessment), validity increased to 0.61, explaining 37% of performance variance. The study concluded: "Although selection centres are relatively expensive selection tools, the results show that the selection centre adds significant incremental validity over the two short-listing tests in predicting subsequent job performance... especially for domains such as empathy and communication, which are crucial" [15].

The pattern is consistent across domains: multimethod assessment combining cognitive testing, work samples, and structured evaluation shows substantially higher predictive validity (0.63–0.65) than any single method alone [14, 16]. Work samples—realistic simulations of job tasks—are particularly valuable because they capture aspects of performance (contextual judgment, procedural skill, adaptation to constraints) that cognitive tests omit. As one synthesis notes: "When there is no previous experience in a similar type of work, the best predictor of success is cognitive abilities (GMA)" [16], but for experienced practitioners and complex roles, work samples become essential.

5.3 Implications for Interview Design

The psychometric evidence points to several conclusions:

Isolated algorithmic assessments have modest validity. With estimated validity coefficients of 0.31–0.42, such assessments explain 10–18% of job performance variance. This is not negligible—it is comparable to many other selection methods—but it is far from sufficient for high-stakes selection decisions where minimizing false positives and false negatives is critical.

Multimethod approaches substantially improve validity. Combining cognitive assessment with work samples and structured interviews increases validity to 0.63–0.65, explaining 40–42% of variance. This improvement is substantial: moving from 15% to 40% explained variance more than doubles predictive power.

Work samples capture distinct competencies. The incremental validity of work samples over cognitive tests indicates they measure capabilities not captured by algorithmic problem-solving. In software engineering contexts, these likely include: integration of multiple knowledge domains, adaptation to ambiguous requirements, debugging in complex systems, and communication of technical reasoning.

Assessment should match job requirements. High validity requires alignment between assessment tasks and actual job tasks. As job tasks evolve—incorporating more collaborative

development, AI-augmented workflows, and legacy system navigation—assessments must evolve correspondingly or risk measuring increasingly irrelevant constructs.

The psychometric evidence does not support eliminating cognitive assessment entirely; rather, it indicates that cognitive assessment alone is insufficient. A valid selection process would combine algorithmic problem-solving (to assess foundational reasoning) with realistic job simulation (to assess applied competency) and structured behavioral interviewing (to assess collaboration and communication). Current practices that rely primarily on algorithmic challenges thus represent a suboptimal approach that discards available gains in predictive validity.

6 Real-World Software Engineering Tasks and Interview Coverage

A fundamental question for any assessment is content validity: does the assessment sample the knowledge and skills required for job performance? We examine the alignment between algorithmic interview tasks and actual software engineering work.

6.1 Characterizing Professional Software Engineering

Contemporary software engineering involves a complex constellation of activities. Survey research and industry analyses identify several dominant task categories:

Legacy code maintenance. A substantial portion of professional software work involves understanding, modifying, and maintaining existing codebases. Key challenges include: inadequate documentation, outdated practices reflecting earlier development eras, absence of original developers, and the need to integrate legacy systems with modern infrastructure (cloud platforms, microservices, contemporary databases) [22]. As one industry analysis notes: "Managing legacy code entails more than just dealing with untidy or outdated code; it involves transforming it into a reliable and efficient foundation that supports future devel-

opment initiatives" [22]. This work requires historical comprehension, architectural reasoning about system boundaries and dependencies, and risk assessment for modifications.

Collaborative debugging. Modern debugging is rarely a solitary activity. Teams use distributed version control, issue tracking systems, continuous integration pipelines, and asynchronous communication tools [23]. Effective debugging collaboration involves: clearly articulating problems, assembling appropriate expertise, establishing shared context using logs and reproduction steps, and coordinating across time zones and team boundaries. Research on debugging collaboration identifies benefits including faster error resolution, knowledge sharing, reduced individual burnout, and improved code quality [23]. These benefits accrue specifically from collaborative rather than individual practice.

