

Convergence Engineering: Commissioning Autonomous Software by Measuring the Rate at Which Novel Failures Trend Toward Zero

Nick Sonnentag · Sunnyday Technologies, Appleton WI · nick@sunn3d.com · ORCID [0009-0002-1897-384X](https://orcid.org/0009-0002-1897-384X)

Date: 2026-04-16 · Version: 1.0 (preprint) · License: CC BY 4.0

Keywords: *convergence engineering · time to silence · commissioning phase · autonomous software systems · LLM coding agents · self-verification · fail-fast · engineering invariants · site reliability engineering · concrete 3D printing · CadQuery*

ABSTRACT

We introduce **Convergence Engineering**, a methodology for commissioning autonomous software systems that treats initial failures as expected calibration data rather than defects. The method has four phases — **Instrument, Commission, Converge, Steady State** — and is governed by a primary metric we call **time to silence**: the elapsed time from first deployment to the first week in which novel failure classes arrive at a team-chosen threshold rate. Reliability is defined not as the absence of failures but as the negative first derivative of unique failure classes with respect to time. A system whose curve is trending negative is becoming reliable; a system whose curve is flat because its failures are silently caught and suppressed is not reliable — it is lying.

We ground the methodology in a public case study: **M3-CRETE**, an open-source motion system for a meter-scale concrete 3D printer (CERN-OHL-W-2.0, github.com/sunnyday-technologies/M3-CRETE), built by a single human engineer using an LLM coding agent as a collaborator. Mining six consecutive build sessions from 2026-04-11 to 2026-04-15 — 1,062 tool-use events, two architectural pivots, an invariant-suite trajectory from 60 through 313 and consolidating at 231 — we recover the convergence curve in Figure 1: peak novel failure classes per session declining from 8 (during the v3 architectural pivot) to under 2 within four calendar days. The measured time to silence is approximately two to three days from the Phase 2 session on 2026-04-11. The sample is small and the kinetics are project-specific; we publish to register the methodology, the metric, and the verification path, not to claim a benchmark.

Convergence Engineering is not a novel algorithm. Its components — fail-fast [1], Andon and *jidoka* [2], SRE error budgets [3], Chaos Engineering [4], Let-It-Crash [7] —

exist in isolation in the literature. Our contribution is the synthesis, the commissioning-phase framing, the time-to-silence metric, and a publicly verifiable demonstration that the combined discipline produces engineering-grade output at one-founder scale.

Contents

1. Introduction: The Fallback Trap
2. Related Work and Intellectual Lineage
3. The Methodology: A Four-Phase Lifecycle
4. Case Study: M3-CRETE
5. The Ethics of Unchecked Autonomous Output
6. Implementation Guide
7. Limitations and Future Work
8. Conclusion

1. Introduction: The Fallback Trap

The natural engineering instinct, when a system fails, is to add a fallback. When the fallback fails, we add another. Over time, systems accumulate layers of degraded-but-still-running behavior until the primary path is no longer distinguishable from any of the alternates. The system still reports `200 OK`, but what it is actually *doing* has drifted away from what its operator asked it to do.

In most software domains this is a quality problem with observability consequences. It becomes a safety problem the moment the system's output acts on the physical world. An LLM agent that falls back from a large model to a small one, then to a cached response, then to a canned string, produces text that nobody reviews in detail because the envelope still looks healthy. If that text is a shopping list, the cost of the fallback chain is a mild irritation. If the text is a mix design for a concrete batch plant or a G-code file destined for a 1.5 kg printhead moving a meter per second, the cost is structural integrity and operator safety.

The factory analog is worth restating. When Toyota commissions a new production line, the engineers expect it to trigger the Andon cord [2]. The Andon is not failure; it is the line surfacing the problems that have to be fixed permanently before the system is allowed to run without oversight. A line that never triggers the Andon on day one is either trivial or suppressing signal.

Suppressed signal is worse than loud failure because the people responsible for the line no longer know what it is doing.

The equivalent in autonomous software is the self-verification harness that fails loudly — not because we enjoy pager fatigue, but because every suppressed alert is either a bug we have not fixed or a lie the system is telling about its own state. **Reliability is not the absence of failures; it is the rate at which novel failure classes converge toward zero.** A system whose derivative is zero because it is new, small, and has not been stressed is different from a system whose derivative is zero because every failure class has been surfaced, root-caused, and fixed. Only the second system is reliable. The two are indistinguishable from the outside unless you instrument for the difference.

1.1 Scope

Convergence Engineering is proposed for autonomous or semi-autonomous systems whose outputs act on the physical world or on customer-visible contracts. It is not an argument against advisory LLM tools where a human operator reviews every output, nor against systems whose failures are purely internal and observable only to the operator. The ethical weight of silent degradation scales with the reversibility of the system's downstream effects; construction, manufacturing, and batch-process chemistry live at the low-reversibility end of that axis. The rest of this paper focuses on that end of the axis.

