WHITE PAPER

The Economics of Changeability

Why Every Feature Should Cost the Same

Damir Majer
ASE Academy, Munich

May 2026
ASE ACADEMY

Contents

Who This Paper Is For

CTOs, engineering managers, and technical leaders who see delivery velocity declining as their codebase grows. This paper presents a measurement-backed model for maintaining flat cost per feature through structural design principles. It completes the ASE Academy trilogy: the ACE Model (Analytical, Creative, Emotional) develops the developer, Constraint-Driven Development (CDD) enforces quality structurally, and this paper explains what to optimize for.

Abstract

As software systems age, the cost of delivering the next feature often rises. This paper sets out the Economics of Changeability (EoC): it argues that changeability is the economically central quality attribute and proposes the Feature Cost Index (FCI) as a practical signal of modification pressure per delivered slice. The paper integrates systems-engineering work on changeability, software-design literature on cost of change, and practitioner observations from coached projects. The central claim is operational rather than absolute: if marginal feature delivery cost remains stable over time, the architecture is likely resisting structural degradation. The model makes empirical predictions, but the current evidence base remains early.

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

- Healthy systems aim to keep marginal feature cost from rising with age. Later features should not become systematically more expensive solely because the codebase has grown.
- Changeability is four things, not one. Robustness, Flexibility, Agility, and Adaptability are distinct aspects requiring different design investments.
- Nine principles, not opinions. Three basic and six extending principles transfer from systems engineering into software design.
- The crossover may arrive earlier than teams expect. Fowler's Design Stamina Hypothesis suggests that structural investment can pay off surprisingly early in non-trivial projects.
- The Feature Cost Index operationalizes modification pressure. Rising FCI is an early warning signal. Flat FCI is consistent with architectural health, though not sufficient on its own to prove it.

1. The Cost Paradox

Every organization I work with tells the same story. The first features ship fast. The team is productive. The stakeholders are happy. Then, gradually, each new feature takes longer. Not because the team got worse. Because the codebase got harder to change.

This is not a new observation. Lehman [16] formalized it as the Law of Increasing Complexity: a program that is used undergoes continual change, and that change increases its complexity unless work is done to reduce it. Cunningham [17] gave us the metaphor: technical debt. Robert C. Martin [1] describes it as "the signature of messy code": a graph where cost per feature rises asymptotically, approaching infinite cost. The counter-graph, the one every CTO wants, shows cost per feature as a flat line. Martin argues that structure is more valuable than behavior because a system that works but cannot be changed is eventually worthless.

The question is: is the rising curve inevitable?

Quality shows up economically as stable marginal cost of change over time

The industry measures quality by defect counts, test coverage, and deployment frequency. These are useful proxies. The business-facing question is more direct: does the next feature become more expensive simply because the system is aging? If yes, something structural may be degrading. This is the economic counterpart to CDD's structural definition: constraint coverage prevents defect classes; stable cost confirms the investment is working. (Throughout this paper, feature and standardized slice are used interchangeably for the unit of delivered behavior; Section 5 provides the operational definition.)

2. Flatten the Curve

> Flatten the curve.
>
> Francesco Cirillo, in coaching sessions

Five years before this paper was written, Francesco Cirillo started using this phrase in coaching sessions. Not "keep the curve flat." Flatten it. The verb matters. It implies the curve naturally rises. Every codebase, left to entropy, accumulates coupling, complexity, and rigidity. The cost per feature goes up. The question is not whether the curve will rise. It will. The question is whether you actively flatten it.

The structural foundation behind this instruction has a separate origin. Years before working with Cirillo, I learned the Integration/Operation Segregation Principle [12] from Ralf Westphal. IOSP became the North Star: every function is either an integration (orchestrates other functions) or an operation (pure computation), never both. This single structural discipline enforces Simplicity and Autonomy simultaneously. It is the practice that keeps the curve flat. When I began working with Cirillo in 2020 and heard "flatten the curve," the connection became clear: IOSP was the structural mechanism behind the coaching goal.