System architecture and design. Industry reports consistently identify a shortage of mid-level to senior engineers capable of architectural decision-making [24]. While junior developers are abundant, the market particularly values engineers who can "lead projects, architect complex systems, and mentor teams" [24]. Architectural work requires understanding tradeoffs between competing concerns (performance, maintainability, cost, scalability), anticipating future evolution of requirements, and communicating design rationale to diverse stakeholders.

Cross-generational knowledge integration. Software teams often span multiple career stages, creating potential knowledge gaps. Senior developers bring deep domain knowledge and extensive computer science experience but may be less familiar with emerging technologies, while junior developers demonstrate facility with new tools but lack appreciation for legacy system constraints [25]. Effective engineering requires bridging these perspectives.

6.2 Coverage Gap Analysis

Comparing these professional task characteristics with algorithmic interview content reveals substantial gaps:

The gaps are not merely quantitative (problems are larger in production) but quali-

Table 3: Interview Task vs. Job Task Comparison

Algorithmic Interview Tasks	**Professional Software Tasks**
Novel, well-defined problems	Ambiguous requirements, evolving constraints
Individual performance	Team collaboration, asynchronous coordination
Time-bounded (30–60 minutes)	Extended timescales (weeks to months)
Optimal solution expected	Satisficing under resource constraints
Recent implementation from scratch	Modification of existing systems
Full context provided	Incomplete information, discovery required
Clean slate assumptions	Legacy constraints, technical debt
Algorithmic correctness primary criterion	Maintainability, performance, cost tradeoffs

tative. Production software engineering centrally involves navigation of existing systems, interpretation of historical decisions, coordination with other humans, and reasoning under uncertainty. Algorithmic interviews systematically exclude or minimize these dimensions. A candidate could excel at algorithmic problem-solving while lacking the historical comprehension, collaborative skill, or architectural judgment required for professional effectiveness.

6.3 Implications of Misalignment

The coverage gap has several consequences. First, it creates adverse selection pressure: optimizing interview performance requires different skills than optimizing job performance. Candidates may rationally invest in interview-specific preparation (algorithmic pattern recognition) rather than skills with broader professional value (system design, debugging methodology, communication). The empirical finding that LeetCode preparation hours show no correlation with interview readiness [1] may reflect this misalignment—candidates are practicing the wrong skills even for interview success, let alone job success.

Second, the gap particularly disadvantages candidates with practical experience. A developer with 10 years of production experience may have extensive skill in debugging complex systems, navigating legacy code, and architectural decision-making, yet perform worse on algorithmic interviews than a recent graduate who has extensively practiced LeetCode

problems. This inverts the expected relationship between experience and assessment performance.

Third, the gap may contribute to workforce diversity challenges. Research on selection bias suggests that assessments emphasizing narrow technical performance under time pressure may systematically disadvantage candidates who excel in collaborative, contextual reasoning—skills that may be more evenly distributed across demographic groups than speed-based algorithmic performance. While we lack direct empirical evidence on this specific claim in software engineering contexts, the broader selection literature documents that multimethod assessment tends to reduce adverse impact compared to single-method cognitive testing.

7 Industry Response and Emerging Practices

Major technology companies have begun reconsidering interview formats in response to AI capability and recognition of the limitations discussed in previous sections. While systematic empirical evaluation of these new approaches remains limited, early industry reports provide insight into emerging directions.

7.1 Explicit Permission of AI Tools

In June 2025, Canva announced a significant policy shift: the company would not only permit but actively encourage candidates to use AI tools during technical interviews [7]. The company's engineering blog explained that the traditional format had become trivially solvable by AI without demonstrating human skill, and that the format no longer effectively differentiated candidates. Canva redesigned its assessments to test "thoughtful AI leveraging" and complex problem decomposition, focusing on design rationale and collaboration rather than algorithmic correctness alone. According to the company's report, performance signal improved post-redesign, with better discrimination between candidate capabilities [7].