The contribution is a named methodology that combines fail-fast instrumentation, a commissioning phase modelled on manufacturing line start-up, and a primary metric (time to silence) that is comparable across systems, teams, and domains. The components are not novel. The synthesis, the framing, and the publicly verifiable demonstration are.

2. Related Work and Intellectual Lineage

Convergence Engineering is assembled from pieces of six existing bodies of work, and none of the components would be novel in isolation. Honest lineage:

Fail-Fast. Shore's 2004 *IEEE Software* article [1] established the principle that software should surface failures at the earliest possible boundary. Fail-fast systems do not try to muddle through an invalid state; they crash. Convergence Engineering treats fail-fast as a precondition, not a conclusion — you cannot measure convergence in a system that silently continues past errors.

Toyota Production System and Jidoka. Ohno's 1988 formalization of the Andon cord and *jidoka* (automation with a human touch) [2] is the direct manufacturing analog to what we propose for software. A line that stops itself at the first detected defect is what a self-verifying autonomous

system should look like. Our *Commission* phase is the software equivalent of the production-line ramp-up, where failures are expected, logged, and converted into permanent fixes.

Google SRE Error Budgets. Beyer et al. [3] introduced the idea that reliability is a measurable quantity with an explicit budget, and that decisions about feature velocity should be bounded by that budget. Error budgets are the closest existing quantitative frame to what we propose. We differ on two points: error budgets are consumed by *all* failures, while we care specifically about *novel* failure classes; and error budgets are a steady-state instrument, while our time-to-silence metric is a commissioning-phase instrument designed to measure the transient from deploy to operational maturity.

Chaos Engineering. Basiri et al. [4] and the Netflix SRE discipline of injecting faults to discover weaknesses is complementary, not competitive. Chaos Engineering probes hypothetical failure paths; Convergence Engineering consumes the real failures that commissioning inevitably produces. A mature team runs both: chaos for the unknown unknowns, convergence for the known-but-unfixed. Our methodology is cheaper per unit of reliability early in a system's life, because the real failures are already there and free — you just have to instrument for them.

Circuit Breaker Pattern. Nygard [6] describes the circuit breaker as a necessary defense against cascading failures from external dependencies. We take no issue with the pattern and consider it orthogonal. A circuit breaker protects the caller from a transient dependency failure; it is not a substitute for fixing the reason the dependency fails.

Erlang/OTP "Let It Crash". Armstrong's 2003 thesis [7] proposes that fault tolerance is best achieved by letting processes crash and restart cleanly, isolating failures at the process boundary. Convergence Engineering borrows the philosophical stance — do not paper over failures — and extends it with the measurement discipline needed to know whether the crashes are declining over time or merely being counted.

Antifragility. Taleb's 2012 work [5] supplies the philosophical vocabulary for systems that gain from disorder. We agree with the framing and note that it is not a methodology. Convergence Engineering is our attempt to operationalize something antifragility-adjacent: a system that becomes more reliable through the very process of being forced to fail.

Positioning. Convergence Engineering does not replace any of the above. It names a specific synthesis — fail-fast instrumentation plus a Toyota-style commissioning phase plus SRE-style quantitative tracking — and introduces a single primary metric (time to silence) that is comparable across implementations. The novelty claim is narrow: the framing, the metric, and a reproducible case study at one-founder scale.

3. The Methodology: A Four-Phase Lifecycle

We describe Convergence Engineering as four distinct phases. The phase boundaries are definitional, not procedural — a team moves between them based on the shape of its failure data, not on a calendar. Architectural pivots cause a team to re-enter Phase 2 *locally* for the scope of the pivot, even while the system as a whole is globally in Phase 3 or 4; a system can be in two phases simultaneously at two different scopes. This is the expected mode of a long-lived project, not an exception.

3.1 Phase 1 — Instrument

Every boundary in the system gets a fail-fast assertion before the system is allowed to do any work. Boundaries include: module entry, I/O operations, inter-service calls, data schema transitions, and — critically for autonomous systems — every point at which the autonomous component's output is committed to an external system. The instrumentation captures structured events: source, type, message, fingerprint for dedup, severity, occurrence count, first-seen and last-seen timestamps.

The defining rule of Phase 1 is that **there are no silent catches**. An exception handler that logs a warning and continues is acceptable only if the logging is structured and the subsequent behavior is documented as an accepted degradation. An exception handler that swallows the failure and returns a default value is always a bug during the commissioning phase, even if it will be the right answer in steady state.

In M3-CRETE, Phase 1 is the engineering-invariant suite in `CAD/preview_assembly.py` (231 invariants at the time of this writing) plus the category-level gates in `CAD/self_check.py` (five independent checks covering inventory, joint consistency, corner-connector thickness, motor-bracket proximity, and plate-vs-C-beam interference). Every invariant is a falsifiable claim: a dimension is 3 mm; a wheel engages a rail by 1.5 to 2.5 mm; the motor sits within 50 mm of its bracket. When an invariant fails, the failure message tells the agent exactly what is wrong.