This was the starting point where multiple research threads began to integrate. Martin's cost curve [1] described the problem. Fricke and Schulz's Design for Changeability (DfC) [2] formalized the theory. Fowler [3] established the timing. IOSP and CRAFT Calisthenics provided the structural practices. But the coaching insight came first: flatten the curve as a daily discipline, not a one-time architectural decision.

Martin Fowler's Design Stamina Hypothesis [3] supports the economic intuition. The crossover point where design discipline begins to pay back can arrive surprisingly early in non-trivial projects. The exact timing is illustrative rather than universal.

The rest of this paper provides the vocabulary, the principles, the measurement, and the evidence. But the action was always simple: flatten the curve.

---

3. Four Aspects, Nine Principles

Changeability is not one thing. Fricke and Schulz [2], working in systems engineering at the Technical University of Munich, decomposed it into four distinct aspects. Ross and Rhodes [15] extended the taxonomy to system-of-systems scale.

ASPECT	DEFINITION	INITIATOR	CHARACTER
Robustness	System absorbs environmental change without modification	Nobody	Passive
Flexibility	System can be changed easily by a human	Developer	Low effort
Agility	System can be changed rapidly by a human	Developer	Low time
Adaptability	System adjusts itself to new conditions	The system	Autonomous

Flexibility and Agility both require a human to change the system, but they measure different things. A plugin architecture is flexible (adding a payment provider is easy). Feature flags make a system agile (changes deploy fast). Auto-scaling makes it adaptable (no human intervention needed). Confusing these aspects leads to wrong architectural investments.

The Nine Design Principles

Fricke and Schulz propose nine principles in two tiers. The first three are basic: they support all four aspects. The remaining six are extending: they support selected aspects.

Basic Principles (support all four aspects)

1. Simplicity

Fewer parts, fewer interactions, fewer things that break. The simplest system that meets current needs is the most changeable.

2. Independence

Changing component A should not require changing component B. Minimize coupling so changes do not cascade.

3. Modularity

Decompose into well-bounded units with clear interfaces. Modules are the unit of change.

Extending Principles (support selected aspects)

PRINCIPLE	SOFTWARE EXAMPLE	SUPPORTS
4. Integrability	Plugin systems. Strategy patterns. Versioned APIs	Flex + Agil
5. Autonomy	Self-contained services with own data. Pure functions. IOSP [12]	Rob + Adapt
6. Scalability	Stateless services. Horizontal scaling. CDN layers	Flex + Agil
7. Non-hierarchical	Event-driven. Pub/sub. Choreography over orchestration	Agil + Rob
8. Decentralization	Each service owns its data and logic. Team ownership	Rob + Adapt
9. Redundancy	Multi-AZ. Circuit breakers. Queue-based resilience	Rob

These nine principles are not opinions. They come from systems engineering, where changing a deployed satellite costs millions. The principles transfer directly to software. The only difference is what “module” and “interface” mean at each scale.

> The design principles are domain-independent. The question is always: which changes are likely, and how can the system accommodate them at minimal cost?
>
> Fricke & Schulz [2]

4. The U-Curve

Changeability has a cost. Too little investment means every modification is surgery. Too much means over-engineering that is never used. Fricke and Schulz [2] model this as a U-curve with an optimal point.