This approach represents a fundamental shift in assessment philosophy: rather than attempting to isolate human capability from AI capability—an increasingly artificial distinction given that production software development now routinely involves AI-augmented workflows—the assessment evaluates hybrid human-AI performance. This aligns more closely with actual job requirements, where engineers increasingly use AI tools for code generation, documentation, and debugging assistance.

7.2 Broader Industry Patterns

Industry reporting indicates that Canva's policy is part of a broader trend. Fortune reported in July 2025 that major technology companies including Meta, Amazon, and Intuit were moving away from pure algorithmic puzzles, emphasizing instead real-world tasks, collaboration assessment, and context adaptation [8]. The shift involves allowing AI tools in interviews while redesigning tasks to focus on how candidates use such tools rather than whether they can solve problems independently. This parallels the reality that software engineers are evaluated on project delivery, not on whether they consulted documentation or used code assistance tools during development.

The timing of these changes is noteworthy. While practitioners have long questioned the validity of algorithmic interviews, substantive format changes accelerated only after LLM capabilities became widely apparent. This suggests that AI disruption served as a catalyst for addressing longstanding concerns about assessment validity that might otherwise have persisted due to institutional inertia.

7.3 Characteristics of Reformed Assessments

Emerging best practices, as reported in industry sources, share several characteristics:

Real-world task simulation. Rather than novel algorithmic puzzles, assessments involve debugging existing code, refactoring legacy systems, or designing components that integrate with specified architectures [8]. These tasks more closely resemble professional work and are

less susceptible to pattern-matching solutions.

Communication and reasoning explanation. Assessments explicitly evaluate how candidates explain their reasoning, handle ambiguity, and communicate tradeoffs. The assumption is that while AI can generate plausible code, human judgment remains essential for evaluating whether generated solutions meet contextual requirements.

Collaboration simulation. Some companies are experimenting with pair programming assessments or scenarios requiring coordination with simulated team members, recognizing that professional effectiveness depends substantially on collaborative capability [8].

Multimethod approach. Reformed assessments typically combine multiple evaluation methods—coding tasks, system design discussion, behavioral interviews, and work samples—rather than relying primarily on algorithmic problem-solving.

7.4 Evidence Gaps and Research Opportunities

While these developments are promising, rigorous evaluation remains limited. Industry reports describe observational data on candidate performance and hiring manager satisfaction, but we lack randomized controlled trials comparing reformed assessment formats with traditional approaches on criterion validity (correlation with subsequent job performance). Key research questions include:

- Do AI-augmented assessments show higher predictive validity than traditional formats when measured against job performance ratings?
- How do reformed assessments affect candidate diversity and adverse impact across demographic groups?
- What is the cost-effectiveness of different assessment approaches considering development time, interviewer training, and candidate experience?
- Do candidates prepared using traditional methods (LeetCode focus) perform worse on

reformed assessments, and if so, does this reflect genuine skill gaps or merely transitional adaptation challenges?

Answering these questions will require longitudinal research designs tracking candidates through hiring, onboarding, and subsequent performance evaluation. The software engineering research community has an opportunity to conduct rigorous evaluation of these natural experiments as companies adopt varied assessment approaches.

8 Discussion and Synthesis

We now synthesize the evidence presented across multiple domains and discuss implications for theory and practice.

8.1 Convergent Evidence for Assessment Inadequacy

Five independent lines of evidence converge on the conclusion that algorithmic interviews have substantial limitations as predictors of software engineering competency:

Historical analysis reveals that the format emerged from 1980s–1990s practical constraints rather than systematic job analysis or validity research. The standardization of whiteboard coding reflected convenience and institutional imitation, not evidence-based design. Format persistence despite technological change represents cultural inertia rather than demonstrated effectiveness.

AI capability assessment shows that LLMs achieve 32–68% success rates on algorithmic problems and produce code that executes faster than 63% of human submissions, yet these same systems struggle with production software engineering tasks involving design, legacy integration, and stakeholder communication. This divergence undermines the construct validity of algorithmic interviews: if AI masters the interview format while failing at the job, the interview measures the wrong construct.