3.2 Phase 2 — Commission

Deploy and expect noise. The system *will* alert constantly. This is healthy. During Phase 2, three things are tracked and nothing else matters as much:

1. **Unique failure classes per session or per day.** A single fingerprint counts once per day no matter how many times it occurs.
2. **Alert-to-fix latency.** The time between an alert's first occurrence and a code change that makes it no longer reproducible.

3. **Root-cause fix ratio.** The fraction of alerts resolved by changing code upstream versus the fraction closed by adding a fallback.

The defining rule of Phase 2: **no fallbacks added during the phase**. Every alert is resolved either by a root-cause fix or by an explicit, logged decision to accept the degradation, with the accepted degradation becoming part of the system's documented contract. Fallbacks added during Phase 2 defeat the entire methodology because they reset the convergence clock and hide the failure class from future tracking.

In M3-CRETE, Phase 2 is visible in the session logs from April 11 through April 13 (sessions `746b75e6` and `344260c2`). The Phase C session alone shows the invariant suite growing from 60 assertions to 313 assertions in a single 216-tool-call session, with two mid-session failure episodes — a 2-of-210 and a 5-of-294 — each resolved within the session itself. The sessions ran loud. The Andon was triggered repeatedly. Every alert was a root-cause fix or an invariant refinement.

3.3 Phase 3 — Converge

Error classes decline. The convergence curve — unique failure classes per session, plotted over time — has a negative first derivative. Resolution patterns accumulate: specific, named fixes that prevent entire classes of recurrence.

In M3-CRETE the learned patterns include:

- **Probe-then-place.** Never trust a STEP file's orientation. Before placing any component, run `probe_steps.py` to measure its bounding box and determine which axis is long, short, or grooved. This pattern retired the orientation-error class wholesale.
- **Sorted-dimension invariants.** Check `sorted([dx, dy, dz])` rather than axis-specific dimensions. A motor is a motor no matter which way it is pointing. This pattern retired the "user moved the motor, every axis-specific check broke" class.
- **The `RELOCATED` set.** Explicitly mark components whose position has been moved outside the nominal envelope by human judgment. Relocated components skip position-range checks but retain dimensional and relationship checks. This pattern retired the "relax the check" anti-pattern and replaced it with surgical opt-outs whose scope is auditable.
- **Shape-signature classification.** In the `v3` assembly script, every loaded STEP solid is classified by its sorted bounding box signature against a `SIG_COLORS` table. This pattern retired the "Fusion renamed the part, our match broke" class.

A team is in Phase 3 when the failure-class derivative is reliably negative and the resolution-pattern library is growing faster than the novel-class list. In M3-CRETE this was April 12–14.

3.4 Phase 4 — Steady State

New error classes per week fall below a team-chosen threshold. The threshold is domain-dependent; for M3-CRETE we use two, chosen empirically because the observed Phase 4 baseline was zero to one novel class per week and we wanted detection headroom rather than a celebratory threshold. Teams in higher-throughput domains may prefer a rate-normalized threshold (e.g., two per thousand autonomous operations).

Alerts are now rare and meaningful. Each new alert is either a genuine novel failure, which triggers a fix and potentially a new invariant, or a regression, which indicates the invariant suite was insufficient and needs strengthening.

The **time-to-silence** metric is the elapsed time from first Phase 2 deployment to the first week in Phase 4. It is comparable across systems because it measures the *rate* at which a team can surface and fix failure classes, not the absolute count. A simple system with fast iteration reaches silence faster than a complex system with slow iteration, but both systems can be characterized on the same axis.

In M3-CRETE, the Phase 4 signal appears in the post-2026-04-13 sessions. The `698d76b7` and `27d97cdf` sessions (103 combined tool calls) show no mid-session invariant failures — the harness runs green from start to finish. The subsequent sessions (`bd4a112a` at 193 tool calls, `3fd7d82e` at 81 tool calls) include a mid-pivot architectural sweep (2080/2040 extrusions replaced wholesale with 4080 C-beam at a single SKU), and the failure mode shifts: the granular per-part invariants stay green, while the category-level `self_check.py` surfaces four failures across sixteen runs, each resolved within the session. Convergence at a coarser granularity is still convergence, and the mixed phase state (Phase 4 at the per-part scope, Phase 2 locally at the architectural-sweep scope) is exactly the pattern §3's opening paragraph predicts.

