Emergence: A Programming Paradigm for Constraint-Shaped Agency

Jeremy McEntire
Cage & Mirror Press
jmc@cageandmirror.com

March 2026

Abstract

The programmer does not write the program. The programmer writes the conditions under which the program emerges. We propose *Emergence* as a programming paradigm in which the fundamental unit of computation is an autonomous agent that interprets messages within a constraint architecture—schemas, thresholds, scoring functions, and topological rules—and the system's behavior arises from the interaction of agents operating within those constraints. This paradigm recovers Alan Kay's original vision of object-oriented programming, which required a computational substrate with contextual understanding that did not exist until large language model agents.

We formalize four well-formedness conditions—completeness, liveness, dissipation, and contraction—that distinguish engineering from hope. The fourth condition, contraction, is the key: we show that the Q-dynamics operator governing the system's acceptability distribution is a contraction mapping, guaranteeing convergence to a unique fixed point. This contraction result simultaneously resolves the control problem: an Emergence system converges to a predictable operating regime not because a controller forces it there, but because the constraint architecture's own dynamics are contractive.

We ground the paradigm in convergent results from cybernetics, modular design theory, organizational theory, and complex systems. We validate it against five empirical studies, two existence proofs—a working stigmergic mesh and Tessera, a self-validating executable document format—and the multi-agent systems literature. We address the paradigm's real limitations—latency, cost, non-determinism, and the genuinely new discipline of designing constraint spaces—with equal honesty.

1 Introduction: The Programming Paradigm Gap

Every programming paradigm answers one question: *where does the intelligence live?* In procedural programming, it lives in the sequence of instructions. In functional programming, it lives in the composition of transformations. In object-oriented programming—as practiced, not as conceived—it lives in the class hierarchy.

In each case, the programmer encodes the intelligence directly. The system does what it is told. This works when the problem is fully specifiable in advance. It fails when the problem space exceeds the programmer's capacity to enumerate its states—which is to say, it fails for every interesting problem in every sufficiently complex domain.

The arrival of large language model (LLM) agents creates a genuinely new possibility. For the first time, the computational unit itself possesses contextual understanding. An LLM agent assessing a code review comment does not pattern-match against a list of phrases; it understands what code review is *for*, recognizes when a comment is substantive versus ceremonial, and distinguishes a genuine technical concern from a territorial objection. This capability is not incremental improvement. It is a qualitative change in what a computational unit *can be*.

We call the resulting paradigm **Emergence**. The taxonomy is clean:

> *Procedural programming: you write the steps. Object-oriented programming: you write the entities. Functional programming: you write the transformations. Emergence: you write the constraints, and the behavior writes itself.*

The program is not the behavior. The program is the physics, the constraints, the container, the field equations. The behavior is the emergent property of the system the programmer designed—not the system itself. This shift implies not merely a new tool but a new discipline: the skill of designing constraint spaces that reliably produce useful behavior is neither programming as currently practiced nor management as currently understood, and its development is as much a part of the paradigm's contribution as its technical architecture (Section 10.4).

2 The Kay Recovery

Alan Kay has spent decades clarifying what he meant—and what he did not mean—by "object-oriented programming." The clarifications have been largely ignored.

In a 2003 email, Kay wrote: "I'm sorry that I long ago coined the term 'objects' for this topic because it gets many people to focus on the lesser idea. The big idea is messaging" [Kay, 2003]. In his Smalltalk work and subsequent reflections, Kay consistently described three properties:

1. Each object is an autonomous entity—"a recursion of the whole computer"—with its own state, behavior, and interpretation of incoming messages.
2. Communication happens exclusively through messages, and the *receiver* decides what a message means.
3. The system's behavior emerges from the interaction of these autonomous message-interpreting entities, not from a central controller.

What the industry built was different. Method dispatch replaced message interpretation: a message became a function call, resolved at compile time or through a vtable, with the caller knowing exactly what would happen. Class hierarchies replaced autonomy: objects became instances of types, their behavior determined by inheritance chains. Design patterns replaced emergence: the Gang of Four catalog [Gamma et al., 1994] codified strategies for managing complexity that the paradigm was supposed to dissolve.

The misimplementation was not stupidity. It was a technological constraint. In 1972—or 1995, or 2015—there was no computational substrate that could *interpret* a message. A function can parse one. A pattern matcher can classify one. A rule engine can apply conditions to one. But none of these *understands* one. The receiver cannot decide what a message means if the receiver has no capacity for meaning.

Kay's vision required a computational unit with genuine contextual understanding. It required exactly what an LLM agent provides.

3 Emergence Architecture: Constraints, Runtime, Game State

If the agents provide the flexibility, what provides the reliability? This is the central engineering question of the paradigm, and its answer is the paradigm's central insight: **the program is the boundary, not the behavior.**

In Emergence, the programmer designs the constraint architecture within which agents operate: *rigid at the interfaces, flexible in the interiors.*

3.1 The Agent as Computational Primitive

An LLM agent is not a better function. It is a different kind of computational entity. The distinction matters because it is the distinction Kay was trying to draw.

