

SDLC Maturity Model

L0 → L4

Where your engineering organisation sits today, and what each level looks like when the rest of the world has already moved to agentic delivery.

FOR

CTOs, VP Engineering, software-led SMEs

FROM

Gordon Mickel • mickel.tech

PRACTICE

Agentic SDLC • Independent expert • AI systems

USE

Read, score yourself, share with the leadership team

A maturity ladder, not a tooling list.

Most engineering organisations are still running yesterday's methodology against agents that already outpace them. The first mistake is to mistake the symptom (slow delivery) for the cause (a methodology that pre-dates agents).

This guide describes five levels of agentic SDLC maturity, L0 through L4, and what each one looks like in practice. It is deliberately short. Read it in twenty minutes, score yourself against the five-pillar self-assessment on page nine, and you have a defensible starting point for any conversation with your engineering leadership, your board or your owners.

Maturity here spans the whole development lifecycle: requirements, review, testing, documentation, deployment and operations. Code generation is one step of the seven on page ten. Once agents write production code in minutes, the constraint moves into the other six, so an organisation that scores only its coding step is measuring the part of the pipeline that no longer holds the work back.

The model has been refined across multiple multi-product engineering organisations of 50-250+ engineers, in regulated and unregulated sectors. It is the same scoring framework used in paid SDLC diagnostics.

AI made teams faster. That is the problem. The cost of producing code, tests and docs collapsed. The cost of owning them did not. The methodology you choose decides whether new throughput is leverage or maintenance debt the next quarter cannot absorb.

Three things this guide does not promise

It does not tell you which tool to buy. Tools change every quarter. The methodology does not.

It is not a sales document. If you can take this and run with it internally, that is a successful outcome.

It does not promise a 10x overnight. L1 → L2 is 90 days of disciplined work. L2 → L3 is the harder shift -- methodology and review architecture. L4 is a destination, not a starting point.

Five levels, one direction.

Most engineering organisations sit between L0 and L1. The interesting work begins at L2. All throughput numbers assume constant or better DORA change-failure rate; throughput without quality is a downgrade.

L0	Legacy No agentic tooling, no methodology change, ticket-driven delivery, manual everything. The cycle-time ceiling is wherever the slowest engineer happens to be.	Baseline NO AI UPLIFT
L1	Assistant Cursor, Claude Code, OpenAI Codex or similar in active use. Same process, faster typing. No methodology change, no measurement, no portfolio strategy. Productivity gains are real but uneven and uncaptured.	10-15% INDIVIDUAL LIFT
L2	Foundation Standard tooling, training and a baseline KPI framework deployed. Spec-first habits forming. First measurable cycle-time wins. Centralised guardrails for prompts and data. AI-assisted reviews and a documented methodology.	20-30% FIRST 90 DAYS
L3	Spec-driven Agents drive most code production from specs. Cross-model review and eval gates enforce quality. Everything-as-code: prompts, evals, guardrails versioned. Lighthouse lanes for the most autonomous workflows.	2-3× SUSTAINED THROUGHPUT
L4	Factory Autonomous agents ship features end-to-end, review-gated. Humans set priorities; agents execute. Continuous evaluation flywheels feed observability back into specs. Engineering organisation reshaped around product / spec leadership.	5-10× RARE, REVIEW-GATED

Reading the ladder in a brownfield codebase

L0 names a way of working rather than the age of the code. A team on a twenty-year-old system can reach L3; a greenfield team can sit at L1 for years. What changes in brownfield is the order of the work. The safety net comes first. Reproducible environments and tests an agent can run are what make its output checkable, and in an old codebase both usually have to be built before any methodology change pays off.

Score the codebase you actually work in, and score it per system. A brownfield organisation is routinely L3 in a young service and L0 in the core that carries the revenue. The gap between those two numbers is usually the more useful finding. Modernising that core with agents is its own discipline: mickel.tech/sdlc/legacy-modernisation.

Where most engineering orgs actually sit.

L0: Legacy

No agentic tooling. Code is hand-written, line by line, by humans. Reviews are slow and inconsistent. The pipeline is whatever someone built when the team was four people. Tickets drive the work; specs are an afterthought when they exist at all.

You are here if: you cannot point to a single engineer using an agentic coding tool in production this week, or you can but your CI/CD pipeline does not run any agent. The baseline.