4. Case Study: M3-CRETE

4.1 System Under Test

M3-CRETE is a cartesian motion system for a 1 m³-class concrete 3D printer, built from 40 x 80 aluminum C-beam extrusion (single SKU across the frame, with short 2040 reinforcement sections inside the channel at the X-axis splice), GT2 belts on polycarbonate V-wheels, and seven NEMA23 stepper motors driven by a Bigtreetech Kraken controller running Klipper. The CAD source is on GitHub at github.com/sunnyday-technologies/M3-CRETE under CERN-OHL-W-2.0. The bill of materials is public at m3-crete.com/bom.

The assembly is built with CadQuery 2.7 and verified by a two-layer harness: `preview_assembly.py` (231 per-part invariants plus three orthographic PNG renders) and

`self_check.py` (five category-level gates including OCC solid-boolean interference checks on plate-vs-C-beam pairs). The detailed harness architecture is described in [10] and not repeated here; this section focuses on the convergence data the harness produces.

Measured time to silence. By our own Phase 4 definition (fewer than two novel failure classes per week), M3-CRETE reached Phase 4 between 2026-04-12 and 2026-04-13 — approximately two to three days from the Phase 2 session on 2026-04-11. The small sample should be read accordingly: this is one project's kinetics, not a benchmark. A system with different complexity, team size, or iteration cadence will produce different numbers. Publishing the metric lets future case studies land on a comparable axis.

4.2 Session-Level Convergence Data

Table 1 summarizes six consecutive M3-CRETE build sessions from 2026-04-11 to 2026-04-15, mined from archived LLM coding-agent session transcripts. *Invariants trajectory* shows the low-to-high range of invariant-suite size observed across that session's harness runs. *Failure episodes* shows distinct `N/M passed` snapshots where $N < M$ — moments during the session when one or more invariants failed and triggered a repair. *Tool calls* is the count of `tool_use` events in the session transcript.

Table 1. M3-CRETE convergence data, April 11–15, 2026.

Date	Session (hash prefix)	Tool Calls	Invariants Trajectory	Failure Episodes
04-11	Phase C (746b75e6)	216	60 → 125 → 131 → 208 → 214 → 251 → 289 → 290 → 313	208/210 (2 fails), 289/294 (5 fails)
04-12	v3 pivot (344260c2)	469	229 → 231 → 284 (then pruned to 231)	221/229 (8 fails), 227/231 (4 fails)
04-12	Hackaday (https://hackaday.io/project/205625-m3-crete) (698d76b7)	34	231/231	none
04-13	Whitepaper v0.1 (27d97cdf)	69	231/231	none
04-14	4080 pivot (bd4a112a)	193	231/231 + 5-gate self_check	4 self_check FAIL / 12 PASS
04-15	Joining plates (3fd7d82e)	81	231/231 + 5-gate self_check	2 self_check FAIL / 2 PASS
Total		1,062	60 → 231 + 5-gate	

Figure 1 plots the peak novel failure classes per session (top panel) against the invariant-suite size at session end (bottom panel). The curve is not monotonic: session 2 (the v3 pivot) shows the highest peak failure count (8), and session 5 (the 4080 pivot) shows a secondary peak. This is the expected shape. An architectural pivot is a deliberate Phase-2 re-entry for the scope of the change; the curve is still trending negative if the duration-to-recovery stays short, and in both cases the pivot session closed with its respective harness green.