Empirical preparation research documents zero correlation ($r = 0.0$, $p > 0.05$, n=131) between LeetCode preparation hours and interview readiness, while communication practice and peer collaboration show approximately 2× effectiveness. If the interview measured skills being practiced, practice volume should correlate with performance. The null finding suggests interviews measure capabilities not developed through algorithmic practice.

Psychometric validity research indicates that isolated cognitive tests have validity coefficients of 0.31–0.42, explaining only 10–18% of job performance variance. Multimethod assessments combining work samples, cognitive testing, and structured interviews achieve validity of 0.63–0.65, explaining 40–42% of variance. Current algorithmic-focused practices thus discard available gains in predictive validity.

Job task analysis reveals fundamental mismatches between interview content (novel, well-defined algorithmic problems) and professional work (ambiguous requirements, legacy constraints, collaborative debugging, architectural tradeoffs). The coverage gap is not merely quantitative but qualitative, involving dimensions systematically excluded from algorithmic assessments.

These findings are mutually reinforcing. The historical contingency of the format explains why systematic validity research was never conducted. The AI capability divergence provides a contemporary demonstration that algorithmic skill does not correspond to software engineering mastery. The empirical preparation findings show that even candidates investing substantial resources recognize the disconnect between practice and outcomes. The psychometric evidence establishes that better assessment methods exist and have known effectiveness. The job task analysis specifies what those better methods should measure.

8.2 Reductio Argument Revisited

The logical structure of our argument can be formalized:

1. Premise: If algorithmic puzzle-solving mastery corresponds to software engineering competency, then systems that master algorithmic puzzles should demonstrate software

engineering competency.

2. Premise: LLMs achieve 32–68% success on hard algorithmic problems (substantial mastery).

3. Premise: LLMs struggle systematically with software engineering tasks requiring design, legacy integration, architectural reasoning, and stakeholder communication.

4. Conclusion: Therefore, algorithmic puzzle-solving mastery does not correspond to software engineering competency.

This argument does not claim that algorithmic skills are irrelevant or that individuals who excel at algorithmic problem-solving cannot be excellent software engineers. Rather, it claims that the relationship is neither necessary (one can be an excellent engineer without exceptional algorithmic speed) nor sufficient (one can master algorithmic puzzles, as AI demonstrates, without mastering software engineering). The practical implication is that algorithmic assessments should be supplementary components of evaluation, not primary determinants.

8.3 Implications for Stakeholders

The evidence has distinct implications for different stakeholder groups:

8.3.1 Implications for Educators

Computing education programs should reconsider how they prepare students for professional practice and hiring processes. Several recommendations emerge:

Integrate authentic software engineering practice. Curricula should include substantial experience with legacy code, debugging in complex systems, collaborative development using industry-standard tools (version control, continuous integration, issue tracking), and architectural decision-making under constraints. These experiences are valuable for professional

development regardless of interview format, but they also develop capabilities that reformed assessments will increasingly evaluate.

Teach interview preparation as a professional skill. Rather than treating interview preparation as separate from coursework, programs could integrate preparation for technical assessment into software engineering courses. This includes: explaining reasoning clearly, thinking aloud while problem-solving, receiving and responding to feedback, and collaborative debugging. The empirical finding that communication practice shows 2× effectiveness compared to isolated coding practice suggests that these skills merit explicit instruction.

Provide structured guidance. The Bell et al. study found that candidates perceive computing education as unsupportive of interview preparation [1]. Programs could address this through: structured mock interviews with feedback, explicit discussion of assessment formats and effective preparation strategies, and alumni mentoring programs. These need not involve teaching LeetCode-specific tricks but rather general professional communication and problem-solving demonstration.

Balance algorithmic foundations with applied competencies. Data structures and algorithms remain foundational knowledge, but their instruction should emphasize understanding over speed-based recall. Students benefit more from understanding when and why to apply different algorithmic approaches than from memorizing optimal solutions to 500 problems.