L1: Assistant

Engineers use Cursor, Claude Code, OpenAI Codex or similar opportunistically. Productivity gains are real but uneven. Some engineers swear by the tools, some are skeptical, some have stopped after the first frustrating session. There is no methodology, no metrics, no portfolio strategy. License management is wherever someone bought seats; prompt management is whatever each engineer keeps in a Notion page.

You are here if: tools are in active use but you cannot attribute productivity uplift to them in numbers, and there is no central guidance on which tools to use, when, or how they fit together.

The trap at L1 is concluding that "AI does not work for us" when what is failing is the absence of methodology, not the presence of agents.

The shift that actually compounds.

L2: Foundation

Standard tooling deployed across the engineering organisation. Internal playbooks. Prompt patterns. A code-review pattern that uses agents on first pass and humans on the high-leverage decisions. License management is centralised; model selection is documented; observability covers AI usage as well as application health. Cycle time and review time are baselined in week one and tracked monthly.

You are here if: there is a documented methodology any new engineer can follow, AI usage is measured rather than felt, and the first measurable wins (20-30% throughput uplift) are visible inside 90 days.

L3: Spec-driven

Specs are the unit of work. Agents drive most code production from specs and other agents review their output before it reaches a human. Eval gates enforce quality on every meaningful change. Cross-model review catches single-model blind spots. An everything-as-code repository holds prompts, evals, guardrails and agent skills under version control. Lighthouse lanes deploy autonomously.

You are here if: throughput is sustained at 2-3× the L2 baseline, agent-vs-human authorship ratio is tracked (and shifting toward agents), and the eval suite gates every merge.

L1 → L2 is a 90-day disciplined rollout. L2 → L3 is a methodology shift that takes one to three months and changes how the engineering organisation thinks about its own work.

Where engineering becomes orchestration.

L4: Factory

Autonomous agents ship features end-to-end. Reviews are automated and gated, with humans pulled in only for highest-risk decisions. Most production code is reviewed (or written) by agents. The engineering organisation is reshaped around product and spec leadership rather than line-by-line implementation. Senior engineers spend their time on edge cases, judgement calls, agent supervision and product framing.

You are here if: review gates run autonomously, most merges are agent-authored, your engineering org has shifted its hiring profile away from line implementers and toward spec-leads and reviewers, and the cost of producing the tenth feature in a sprint is materially lower than the cost of the first.

Counter-question for L4

The right question at L4 is not "how do we get even faster?" but "what new product surface does this enable that was previously uneconomic?" Most organisations underprice the second-order effect, features that were not on the roadmap because they were not viable at L0/L1 cost suddenly become obvious. Plan for that.

A note on numbers

The percentages and multipliers in this model are operating ranges observed across multiple engineering organisations, and they measure what a whole organisation ships once specs, review and release are rebuilt around agents; careful measurements of a single engineer's gain come in well below them. L4 is still rare, so read its 5-10× as the top of the range rather than a target. The spread within a level is often larger than the gap between levels; what matters is the direction of travel and the shape of the methodology.

Five pillars. Five questions each. Twenty minutes.

Score each statement as a star count (☐ = 0, ☑ = 1). Five statements per pillar; maximum five stars per pillar. The honest scoring is more valuable than the generous one.

AVERAGE PILLAR SCORE	MATURITY LEVEL
< 1.0	L0: Legacy
1.0-1.9	L1: Assistant
2.0-3.0	L2: Foundation
3.1-4.0	L3: Spec-driven
4.1-5.0	L4: Factory

Three caps that override the average

If CI/CD pillar score is below 2: cap the overall result at L1. You cannot run agentic delivery on a pipeline that is click-ops or that does not roll back.

If Monitoring pillar score is below 2: cap the overall result at L2. Without observability you cannot close the eval flywheel that L3 depends on.

If Requirements & Planning pillar score is below 2: cap the overall result at L1. Specs are the unit of work at L2 and above; without them, agents have nothing to drive from.

Requirements, Code.

01

Requirements & Planning

- □ □ □ □ Written specs with acceptance criteria before coding starts.
- □ □ □ □ Agent generates implementation plan from spec (tasks, dependencies, risks).
- □ □ □ □ Agent reviews plan for gaps, conflicts, missing edge cases.
- □ □ □ □ Duplicate or conflicting requirements auto-detected across the backlog.
- □ □ □ □ Planning accuracy measured and fed back into the estimation model.