A function *computes*: given inputs, it produces outputs according to a fixed algorithm. An agent *assesses*: given a situation, it produces a judgment shaped by its understanding and context. A function that detects scope creep by counting changed lines is doing something fundamentally different from an agent that reads a pull request and recognizes that what was filed as a "bug fix" is actually a feature rewrite touching authentication, database schema, and three API contracts simultaneously. The function applies a rule. The agent understands a situation.

This distinction enables three properties previously unavailable:

Genuine message interpretation. When an agent receives a signal—a pull request, a customer ticket, a deployment log—it does not merely parse the text. It interprets meaning in context. A large change touching authentication logic receives different attention than an identically sized change adding stylesheets, not because of a rule about filenames, but because the agent understands what authentication code is and why changes to it carry risk.

Judgment under ambiguity. Traditional computation requires the programmer to anticipate every case. An agent exercises judgment in situations the programmer never considered, because its training encompasses the breadth of human knowledge about how to reason in context. This is the same capability that allows a new employee to handle situations not covered by the employee handbook.

Compositional emergence. When multiple agents, each interpreting messages within their own context, interact through a shared environment, the system-level behavior is not the sum of the agents' behaviors. Patterns, correlations, and structures arise that no individual agent was designed to detect. This is the property that Kay described and that class-hierarchy OOP could never deliver.

3.2 Rigid Boundaries

The boundaries are non-negotiable structural elements that ensure coherence:

Schemas. The data structures agents consume and produce are strictly typed. A signal has a source, timestamp, content, and metadata. An assessment has a confidence score, a set of claims, and a provenance chain. The agent decides what to *put* in these fields; the schema decides what fields *exist*.

Thresholds. Numerical parameters that gate behavior without prescribing it. A vigilance parameter determines how similar a signal must be to existing knowledge before an agent accepts it. An energy budget determines how many signals an agent can process before it must yield. The agent exercises judgment *within* these bounds; the bounds themselves are fixed.

Scoring functions. Mathematical functions that evaluate outputs without understanding them. A familiarity score combines lexical overlap, semantic similarity, structural alignment, and temporal recency into a single number. The agent does not know it is being scored. The scoring function does not know what the agent is thinking. The decoupling is the point: the meta-system shapes selection pressure without micromanaging cognition.

Topological rules. Rules governing how agents connect, communicate, and compete. Which agents see which signals. How routing priority is determined. When agents fork, merge, or decay. These rules define the *structure* of interaction without constraining the *content*.

3.3 Flexible Interiors

Within the boundaries, agents operate with genuine autonomy. They interpret signals, form assessments, develop specializations, and generate insights without procedural instruction. The programmer does not specify *how* an agent should assess whether a team's communication patterns are degrading. The programmer specifies *that* communication health is a property to assess, provides a schema for reporting it, and trusts the agent's contextual understanding to do the rest.

This is the inversion that defines the paradigm. In traditional programming, the interior is rigid (algorithms, control flow, explicit logic) and the boundaries are flexible (APIs change, schemas evolve). In Emergence, the boundaries are rigid and the interiors are flexible. The programmer's job shifts from *writing behavior* to *shaping the space in which behavior occurs.*

3.4 Recursive Constraint Interaction

The constraint architecture is not a thermostat with one feedback loop. It has many degrees of freedom (one per agent per context) and many feedback loops (scoring, routing, lifecycle, consensus) that *interact.* The vigilance threshold affects which signals an agent accepts, which affects its specialization, which affects the topology, which affects which signals it receives, which affects whether the vigilance threshold should change.

This recursive interaction is the mechanism of emergence. The constraint architecture creates conditions for self-organization; the self-organization produces capabilities that no individual constraint was designed to produce.

4 Formal Well-Formedness

If the paradigm is to be engineering rather than aspiration, constraint architectures must have well-formedness conditions. We define four conditions that, taken together, guarantee that an Emergence system is bounded, alive, dissipative, and convergent. The first three were identified informally in prior work; here we state them precisely and add the fourth—contraction—which is the condition that transforms the paradigm from a design philosophy into a formal system with convergence guarantees.

Definition 4.1 (Constraint Architecture)**.** A constraint architecture $\mathcal{A} = (C, L, E)$ consists of a set of constraints C operating on state space $S = (\text{Agents}, \text{Signals}, \text{Topology}, \text{Energy})$, lifecycle rules $L = \{\text{fork}, \text{decay}, \text{merge}\}$, and an energy function $E\colon S \to \mathbb{R}_+$. The constraints decompose into four primitive types:

- **Schema constraints** $\sigma\colon \text{Signals} \to \{\text{valid}, \text{invalid}\}$: deterministic, stateless, compositional.
- **Threshold constraints** $\tau = (f, \theta)$: a scoring function $f\colon S \to \mathbb{R}$ and threshold $\theta \in \mathbb{R}$ gating behavior.
- **Scoring constraints** $\kappa\colon \text{Outputs} \to [0, M]$: total, deterministic, bounded functions assigning scores to agent outputs.
- **Topological constraints** $\rho\colon \text{Topology} \times \text{Signals} \to \text{Topology}$: the only stateful primitive, governing how agent topology evolves.