8.3.2 Implications for Employers

Organizations conducting technical hiring should consider:

Adopt multimethod assessment. The psychometric evidence consistently shows that combining work samples, structured interviews, and cognitive assessment yields validity coefficients of 0.63–0.65 compared to 0.31–0.42 for isolated cognitive tests. Multimethod assessment requires more development time initially but provides substantially better prediction of job performance.

Emphasize realistic job simulation. Assessment tasks should sample actual job require-

ments: debugging in existing codebases, architectural design discussions, explaining technical decisions to non-technical stakeholders, navigating ambiguous requirements. These tasks are harder to prepare for using pattern-matching, provide richer signal about professional capabilities, and align with the actual work successful candidates will perform.

Permit and evaluate AI tool use. Given that production software engineering increasingly involves AI-augmented workflows, assessments should evaluate candidates' effectiveness in using such tools rather than attempting to isolate human-only performance. This includes assessing: prompt engineering skill, critical evaluation of AI-generated solutions, integration of AI suggestions with domain knowledge, and recognition of when AI assistance is inappropriate.

Invest in interviewer training. Structured interviews show higher validity than unstructured interviews, but this benefit accrues only when interviewers are trained in consistent evaluation criteria, behavioral anchors for rating scales, and bias mitigation. Organizations often invest substantially in developing assessment content while underinvesting in interviewer preparation.

Conduct validity research. Organizations with sufficient hiring volume should conduct internal validity studies: correlate interview performance with subsequent job performance ratings, analyze predictive validity across different assessment components, and test whether reformed assessment formats improve prediction. This research would provide valuable evidence for the broader community while improving organizational hiring effectiveness.

8.3.3 Implications for Candidates

Individuals preparing for technical interviews face a challenging environment where assessment formats are in flux. Several strategies emerge from the evidence:

Prioritize authentic practice. The empirical finding that communication practice and peer collaboration show 2× effectiveness compared to isolated LeetCode grinding suggests reallocation of preparation time. Effective preparation includes: mock interviews with peers

or mentors, explaining solutions aloud, receiving critical feedback, and practicing in time-constrained collaborative settings that simulate actual interview conditions.

Develop breadth of competencies. While algorithmic problem-solving remains part of many interview processes, the trend toward multimethod assessment means candidates benefit from developing: system design and architectural reasoning, debugging methodology and tooling proficiency, communication of technical concepts to varied audiences, and demonstrated experience with collaborative development (visible through open source contributions, team projects, or work experience).

Understand the assessment landscape. Different organizations are adopting different interview formats at different rates. Candidates should research specific company practices, understand what each assessment type evaluates, and prepare accordingly. This may involve: asking recruiters about interview format, reading company engineering blogs describing hiring philosophy, and connecting with recent interviewees.

Leverage AI tools in preparation. Given emerging practices that permit AI use in interviews, preparation should include practicing with AI coding assistants: developing effective prompts, critically evaluating generated solutions, and integrating AI suggestions with domain knowledge. This aligns preparation with actual assessment format while developing professionally valuable skills.

8.3.4 Implications for Interview Preparation Platforms

Platforms providing interview preparation resources should consider:

Shift emphasis toward authentic task practice. While algorithmic problem catalogs serve a role, platforms could provide greater value through: simulated collaborative debugging scenarios, architecture design exercises with constraint analysis, mock behavioral interviews with structured feedback, and peer-to-peer practice facilitation.

Incorporate communication skill development. The strong effect of communication practice on interview readiness suggests platforms should provide: structured prompts for ex-

plaining reasoning, video recording and self-review tools, peer review and feedback mechanisms, and explicit instruction in technical communication.

Provide evidence-based guidance. Platforms could help candidates make informed preparation decisions by presenting empirical evidence on practice effectiveness, highlighting the null correlation between problem volume and outcomes, and recommending time allocation strategies grounded in learning science (spaced repetition, authentic simulation, deliberate practice with feedback).