02

Code Generation & Refactoring

- □ □ □ □ Agentic coding-tool adoption tracked across the engineering org.
- □ □ □ □ Agents build from spec, not just autocomplete.
- □ □ □ □ Standards enforced by AI: lint, architecture, docs, naming.
- □ □ □ □ Auto-generated documentation kept in sync with code changes.
- □ □ □ □ Codebase-drift prevention (spec vs implementation).
- □ □ □ □ Legacy migration assisted by spec-first agentic workflows.

Testing, CI/CD.

03

Testing & Quality

- □ □ □ □ Automated test coverage on every change.
- □ □ □ □ Tests auto-generated from specs and acceptance criteria.
- □ □ □ □ AI reviews 100% of PRs and surfaces critical paths for human review (10-20%).
- □ □ □ □ Automated vulnerability scanning on every commit.
- □ □ □ □ Agents self-iterate (run, fail, fix, re-run) on broken builds.
- □ □ □ □ AI-powered visual QA and end-to-end testing in the pipeline.

04

CI/CD & Deployment

- □ □ □ □ Pipeline as code (not click-ops); agents can read and modify it.
- □ □ □ □ Agents trigger deploys, read failures, auto-fix and re-run.
- □ □ □ □ Feature flags for progressive rollout and instant rollback.
- □ □ □ □ Infrastructure as code; agents provision environments.
- □ □ □ □ Environment parity (dev = staging = prod).
- □ □ □ □ DORA metrics tracked and visible to the engineering org.

Monitoring & Operations.

05

Monitoring & Operations

□ □ □ □ □ DORA metrics tracked: lead time, MTTR, deploy frequency, change-failure rate.

□ □ □ □ □ Agents watch logs and errors, auto-file bugs with root cause.

□ □ □ □ □ Production incidents auto-triaged and routed.

□ □ □ □ □ Usage data feeds back into product decisions and feature proposals.

□ □ □ □ □ Observability flywheel: monitor → detect → fix → learn → improve.

□ □ □ □ □ Alert noise systematically reduced (signal vs noise).

Now do this

Score each pillar honestly. Average the five.

Apply the three caps from page six.

Locate yourself on the L0-L4 ladder.

Read the "what good looks like" pipeline on the next page against your weakest pillar.

Pick one thing to change in the next 30 days, not five.

Seven steps with agents in every loop.

The L3 reference pipeline. Agents touch every step; humans own the merge and the priorities.

1	Plan	Agent reads spec and code, flags ambiguities, maps dependencies, estimates, writes subtasks to your work-management system. Spec prompts are version-controlled.
2	Design	Multimodal agent turns mockups into component stubs with design tokens and a11y fixes. Emits a written PLAN.md that humans review.
3	Build	Agent edits multiple files, respects AGENTS.md conventions, adds tests and docs, runs format / lint. Best-of-N for risky diffs.
4	Test	Agent generates and maintains tests. Gold-set + regression harness. Flake alerts. Drift checks weekly.
5	Review	AI first-pass on P0/P1 + PII / secret / perf. Risk tiers (Low: auto, Medium: human + agent, High: two humans + security). Human owns the merge.
6	Deploy	Trunk-based + flags. Staged rollout. CI rollback job. Pre-flight policy check (residency, secrets, observability hooks).
7	Operate	Agent correlates logs and metrics back to commits. Proposes hotfix PRs. Drafts incident timeline. Cost and latency budget alerts.

Two stages, not a big bang.

The methodology shift fails the same way every time when it is attempted as a single transformation programme. The alternative is a two-stage rollout that compounds.

Stage 1: Foundation (2-4 weeks)

Baseline metrics: cycle time, review time, deploy frequency, change-failure rate.

Standard tooling deployed: editor stack, model selection, license management, prompt registry.

One or two lighthouse lanes shipped end-to-end with the new methodology, alongside the team.

Internal methodology playbook written.

OUTCOME • 20-30% measurable cycle-time reduction in the first 90 days.

Stage 2: Methodology shift (1.5-3 months)

Eval-gated agentic delivery in production.