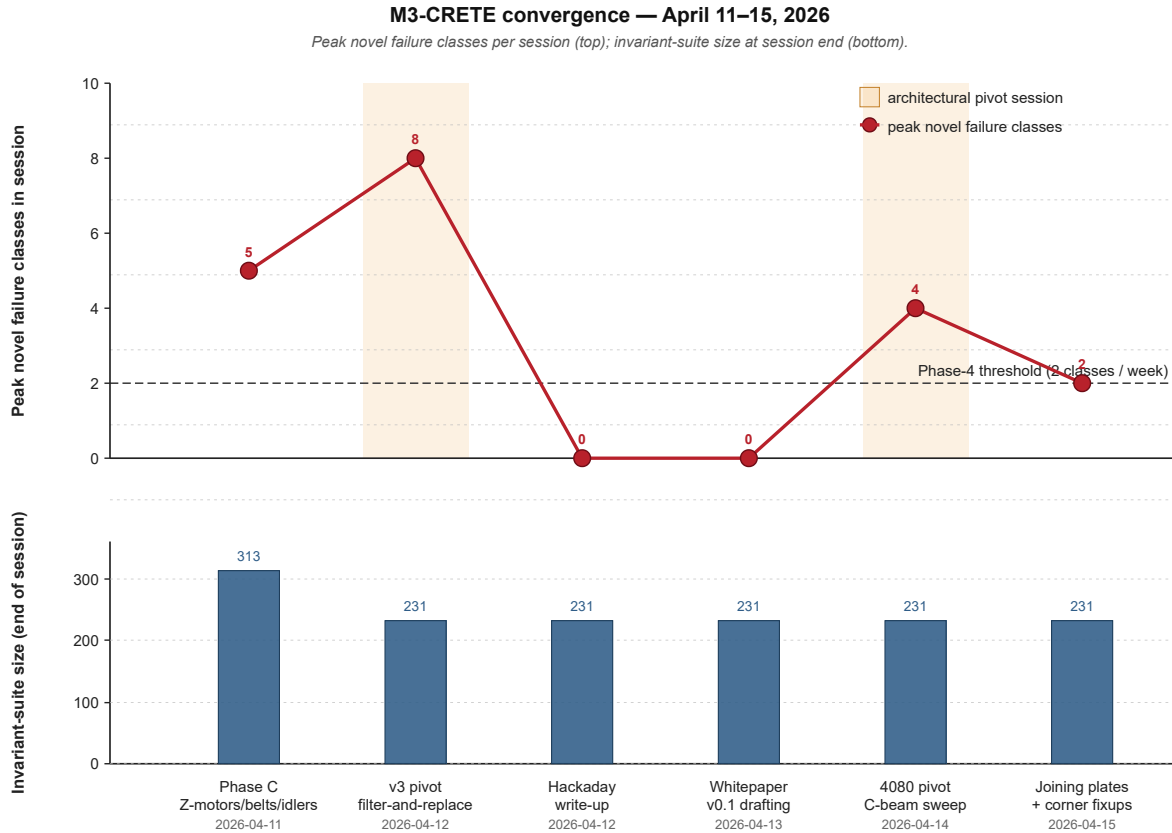


Figure 1. M3-CRETE convergence curve across six April 11–15, 2026 sessions. Top panel: peak novel failure classes per session, with architectural-pivot sessions shaded; the Phase-4 threshold (two classes per week) is plotted as a dashed reference line. Bottom panel: invariant-suite size at session end — a companion density curve showing that instrumentation breadth tracks project scope. Data source: `preview_assembly.py` and `self_check.py` snapshots in the archived agent session transcripts for each listed session.

4.3 What the Harness Caught vs. What the Human Caught

Across the six sessions, the harness caught the following classes of failure before any human visual review:

- **Rotation errors** (≥ 8 instances across Phase C and v3). A part placed 90° off; sorted-dimension signature correct but axis-specific placement wrong.
- **Position drift** (≥ 4 instances). Part center outside expected range after a refactor of the topology constants.
- **Engagement-depth errors** (≥ 3 instances). V-wheel failing to overlap the C-beam post face by 1.5 to 2.5 mm.
- **Belt-span errors** (≥ 2 instances). Belt Z-bottom or Z-top not at the tangent point of its idler or pulley.

- **Missing-part errors** (≥ 2 instances). Belt strand count not equal to eight; bracket count not equal to four.
- **Interference errors** (during 4080 pivot). Plate clipping through C-beam by more than 0.5 mm^3 of boolean-common volume.

The following classes were caught only by human visual review:

- **Sandwich-vs-butt-joint configuration** (1 instance). Geometrically indistinguishable; semantic intent only.
- **C-beam channel direction** (1 instance). The V-groove faced outward when it should have faced inward; the invariant checked dimensions but not channel orientation. Caught by human, fixed by adding a `channel_dir` parameter to the C-beam builder.
- **Aesthetic color assignment** (multiple instances). The harness does not check appearance.

The companion paper [10] reports the catch ratio at approximately 4:1 in favor of the harness across an earlier dataset; we have not re-derived the ratio for the six-session window presented here. The qualitative claim — that roughly 80% of geometric failures in this case study are automatable and the remainder require human semantic judgment — is consistent across both datasets.

4.4 The Architectural-Pivot Signature

Two architectural pivots are visible in the data: the `v3` filter-and-replace pivot on 2026-04-12 and the 4080 C-beam sweep on 2026-04-14. Both pivots have the same signature on the convergence curve: a sharp spike in novel failures localized to the pivot session, followed by convergence within the same session. This is what a healthy local Phase-2 re-entry looks like. A pivot that did not show the spike would indicate invariant suite insufficient; a pivot whose spike failed to converge within the session would indicate scope too large to ship.

The 4080 pivot is particularly instructive because it was driven by a non-technical constraint: shipping pallet sizes and single-SKU procurement. An engineering decision external to the CAD model forced a sweep of every reference to `2080`, `2040`, and `1200mm` in the codebase. The harness caught every stale reference without a human having to list them. The commit `ce03b57`, landed on 2026-04-14, carries the terse message *"fix: sweep all stale 2080/2040/1200mm refs to 4080 C-beam 1000mm"* — a commit message that, read without the convergence framing, looks like ordinary cleanup, and read with the framing, reveals a Phase-2 re-entry that resolved in a single session because the harness itemized exactly what was stale.