8.4 Limitations and Future Research

Several limitations qualify our conclusions. First, while we have substantial evidence on each component of the argument, we lack large-scale randomized controlled trials directly comparing traditional algorithmic interviews with reformed multimethod approaches on predictive validity. The Bell et al. study provides the best available empirical evidence on preparation effectiveness, but with n=131 and self-reported measures, replication with larger samples and objective performance criteria would strengthen conclusions.

Second, the field is in transition. Industry reports describe emerging practices at major technology companies, but systematic evaluation of these approaches is ongoing. We cannot yet definitively state that AI-augmented assessments have higher predictive validity than traditional formats, though the psychometric literature on multimethod assessment strongly suggests they should.

Third, our analysis focuses primarily on entry-level to mid-level software engineering positions at technology companies. The relevance of algorithmic interviews may differ for specialized roles (e.g., algorithmic trading, computational science, machine learning engineering) where algorithmic optimization is central to job requirements. Assessment validity likely varies by role context.

Fourth, we lack comprehensive data on cost-effectiveness. Multimethod assessment requires more development time, interviewer training, and evaluation effort than standardized

algorithmic testing. Whether the improvement in predictive validity justifies the additional cost depends on organizational context: high-volume hiring may favor more scalable methods even with lower validity, while critical technical roles may warrant substantial assessment investment.

Future research should address several questions:

Longitudinal validity studies. Track candidates through reformed assessment processes and correlate interview performance with subsequent job performance ratings at 6 months, 1 year, and 2 years post-hire. Compare predictive validity across different assessment formats.

Learning science experiments. Conduct randomized trials of preparation methods: compare outcomes for candidates using traditional LeetCode practice versus authentic simulation with peer feedback versus hybrid approaches. Measure both interview performance and skill transfer to job contexts.

Diversity and adverse impact analysis. Examine whether reformed assessment formats affect demographic representation in hiring outcomes. Test whether multimethod approaches reduce adverse impact compared to algorithmic-only formats.

AI capability boundary research. Systematically characterize which software engineering tasks AI systems perform well versus poorly. Update these characterizations as AI capabilities evolve. Use findings to inform assessment design, ensuring interviews evaluate capabilities where human judgment remains distinctively valuable.

Curriculum integration studies. Evaluate whether software engineering programs that integrate authentic practice, interview skill development, and collaborative projects produce candidates who perform better in reformed assessments and in subsequent job performance.

9 Conclusion

This paper examined the validity of algorithmic technical interviews in software engineering hiring, synthesizing evidence from historical analysis, AI capability assessment, empirical

preparation research, psychometric validity studies, and job task analysis. Five converging lines of evidence suggest that the traditional format has substantial limitations: it emerged from historical contingency rather than evidence-based design, measures capabilities that AI systems increasingly replicate while omitting capabilities where human judgment remains essential, shows zero correlation with preparation effort in empirical research, achieves only modest predictive validity compared to multimethod alternatives, and systematically excludes the collaborative, contextual reasoning that characterizes professional software engineering.

The arrival of large language models with substantial coding capability serves as a catalyst exposing these longstanding limitations. When AI systems achieve 32–68% success on hard algorithmic problems yet struggle with production software engineering tasks, the divergence reveals that algorithmic interviews measure the wrong construct. The appropriate response is not to make algorithmic problems harder—an arms race that becomes increasingly disconnected from job requirements—but rather to redesign assessments around authentic task simulation, multimethod evaluation, and explicit assessment of capabilities where human expertise remains distinctively valuable.

Major technology companies have begun this transition, permitting AI tool use in interviews and redesigning assessments to evaluate thoughtful AI leveraging, communication, and complex problem decomposition. While systematic evaluation of these reformed approaches remains limited, the psychometric literature provides strong theoretical grounds for optimism: multimethod assessments combining work samples, structured interviews, and cognitive evaluation consistently show predictive validity of 0.63–0.65 compared to 0.31–0.42 for isolated cognitive tests. These are not marginal improvements but represent doubling of explained variance in job performance.