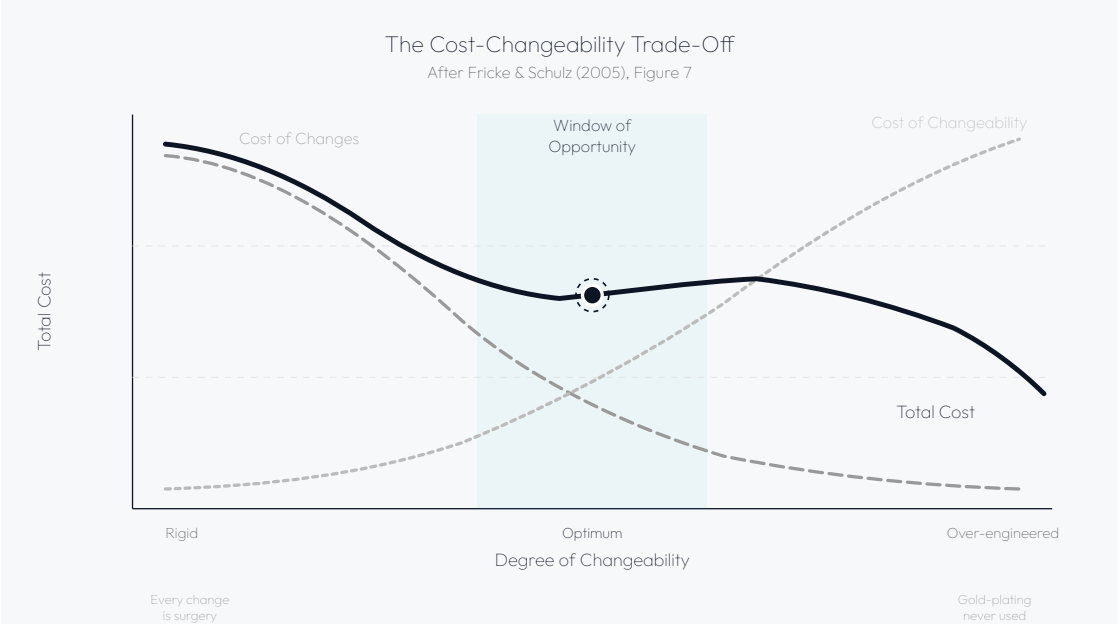

Fig. 1. The cost-changeability trade-off. Two curves cross at the optimum: the "window of opportunity" where total cost is minimized. After Fricke & Schulz (2005).

The left side of the curve is familiar to every team that has inherited a rigid codebase. Every change requires touching ten files. The right side is familiar to teams that invested six months in a plugin framework that supports two plugins.

Kleiser [10] maps this to software practice. Low-cost changeability investments that every codebase should have: externalized configuration, named constants, facades and interfaces, good naming. High-cost investments that require deliberate decision: domain-driven design, messaging systems, ORM abstractions. The distinction is economic, not technical: invest in changeability where change is likely. Accept rigidity where requirements are stable.

This maps directly to the YAGNI principle. YAGNI is not an argument against design. It is an argument for targeted design: invest at the optimum, not at the extremes.

Fowler's Timing Argument

How quickly does the investment pay off? Fowler's Design Stamina Hypothesis [3] suggests that the crossover from undisciplined speed to design-enabled speed can occur surprisingly early. This is a conceptual argument, not a universal timing law.

> The pseudo-graph is illustrative, but the message is the important thing. Design up front invests to help build speed later on.
>
> Martin Fowler [3]

5. The Complete Chain

What was missing was the connection: a single chain from problem to evidence with practices and measurement in between. This paper's contribution is integration and operationalization: connecting independently established results into one falsifiable argument and providing the Feature Cost Index as a practical measurement instrument.

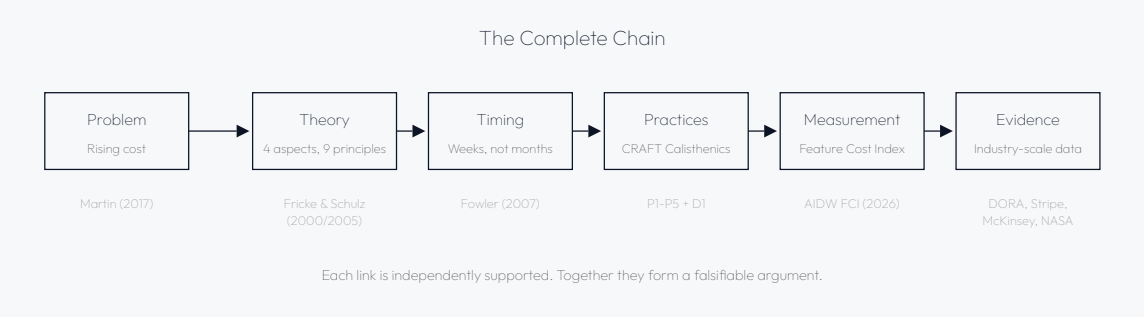

Fig. 2. The Complete Chain. Six independently supported links from problem to evidence.

