Trustworthy Evaluation of AI-Generated Code

A multidimensional framework for correctness, security and production readiness

Sunday Adetona Aderinto

Global IT Specialist | Software Engineer | AI Technical Evaluator | Computer Engineering Researcher

Professional Insight Paper	September 2026
Domains	Software engineering | Trustworthy AI | Distributed systems

Core proposition

Generative AI can accelerate software development, but fluent code is not necessarily correct, secure or maintainable. This paper proposes a practical evaluation framework that combines executable evidence, static analysis, security review, architectural assessment and calibrated human judgement. It argues that evaluation must move beyond pass rates to examine whether code is fit for sustained use in production environments.

Executive abstract

Generative AI can accelerate software development, but fluent code is not necessarily correct, secure or maintainable. This paper proposes a practical evaluation framework that combines executable evidence, static analysis, security review, architectural assessment and calibrated human judgement. It argues that evaluation must move beyond pass rates to examine whether code is fit for sustained use in production environments.

1. The evaluation problem

AI-generated code can satisfy a narrow prompt while failing assumptions that were not explicitly represented. Common failure modes include incomplete error handling, fabricated dependencies, insecure defaults, weak input validation, hidden performance costs and code that passes a local test but conflicts with the wider system. An evaluator must therefore distinguish syntactic validity, functional adequacy and production fitness.

2. A six-dimensional model

A robust assessment should test six dimensions: functional correctness; security and abuse resistance; maintainability; architectural fit; operational behaviour; and explanatory integrity. Each dimension requires explicit evidence. Correctness needs tests and edge cases. Security needs threat-aware review. Maintainability needs clarity, modularity and change cost. Architectural fit asks whether the output respects interfaces, data ownership and system constraints.

3. Evidence hierarchy

The strongest evidence is reproducible: compilation, execution, unit and integration tests, property-based tests, mutation testing, static analysis and dependency checks. Human review remains essential where requirements are ambiguous or where quality depends on trade-offs. A score should never conceal a critical security defect; gating rules should override aggregate averages for severe failures.

4. Human calibration

Expert reviewers need anchored rubrics, worked examples and periodic calibration. Independent scoring followed by adjudication exposes ambiguous criteria and reduces reviewer drift. Organisations should record not only the final judgement but the reason, severity, affected component and remediation path. This creates a learning asset for both engineering teams and model improvement.

5. Deployment governance

AI-generated contributions should enter the same controlled delivery pipeline as human code: version control, review, automated tests, security scanning, staged deployment, observability and rollback. Higher-risk domains require stronger evidence and clearer accountability. The developer who accepts the output remains responsible for its consequences.

6. Practical implementation

Begin with a representative task set and a short rubric. Measure disagreement, escaped defects and review time. Expand the benchmark by language, task and risk class. The objective is not to eliminate human judgement but to make it consistent, inspectable and better supported by automation.

Practitioner takeaways

• Evaluate production fitness, not surface plausibility.

• Combine executable evidence with expert judgement.

• Treat critical security failures as gates, not small deductions.

• Use reviewer disagreement as data for improving the rubric.

Conclusion

Responsible technology leadership requires evidence, explicit trade-offs and accountable human judgement. The frameworks in this paper are intended to support disciplined implementation and to create a foundation for deeper empirical research.

Selected references

NIST, Secure Software Development Framework (SSDF).

NIST, Artificial Intelligence Risk Management Framework (AI RMF 1.0).

OWASP, Top 10 for Large Language Model Applications.

ISO/IEC 25010, Systems and software quality models.

About the author

Sunday Adetona Aderinto is a senior software engineer and Computer Engineering researcher with more than ten years of experience across full-stack systems, APIs, databases, infrastructure, technical instruction and AI-generated code evaluation. His interests include trustworthy AI, federated learning, differential privacy and production software quality.