thinkia

AI ENGINEERING · PLAYBOOK 2026

AI-SDLC. *From code* to intent.

The operating system Thinkia uses to ship software when code is no longer the bottleneck.

EDITORIAL

Not an improvement. *A shift in scale.*

We are living through the largest change in how software is built since software has existed. It is not a new tool. It is not another framework. It is a shift in practice, scale and nature — all at once.

The exponential curve is not a slogan. It is what we have been watching every quarter since 2023: each generation of models collapses another layer of engineering. First the code. Then testing. Now architecture. Soon, the whole cycle. What used to take six months gets built in two weeks. What used to need a team of twelve is done with three. What used to be out of reach becomes trivial.

Organisations reading this moment as *"a productivity improvement"* will arrive late. Not because they are slow. Because they are measuring the wrong thing. The question is not how much more code you can produce. It is what kind of organisation makes sense when producing code is no longer a bottleneck.

AI-SDLC was born from that question. Ten versions later, we are publishing it. Because keeping it private no longer makes sense, and because the sector moves faster when we share the operating system, not just the outputs.

What you are holding is not a proposal. It is what we use.

David Alejano
Chief Strategy Officer · Thinkia · Madrid, May 2026
linkedin.com/in/dalejano

EXECUTIVE SUMMARY

The traditional software development lifecycle assumes the engineer's primary activity is writing code. AI-SDLC redefines that assumption: the highest-value human activity becomes specifying intent and validating outcome, and code becomes a by-product of well-written specifications generated by AI agents.

This playbook describes the complete model: six temporal phases (F0 to F5) plus a transverse architecture track; a four-level hierarchy with its own artifacts, owners and governance; an executable Golden Spec as source of truth; a system of Health Reports at three moments in the flow; and an MCP-compatible Skill Execution Model that lets the methodology run from any orchestrator.

The cultural shift matches the technical one in magnitude. The engineer's time distribution inverts: from 60 % to 10 % writing code, from 20 % to 50 % reviewing and validating. Planning becomes Kanban-first out of necessity. And governance becomes part of the flow — Quality Gates, DLP, traceability and compliance stop being layers that slow delivery and become edges of the process itself.

6

LIFECYCLE PHASES

F0 Input → F5 Delivery, plus a transverse architecture track

10×

GENERATION ACCELERATION

AI generates code 5–10× faster; the bottleneck moves to human review

5

GOLDEN SPEC SIGNATURES

Without an approved Golden Spec there are no stories and no code

CONTENTS

01
The paradigm *shift.*

The traditional SDLC spent four decades optimising an implicit hypothesis: the engineer's highest-value activity is translating requirements into code. Every framework, every methodology, every inherited tool assumes that activity as the core.

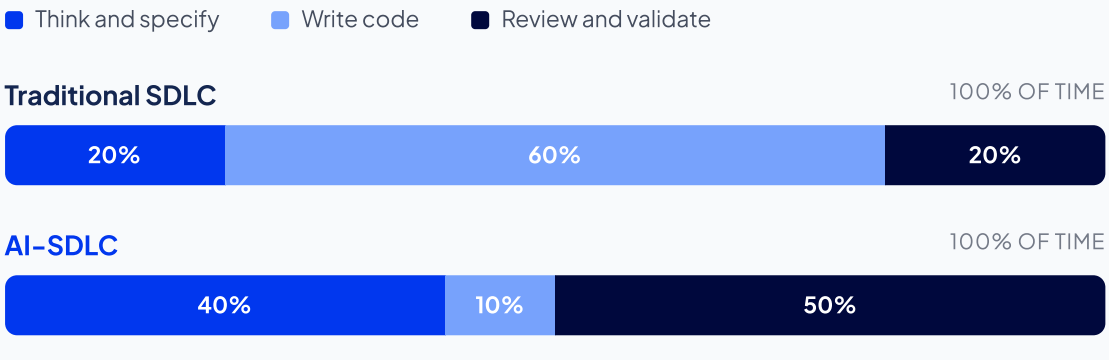
AI-SDLC starts from a different assumption. When an AI agent can generate correct code from a sufficiently precise specification, the highest-value human activity stops being *translating* and becomes *specifying* and *validating*.

This is not an incremental optimisation. It is a shift in nature that affects the phases of the cycle, team roles, artifacts produced, applied governance and the engineering culture of the organisation.

