SYNTHESIS

The Five Rings

Why Craftsmanship Matters More Now, and Where It Leads

Damir Majer
ASE Academy, Munich

May 2026
ASE ACADEMY

Contents

1 The Amplifier Thesis 3

2 When Constraints Are Removed 4

3 The Convergence 6

4 Five Rings 7

Who This Paper Is For

Technical leaders, engineering managers, and senior developers who have read the three preceding papers and want to understand how they connect. This synthesis asks the question the trilogy leaves open: if developer discipline, structural constraints, and economic optimization all matter, why do they matter more in an era of AI-assisted development?

Abstract

The three preceding papers address developer capability (the ACE Model), structural quality (Constraint-Driven Development, CDD), and software economics (the Economics of Changeability, EoC). This synthesis proposes that these themes become more tightly coupled under AI-assisted development. The central thesis is directional: AI appears to amplify existing strengths and weaknesses more than it replaces judgment. If that is correct, then developer maturity, structural constraints, and changeability economics are not separate concerns but interacting parts of one system. The convergence described here is conceptual and suggestive rather than conclusively demonstrated.

Foreword

This series marks twenty years of my own firm, Majer Consulting (majcon). It is not a survey of the field but the set of ideas that two decades of practice and training have most shaped, set down plainly. Read them as a practitioner's case, offered in good faith and open to challenge.

Key Takeaways

- AI appears to amplify existing discipline. Stronger developers and teams may gain leverage from AI assistance, while weaker structural practices can also scale faster.
- Generation and judgment are asymmetrical. AI may handle a large share of first-draft code, while architecture, integration, and evaluation remain stubbornly human-intensive.
- Three paths appear to converge. Psychology (ACE), constraint theory (CDD), and software economics (EoC) point toward a common structural insight: discipline is a multiplier, not overhead.
- Judgment is the meta-skill. The developer who understands when to accept AI output and when to reject it exercises the same ACE capacities: analytical rigor, creative evaluation, emotional calibration.
- Craftsmanship is not nostalgia. It is the economic strategy that keeps the cost curve flat when AI accelerates everything else.

1. The Amplifier Thesis

The three preceding papers made claims about developers, constraints, and economics. Each claim was scoped to its own domain. This synthesis begins with a question none of them addressed directly: what happens to these ideas when AI enters the picture?

The prevailing narrative treats AI as a uniform productivity multiplier. The currently available evidence suggests a more uneven picture.

The Discipline Divide

Early studies of AI-assisted development do not show a uniform productivity dividend. Some developers appear to gain leverage; others incur verification overhead, misplaced confidence, or quality regressions. A randomized controlled trial by Koning at Harvard Business School tracked developers using AI coding tools over three months [1]. Reported gains were uneven across performance groups, which is at least directionally consistent with an amplification effect rather than a simple replacement story.

The METR study [2] added a counterintuitive finding: experienced open-source developers using early-2025 AI tools were 19% slower on familiar tasks, while believing they were faster. The gap between perceived and actual performance is itself suggestive. It does not prove the trilogy's framework, but it does reinforce the importance of judgment and evaluation.

> AI raises the ceiling fastest for teams that already have a floor to stand on. Its gains compound with the discipline a team brings.
>
> Amplifier Thesis — inferred from the evidence reviewed in this trilogy

The 70% Problem

Osmani identifies a structural pattern in AI-assisted development [3]: generated code can cover a large share of the first draft, while integration, architecture, error handling, and edge cases remain disproportionately human-intensive. Whether the ratio is exactly 70/30 matters less than the asymmetry.

This is the Amplifier Thesis in miniature. Generation accelerates the easy-to-express portion of the work without resolving the harder problem of structural judgment. If anything, it can intensify that problem by increasing the volume of plausible code that still requires evaluation.

DORA: The Amplifier in the Data

DORA's research on AI-assisted software development [17] reaches the same conclusion from large-scale data. It describes AI as "an amplifier, magnifying an organization's existing strengths and weaknesses," and reports that the greatest returns come "not from the tools themselves, but from a strategic focus on the underlying organizational system." Where teams have not scaled their foundations (automated testing, CI/CD, review processes) to match AI-accelerated generation, delivery stability declines. This is the Amplifier Thesis in data form. Without the structural discipline that CDD describes and the judgment that ACE develops, AI-generated volume overwhelms the system. CodeScene's 2026 peer-reviewed study shows the same in the code itself: across 5,000 AI-refactored programs, AI-introduced defect risk was markedly higher in less-healthy code [23].