The implications extend beyond interview reform to broader questions about computing education and professional development. If traditional interview formats misalign with job requirements, then educational programs optimizing for interview performance may inad-

vertently misalign with professional preparation. The empirical finding that communication practice and peer collaboration show 2× effectiveness compared to isolated algorithmic practice suggests that both education and interview preparation should emphasize authentic simulation of professional contexts.

We conclude with a call for evidence-guided practice. The algorithmic interview format was never validated through systematic research; it persisted through historical contingency and cultural momentum. As the field responds to AI disruption, we have an opportunity to redesign assessment based on empirical evidence about predictive validity, learning science principles about effective preparation, and clear-eyed analysis of actual job requirements. This redesign will require investment—developing work samples, training interviewers, conducting validity research—but the investment promises substantial returns in hiring effectiveness, candidate experience, and alignment between assessment and professional practice.

The question is not whether algorithmic skill matters for software engineering—it clearly does—but whether isolated algorithmic puzzle-solving under time pressure is the optimal method for evaluating professional competency. The evidence consistently suggests it is not. A more valid approach would combine foundational algorithmic assessment with realistic job simulation, structured evaluation of communication and collaboration, and explicit assessment of capabilities that remain distinctively human even as AI capabilities expand. The transition to such approaches is underway; the research community's task is to rigorously evaluate their effectiveness and guide evidence-based optimization of technical hiring practices.

Acknowledgments

My deepest gratitude to Mr. Krishna, whose constancy forms the foundation upon which all my work, including this, quietly rests.

Salutations to the Goddess who dwells in all beings in the form of intelligence. I bow to her again and again.

Statements and Declarations

Funding Declaration

No funding was received to assist with the preparation of this manuscript.

Author Contribution

L.A.R. was responsible for all aspects of this manuscript, including conceptualization, methodology, writing the original draft, and review and editing.

Competing Interests

The author declare no competing interests.

References

[1] B. Bell, J. Thomas, et al., "How do Software Engineering Candidates Prepare for Technical Interviews?" *arXiv preprint arXiv:2507.02068*, 2025.

[2] C. Treude and M.-A. Storey, "Generative AI and Empirical Software Engineering: A Paradigm Shift," *IEEE Transactions on Software Engineering (preprint arXiv:2502.08108v2)*, 2025.

[3] L. Wang, C. Shi, S. Du, Y. Tao, Y. Shen, H. Zheng, and X. Qiu, "Performance Review on LLM for Solving LeetCode Problems," *arXiv preprint arXiv:2502.15770*, 2025.

[4] "Software engineering education in the era of conversational AI," *Frontiers in Artificial Intelligence*, vol. 7, no. 1436350, 2024. DOI: 10.3389/frai.2024.1436350

[5] "How much does AI impact development speed? An enterprise study," *arXiv preprint arXiv:2410.12944v1*, 2025.

[6] J. Becker et al., "Measuring the Impact of Early-2025 AI on Software Development," *arXiv preprint arXiv:2507.09089*, 2025.

[7] Canva Engineering, "Yes, You Can Use AI in Our Interviews. In fact, we insist," *Canva Developer Blog*, June 2025. [Online]. Available: https://www.canva.dev/blog/engineering/yes-you-can-use-ai-in-our-interviews

[8] "How to vet software developer candidates in the age of AI," *Fortune*, July 2, 2025.

[9] "Technical Interviews in 2025: How to Stand Out in the Age of AI," *CourseReport Blog*, July 8, 2025. [Online]. Available: https://www.coursereport.com/blog/technical-interviews-in-2025-how-to-stand-out-in-the-age-of-ai

[10] "Whiteboard Interview Guide: The Good, The Bad, and The Ugly," *CoderPad Blog*, April 24, 2024. [Online]. Available: https://coderpad.io/blog/interviewing/whiteboard-interview-guide/

[11] "Whiteboard Interview – Definition, Overview & FAQ," *Recruiteze Blog*, December 6, 2023. [Online]. Available: https://recruiteze.com/glossary/whiteboard-interview/