THE FOUR PILLARS

- **Specification is the new source code.** What used to live as tacit knowledge in a senior's head becomes explicit artifact. The spec becomes the contract the code implements.
- **Review is the central activity.** From 20 % to 50 % of the engineer's time goes into validating, not writing.
- **Architecture is designed to be readable by agents.** Patterns and conventions live in versioned artifacts, not conversations.
- **Governance is integrated into the flow.** Quality Gates, DLP and traceability are edges of the process, not extra layers.

The new distribution of effort.



Writing code shrinks 6×. Thinking and validating multiplies 2.25×. *This is not about delegating the boring stuff — it is about changing the cognitive economy of the work.*

The inversion is structural. What becomes scarce is clear attention — the kind that produces a precise spec, that detects a contradiction in a design, that understands what the code *should* do better than the code itself expresses it.

The bottleneck stops being coding speed and becomes specification quality and review rigor. And here appears the most silent risk of the paradigm: code arrives fast, tests pass, the linter is green — and nobody on the team can explain exactly what that function generated three days ago does.

If you can't explain what the generated code does, don't merge it.

AI-SDLC's golden rule — regardless of zone, of whether tests pass, or of whether the linter is green.

This risk has a name: **comprehension debt**. It is the most dangerous debt in the new paradigm because it accumulates invisibly. It does not show up on test coverage dashboards. SAST does not detect it. CI does not block it. It only surfaces when the system needs to change — and the team discovers it does not understand it.

Unlike classical technical debt, comprehension debt is not a code quality problem: the code can be impeccable. It is a problem of cognitive ownership. The team lost the connection between intent (the spec) and execution (the code). And in an environment where AI can regenerate code in minutes, that connection is the only thing that guarantees the system does what the business needs.

To measure and manage it, AI-SDLC introduces the **Comprehension Score**: a metric that periodically evaluates the team's capacity to explain the code's behaviour without reading it line by line. When it falls below the threshold, a mandatory ownership session is triggered — the equivalent of a simulator drill with the autopilot disconnected.

It is not a bureaucratic process. It is the difference between a team that operates its system and a team that merely hosts it.

02
The journey of AI-assisted *software.*

Not every organisation starts at the same point. There are five stages in the maturity of AI-assisted software development — and knowing where you are defines what you should do next.

Traditional Development
Software is built manually, with full control but at a high cost.

VibeCoding
Software is generated fast… but without control or consistency.

Coding 3.0
Software is accelerated with AI, but still relies on the developer.

Spec Driven Development
Software is defined through structured intent within a rules framework.

Enterprise AI SWLC
Software is orchestrated as a governed system of rules, specs and capabilities.

The critical jump: from VibeCoding to Spec Driven

The biggest mistake organisations make is confusing VibeCoding with transformation. Generating code fast without governance creates technical debt at AI speed. The key is not generation speed — it is the quality of the specification that precedes it.

Enterprise AI SWLC: the destination

At the Enterprise stage, software stops being an artisanal product and becomes the governed output of a system of rules, specs and capabilities. AI does not replace the engineer — it redefines what kind of engineer you need to be.

03
Anatomy of the *system.*

Work is organised in four levels. Each one answers a different question and produces a different artifact.

Cap

Capability — What business ambition are we pursuing?

Vehicle for strategic multi-feature ambitions. Lightweight document containing strategic intent, business case, personas, outcomes and KPIs, the features that make it up and an executable delivery plan. It does not go through the feature's five signatures.

Ftr

Feature — What do we deliver to the user and how is it built?

The Golden Spec in three files: `.spec.md` (what it does), `.design.md` (how it is structured) and `.ui-spec.md` (UI). Requires five mandatory signatures before generating a single line of code.

Str

Story — What concrete chunk does the agent generate now?

The agent's executable unit. Lightweight because it inherits from the feature. A single automation zone, a single bounded context, acceptance verifiable in less than two hours of human review. If it needs more, it must be split — the context is poorly scoped.

Tsk

Task — What technical steps does this story have?

A checkbox list inside the story. Executed by the agent or the developer depending on the zone. The full story has a single zone — if a story crosses green → amber → red, it gets split before execution.

Las seis fases y el track transversal.