Definition 4.2 (Well-Formedness Conditions)**.** A constraint architecture $\mathcal{A}$ is *well-formed* when it satisfies conditions WF1–WF4:

(WF1) Completeness. For every agent a and every output o produced by a, there exists at least one scoring constraint κ such that $\kappa(o)$ is defined. No agent can produce assessments that bypass evaluation. This is the analog of type safety: the system guarantees that all behavior is bounded, even if the specific behavior is not predicted.

(WF2) Liveness. The lifecycle rules $L = \{\text{fork}, \text{decay}, \text{merge}\}$ are non-trivially reachable: for each rule $l \in L$, there exists a reachable state $s \in S$ such that l is triggered at s. A system that can only grow or only shrink is not well-formed.

(WF3) Dissipation. The energy function $E \colon S \to \mathbb{R}_+$ is bounded, monotonically non-increasing between external input events, and strictly decreasing whenever an agent processes a signal. Formally: $E(s_{t+1}) \leq E(s_t)$ for all t, with strict inequality when any agent acts. No agent can process signals indefinitely. This prevents monopoly (one agent capturing all signals) and guarantees that the system must continuously earn its computational expenditure.

(WF4) Contraction. The Q-dynamics operator $\Phi_{\mathcal{A}}$ (Definition 5.2) is a contraction in the Wasserstein-1 metric:

$$W_1\big(\Phi_{\mathcal{A}}(Q),\, \Phi_{\mathcal{A}}(Q')\big) \leq \lambda\, W_1(Q, Q')$$

for some $\lambda \in [0, 1)$ and all distributions Q, Q' in the feasible set.

The first three conditions prevent pathological behavior: unscored outputs, frozen topologies, unbounded computation. The fourth guarantees *convergence*—that the system's distribution of acceptable outputs contracts toward a unique fixed point rather than drifting indefinitely.

Remark 4.3. Unlike WF1–WF3, which can be verified by inspection of the constraint architecture, WF4 requires either analytical derivation of λ from the architecture's properties or empirical estimation. Section 5 provides the analytical derivation for the Gaussian case. Section 7 reports empirical confirmation for the full system.

5 Q-Dynamics as Convergence Mechanism

WF1–WF3 tell us what an Emergence system *cannot* do. They do not tell us what it *will* do. The missing piece is a mechanism that explains *how* the system converges to a useful operating regime. Q-dynamics provides that mechanism.

Definition 5.1 (Acceptability Distribution)**.** For a constraint architecture $\mathcal{A}$ operating on state space S, the acceptability distribution Q_t is the empirical distribution of outputs that score above the system's minimum acceptance threshold at time t:

$$Q_t = \frac{1}{|O_t|} \sum_{o \in O_t} \delta_o$$

where $O_t = \{o : \kappa(o) \geq \theta_{\min} \text{ for all active scoring constraints } \kappa\}$ and δ_o is the point mass at o.

The acceptability distribution captures what the system currently treats as good output. It is not a parameter set by the designer; it is an emergent property of the system's own operation. The question is whether Q_t converges or drifts.

Definition 5.2 (Q-Dynamics)**.** The evolution of Q is governed by:

$$Q_{t+1} = \Phi_{\mathcal{A}}(Q_t)$$

where $\Phi_{\mathcal{A}}$ is the operator induced by constraint architecture $\mathcal{A}$: given the current acceptability distribution, $\Phi_{\mathcal{A}}$ produces the next-period distribution through two steps:

1. **Scoring-selection:** reweight Q_t by the scoring function κ, renormalize. Outputs that score well receive more mass; outputs that score poorly lose mass.
2. **Lifecycle injection:** fork events, external signal arrival, and agent non-determinism inject variance, preventing collapse to a degenerate point mass.

5.1 The Gaussian Contraction Result

For the analytically tractable case where Q_t is modeled as Gaussian and the scoring function is Gaussian with precision $1/\sigma_\kappa^2$ centered at target μ_*, the two steps compose into a map $T\colon (\delta, \sigma^2) \mapsto (\delta', \sigma'^2)$ where $\delta = \mu - \mu_*$:

$$\delta' = \delta \cdot r(\sigma^2), \qquad \text{where } r(\sigma^2) = \frac{\sigma_\kappa^2}{\sigma^2 + \sigma_\kappa^2}, \tag{1}$$

$$\sigma'^2 = h(\sigma^2), \qquad \text{where } h(\sigma^2) = \frac{\sigma^2\,\sigma_\kappa^2}{\sigma^2 + \sigma_\kappa^2} + \sigma_L^2. \tag{2}$$

Here $\sigma_L^2 > 0$ is the lifecycle variance injection, bounded by dissipation (WF3).

