thinkia

AI ENGINEERING · PLAYBOOK 2026

AI-SDLC. *De código* a intención.

El sistema operativo con el que Thinkia lleva software al mercado cuando el código deja de ser el cuello de botella.

EDITORIAL

No es una mejora.
Es un cambio de escala.

Estamos viviendo el mayor cambio en cómo se construye software desde que existe el software. No es una herramienta nueva. No es un framework más. Es un cambio de práctica, de escala y de naturaleza, simultáneamente.

La curva exponencial no es un eslogan. Es lo que vemos cada trimestre desde 2023: cada generación de modelos colapsa otra capa de la ingeniería. Primero el código. Después el testing. Ahora la arquitectura. Pronto, todo el ciclo. Lo que tardaba seis meses se construye en dos semanas. Lo que costaba un equipo de doce se hace con tres. Lo que era inalcanzable se vuelve trivial.

Las organizaciones que están leyendo este momento como *"una mejora de productividad"* llegarán tarde. No porque sean lentas. Sino porque están midiendo la cosa equivocada. La pregunta no es cuánto código más puedes producir. Es qué tipo de organización tiene sentido cuando producir código deja de ser un cuello de botella.

AI-SDLC nació de esa pregunta. Diez versiones después, lo publicamos. Porque guardarlo ya no tiene sentido y porque el sector avanza más rápido cuando se comparte el sistema operativo, no solo sus outputs.

Lo que tienes en las manos no es una propuesta. Es lo que usamos.

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

RESUMEN EJECUTIVO

El ciclo de desarrollo de software tradicional asume que la actividad principal del ingeniero es escribir código. AI-SDLC redefine ese supuesto: la actividad de mayor valor humano pasa a ser especificar intención y validar resultado, y el código se convierte en subproducto de especificaciones bien escritas, generadas por agentes de IA.

Este playbook describe el modelo completo: seis fases temporales (F0 a F5) más un track transversal de arquitectura; una jerarquía de cuatro niveles con artefactos, dueños y gobernanza propios; una Golden Spec ejecutable como fuente de verdad; un sistema de Health Reports en tres momentos del flujo; y un Skill Execution Model MCP-compatible que permite operar la metodología desde cualquier orquestador.

El cambio cultural es de la misma magnitud que el técnico. La distribución del esfuerzo del ingeniero se invierte: del 60 % al 10 % en escribir código, del 20 % al 50 % en revisar y validar. La planificación se vuelve Kanban-first por necesidad. Y la gobernanza se integra en el flujo — Quality Gates, DLP, trazabilidad y compliance dejan de ser capas que ralentizan para convertirse en aristas del propio proceso.

6

FASES DEL CICLO

F0 Input → F5 Delivery, más track transversal de arquitectura

10×

ACELERACIÓN EN GENERACIÓN

La IA genera código 5–10× más rápido; el cuello pasa al review humano

5

FIRMAS EN LA GOLDEN SPEC

Sin Golden Spec aprobada no hay stories ni código

CONTENIDOS

01	El cambio de paradigma	p. 05
02	El viaje del desarrollo software con IA	p. 07
03	Anatomía del sistema: fases y jerarquía	p. 08
04	La Golden Spec y las zonas de automatización	p. 10
05	Los artefactos críticos: AI-BOM, Health Reports y Context Package	p. 12
06	El Skill Execution Model y compatibilidad MCP	p. 14
07	Gobernanza integrada y Quality Gates	p. 16
08	Planificación Kanban-first y métricas de flujo	p. 17
—	Conclusión	p. 18

01
El cambio de *paradigma.*

El SDLC tradicional optimizó durante cuatro décadas una hipótesis implícita: la actividad de mayor valor del ingeniero es traducir requisitos a código. Cada framework, cada metodología, cada herramienta heredada asume esa actividad como núcleo.

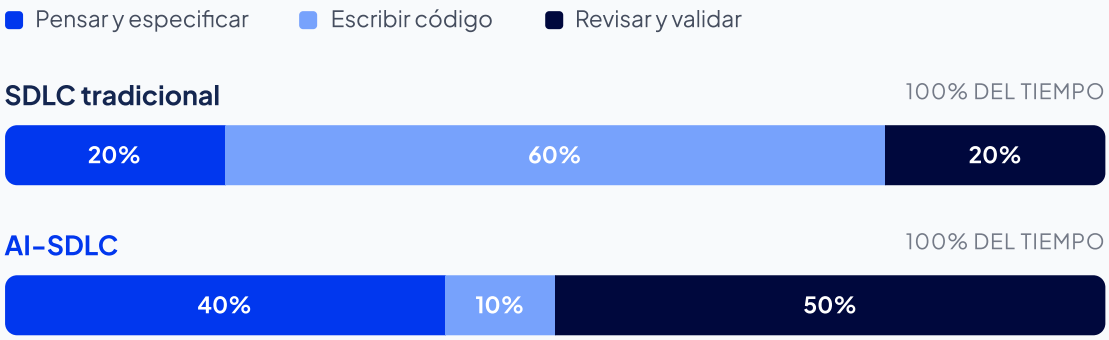
AI-SDLC parte de un supuesto distinto. Cuando un agente de IA puede generar código correcto a partir de una especificación suficientemente precisa, la actividad de mayor valor humano deja de ser *traducir* y pasa a ser *especificar* y *validar*.