The Asymmetry of Verification

The structural bottleneck of AI-assisted development is the Asymmetry of Verification. While LLMs have collapsed the cost of generating plausible code, they have simultaneously increased the cognitive load of evaluation. In this environment, the role of the developer shifts from writer to validator. Without the structural guardrails provided by Constraint-Driven Development, the volume of generated code risks overwhelming the human capacity to verify its integrity, leading to a verification debt that can eventually stall delivery.

The Quality Tax

Several recent sources point in the same direction, though with different methods and levels of maturity. Some reports suggest that AI-generated code produces more maintainability, logic, or security issues than carefully reviewed human-written code [15]. GitClear's large-scale analysis also reported increases in code cloning and churn in the AI era [16]. DORA's AI-assisted development findings [17] add a system-level caution: faster code generation does not automatically translate into improved delivery stability. Across these sources, a directional pattern appears: code generation can increase output without reliably improving system-level outcomes.

> AI amplifies whatever discipline exists. It does not create discipline where none was present.

2. When Constraints Are Removed

The CDD paper [5] argues that calibrated constraints produce quality. Remove them without replacement and quality degrades. AI-assisted development is, in part, an experiment in constraint removal.

What AI Removes

Traditional software development imposes constraints through scarcity. Writing code is slow. Compilation catches syntax errors. Code review forces a second pair of eyes. Testing requires effort. Each friction point is a constraint that, intentionally or not, enforces a minimum quality bar.

AI removes several of these frictions simultaneously. Code generation becomes near-instant. But the constraints that made quality possible, the thinking time, the design consideration, the effort that forced prioritization, those vanish too.

> Every constraint that AI removes was also a quality gate. The question is not whether to use AI. The question is which constraints to reintroduce deliberately.

The Lean Parallel

The Poppendiecks observed something similar in manufacturing [6]. When Toyota removed waste from its production system, it simultaneously added constraints: pull-based flow, work-in-progress limits, andon cords. Removing waste without adding discipline produces chaos. Software development faces the same dynamic. AI removes the waste of manual code writing. The discipline that must be reintroduced is structural: CDD provides the guardrails, the ACE Model develops the human judgment to use them, and EoC provides the economic telemetry to measure their effectiveness.

From Individual to Orchestrator

The team structure itself is changing. Anthropic's 2026 Agentic Coding Trends Report found that developers integrate AI into 60% of their work but can fully delegate only 0-20% of tasks [18]. The role is shifting from writing code to orchestrating agents that write code. This is a different skill. It requires the judgment to evaluate generated output, the architectural sense to specify what should be built, and the discipline to reject what looks correct but is structurally unsound.

McKinsey is the most visible case study of this transformation. The firm reports deploying around 25,000 AI agents in under two years, with the stated goal of pairing every consultant with at least one agent [19]. Its hiring is reportedly shifting toward liberal arts majors, on the logic that creativity and judgment matter more than raw analytical skill when agents handle the analytical work. McKinsey's AI division, QuantumBlack, now advocates Spec-Driven Development: structured specifications drive agent output, while deterministic workflow engines control sequencing [20]. The parallels to CDD's constraint stack are not accidental. Both recognize that unconstrained generation produces volume, not value.

The implication for software teams may be significant: individual leverage can increase sharply when developers orchestrate capable tools well, but only when they also bring architectural judgment and disciplined evaluation.

The Five Rings

The trilogy examined three rings: who develops well, how to enforce quality, and what to optimize for. This synthesis adds two more. The fourth is the Amplifier: AI changes the stakes without changing the structure. The fifth is the Convergence: independent lines of research arriving at the same place.

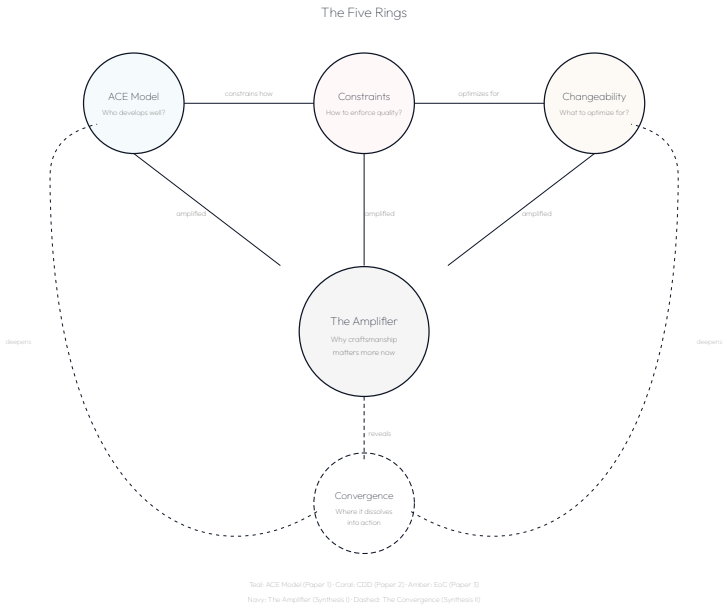

Fig. 1. The Five Rings. Three trilogy papers (teal, coral, amber) feed into the Amplifier (navy). The Convergence (dashed) emerges from their interaction and deepens each ring in return.