5. The Ethics of Unchecked Autonomous Output

A fallback chain that silently succeeds produces artifacts that nobody audited. In a pure software domain, this is an observability failure: the customer thinks they asked the system to do X and the system reports having done X, but the system actually did Y and nobody, including the system's operators, currently knows.

The ethical stakes change when the artifact acts on the physical world. In concrete 3D printing — the domain of M3-CRETE's downstream user — the artifacts include CAD destined for CNC or water-jet fabrication, mix designs destined for a concrete batch plant, toolpaths destined for a machine moving a printhead at one meter per second through space occupied by human operators. An autonomous system that silently falls back to a degraded path has, in this domain, committed a specific and bounded act: it has delivered an artifact different from the one specified without telling anyone. The downstream human operator acts on the artifact believing it is the one they asked for. If the artifact is wrong, the concrete is poured wrong, the wall is crooked, or the printhead collides with the formwork, the attribution chain is broken. Nobody knows which step of the chain produced the error because the silent fallback was, by design, not logged as a deviation.

We propose the following principle: **if an autonomous system's output affects the physical world or a customer's expectations, verification infrastructure is not optional — it is the engineering contribution. The agent is the means; the invariants are the artifact.** This is a specific claim, not a general one. We are not arguing against transient retries, circuit breakers, or graceful degradation in domains where the degradation is logged, disclosed, and bounded. We are arguing that every fallback path must be either a documented degradation (with a consumer-facing disclosure contract) or a root-cause fix still pending.

The regulatory analog exists already. Construction codes require stamped drawings because the downstream consequences are physical and the attribution chain must survive review. A professional engineer who stamps a drawing with unreviewed CAD carries liability. An autonomous system that produces unreviewed CAD absent verification infrastructure is, by this standard, not a tool; it is a liability transfer mechanism — one that moves engineering liability from the system builder to a downstream human operator who was not given the information needed to catch the error. Convergence Engineering is, in part, an argument that the verification harness is the equivalent of the PE stamp — the artifact that makes the autonomous system's output reviewable and therefore shippable.

6. Implementation Guide

This section describes the minimum viable implementation of Convergence Engineering. The schemas and patterns below are generic; a reference implementation for any given team will

differ in vocabulary but not in structure.

6.1 Error-Event Schema

Every structured failure event carries at least the following fields:

```
CREATE TABLE error_events (  
  id          UUID PRIMARY KEY,  
  fingerprint TEXT NOT NULL,      -- dedup key (hash of type + normalized message)  
  source      TEXT NOT NULL,      -- module / service / agent that raised it  
  type        TEXT NOT NULL,      -- coarse category (e.g. "invariant_fail")  
  message     TEXT NOT NULL,      -- raw human-readable message  
  severity    TEXT NOT NULL CHECK (severity IN ('INFO','WARNING','CRITICAL')),  
  first_seen  TIMESTAMPTZ NOT NULL,  
  last_seen   TIMESTAMPTZ NOT NULL,  
  occurrence  INT NOT NULL DEFAULT 1,  
  resolved_at TIMESTAMPTZ,  
  resolved_by TEXT                -- commit SHA or PR reference  
);  
CREATE INDEX idx_error_events_fingerprint ON error_events (fingerprint);  
CREATE INDEX idx_error_events_first_seen  ON error_events (first_seen);
```

The `fingerprint` column is the key to convergence tracking. A fingerprint collapses all occurrences of the same root failure into a single row. *Novel failure classes per week* is a simple count of `fingerprint` values with `first_seen` in the last seven days.

6.2 Resolution-Pattern Schema

When the same fingerprint is resolved by the same fix repeatedly, the fix becomes a pattern worth naming:

```
CREATE TABLE resolution_patterns (  
  id          UUID PRIMARY KEY,  
  name        TEXT NOT NULL UNIQUE, -- e.g. "probe_then_place"  
  matches_type TEXT NOT NULL,      -- error_events.type this pattern applies to  
  fix_action   TEXT NOT NULL,      -- terse procedural description  
  attempts    INT NOT NULL DEFAULT 0,  
  successes    INT NOT NULL DEFAULT 0,  
  created_at   TIMESTAMPTZ NOT NULL,  
  authored_by  TEXT  
);
```

When a new `error_events` row arrives, the agent (or the human on-call) first checks whether an existing pattern matches. A matched pattern is applied directly, bypassing re-diagnosis. The `successes / attempts` ratio tells the team whether a pattern is still load-bearing or has drifted and needs retirement.

6.3 The Convergence Dashboard

One chart. Unique failure fingerprints per day (or per session, for agent-driven systems) plotted against time. This is the single most important operational display during the commissioning phase. If the curve is flat or rising, the team is adding features faster than it is fixing failures; this is a management problem, not a technical one. If the curve is monotonically declining, the team is converging. If the curve shows sharp spikes localized to architectural pivots and returns to baseline within a session or a week, the team is converging and the harness is doing its job.