F0 — Input Bundle standardises what context the Framing phase receives, regardless of how the project originated. Path A (external RFP) or Path B (internal Discovery) — both produce the same three canonical files with the same structure and signatures.

F1 — Problem Framing answers what and why, never how. Discovery Brief, Capability Brief and Spike Spec cover the three possible cases. If a mockup or a component diagram appears in Framing, it is out of phase.

F2 — Spec Engineering is the most transformed phase. The Golden Spec stops being a requirements document and becomes an executable artifact. Without a Golden Spec approved with five signatures, there are no stories and no code. This gate is the most powerful control mechanism in AI-SDLC.

F3 — AI Generation inverts the ratio. The engineer does not write code — they supervise the agent's execution, adjust prompts and context if the output drifts, and iterate until the story is met. Agents run in isolated Docker sandboxes without access to the corporate network.

F4 — Review & Validation becomes the main engineering activity. Three dimensions: correctness vs. spec, comprehensibility and security. Review is not a formality — it is the last line of defence against comprehension debt.

F5 — Delivery is AI-augmented end-to-end. Deploy-as-Spec, predictive Change Impact Analysis, rollout with auto-rollback and a feedback loop to the Production Context close the cycle spec → generation → deploy → operation → spec.

04
The Golden Spec.

The executable specification. Three files, three owners, five signatures. The system's source of truth.

spec.md

What it does

Intent, Behavior scenarios (Given-When-Then), Acceptance, NFRs, Regulatory tier, Automation Zone Summary, Integration Contracts, Story Breakdown. Owner: Product Owner + Spec Architect.

design.md

How it is structured

Domain Model, Component Design, Automation Decision Map, Integration Design, Data Design, Local ADRs, Promotion Candidates. Owner: Spec Architect + Dev Senior.

ui-spec.md

The interface

Mockups, user flows, states (loading, error, empty, success), design system references, accessibility rules. Owner: Product Designer. Only when the feature has UI.

THE FIVE MANDATORY SIGNATURES

Without an approved Golden Spec, there are no stories and no code.

Product Owner certifies business intent. Spec Architect certifies coherence and correct design. Dev Senior certifies feasibility with the current architecture. Security Engineer certifies that it introduces no risk. QA Lead certifies that the scenarios cover the full behaviour. If the feature has UI, the Product Designer signs a sixth time.

The automation zones.

ZONE	AGENT AUTONOMY	HUMAN REVIEW	EXAMPLES
Green	Autonomous generation	Spot-check (~10 %)	CRUD, scaffolding, unit tests, API docs
Amber	Supervised generation	Line-by-line review (100 %)	Business logic, integrations, event handlers
Red	Human authorship (AI assists)	Pair programming + tests	Core algorithms, auth/crypto, critical compliance

The Automation Decision Map lives in `.design.md`, not in `.spec.md`. Its place is where components are enumerated — green/amber/red classification requires knowing the feature's components, and components are enumerated in the design.

The story automatically inherits the zone of the most restrictive component it touches. If it touches an amber component, the whole story is amber. A story cannot be half green and half red — if it crosses zones, it gets split before execution.

Each project declares a **Project Zone Profile** in the repository constitution: which zones it admits. Green-only for internal portals and dashboards. Green + Amber for most SaaS products. Green + Amber + Red for platforms with auth or compliance components. Amber + Red for platform kernels.

A story with a zone outside the declared profile automatically escalates as a blocker in Health Reports. The team decides whether to change the profile via Architecture Proposal, split the story, or move it to another repo with a different profile. There are no silent exceptions.

05
The critical *artifacts.*

AI-BOM, Health Reports, layered Context Package and Coverage Guarantees form the traceability chain that turns assisted generation into an auditable practice.

Granular AI-BOM per story.

Each story includes an AI Bill of Materials: models used (name, version, provider), tokens consumed, manual post-generation changes (diff), effective prompts, test coverage achieved. Per-story granularity allows attributing tokens, models and edits to the real unit of generation — the same one the release AI-BOM aggregates.

It answers the question every audit will ask: who or what wrote this code, and under what criteria? Without AI-BOM, nothing is merged and no feature is closed.

Distributed Health Reports.

Three moments: Step Health Check, Artifact Health Report and Spec Health Report. Same structure — Blockers, Warnings, Suggestions.