Theorem 5.3 (Gaussian Contraction)**.** *For the Q-dynamics map T with Gaussian scoring (variance $\sigma_\kappa^2 > 0$) and lifecycle variance injection ($\sigma_L^2 > 0$):*

(i) *The variance map h is a global contraction: $h'(\sigma^2) = r(\sigma^2)^2 < 1$ for all $\sigma^2 > 0$.*

(ii) *T has a unique fixed point ($\delta_* = 0,\ \sigma_*^2$) where:*

$$\sigma_*^2 = \frac{\sigma_L^2 + \sqrt{(\sigma_L^2)^2 + 4\,\sigma_L^2\,\sigma_\kappa^2}}{2}.$$

(iii) *For any initial condition, $(\delta_t, \sigma_t^2) \to (0, \sigma_*^2)$ at geometric rate:*

$$\lambda_* = \frac{\sigma_\kappa^2}{\sigma_*^2 + \sigma_\kappa^2} = \frac{2\,\sigma_\kappa^2}{\sigma_L^2 + \sqrt{(\sigma_L^2)^2 + 4\,\sigma_L^2\,\sigma_\kappa^2} + 2\,\sigma_\kappa^2}.$$

(iv) *$\lambda_* < 1$ whenever $\sigma_\kappa^2 > 0$ and $\sigma_L^2 > 0$.*

Proof. The variance map h maps $(0, \infty)$ into $[\sigma_L^2, \sigma_\kappa^2 + \sigma_L^2)$ and satisfies $h'(\sigma^2) = (\sigma_\kappa^2)^2/(\sigma^2 + \sigma_\kappa^2)^2 < 1$. By Banach's fixed-point theorem [Banach, 1922], h has a unique fixed point and convergence is global. The fixed-point equation $h(\sigma_*^2) = \sigma_*^2$ yields the stated quadratic; the unique positive root is as given. The mean deviation satisfies $|\delta_t| = |\delta_0| \prod_{k=0}^{t-1} r(\sigma_k^2)$, and since $r(\sigma_k^2) < 1$ uniformly for $\sigma_k^2 \geq \sigma_L^2$, the product vanishes geometrically. The asymptotic rate is $\lambda_* = r(\sigma_*^2)$. Since $\sigma_*^2 > 0$, we have $\lambda_* < 1$. □

The contraction parameter λ_* is not assumed—it is *computed* from the scoring function's precision and the lifecycle's variance injection. The system converges because the scoring function's curvature pulls the distribution inward faster than the lifecycle's variance injection pushes it outward. The balance between these two forces determines the fixed point.

Table 1: Effect of parameter changes on the contraction rate λ_*.

Parameter change	Effect on λ_*	Interpretation
$\sigma_\kappa^2 \to 0$ (sharper scoring)	$\lambda_* \to 0$	Stronger selection = faster convergence
$\sigma_\kappa^2 \to \infty$ (flatter scoring)	$\lambda_* \to 1$	Weaker selection = slower convergence
$\sigma_L^2 \to 0$ (less lifecycle noise)	$\lambda_* \to 1$	Less variance injection = fixed point degeneracy
$\sigma_L^2 \to \infty$ (more lifecycle noise)	$\lambda_* \to 0$	More noise = faster contraction, but Q^* has high variance

5.2 General Convergence

Theorem 5.4 (General Convergence)**.** *Let $\mathcal{A}$ be a constraint architecture satisfying WF1–WF4 with contraction parameter $\lambda < 1$. Then there exists a unique fixed-point distribution Q^* such that $\Phi_{\mathcal{A}}(Q^*) = Q^*$, and for any initial Q_0:*

$$W_1(Q_t, Q^*) \leq \lambda^t \, W_1(Q_0, Q^*).$$

Convergence is geometric at rate λ.

Proof. Banach fixed-point theorem on $(\mathcal{P}(\text{Output}), W_1)$, which is complete for distributions with finite first moment over a Polish space [Villani, 2009]. □

The Gaussian case proves that WF4 *can* be derived from the architecture's properties. The general theorem shows that whenever WF4 holds—whether derived analytically or confirmed empirically—convergence follows. Together, they establish Q-dynamics as the convergence mechanism of the Emergence paradigm.

6 The Control Problem and Contraction

The most serious objection to the Emergence paradigm is the control problem: if behavior is emergent, how do you ensure the system does what you need? If agents are autonomous and trajectories are non-deterministic, what prevents the system from drifting into dysfunction?

The Q-dynamics contraction result provides the answer directly. The system converges because *the constraint architecture is a contraction mapping on the space of possible behaviors.* This is not a metaphor. It is a mathematical fact with a computed rate.

Consider the objection in its strongest form: "Emergent behavior is unpredictable, therefore uncontrollable." The contraction theorem decomposes this into two claims, one true and one false:

1. *Trajectories are unpredictable.* True. The same constraints, agents, and signals will not produce identical trajectories. The specific path through the constraint space varies because agents exercise genuine judgment.
2. *The operating regime is unpredictable.* False. The Q-dynamics operator contracts toward a unique fixed point Q^*. Different trajectories converge to the same distribution of acceptable outputs. The specific outputs vary; the statistical character of the outputs does not.

The control mechanism is *the constraint architecture itself.* Schemas ensure structural validity. Scoring functions calibrate judgments. Lifecycle rules ensure that consistently wrong agents decay. And the contraction theorem guarantees that these mechanisms, composed, produce convergence at a computable rate.

This reframes what "control" means in an Emergence system. The designer does not control trajectories; the designer controls the fixed point. The scoring function's center μ_* determines *where* the system converges. The scoring function's precision σ_κ^2 determines *how tightly* it converges. The lifecycle variance σ_L^2 determines how much exploration the system maintains around the fixed point. These are the control knobs. They are few, interpretable, and their effects are analytically characterized (Table 1).