Each link is independently supported. Together they form a falsifiable argument.

From Principles to Practice

CRAFT Calisthenics enforce Fricke's three basic principles through practice constraints. Integration/Operation Segregation [12] enforces Simplicity (each function does one thing) and Autonomy (operations have zero dependencies). The companion paper on Constraint-Driven Development [14] describes how these constraints can prevent recurring categories of defects. This paper adds the economic argument: the same constraints that flatten the cost curve are also quality tools. They are economic instruments.

Feature Cost Index

We introduced the Feature Cost Index (FCI) as an operational signal:

$$\text{FCI} = \text{filesModified} / \text{slicesDelivered}$$

FCI measures modification pressure per delivered slice. filesModified counts existing files changed during a development cycle. slicesDelivered counts standardized feature slices completed in that cycle. A companion diagnostic, extensionRatio = new files / total files touched, helps interpret whether delivery happened primarily through extension or modification.

- Rising FCI = structural friction may be increasing.
- Flat FCI = modification pressure is stable. Consistent with healthy architecture, though not sufficient on its own to prove it.
- Falling FCI = refactoring or clearer modularity may be paying off.

FCI is a lightweight operational signal, not a complete theory of architectural health. It becomes more useful when interpreted alongside extension ratio, defect trends, and architectural context.

6. The Evidence

The individual claims are backed by large-scale data.

FINDING	NUMBER	SOURCE
Developer time lost to technical debt	42% of work week	Stripe [4]
Bug fix cost: production vs review	30-100x more expensive	NASA, McConnell [6]
Elite vs low: deployment frequency	182x more frequent	DORA [5]
Elite vs low: change failure rate	8x lower	DORA [5]
Defect reduction from DevEx investment	20-30%	McKinsey [8]
Defects in low-quality vs high-quality code	15x more	CodeScene [9]
Annual cost of poor software quality (US)	$2.41 trillion	CISQ [7]
Time on new work: high vs low performers	49% vs 38%	DORA [5]
DevEx score to time saved	1-pt DXI = 13 min/week/dev	Forsgren & Noda [20]
SCENARIO ESTIMATE (not empirical)		
Illustrative slowdown scenario in rigid codebases	2-3x slower	Scenario estimate informed by [4], [5]

The DORA finding is critical: speed and stability are not trade-offs. Elite teams deploy 182 times more frequently and have 8 times lower change failure rates. Changeability is not a luxury that slows you down. It is a structural property that enables both speed and stability simultaneously.

> Speed and stability are not trade-offs. Elite teams deploy 182 times more frequently and have 8 times lower change failure rates.

The ROI Model

The following model is best read as a scenario tool, not a valuation formula. For a team of N developers at average salary S:

1. Waste: N × S × 0.42 = annual cost of debt (Stripe)
2. Bug tax: Production bugs × 30-100x multiplier (NASA/McConnell)
3. Velocity tax: Features take 2-3x longer in rigid codebases
4. Turnover risk: 23% attrition, each costing 1.5-2x salary

For a 20-developer team at EUR 110K average, the model yields roughly EUR 1.8M per year in theoretical friction cost under the stated assumptions. Because the inputs are heterogeneous (self-reported surveys, mission-critical contexts, correlational data), this should be read as an order-of-magnitude scenario rather than a precise forecast. A healthy system minimizes but cannot eliminate this baseline.

Forsgren and Noda [20] put a number on this from a different angle: every 1-point gain in an organization's Developer Experience Index (DXI) saves about 13 minutes per week per developer, which compounds to roughly 10 hours per year. At Etsy, framing deployment friction in business terms ("50% of engineering hours spent waiting for deployment") translated into roughly 500 hours of engineering time recovered. DXI and FCI measure the same

thing at different scales. DXI captures perceived friction. FCI captures structural friction. Both tell you whether the cost curve is flattening.