Without distributed detection, contradictions appear in production. With it, the cost stays where the problem was introduced.

Coverage Guarantees: what is missing, not what is wrong.

Health Reports catch what is *wrong* in what we wrote. Coverage Guarantees catch what is *missing* relative to the input we received. An artifact can be coherent and still incomplete. Both dimensions must be validated.

- F0 → F1 — every relevant paragraph of the RFP appears in Discovery brief or requirements.
- F1 → F2 — every feature in the capability brief has its signed feature spec.
- spec/design → stories — every scenario and component is covered by at least one story.

Layered Context Package.

The context chain flows from the most stable and general to the most specific. Without this complete chain, the agent hallucinates. The expensive layer is compiled once per feature; the thin layer, once per story.

```
Platform Registry + Corporate constitution + Corp. prompts library
  ↓ (organisational — stable, shared across all projects)

Repo constitution + 7 views of Application Architecture + Patterns
  ↓ (project — stable inside the repo)

feature.spec.md + feature.design.md + feature.ui-spec.md
  ↓ (feature — changes with each signed Golden Spec)

story-NNN.md                ← generation order
  ↓ (story — thin, fast to compile)

Production Context slice    ← real system behaviour
  ↓

AI Agent → generates 4-code/
```

The **expensive layer** (`feature.context.yml`) is compiled once per feature and only recompiled if the feature changes or there is a refresh of organisational sources. It is the big investment — organisational + repo + golden spec + production context slice.

The **thin layer** (`story.context.yml`) references the expensive layer as parent and adds only the specific slice of components and scenarios for the current story. The agent moves fast between stories without recompiling anything expensive.

The **Production Context** is a per-service artifact that captures real system behaviour in production. It does not live in the repo — it lives in the operation platform. Runtime metrics, incident history with root causes, observed behaviour, recent deploys. It is a *source* feeding the Context Package, not a compilation.

The distinction matters: confusing Production Context with Context Package leads to not knowing who maintains what. The Platform Team and SRE maintain the Production Context Store; the AI Engineering Office maintains the Context Package compilation skills.

06
Skill Execution *Model.*

V10 — MCP COMPATIBLE

The methodology decouples from the *orchestrator.*

Any engine compatible with the Skill Contract can execute AI-SDLC. Claude Code, Cursor, Windsurf, an IDE plugin, or a custom agent in LangGraph or semantic-kernel — all of them can invoke skills as MCP tools without needing a proprietary runner.

atomic

Transformation that produces a section of an artifact or a full artifact. The majority of pipeline prompts.

validator

Cross-artifact verification. Produces no content — produces a Health Report. output: null, healthChecks: [...].

context

Context Package compilation. Cacheable by input hash. Prerequisite for any generation skill.

Each atomic prompt exposes a YAML manifest with a Skill Contract: `skill-id`, `skill-version` (semver), typed `inputs`, `outputs` with fixed schema `{output, healthChecks}`, `model-zone` and `prompt-hash` (SHA-256). The fixed output schema guarantees any orchestrator can consume the result without custom logic.

Model routing by zone.

ZONE	RECOMMENDED MODEL	TEMPERATURE	USE CASES
Green	Fast, cheap model	0.1	Extractions, mappings, deterministic derivations
Amber	Balanced model	0.2	Synthesis, technical decisions, cross-artifact validation
Red	Powerful model	0.1	Complex reasoning, architecture, critical validators

The `model-zone` declared in the manifest becomes corporate LLM Gateway policy. The Gateway decides which model to invoke based on the zone — this logic was the runner's responsibility in earlier versions; now it is centralised. Any external orchestrator respects the policy without replicating logic.

Workflows are declarative compositions of skill invocations with `parallel-with` and `after` for orchestration. Strict DAG, typed references, explicit barriers and declarative `on-health-blocker` policy. Workflow YAML is standard — any MCP-compatible engine executes it.

The **LLM Gateway** is the platform's most critical layer. A single point through which every AI model call passes. It centralises DLP filtering, intelligent routing, complete auditing (prompt hash, model, user, project, timestamp feeding the AI-BOM), per-team and per-project metering, and multi-provider policy to avoid vendor lock-in.

In environments with sensitive data, the Gateway is deployed in VPC or on-premise for data sovereignty. Non-negotiable in companies with European regulatory compliance — EU AI Act, GDPR, financial or healthcare sectors.