Overlay a second series: unique fingerprints *resolved* per day. The gap between the two curves is the open-failure backlog. A sustained gap indicates either insufficient fix velocity or a suite of accepted degradations that should be promoted to documented contracts.

6.4 Decision Tree: Fallback vs. Root-Cause Fix

When an alert arrives:

1. **Is the failure transient and external?** (Network blip, third-party API timeout.) Yes → retry with backoff is acceptable. Log the event. Count it against the external-dependency error budget, not the convergence curve.
2. **Is the failure persistent and internal?** Yes → root-cause fix. No fallback. The convergence curve is watching.
3. **Is the failure persistent and external?** Yes → circuit breaker is acceptable; document the degraded behavior as part of the system's *customer-facing* contract, not just a log line. If the circuit-breaker state is visible to downstream consumers (customers, operators), the degradation is disclosed; if not, it is silent, and you are in the fallback trap.
4. **Is the failure a novel geometric, physical, or semantic class the harness does not cover?** Yes → add an invariant, then root-cause fix. The invariant is the permanent record; the fix is the immediate outcome.

6.5 Alert-Fatigue Mitigation

The commissioning phase is noisy by design. Three discipline rules make the noise tolerable:

1. **Fingerprint dedup.** Same fingerprint in the same cooldown window produces one alert. Occurrences roll up silently.
2. **Severity tiers.** `CRITICAL` pages a human; `WARNING` appears in the dashboard; `INFO` lands in a metrics counter. Wrong-tier assignment is itself a bug — miscategorized severity destroys the signal faster than noise does.
3. **Expected end date.** Phase 2 has a documented expected duration. The team knows the noise is temporary. A commissioning phase that has been in Phase 2 for more than six months with no declining derivative is an organizational problem, not a technical one, and should be surfaced to management explicitly.

7. Limitations and Future Work

Sample size. This paper reports on a single primary case study from one organization. The methodology needs validation across additional architectures, team sizes, and domains before its generality can be claimed. We consider this paper a registration-of-claim document; we invite replication.

Retrospective reconstruction. The convergence curve presented here was reconstructed retrospectively from agent transcripts that were archived for other purposes. The methodology section describes the prospective instrumentation a team would deploy from day one; we did not have that dashboard running during the M3-CRETE sessions. Future case studies should report prospective data collected from a purpose-built `error_events` table; the present work demonstrates that even post-hoc reconstruction yields a legible convergence curve, but prospective collection will be higher-fidelity.

Survivorship bias. We are reporting on a system that reached Phase 4. Systems that fail during Phase 2 and get abandoned do not publish case studies. The time-to-silence metric, in particular, may appear more favorable in the literature than it is in the population of attempts, because the unsuccessful attempts are not observable. A disciplined use of the methodology requires reporting time-to-abandonment as well as time-to-silence.

Cultural prerequisite. The commissioning phase requires a team — or a founder, as in the M3-CRETE case — willing to tolerate loud failure output during the early life of a system. Teams that are measured by uptime SLAs during commissioning will be rewarded for suppressing alerts, which defeats the methodology. Convergence Engineering is therefore not a purely technical discipline; it requires a management decision to defer steady-state metrics until the commissioning phase is complete.

Fix velocity. The shape of the convergence curve is dominated by the team's mean time to root-cause fix. A team that takes three days per fix will have a curve with different kinetics than a

team that fixes in three hours. The methodology does not specify fix velocity; it only specifies that the fixes must be root-cause rather than fallback.

Formal characterization of the curve. We have not yet established whether the expected shape of the convergence curve is exponential decay, a power law, or some domain-dependent family. Collecting convergence data from additional case studies is the first-order requirement for making this claim formal.

Second case studies. The methodology was developed across multiple autonomous systems beyond the one reported here. Publication of those systems' convergence data — including the Open3DCP canonical schema project [11], whose field-level validation rules display the same migration-driven convergence pattern in a data-engineering rather than CAD-engineering domain — is future work.

8. Conclusion

Convergence Engineering is a synthesis, not an invention. Its components — fail-fast assertions, manufacturing-style commissioning phases, structured error events, root-cause discipline, and quantitative reliability tracking — have been published in isolation for two decades. The contribution is the combination and the framing: the claim that reliability is a derivative rather than an absolute, the naming of *time to silence* as the primary commissioning metric, and the explicit argument that silent fallbacks in autonomous systems acting on the physical world are not a quality compromise but an ethical one.

The M3-CRETE case study demonstrates that a single founder paired with an LLM coding agent can build and verify an engineering-grade mechanical assembly — public, open-source, under CERN-OHL-W-2.0 — by instrumenting every geometric boundary with falsifiable invariants, running the harness on every build, fixing root causes rather than adding fallbacks, and tracking the convergence curve as the primary signal that the system is becoming commissioned. The convergence data are on GitHub. The invariants are in `CAD/preview_assembly.py` and `CAD/self_check.py`. The session transcripts are archived. Every claim in this paper is verifiable against the repository at github.com/sunnyday-technologies/M3-CRETE.