7. Evidence and Honest Limits

What We Know

The evidence in Section 6 suggests the pattern at industry scale, but it does not test this paper's full argument end to end.

Fricke and Schulz's framework is well-cited. The software translation is newer but grounded in established research [10, 11]. Feathers [18] provides practitioner support for the underlying idea that cost of change depends on structural properties of the code, not only on feature size.

AI-era data is directionally consistent with the pattern. DORA's research on AI-assisted software development [19] describes AI as "an amplifier, magnifying an organization's existing strengths and weaknesses," and reports that delivery stability tends to decline when AI-driven code generation outpaces a team's structural practices. This does not prove the EoC model, but it is consistent with the claim that faster generation without structural discipline can increase change failures.

Coaching observation. Eight progressive cycles on a verification project (Werkstatt, four backend cycles, four frontend cycles) show relatively stable modification pressure despite growing codebase complexity. FCI was reconstructed from git history for each cycle:

CYCLE	NEW	MOD.	SLICES	EXT. RATIO	FCI
Backend FL	8	0	6	1.00	0.00
Backend CL	7	4	7	0.64	0.57
Backend AL	7	2	7	0.78	0.29
Backend PL	11	0	8	1.00	0.00
UI FL	8	0	4	1.00	0.00
UI CL	5	2	5	0.71	0.40
UI AL	5	4	6	0.56	0.67
UI PL	8	3	7	0.73	0.43

Backend FCI ranges from 0.00 to 0.57, with the notable result that PL (the most architecturally complex cycle, introducing bounded contexts and domain events) achieved FCI = 0.00: zero existing files required modification. Frontend FCI peaks at 0.67 during AL, when approval and warranty features needed to surface in existing views. Across 8 cycles, each slice modified fewer than one existing file on average. The majority of work was extension. Data was reconstructed from git history; source file counts exclude tests, config, and infrastructure. This is illustrative single-project evidence; replication is needed.

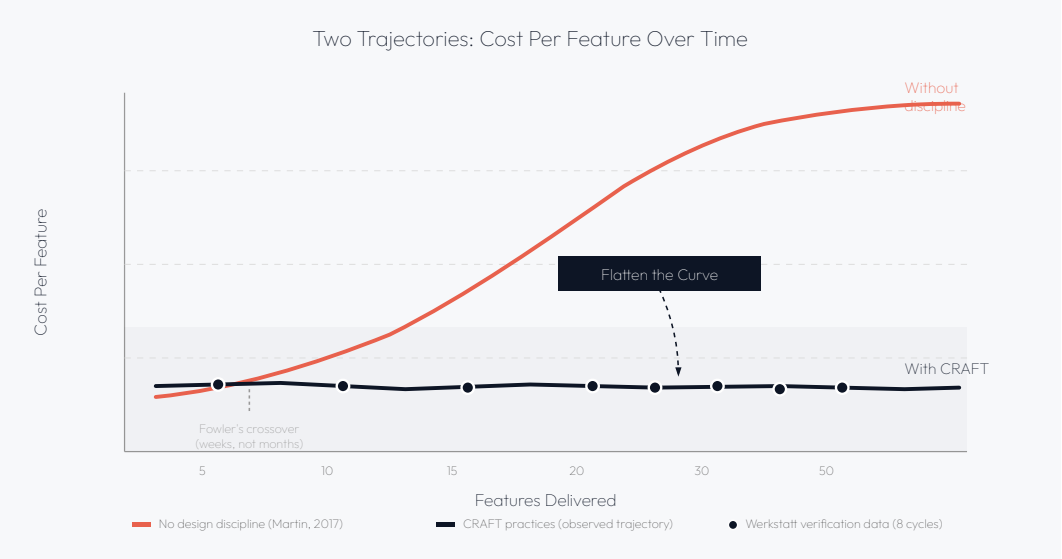

Fig. 3. Two trajectories. The rising curve represents Martin's asymptotic model. The flatter line represents observed behavior under CRAFT Calisthenics in one verification project. FCI data reconstructed from git history (source files only, tests excluded). Single-project evidence; replication needed.