Cross-model review pipeline.

Everything-as-code repository for prompts, evals, guardrails, agent skills.

Monthly steering until the team owns the system end-to-end.

OUTCOME • 2-3× sustained throughput. Engineering org reshaped around product / spec leadership.

A common failure mode

Skipping Stage 1 and going straight to L3 / L4 patterns on a team that has not yet adopted spec-first habits or built the eval scaffolding. The result is theatre, agentic-looking PRs without the discipline underneath. The fastest way to L3 is the disciplined route through L2.

Numbers a board can argue with.

Anecdote does not survive a board review. Baseline these in week one, report them monthly.

LAYER	METRIC	WHAT IT TELLS YOU
Delivery	Lead time for changes	Spec to production. Direct proxy for cycle time.
Delivery	Deploy frequency	Trunk health and rollback discipline.
Delivery	Change-failure rate	Quality of what you ship, not just speed.
Delivery	MTTR (mean time to restore)	Operational maturity.
SDLC	Spec quality score	How often agent-generated plans need rewrites.
SDLC	Eval-gate pass rate	How often agent output meets the bar on first pass.
SDLC	Agent : human authorship ratio	Where the work is actually getting done.
SDLC	Observability → spec feedback latency	How fast the flywheel closes.
Adoption	% PRs with AI assist	Real adoption, not seat count.
Quality	Gold-set accuracy and drift delta	Whether your evals are still meaningful.

Nine places it usually breaks.

Patterns that show up across multiple engagements. Worth checking against your own programme.

Tool rollout, no methodology change. Cursor licenses bought, methodology untouched. The tools become an individual-productivity uplift (10-15%) rather than an org uplift. Stalls at L1.

Specs treated as a documentation chore. Specs are the unit of work at L2+. If they are written after the code, the agent has nothing to drive from.

No eval pipeline. Without evals, agent output quality is felt rather than measured. Agents drift, prompts rot, and there is no signal until production breaks.

Single-model dependence. Cross-model review catches what a single model misses. Picking one frontier model and trusting it is the equivalent of code review by the same person who wrote the code.

Review by humans on every change. At L3+ humans should review the high-risk 10-20%, not the routine 80%. Reviewing everything is the L1 trap dressed up as quality discipline.

No observability flywheel. Production data should feed back into specs and evals. If it does not, you are running L1 with extra steps.

Output without back-pressure. An agent can produce six months of code in a week. If the engineering system cannot review, understand and own that output, it becomes the most expensive maintenance debt the next quarter inherits. Production cheaper, maintenance not.

Agent-blind codebase. No AGENTS.md, no fast CI, no tests an agent can run to verify itself. The agent still produces confident output that looks like it worked when it did not. Agent-readiness is codebase health; the boring fundamentals gate the leverage.

Cultural mandate without cultural work. Senior engineers already have a working system. Pushing agents on them without demonstrating leverage on their problem feels like training wheels and breeds quiet refusal. Meet them where they are, prove it on their codebase, then scale.

Three ways to use this.

Score yourself and run with it. The model is yours to use internally. If the methodology shift makes sense and you have the bandwidth to drive it in-house, this guide plus the pipeline reference on page ten is enough to start.

Score yourself and bring me in for the diagnostic. A two-week paid SDLC Diagnostic produces a defensible baseline, a 90-day plan and an executive readout, using this same model. From CHF 15,000.

Run the Agentic SDLC Programme. Three modules, together or one at a time: a Code Factory Build for one team, train-the-trainer so your team owns it; the R&D Operating Model for the whole org, at mickel.tech/sdlc/rnd-operating-model; and Legacy Migration for the legacy core, with its own measure and fee.

If the binding constraint is the old system rather than the way of working, start with Legacy Migration. mickel.tech/sdlc/legacy-modernisation covers it, and the readiness checklist on that page tells you whether the codebase is in shape for it.

ENGAGE

A 30-minute discovery call is the cheapest way in.

On the call we agree whether agentic SDLC is the right shape for your team, what an engagement looks like, and what the next step would be. No deck, no pre-work. If it is not a fit, I say so on the call.

BOOK

cal.com/gmickel/discovery30

EMAIL

gordon@mickel.tech

READ

mickel.tech/sdlc

PRACTICE

Binningen, CH · DE / EN