We do not claim Convergence Engineering is the only way to build reliable autonomous software. We claim it is *a* way — a specific, named, measurable discipline — and that the alternative, fallback-by-default, produces systems that are indistinguishable from reliable ones until the moment they are not.

Silent success *unearned* is the failure mode. Loud commissioning, earned silence, and a negative first derivative on the failure-class curve are the signs of a system that has earned its quiet rather than faked it.

Acknowledgments

Drafting assistance was provided by an LLM writing collaborator. The author conceived the methodology, mined the convergence data from his own session transcripts, and is accountable for every claim, numeric value, and framing decision in this paper. Reviewer feedback is invited and will be incorporated in subsequent versions.

M3-CRETE stands on the V-slot aluminum-extrusion ecosystem (OpenBuilds-compatible components, CC BY-SA 4.0) and the open-source CAD ecosystem (CadQuery 2.7, build123d). The BigTreeTech Kraken controller and Klipper firmware make the motion system buildable with commodity hardware.

References

- [1] Shore, J. (2004). "Fail Fast." *IEEE Software*, 21(5), 21–25. DOI: 10.1109/MS.2004.1331296 .
- [2] Ohno, T. (1988). *Toyota Production System: Beyond Large-Scale Production*. Productivity Press.
- [3] Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (2016). *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media.
- [4] Basiri, A., Behnam, N., de Rooij, R., Hochstein, L., Kosewski, L., Reynolds, J., & Rosenthal, C. (2016). "Chaos Engineering." *IEEE Software*, 33(3), 35–41. DOI: 10.1109/MS.2016.60 .
- [5] Taleb, N. N. (2012). *Antifragile: Things That Gain from Disorder*. Random House.
- [6] Nygard, M. T. (2018). *Release It! Design and Deploy Production-Ready Software* (2nd ed.). Pragmatic Bookshelf.
- [7] Armstrong, J. (2003). *Making reliable distributed systems in the presence of software errors*. PhD thesis, Royal Institute of Technology, Stockholm.
- [8] Kim, G., Humble, J., Debois, P., & Willis, J. (2016). *The DevOps Handbook*. IT Revolution Press.
- [9] Poppendieck, M., & Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit*. Addison-Wesley.
- [10] Sonnentag, N. (2026). "Self-Verifying LLM-Driven Parametric CAD Assemblies: Lessons from a Meter-Scale Concrete 3D Printer." Sunnyday Technologies. Companion paper; cited for harness architecture details. Zenodo DOI: (*pending*).
- [11] Sonnentag, N. (2026). "Open3DCP: A Canonical Data Schema for Machine Learning in Extrusion-Based 3D Concrete Printing." Sunnyday Technologies. Zenodo DOI: (*pending*).

Appendix A — Reproducing the Convergence Curve

The data behind Table 1 are mined from archived LLM coding-agent session transcripts. Each session is a JSONL file where every line is a conversation event. The extraction pipeline:

```
# Count tool-use events per session
for f in *.jsonl; do
    echo -n "$f: "
    grep -c '"type":"tool_use"' "$f"
done

# Extract unique "Sanity check: N/M passed" snapshots
for f in *.jsonl; do
    echo "=== $f ==="
    grep -o "Sanity check: [0-9]*/[0-9]* passed" "$f" | sort -u
done

# Extract self_check.py outcomes
for f in *.jsonl; do
    echo -n "$f: "
    grep -o "SELF-CHECK [A-Z]*" "$f" | sort | uniq -c
done
```

Transcripts are archived locally at paths that are environment-specific. A sanitized extract — containing only the lines required to reproduce Table 1 and Figure 1 — is published as a Zenodo supplementary dataset accompanying this paper; the dataset DOI is recorded in the final deposit metadata.

Appendix B — Links

- M3-CRETE repository: github.com/sunnyday-technologies/M3-CRETE
- M3-CRETE site and BOM: m3-crete.com , m3-crete.com/bom
- Companion paper (harness architecture): reference [10]
- Open3DCP schema project: open3dcp.org
- Zenodo deposit for this paper, source, figure, and sanitized convergence data:
(*pending*)

Preprint notice. This is version 1.0 (preprint). Subsequent versions will be deposited at the same Zenodo concept DOI and, if an arXiv endorsement is obtained, cross-posted to arXiv [cs.SE](https://arxiv.org/abs/cs.SE) with identical content.

How to cite. Sonnentag, N. (2026). *Convergence Engineering: Commissioning Autonomous Software by Measuring the Rate at Which Novel Failures Trend Toward Zero*. Sunnyday Technologies preprint v1.0. Zenodo. (DOI pending).

License. CC BY 4.0 — you may redistribute and adapt the text provided you cite the author and version.