The paradigm does not dodge the control problem. It dissolves it by showing that convergence is a structural property of well-formed constraint architectures, not an additional requirement imposed from outside. The contraction *is* the control.

6.1 Compositional Guarantee

A system built from well-formed components must itself be well-formed, or the paradigm does not scale. The compositional result provides this guarantee.

Theorem 6.1 (Compositional Well-Formedness). *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be well-formed constraint architectures satisfying WF1–WF4 with contraction parameters λ_1, λ_2. If the constraint sets are type-compatible (no circular dependencies) and the energy functions are additively separable, then the composed architecture $\mathcal{A}_{12}$ satisfies WF1–WF4 with contraction parameter $\lambda_{12} = \max(\lambda_1, \lambda_2)$.*

Proof. WF1: scoring constraints cover the union of agents. WF2: additive separability ensures cross-architecture constraints do not block lifecycle triggers. WF3: the sum of non-increasing, strictly decreasing energy functions is non-increasing and strictly decreasing. WF4: for sequential composition, $\lambda_2\lambda_1 \leq \max(\lambda_1, \lambda_2)$; for parallel composition on disjoint output spaces, W_1 decomposes and the joint rate is the maximum of the component rates. □

Well-formedness composes. This means Emergence systems can be built modularly: each subsystem is verified independently, and composition preserves all four guarantees.

7 Existence Proofs

7.1 The Kay Recovery: Stigmergic Mesh

Theory without implementation is speculation. We have built a working system that instantiates the Emergence paradigm: a stigmergic mesh for ambient structure discovery in organizations [McEntire, 2026a].

The system ingests work artifacts—pull requests, issue tickets, deployment logs, chat messages—as signals. No one formulates a query. No one specifies what to look for. The system processes the ambient exhaust of organizational work and surfaces structural patterns that no one asked about.

Architecture. The mesh consists of worker agents, each bound to a context representing a region of specialization. Signals arrive and are routed via breadth-first search through the worker topology. The first worker whose familiarity score exceeds a vigilance threshold accepts the signal; the remaining workers never see it. Accepted signals update the worker's context through one of three learning modes: full storage (indexed), compressed summary, or lossy weight shift. Workers that accept too many signals fork. Workers that accept too few decay. Redundant workers merge. The topology self-organizes.

The rigid boundaries are strict: immutable typed signals, assessment schemas with required confidence scores, a five-component familiarity function, exponential energy decay with activity-based recharge. The flexible interiors are genuine: when an agent assesses a pull request, it brings the full weight of its contextual understanding to bear on what the change means, what risks it carries, and what patterns it participates in.

What emerged. During live deployment against a real engineering organization (dozens of repositories, multiple chat channels, several project tracking teams), the system produced detections that no individual component was designed to generate:

- A policy intervention flagged a pull request that modified security-sensitive code buried inside a large feature branch—not because of a rule about filenames, but because the agent recognized that a sensitive surface was being modified incidentally, inside a change scoped as something else.
- A churn detector surfaced that a repository had exceeded its weekly diff budget, flagging not the volume itself but the *pattern*: a single large PR pushing cumulative churn past a threshold, suggesting scope creep.
- Workers self-organized into specializations that no one configured. Over successive runs, individual workers developed affinities for infrastructure concerns, product-facing behavior, and cross-team coordination.
- Cross-source correlations emerged. The same organizational pattern—a team under sustained pressure—manifested differently across version control, project tracking, and chat. No individual source showed the pattern clearly. The convergence across independently specialized agents surfaced it.

Well-formedness verification. The mesh satisfies all four well-formedness conditions:

- **WF1 (Completeness):** All assessments scored by the familiarity function. *Verified.*
- **WF2 (Liveness):** Fork (queue overflow), decay (energy depletion), merge (specialization similarity). *Verified.*
- **WF3 (Dissipation):** Energy = budget − costs. Monotonically decreasing, bounded below. *Verified.*
- **WF4 (Contraction):** Empirically confirmed—see Section 7.3.

The system operates with over one thousand passing tests. A full ingestion cycle costs under one dollar and completes in minutes.

7.2 Tessera: Self-Validating Executable Documents

Tessera [McEntire, 2026c] is a self-validating executable document format that provides a second, structurally distinct existence proof of the Emergence paradigm. Where the stigmergic mesh instantiates Emergence in a multi-agent coordination system, Tessera instantiates it in a single-document runtime.

Tessera documents are chains of cryptographically signed sections (Ed25519 signatures, SHA-256 hash chains) with embedded Rhai scripts that execute within the document's own constraint environment. The mapping to Emergence is direct:

Constraints = Document schema. The hash chain and cryptographic signatures enforce structural integrity. Each section's content must validate against its declared schema. The constraints are rigid: a section that fails validation breaks the chain, and the document rejects it.

Runtime = Embedded engine. The Rhai scripting engine executes within the constraints defined by the document itself. Scripts can read document state, perform computations, and produce outputs—but only within the boundaries the document's schema permits. The runtime is flexible: different scripts produce different behaviors within the same constraint envelope.

Game state = Emergence. The document's state evolves as sections are added, scripts execute, and the hash chain extends. The "behavior" of a Tessera document—what it computes, what it validates, what it produces—is not specified in advance. It emerges from the interaction of the schema constraints, the runtime engine, and the accumulated document state.