[12] "Evolution of Software Development — History, Phases and Future Trends," *GeeksforGeeks Blog*, November 7, 2023. [Online]. Available: https://www.geeksforgeeks.org/software-engineering/evolution-of-software-development-history-phases-and-future-trends/

[13] "The evolution of the coding interview," *Reddit r/ExperiencedDevs*, 2024. [Online]. Available: https://www.reddit.com/r/ExperiencedDevs/comments/1ae4gtw/the_evolution_of_the_cod

[14] "Understanding Scientific Findings in Employee Selection," *Criteria Corp Blog*, June 15, 2022 (updated 2025). [Online]. Available: https://www.criteriacorp.com/blog/updates-employee-selection-science

[15] "The Predictive Validity of Selection for Entry into [Medical Training]," *PubMed Central*, PMID: 3809426, October 27, 2013.

[16] "Validity of Selection Methods in HR," *LogiPass Blog*, 1997 (ongoing updates). [Online]. Available: https://logipass.net/en/blog/validity-of-selection-methods-in-hr

[17] "Validity of Pre-Employment Tests," *Criteria Corp Blog*, September 9, 2025. [Online]. Available: https://www.criteriacorp.com/resources/definitive-guide-validity-of-preemployment-tests/validity-pre-employment-tests

[18] "What is Predictive Validity in HR?" *HR Lineup Blog*, February 6, 2025. [Online]. Available: https://www.hrlineup.com/what-is-predictive-validity-in-hr/

[19] Wikipedia contributors, "Spaced Repetition," *Wikipedia*, August 8, 2001 (ongoing updates). [Online]. Available: https://en.wikipedia.org/wiki/Spaced_repetition

[20] "Spaced Repetition of Practice," *Codecademy Blog*, October 25, 2021. [Online]. Available: https://www.codecademy.com/article/spaced-repetition

[21] "How to Use Spaced Repetition," *Snappify Blog*, September 19, 2024. [Online]. Available: https://snappify.com/blog/spaced-repetition

[22] "Legacy Code: 5 Challenges, Tools and Tips to Overcome Them," *Swimm.io Blog*, September 17, 2025. [Online]. Available: https://swimm.io/learn/legacy-code/legacy-code-5-challenges-tools-and-tips-to-overcome-them

[23] "Debugging Collaboration," *Meegle Blog*, October 24, 2025. [Online]. Available: https://www.meegle.com/en_us/topics/debugging/debugging-collaboration

[24] "Is Software Engineering a Dying Career? [10 Key Factors]," *Digital Defy Blog*, August 20, 2025. [Online]. Available: https://digitaldefynd.com/IQ/is-software-engineering-a-dying-career/

[25] "How to Identify Skill Gaps to Scale for Expansion," *Revelo Blog*, May 25, 2025. [Online]. Available: https://www.revelo.com/blog/how-to-identify-skill-gaps

[26] Meng Tan Hon, "The Real Limits of AI Code Generation—and What Enterprises Must Know," *LinkedIn*, June 29, 2025. [Online]. Available: https://www.linkedin.com/pulse/real-limits-ai-code-generationand-what-enterprises-must-kee-meng-tan-hon1e

[27] "Challenges of Using AI in Software Development," *GUVI Blog*, October 21, 2025. [Online]. Available: https://www.guvi.in/blog/challenges-of-using-ai-in-software-development/

[28] "Challenges and Opportunities of AI in Software Development," *Codewave Blog*, June 22, 2025. [Online]. Available: https://codewave.com/insights/ai-in-software-development-challenges-opportunities/

[29] "How to Evaluate and Benchmark Large Language Models," *Together.ai Blog*, November 3, 2025. [Online]. Available: https://www.together.ai/blog/evaluate-and-benchmark-llms

[30] LiveCodeBench Contributors, "LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code," 2024–2025. [Online]. Available: https://livecodebench.github.io

[31] "Cracking the Coding Interview: Solve 5 Real World Problems," *Educative Blog*, March 3, 2021. [Online]. Available: https://www.educative.io/blog/crack-coding-interview-real-world-problems