What We Do Not Know

FCI is new. The metric has been applied to only a small number of projects. The sample size is insufficient for statistical claims. At present, the argument is operational and mechanistic rather than statistically established.

No controlled experiment. There is no randomized trial comparing teams with and without changeability investment. The DORA data is correlational. The coaching observations are uncontrolled. The ROI model is approximate. The Stripe 42% figure is self-reported. The NASA multiplier comes from mission-critical contexts. Extrapolating to all software teams requires caution. The Complete Chain has not been tested end to end. Each link is independently supported to varying degrees, but the combined effect has not been measured in a single study.

> The honesty is deliberate. The claim is directional: structural investment in changeability appears to have economic returns. The exact magnitude varies by context.

8. The Diagnostic

A model earns its keep when it becomes measurable. The Feature Cost Index turns the flywheel in Fig. 4 from a picture into an instrument: track it across delivery cycles and you can see which loop your team is in, and whether a structural intervention moved you off the treadmill.

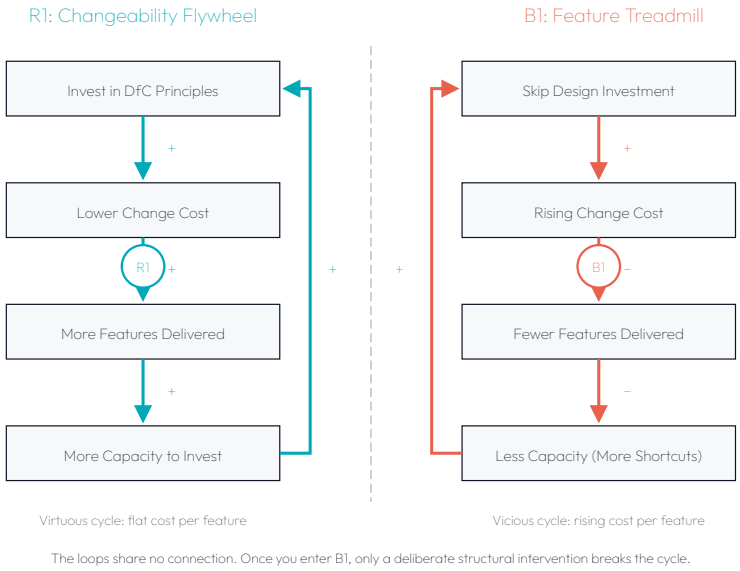

Fig. 4. R1 (teal): the Changeability Flywheel. Investing in DfC principles lowers change cost, enabling more features, freeing capacity for further investment. B1 (coral): the Feature Treadmill. Skipping design investment raises cost, reducing capacity, forcing more shortcuts.

To apply this model as a diagnostic:

1. **Measure Your FCI Baseline**
   For the last 3-5 standardized slices, count: how many existing files were modified? How many new files created? How many slices delivered? Compute FCI. This is your baseline.

2. **Identify Your Weakest DfC Principle**
   If changes cascade: Independence is weak. If nobody understands the code: Simplicity is weak. If changes touch 10 files: Modularity is weak. The symptom points to the principle.

3. **Apply One Structural Intervention**
   Not nine. One. Introduce interfaces at module boundaries (Independence). Extract small functions (Simplicity). Regroup code by domain concept (Modularity). One cycle. One principle.

4. **Compare FCI Next Cycle**
   Compute FCI again after the next 3-5 slices. If it dropped: the intervention worked. If not: the wrong principle was targeted, or the intervention was too shallow. Adjust and repeat.

9. Not Faster Delivery. Flatter Cost.

The question is not whether your team can ship faster. Speed without structural investment is the B1 loop in Fig. 4: each shortcut raises the cost of the next feature. The team sprints while the ground tilts upward.

The deeper question is whether later features become more expensive simply because the system has aged. That is what changeability means in economic terms.