Esto no es una optimización incremental. Es un cambio de naturaleza que afecta a las fases del ciclo, a los roles del equipo, a los artefactos producidos, a la gobernanza aplicada y a la cultura de ingeniería de la organización.

LOS CUATRO PILARES

- **La especificación es el nuevo código fuente.** Lo que vivía como conocimiento tácito en la cabeza de un senior pasa a ser artefacto explícito. La spec se convierte en el contrato que el código implementa.
- **El review es la actividad central.** Del 20 % al 50 % del tiempo del ingeniero se invierte en validar, no en escribir.
- **La arquitectura se diseña para ser legible por agentes.** Patrones y convenciones viven en artefactos versionados, no en conversaciones.
- **La gobernanza se integra en el flujo.** Quality Gates, DLP y trazabilidad son aristas del proceso, no capas adicionales.

La nueva distribución del esfuerzo.



Escribir código se reduce 6×. Pensar y validar se multiplica por 2,25×. *No es delegar lo aburrido — es cambiar la economía cognitiva del trabajo.*

La inversión es estructural. Lo escaso pasa a ser la atención clara — la que produce una spec precisa, la que detecta una contradicción en un diseño, la que entiende lo que el código debería hacer mejor que el código mismo lo expresa.

El cuello de botella deja de ser la velocidad de codificación y pasa a ser la calidad de la especificación y el rigor del review. Y aquí aparece el riesgo más silencioso del paradigma: el código llega rápido, los tests pasan, el linter está verde — y nadie en el equipo puede explicar exactamente qué hace esa función generada hace tres días.

Si no puedes explicar qué hace el código generado, no lo mergees.

Regla de oro del AI-SDLC — independientemente de la zona, de que los tests pasen, o de que el linter esté verde.

Este riesgo tiene nombre: **comprehension debt**. Es la deuda más peligrosa del nuevo paradigma porque se acumula de forma invisible. No aparece en los dashboards de cobertura de tests. No la detecta el SAST. No la bloquea el CI. Solo emerge cuando hay que cambiar el sistema — y el equipo descubre que no lo entiende.

A diferencia de la deuda técnica clásica, la comprehension debt no es un problema de calidad del código: el código puede ser impecable. Es un problema de propiedad cognitiva. El equipo perdió la conexión entre la intención (la spec) y la ejecución (el código). Y en un entorno donde la IA puede regenerar código en minutos, esa conexión es lo único que garantiza que el sistema hace lo que el negocio necesita.

Para medirla y gestionarla, AI-SDLC introduce el **Comprehension Score**: una métrica que evalúa periódicamente la capacidad del equipo de explicar el comportamiento del código sin leerlo línea a línea. Cuando baja del umbral, se activa una sesión de apropiación obligatoria — el equivalente a un simulacro con el autopiloto desconectado.

No es un proceso burocrático. Es la diferencia entre un equipo que opera su sistema y un equipo que lo hospeda.

02
El viaje del desarrollo *software con IA.*

No todas las organizaciones arrancan en el mismo punto. Hay cinco estadios en la madurez del desarrollo software con IA — y saber dónde estás define qué debes hacer a continuación.

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.

El salto crítico: de VibeCoding a Spec Driven

El mayor error que cometen las organizaciones es confundir VibeCoding con transformación. Generar código rápidamente sin gobernanza crea deuda técnica a velocidad de IA. La clave no es la velocidad de generación — es la calidad de la especificación que la precede.

Enterprise AI SWLC: el destino

En el estadio Enterprise, el software deja de ser un producto artesanal para convertirse en el output gobernado de un sistema de reglas, specs y capacidades. La IA no sustituye al ingeniero — redefine qué tipo de ingeniero necesitas ser.

03
Anatomía del *sistema.*

El trabajo se organiza en cuatro niveles. Cada uno responde a una pregunta distinta y produce un artefacto distinto.

Cap

Capability — ¿Qué ambición de negocio perseguimos?