Tessera demonstrates that the Emergence pattern is not specific to multi-agent LLM systems. The same architecture—rigid constraints at the boundary, flexible computation in the interior, behavior emerging from their interaction—appears in a deterministic, cryptographically verifiable document format. The constraints are different (hash chains instead of scoring functions, schemas instead of vigilance thresholds), but the structural pattern is identical: the programmer writes the physics, and the behavior arises within it.

7.3 Empirical Studies

Five empirical studies, reported in full in the companion design calculus paper [McEntire, 2026b], test whether the Q-dynamics contraction captures the essential behavior of the stigmergic mesh. We summarize the key results.

Study 1: Global contraction ($\lambda < 1$). Twelve trials with different initial conditions (varying worker count and vocabulary warmth) processed 5,975 production signals. W_1 between runs decreased 91.6% ($0.182 \to 0.015$). Contraction exhibits two phases: Phase 1 ($\lambda \approx 0.94$) washes out initial structural differences; Phase 2 ($\lambda \approx 0.50$) is structural Q-convergence as worker-level distributions align. **Result: $\lambda < 1$ confirmed.**

Study 2: Predicted vs. measured λ. The Gaussian model predicts $\lambda_* = 0.79$. The mesh measured $\lambda \approx 0.50$ in Phase 2—substantially faster. The systematic overestimation confirms that the vigilance threshold's truncation of distribution tails strengthens contraction beyond the Gaussian prediction. The Gaussian λ_* is an upper bound on the actual contraction rate.

Study 3: Selection balance and Cage verification. Six settings of the discovery/retrieval threshold (controlling effective selection balance β) show vocabulary entropy decreasing monotonically from 12.33 to 10.48 ($r = -0.933$) as selection sharpens. At the most restrictive setting, the mesh collapses to 7 workers with near-instant convergence—the system locks in. The Cage prediction is confirmed: as $\beta \to 0$, the acceptability distribution collapses toward the system's own prior.

Study 4: Scoring curvature controls convergence rate. Five settings of the vigilance ceiling (controlling effective σ_κ^2) show convergence ratio ranging from 0.27 (sharpest scoring) to 0.51 (flattest scoring). Sharper scoring produces 2× tighter convergence, confirming the theorem's prediction that λ_* decreases with scoring precision.

Study 5: Fork injects variance. Twenty-three fork events in 5,977 signals increase between-component variance by 5.5% on average, confirming fork as the σ_L^2 injection mechanism. The 23:1 fork-to-decay ratio indicates the system operates in a growth-dominated regime where variance injection substantially exceeds variance removal.

8 Multi-Agent Systems and the Emergence Distinction

Multi-agent systems have been studied for decades. The canonical references—Wooldridge [2009] and Shoham and Leyton-Brown [2008]—establish the theoretical foundations: communication protocols, coordination mechanisms, game-theoretic equilibria, organizational structures. The question is what Emergence adds.

8.1 The Agent Coordination Problem

The central problem in multi-agent systems is coordination: how do autonomous agents with potentially divergent objectives produce coherent collective behavior? The standard approaches are:

1. **Centralized control:** a coordinator agent assigns tasks and resolves conflicts. Effective but fragile—the coordinator is a single point of failure and a bottleneck.
2. **Negotiation protocols:** agents bargain, bid, or vote. Flexible but expensive—the communication overhead grows with agent count.
3. **Norms and conventions:** agents follow shared rules that reduce the coordination space. Scalable but rigid—the rules must be specified in advance.
4. **Stigmergy:** agents coordinate through a shared environment rather than direct communication. Scalable and flexible, but traditionally limited to simple agents with simple signals.

Emergence takes the fourth approach—stigmergy—and extends it with agents capable of genuine interpretation. The constraint architecture replaces the negotiation protocol, the norm system, *and* the centralized coordinator. The scoring functions are the norms. The lifecycle rules are the coordinator. The schemas are the communication protocol. But unlike their traditional counterparts, these mechanisms do not require the designer to specify the coordination strategy in advance. The strategy emerges from agents interpreting signals within the constraint space.

8.2 Modern Agent Frameworks

The recent proliferation of LLM-based agent frameworks—AutoGen [Wu et al., 2023], CrewAI [CrewAI, 2024], LangGraph [LangChain, 2024]—demonstrates the engineering community's recognition that LLM agents need coordination. But these frameworks overwhelmingly implement *orchestration*, not emergence:

- AutoGen defines "conversation patterns" that prescribe agent interaction sequences.
- CrewAI assigns "roles" and "tasks" to agents in advance.
- LangGraph defines computation as a directed graph with explicit edges between nodes.

In each case, the designer specifies *what happens.* The agents execute within a script. The behavior is predictable because it is prescribed. This is procedural programming with LLM nodes—a useful engineering pattern, but not a new paradigm.

Emergence differs structurally: the designer specifies *what cannot happen* (constraints), and the behavior arises from agents operating within those constraints. The test is simple: if you can read the source and predict the output, it is orchestration. If the output surprises even the designer—while remaining within the constraint envelope—it is emergence.