Here, constraint shifts from the structural sense used in CDD to an economic one: the level of structural discipline that minimizes total cost of change. Structural discipline is not a quest for absolute restriction, but for the Window of Optimal Constraint (CDD introduces this concept in structural terms). Too few constraints produce entropy – recurring defect classes that drain resources. Over-constraint produces rigidity – where evolving the design becomes prohibitive. The economic sweet spot is where constraints are strong enough to prevent recurring defect classes (CDD) but flexible enough to allow for the pivots that changeability requires.

This paper completes a trilogy: the ACE Model [13] develops the developer, CDD [14] enforces quality structurally, and this paper provides the economic argument for what to optimize. Fig. 5 shows the connections.

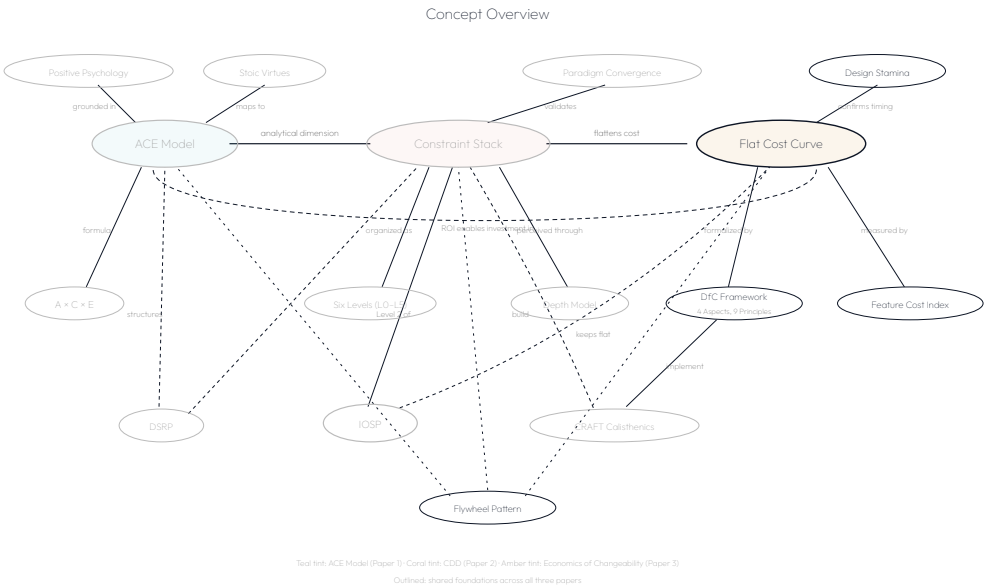

Fig. 5. Concept overview of the trilogy. Teal: ACE Model. Coral: Constraint-Driven Development. Amber: Economics of Changeability (this paper). Navy: shared foundations.

Not faster delivery. Flatter cost. That is the shift.

Appendix: Methods

Feature Cost Index: Computation Protocol

FCI is computed per development cycle (typically 3–5 features or one sprint):

1. Count files modified (existing files that were changed).
2. Count files added (new files created).
3. Count slicesDelivered (merged PRs that modify user-facing behavior and include at least one test; see Standardized Slice definition below).
4. Compute FCI = filesModified / slicesDelivered.
5. Compute extensionRatio = filesAdded / (filesAdded + filesModified) as a companion diagnostic.

A rising FCI across cycles indicates increasing modification pressure. A flat or falling FCI suggests stable or improving architecture.

Operational Definition: The Standardized Slice

To ensure the Feature Cost Index is a reliable metric, the denominator (slicesDelivered) must be standardized. For this model, a Slice is defined as a single Pull Request (PR) that:

1. Modifies Behavior: Adds or changes user-facing or API-level functionality.
2. Is Integrated: Merged into the primary branch (main/master).
3. Is Verified: Includes at least one automated test covering the change.

Pure refactoring PRs, documentation updates, and CI/infrastructure changes are excluded from the slice count. This ensures FCI measures feature delivery cost, not maintenance overhead.