07
Integrated *governance.*

Quality Gates are state machine transition conditions, not layers superimposed on the process. Each artifact has eleven possible canonical states — only `approved` and `implemented` allow generating downstream.

F0 — Documented context + signed platform-dependencies.md

Without documented context and without Architect + PO signature, the Capability brief does not start. Absolute block before time is invested in Framing.

F2 — Artifact Health Report without blockers + 5 mandatory signatures

Open blockers block artifact signature. Warnings not explicitly signed escalate to blockers. Without the Golden Spec's five signatures, there are no stories. Without stories, there is no code.

F3 — Cross-artifact Spec Health Report + automatic deterministic gates

Lint, SAST, test coverage. Red build → the agent iterates or escalates. Cross-artifact blockers → F3 does not start. The Invalidation Impact Report is signed if the commit touches approved artifacts with dependents.

F4 — Human approval + complete AI-BOM

Without human approval with a comprehension check, nothing is merged. Without per-story AI-BOM aggregated per feature, no feature is closed. Both are non-negotiable.

F5 — Deploy Risk Report + canary health check

risk_level: critical blocks the deploy until human review. Violation of rollback conditions declared in .spec.md triggers immediate auto-rollback without manual intervention.

08
Kanban-first and *metrics.*

AI accelerates generation 5–10×. Fixed sprints create shippable inventory that does not get merged for lack of review. The bottleneck moves to human review, which is flow-limited, not time-limited. That is why the canonical planning is Kanban.

5–10×
ACCELERATION
in AI code generation vs. manual authorship

3
BACKLOGS
Roadmap (capabilities), Feature, Story — nested boards

WIP
BY ZONE
Green: 5 gen / 3 review. Amber: 3/2. Red: 1/1.

2*h*
MAX REVIEW
Story poorly sized if its acceptance takes more than 2h to validate

The Story Board has swimlanes per zone. Each zone has its own WIP limit, calibrated to the corresponding human review capacity. If a column hits WIP, no new work enters until something leaves. Stories of different zones are not interchangeable.

The five core flow metrics are: **Lead time** (ready → merged), **Cycle time** (generating → merged), **Throughput** (stories/time per zone), **WIP age** (time in column) and **Blocker rate**. Plus two specific to AI-SDLC: **Health Report blocker rate** (% of stories that generate a blocker) and **Comprehension debt index** (team's understanding of its own code).

The maturity roadmap has four stages. Stage 1 (Week 1): a pilot team can do AI-SDLC with an agentic IDE + canonical structure + minimal constitution. Stage 2 (Month 1–3): multiple teams with LLM Gateway + DLP + automatic AI-BOM. Stage 3 (Month 3–9): the whole organisation with operational AI-augmented delivery. Stage 4 (Month 9–18): autonomous platform with RAG, knowledge graph and Production Context with predictive analysis.

Teams already running Scrum can adapt sprints as replenishment timeboxes, but canonical planning is flow-based. The reason is structural: time-boxed sprints become an artificial bottleneck when generation accelerates 5–10×. Forcing the flow into time-boxes designed for manual coding speeds does not pay off.

Conclusion.

The new software lifecycle does not start with a ticket and end with a merge. It starts with a structured intent — the spec — and ends when the system in production feeds back into the next spec. The code in between is the by-product of that cycle, not its protagonist.

This changes what it means to build software. It changes who does what. It changes when decisions are made and who makes them. It changes what a team must understand about its own system in order to operate it. It changes, in the end, what kind of organisation has the edge.

There is a recurring question when we present this system for the first time: is this going to replace our engineers? The answer is the opposite. It is going to make the engineer, finally, an engineer: someone whose authority is exercised over intent and outcome, not over syntax. Someone whose knowledge is inscribed in readable, versioned artifacts, not in commits nobody remembers writing.

v10 is not the final answer. It is the current shape of a question that will not close: how to build software when code is no longer the bottleneck. There will be a v11, and a v12, and replacements we have not yet imagined. The cycle will keep accelerating. What does not change is the principle: **specify with precision, generate with agents, validate with rigor, operate with traceability, and close the cycle.**

> *"Code is the by-product. The spec is the asset.*

thinkia

thinkia.com