8.3 Subsumption

Emergence subsumes the agent coordination problem rather than solving it in the traditional sense. Traditional coordination asks: "Given these agents with these capabilities, what protocol produces the desired collective behavior?" Emergence asks: "Given these constraints, what collective behavior arises from agents with contextual understanding?"

The inversion is the contribution. Traditional multi-agent systems design coordination to produce behavior. Emergence designs constraints and lets coordination emerge. The well-formedness conditions (WF1–WF4) guarantee that the emergent coordination converges, which is what makes this inversion engineering rather than hope.

9 Theoretical Convergence: The Chronicler Principle

The Emergence paradigm converges with results from four independent traditions, each of which discovered the same structural truth about where rigidity and flexibility must live.

9.1 Ashby's Law of Requisite Variety

Ashby's Law [Ashby, 1956] states that a regulator must have at least as much variety as the system it regulates. In traditional programming, the channel capacity is determined by the programmer's ability to enumerate states. Every `if` branch, every `case` statement is an explicit expansion of the channel.

Emergence achieves requisite variety through a different channel. The agents' contextual understanding *already encompasses* the variety of the problem space. The constraint architecture does not need to enumerate states; it needs to *bound the agents that are already navigating them.*

9.2 Beer's Viable System Model

Beer's Viable System Model [Beer, 1972] identifies the 3–4 homeostat as the central tension in any viable system: System 3 (internal stability) must be dynamically balanced against System 4 (environmental intelligence). Beer's specific contribution is that *System 4 must be a genuine model of the environment, not a summary of internal reports.*

This is precisely the failure mode that Emergence avoids. The agents in the flexible interior are not summarizing internal state; they are interpreting environmental signals with contextual understanding independent of the system's prior beliefs.

9.3 Baldwin and Clark's Design Rules

Baldwin and Clark [Baldwin and Clark, 2000] demonstrated that modular systems outperform integral systems when design rules—the interfaces between modules—are well-specified. In Emergence, agents are modules and constraints are design rules. The option value is not financial but epistemic: the system discovers what specializations are useful rather than requiring the programmer to predict them.

9.4 Kauffman's Edge of Chaos

Kauffman's work on Boolean networks [Kauffman, 1993] identified the regime between order and chaos where adaptive computation occurs. The Emergence constraint architecture controls both connectivity (topological rules) and bias (thresholds). The lifecycle rules continuously adjust these parameters, maintaining the system at the edge without requiring the programmer to calculate the critical regime directly.

9.5 The Chronicler Principle

The paradigm is easiest to understand through analogy.

Consider a system designed to write a novel. The traditional approach programs a plot generator: act structure, character arcs, conflict escalation, prose templates. The programmer encodes the intelligence of storytelling into explicit rules.

The Emergence approach builds a *world*: a physics model that enforces conservation laws, an economy model that enforces scarcity, a character model that enforces psychological consistency, a geography model that enforces spatial constraints. Each model is an agent bound to a context, interpreting events within its domain. No model knows about stories. No model has a plot. But when a character model reports that a character's motivation has shifted, and the economy model reports that a resource has become scarce, and the geography model reports that two factions now share a border—a story emerges from the interaction.

A chronicler agent observes the interactions, recognizes narrative structure in the convergence of model outputs, and produces prose. The chronicler does not invent events. It *witnesses* them. The story is not written; it is *discovered.*

This is not metaphor. It is the literal architecture. Replace "physics model" with "schema validator" and "character model" with "domain expert agent" and "chronicler" with "consensus layer," and you have the architecture of any Emergence system. The programmer builds the world. The agents inhabit it. The behavior is what happens when the world's rules interact.

9.6 The Convergence

Four traditions, developed independently over six decades, arrived at the same prescription: rigid constraints at the boundaries, flexible behavior in the interiors, with feedback loops that maintain the balance. Ashby specified the information-theoretic requirement. Beer described the organizational architecture. Baldwin and Clark proved the economic value. Kauffman identified the computational regime. Even Postel's Law—"be conservative in what you send, liberal in what you accept" [Postel, 1980]—is the same insight applied to protocol design. Emergence generalizes this principle from protocols to entire systems.

The convergence across independent formalisms is the strongest evidence that the paradigm captures something real—not a framework imposed on recalcitrant material, but a pattern the material itself exhibits.

10 Limitations and Future Work

The Emergence paradigm has real limitations that must be acknowledged honestly. Some are fundamental; some are contingent on the current state of technology.

10.1 Latency

Every agent assessment is an LLM inference call. Processing a single signal through reflection, evaluation, and assessment takes seconds, not microseconds. This is not a network optimization problem; it is a fundamental property of the computational substrate. LLM inference is slow in the same way that human judgment is slow—the richness of contextual interpretation has a temporal cost.

Emergence systems are not suitable for real-time control, latency-sensitive APIs, or tight feedback loops. They are suitable for problems where the cost of missing a pattern exceeds the cost of detecting it minutes or hours later. The paradigm does not replace traditional computing; it occupies a different niche.

10.2 Cost

Agent assessments cost money. Each LLM call consumes tokens, and tokens have a price. The constraint architecture must therefore include explicit economic boundaries: token budgets, energy decay functions that limit processing, three-tier learning modes that range from full analysis (expensive) to lossy impression (cheap). Cost management is not an operational afterthought; it is a first-class constraint.