Data Source

The Werkstatt verification project (Section 7) comprises 8 progressive cycles: 4 backend (FL, CL, AL, PL) in Deno/TypeScript and 4 frontend (FL, CL, AL, PL) in Preact/HTM. File counts include domain and feature source files only (tests, config, barrel exports excluded). Slice counts derive from AIDW cooldown reports. FCI was computed per cycle as filesModified / slicesDelivered. This is a single project; replication is needed.

Falsifiability

The model predicts: (1) codebases with stronger structural investment will tend to show more stable modification pressure over time; and (2) codebases with weaker structural investment will be more likely to show rising FCI across delivery cycles. A strong counterexample would be a structurally weak codebase that maintains stable modification pressure over many comparable cycles, or a structurally disciplined codebase whose FCI rises persistently without plausible external explanation.

Limitations

- FCI has been applied to a small number of projects. The metric is new and not independently validated.
- The ROI model combines heterogeneous data sources. The EUR 1.8M figure is an order-of-magnitude estimate.
- The Complete Chain is a logical argument connecting independently supported links. The combined end-to-end effect has not been measured in a single study.
- The Stripe 42% figure is self-reported. The NASA/McConnell 30-100x multiplier comes from mission-critical contexts that may not generalize.

References

[1] Martin, R.C.: Clean Architecture. Pearson (2017). Chapters 1-2: "The signature of messy code"
[2] Fricke, E., Schulz, A.P.: Design for Changeability (DfC). Systems Engineering 8(4), pp. 342-359. Wiley/INCOSE (2005)
[3] Fowler, M.: Design Stamina Hypothesis. martinfowler.com (2007)
[4] Stripe: The Developer Coefficient. stripe.com (2018)
[5] DORA: State of DevOps Report. Google Cloud / dora.dev (2024)
[6] McConnell, S.: Code Complete. Microsoft Press (2004). Ch. 29 (cost of defects)
[7] CISQ: The Cost of Poor Software Quality in the US. Consortium for Information & Software Quality (2022)
[8] McKinsey: Yes, You Can Measure Software Developer Productivity. mckinsey.com (2023)
[9] Tornhill, A., Borg, M.: Code Red — The Business Impact of Code Quality. CodeScene (2022). 39 codebases; low-quality code has 15x more defects. See also Software Design X-Rays (2018) and Borg, Hagatulah, Tornhill, Söderberg: Code for Machines, Not Just Humans (2026), finding AI defect risk markedly higher in unhealthy code
[10] Kleiser, F.: Changeability in Software Architectures. fabian-kleiser.de/blog
[11] Ford, N., Parsons, R., Kua, P.: Building Evolutionary Architectures. O'Reilly (2017)
[12] Westphal, R.: Integration Operation Segregation Principle. ralfwestphal.substack.com
[13] Majer, D., Drumm, C.: The ACE Model. ASE Academy White Paper (2026)
[14] Majer, D.: Constraint-Driven Development. ASE Academy White Paper (2026)
[15] Ross, A.M., Rhodes, D.H.: Defining Changeability. Systems Engineering 11(3), pp. 246-262 (2008)
[16] Lehman, M.M.: Programs, Life Cycles, and Laws of Software Evolution. Proc. IEEE 68(9) (1980)
[17] Cunningham, W.: The WyCash Portfolio Management System. OOPSLA Experience Report (1992). Origin of the "technical debt" metaphor
[18] Feathers, M.: Working Effectively with Legacy Code. Prentice Hall (2004)
[19] DORA: State of AI-assisted Software Development. Google Cloud / dora.dev (2025); ROI of AI-Assisted Software Development (2026). "AI as an amplifier"; delivery stability declines without strong practices
[20] Forsgren, N., Noda, A.: Frictionless: 7 Steps to Remove Barriers, Unlock Value, and Outpace Your Competition in the AI Era. Shift Key Press (2025). DXI metric (1-pt = 13 min/week/dev). Etsy, Block, Capital One case studies

About the Author

Damir Majer coaches development teams on software quality at ASE Academy in Munich. This paper connects a coaching insight by Francesco Cirillo ("flatten the curve") with the economic argument for structural investment. The ACE Model and CDD papers complete the trilogy.

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