3. The Convergence

Three research traditions, pursued independently, arrive at mutually reinforcing conclusions.

From Psychology

The ACE Model [4] draws on positive psychology [7] and the Stoic virtues [8]. Its central claim, in its own words, is that the three dimensions are “mutually enabling rather than simply additive: severe weakness in one can limit the practical impact of the others.” Related traditions (Pestalozzi’s Head-Heart-Hand, Goleman’s emotional intelligence, Seligman’s PERMA model) reinforce the underlying premise that human performance is multi-dimensional.

From Constraint Theory

CDD draws on Westphal’s IOSP [9], Beck’s four rules of simple design [10], and Cabrera’s DSRP [11]. Its central claim is that calibrated structural constraints can prevent some recurring defect classes and materially reduce others. The Constraint Stack is a diagnostic lens: each of six levels targets a recurring defect area, and broader coverage reduces unguarded entry points.

From Software Economics

EoC [21] draws on Fricke and Schulz’s Design for Changeability framework [12], Martin’s cost curve [13], and Fowler’s Design Stamina Hypothesis [14]. Its central claim is that structural investment in changeability is argued to have economic returns. The Feature Cost Index provides a practitioner metric for tracking whether the investment is working. The DORA data now reinforces this from the opposite direction: teams that adopt AI without structural discipline appear to see their delivery stability decline [17], which is directionally consistent with the claim that velocity without structural discipline can accelerate cost growth.

Why do these three converge rather than merely coexist? Because each exposes the same bottleneck from a different angle. Strong constraints (CDD) without the judgment to apply them well (ACE) produce rigid, resented rules. Mature developers without economic feedback (EoC) have no way to tell whether their discipline is paying

off. A concrete case: a team introduces Value Objects (CDD Level 4) but lacks the Creative dimension (ACE) to model the domain cleanly. The constraint exists on paper, the implementation is awkward, adoption stalls, and the cost curve (EoC) keeps rising. Remove any one leg and the other two lose their multiplying effect.

The same logic appears from outside the trilogy. Clean documentation, well-defined interfaces, and explicit constraints help AI agents and human developers alike: the properties that make a system good for people make it good for machines. Deming [22] reached the same structural insight decades earlier from manufacturing quality: a bad system will beat a good person every time, and improving quality improves productivity rather than trading against it. The developer (ACE), the structure (CDD), and the economics (EoC) form a single system. The same properties that make code changeable by humans make it navigable by machines.

> Three questions. Three disciplines. One structural answer: the developer, the constraints, and the economics form a single system. Invest in one without the others and the returns diminish.

The Multiplicative Structure

The convergence is more than metaphorical, but it remains interpretive. Each paper identifies a weakest-link dynamic: ACE treats dimensions as mutually enabling, CDD treats structural levels as coverage-based: broader coverage reduces unguarded entry points, and EoC treats rising change cost as the economic symptom of structural weakness. Across all three, the system is constrained by its weakest component.

> From one thing, know ten thousand things. Study this well.
>
> Miyamoto Musashi, The Book of Five Rings (1645)

4. Five Rings

The trilogy began with a practical observation: technical training alone does not transform teams. Three papers traced the implication across psychology, constraint theory, and economics. This synthesis adds the amplifier and the convergence.

Five rings. Five questions.

1. **Who develops well?**
   The ACE Model. Analytical, Creative, and Emotional dimensions. All three, multiplicatively.

2. **How to enforce quality?**
   Constraint-Driven Development. Six levels: Verification, Code, Design, Architecture, Domain, System. Structure, not willpower.

3. **What to optimize for?**
   Economics of Changeability. Flat cost curve. Feature Cost Index. The economic argument for structural investment.

4. **Why does it matter more now?**
   The Amplifier. AI magnifies existing discipline. High performers gain leverage. Low performers accelerate structural degradation.

5. **Where does the system converge?**
   Diagnose people first (ACE: which dimension is weakest?), then structural gaps (CDD: where do defects cluster?), then economic trajectory (EoC: is FCI rising?). The convergence is seeing all three as one system — not three separate audits but one diagnostic applied through three lenses.