10.3 Non-Determinism

Emergence systems do not produce identical outputs from identical inputs. The *constraints* are deterministic; the *trajectories* are not. The scoring functions produce the same scores for the same inputs. What varies is the path the agents take through the space those constraints define.

For many engineering contexts, non-deterministic trajectories are disqualifying. The paradigm does not apply where exact reproducibility is required. It applies where *structural consistency* is sufficient—where the question is not "did the system produce output X?" but "did the system produce an output within the acceptable set?"

10.4 The Space Between: A New Discipline

The hardest limitation is human, not technical. Emergence requires a skill that does not yet have a name, a training pipeline, or a professional identity. It is not programming—the practitioner does not write algorithms or control flow. It is not management—the practitioner does not direct agents through conversation or feedback. It is something in between: closer to designing a game's rules than to playing it, closer to landscape architecture than to construction, closer to creating an ecosystem than to building a machine.

The practitioner must think in terms of selection pressures rather than instructions, in terms of fitness landscapes rather than control flow, in terms of what behaviors a constraint *space* permits rather than what a specific agent will *do*. This is arguably the paradigm's most important implication. Every previous paradigm shift changed what programmers *wrote*. Emergence changes what programmers *are*.

10.5 Gaussian Idealization

The contraction theorem (Theorem 5.3) is proved for Gaussian distributions. The actual stigmergic mesh uses non-Gaussian scoring (a bounded six-component familiarity function). The gap is classified: near the fixed point, the second-order Taylor expansion of $\log(\kappa)$ gives an effective Gaussian with precision $\Lambda_{\text{eff}} = -D^2 \log(\kappa)$, bridging local stability. Global contraction for non-Gaussian architectures is analytically open. Empirically, it holds (Study 1 confirms $\lambda < 1$ for the full system), but the analytical proof requires closing six identified gaps between the Gaussian model and the mesh [McEntire, 2026b].

10.6 Single Domain

Only the stigmergic mesh and Tessera have been verified against the well-formedness conditions. The paradigm's generality claim requires additional existence proofs in structurally different domains.

11 Conclusion

Programming paradigms arrive when a new capability makes a new relationship between programmer and machine possible. Structured programming arrived when hardware supported abstraction. Object-oriented programming arrived when GUIs demanded encapsulation. Functional programming arrived (the second time) when multicore demanded immutability.

Emergence arrives because large language models have created computational units with contextual understanding. For the first time, the receiver of a message can decide what it means. For the first time, the computational primitive can exercise judgment. For the first time, the programmer can specify *what matters* and trust the system to figure out *how to assess it*.

The contribution is not merely architectural. It is formal. Four well-formedness conditions—completeness, liveness, dissipation, and contraction—transform the paradigm from a design philosophy into an engineering discipline with provable properties. The Q-dynamics contraction theorem shows that the system converges to a unique fixed point at a computable rate. The compositionality theorem shows that well-formed subsystems compose into well-formed systems. The control problem is not dodged; it is dissolved—the contraction *is* the control.

This was Kay's vision. Not classes and inheritance. Not design patterns and factories. Autonomous entities interpreting messages, coordinating through shared environments, producing behavior that emerges from their interaction rather than from central specification.

The programmer does not write the program. The programmer writes the conditions under which the program emerges. The constraints are the artifact. The computation is what arises within them.

References

W. Ross Ashby. *An Introduction to Cybernetics.* Chapman & Hall, 1956.

Carliss Y. Baldwin and Kim B. Clark. *Design Rules: The Power of Modularity.* MIT Press, 2000.

Stefan Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fundamenta Mathematicae*, 3(1):133–181, 1922.

Stafford Beer. *Brain of the Firm.* Allen Lane / The Penguin Press, 1972.

CrewAI. CrewAI: Framework for orchestrating role-playing autonomous AI agents. `https://github.com/joaomdmoura/crewAI`, 2024.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1994.

Stuart A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution.* Oxford University Press, 1993.

Alan Kay. Re: Prototypes vs classes was: Re: Sun's hotspot, 2003. Email to the Squeak mailing list, archived at `http://lists.squeakfoundation.org/pipermail/squeak-dev/2003-July/066092.html`.

LangChain. LangGraph: Building language agents as graphs. `https://github.com/langchain-ai/langgraph`, 2024.

Jeremy McEntire. Ambient structure discovery via stigmergic mesh. Working paper, 2026a.

Jeremy McEntire. A design calculus for emergence: Constraint typing, compositional well-formedness, and endogenous acceptability dynamics. Working paper, 2026b.

Jeremy McEntire. Tessera: Self-validating executable documents. https://github.com/jmcentire/tessera, 2026c.

Jon Postel. DoD standard internet protocol. Technical Report RFC 760, USC/Information Sciences Institute, 1980. The robustness principle: "be conservative in what you do, be liberal in what you accept from others".

Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2008.

Cédric Villani. *Optimal Transport: Old and New*. Springer, 2009.

Michael Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2nd edition, 2009.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In *arXiv preprint arXiv:2308.08155*, 2023.