The fifth ring is different in kind from the first four — it emerges from their interaction rather than standing independently. The developer, the constraints, the economics, and the amplifier form a single system. Each reinforces the others. None works alone.

In trilogy terms, discipline is the combined expression: ACE capacities exercised through CDD constraints, measured by EoC outcomes.

What Remains Open

This synthesis is directional, not definitive. The Amplifier Thesis is the trilogy's central hypothesis, not a result established by the cited studies. It rests on early AI productivity studies and mixed-quality industry evidence. A strong counterexample would be a large-scale study showing that AI-assisted teams with weak structural foundations consistently outperform disciplined teams without AI tools. If such evidence accumulates, the amplification model would need revision. The convergence described here is structural rather than causal. Each component of the trilogy draws on some combination of prior literature and practitioner observation. The claim of the synthesis is that these components form a coherent hypothesis about software development under AI amplification. That hypothesis still requires longitudinal testing across multiple teams.

Where It Dissolves into Action

The question is not whether AI will change software development. That is settled. The question is what kind of developer thrives in an AI-amplified environment. The trilogy answered in parts. The synthesis answers as a whole.

Not the developer who writes the most code. Not the developer who adopts the latest tool. The developer who brings analytical rigor, creative judgment, and emotional intelligence to a structurally constrained practice, optimizing for the long-term cost curve rather than the short-term velocity metric.

That is the shift.

References

[1] Koning, R.: The Impact of Generative AI on Developer Productivity. Harvard Business School Working Paper (2025)
[2] METR: Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity. RCT with 16 developers (2025)
[3] Osmani, A.: Beyond Vibe Coding: From Coder to AI-Era Developer. O'Reilly (2025)
[4] Majer, D., Drumm, C.: The ACE Model. ASE Academy White Paper (2026)
[5] Majer, D.: Constraint-Driven Development. ASE Academy White Paper (2026)
[6] Poppendieck, M., Poppendieck, T.: Lean Software Development: An Agile Toolkit. Addison-Wesley (2003)
[7] Seligman, M.: Flourish. Atria Books (2011)
[8] Marcus Aurelius, Seneca, Epictetus: Stoic virtues. Various works, classical antiquity
[9] Westphal, R.: Integration Operation Segregation Principle. ralfwestphal.substack.com
[10] Beck, K.: Four Rules of Simple Design. Extreme Programming Explained. Addison-Wesley (1999)
[11] Cabrera, D., Cabrera, L.: DSRP Theory: A Primer. Systems 10(2), MDPI (2022)
[12] Fricke, E., Schulz, A.P.: Design for Changeability (DfC). Systems Engineering 8(4), pp. 342-359. Wiley/INCOSE (2005)
[13] Martin, R.C.: Clean Architecture. Pearson (2017). Chapters 1-2
[14] Fowler, M.: Design Stamina Hypothesis. martinfowler.com (2007)
[15] CodeRabbit: State of AI vs Human Code Generation Report. coderabbit.ai (2026)
[16] GitClear: AI Coding Assistant Code Quality Research (2025 update). 211M lines analyzed. gitclear.com (2025). Successor to 2024 report (153M lines)
[17] DORA: State of AI-assisted Software Development. Google Cloud / dora.dev (2025); ROI of AI-Assisted Software Development (2026). "AI as an amplifier"; returns from the organizational system, not the tools
[18] Anthropic: 2026 Agentic Coding Trends Report. resources.anthropic.com (Jan 2026)
[19] McKinsey & Company: AI Agent Deployment at Scale. Reported in Fortune, FinalRound AI (Feb 2026)
[20] QuantumBlack, AI by McKinsey: Agentic Workflows for Software Development. medium.com/quantumblack (Feb 2026)
[21] Majer, D.: The Economics of Changeability. ASE Academy White Paper (2026)
[22] Deming, W.E.: Out of the Crisis. MIT Center for Advanced Engineering Study (1986); MIT Press (2000). Chain reaction: improving quality raises productivity. "A bad system will beat a good person every time" (Deming, 1993 seminar)
[23] Borg, M., Hagatulah, N., Tornhill, A., Söderberg, E.: Code for Machines, Not Just Humans — Quantifying AI-Friendliness with Code Health Metrics. arXiv:2601.02200; ACM (2026). Code health determines AI defect risk

About the Author

Damir Majer coaches development teams on software quality at ASE Academy in Munich. This synthesis emerged from years of parallel research across psychology, constraint theory, and software economics. The trilogy papers are available at ase.academy.

Contact: d.majer@majcon.de | ase.academy