Vehículo para ambiciones estratégicas multi-feature. Documento ligero que contiene intent estratégico, business case, personas, outcomes y KPIs, features que la componen y un delivery plan ejecutable. No pasa por las cinco firmas del feature.

Ftr

Feature — ¿Qué entregamos al usuario y cómo se construye?

La Golden Spec en tres archivos: `.spec.md` (qué hace), `.design.md` (cómo se estructura) y `.ui-spec.md` (UI). Requiere cinco firmas obligatorias antes de generar una sola línea de código.

Str

Story — ¿Qué trozo concreto genera el agente ahora?

La unidad ejecutable del agente. Ligera porque hereda del feature. Una sola zona de automatización, un solo bounded context, acceptance verificable en menos de dos horas de review humano. Si necesita más, hay que partirla — el contexto está mal dimensionado.

Tsk

Task — ¿Qué pasos técnicos tiene esta story?

Lista de checkboxes dentro de la story. Ejecutada por el agente o por el desarrollador según la zona. La story completa tiene zona única — si una story cruza verde → ámbar → rojo, se parte antes de ejecutar.

Las seis fases y el track transversal.

TRACK TRANSVERSAL Architecture & Constitution — evoluciona continuamente en paralelo a las fases

F0	F1	F2	F3	F4	F5
Input Bundle	Problem Framing	Spec Engineering	AI Generation	Review & Validation	Delivery
100% human	85% human	70% human	80% IA	75% human	85% IA

F0 — Input Bundle estandariza qué contexto recibe la fase de Framing, independientemente de cómo se originó el proyecto. Path A (RFP externo) o Path B (Discovery interno) — ambos producen los mismos tres archivos canónicos con la misma estructura y firmas.

F1 — Problem Framing responde a qué y por qué, nunca a cómo. Discovery Brief, Capability Brief y Spike Spec cubren los tres casos posibles. Si en Framing aparece un mockup o un diagrama de componentes, está fuera de fase.

F2 — Spec Engineering es la fase más transformada. La Golden Spec deja de ser un documento de requisitos y se convierte en un artefacto ejecutable. Sin Golden Spec aprobada con cinco firmas, no hay stories ni código. Este gate es el mecanismo de control más potente del AI-SDLC.

F3 — AI Generation invierte la proporción. El ingeniero no escribe código — supervisa la ejecución del agente, ajusta prompts y contexto si el output diverge, e itera hasta cumplir la story. Los agentes ejecutan en sandboxes Docker aislados sin acceso a red corporativa.

F4 — Review & Validation se convierte en la actividad de ingeniería principal. Tres dimensiones: correctitud vs. spec, comprensibilidad y seguridad. El review no es un trámite — es la última línea de defensa frente al comprehension debt.

F5 — Delivery es AI-augmented end-to-end. Deploy-as-Spec, Change Impact Analysis predictivo, rollout con auto-rollback y feedback loop al Production Context cierran el ciclo spec → generación → deploy → operación → spec.

04
La Golden *Spec.*

La especificación ejecutable. Tres archivos, tres dueños, cinco firmas. La fuente de verdad del sistema.

spec.md

Qué hace

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

design.md

Cómo se estructura

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

ui-spec.md

La interfaz

Mockups, user flows, estados (loading, error, empty, success), referencias al design system, reglas de accesibilidad. Dueño: Product Designer. Solo si el feature tiene UI.

LAS CINCO FIRMAS OBLIGATORIAS

Sin Golden Spec aprobada, no hay stories ni código.

Product Owner certifica el intent de negocio. Spec Architect certifica coherencia y diseño correcto. Dev Senior certifica viabilidad con la arquitectura actual. Security Engineer certifica que no introduce riesgos. QA Lead certifica que los scenarios cubren el comportamiento completo. Si el feature tiene UI, el Product Designer firma una sexta vez.

Las zonas de automatización.

ZONA	AUTONOMÍA DEL AGENTE	REVIEW HUMANO	EJEMPLOS
Verde	Generación autónoma	Spot-check (~10 %)	CRUD, scaffolding, tests unitarios, docs API
Ámbar	Generación supervisada	Review línea a línea (100 %)	Lógica de negocio, integraciones, event handlers
Roja	Autoría humana (IA asiste)	Pair programming + tests	Algoritmos core, auth/crypto, compliance crítico

El Automation Decision Map vive en el `.design.md`, no en el `.spec.md`. Su lugar es donde los componentes se enumeran — la clasificación verde/ámbar/rojo requiere conocer los componentes del feature, y los componentes se enumeran en el design.

La story hereda automáticamente la zona del componente más restrictivo que toca. Si toca un componente ámbar, la story es ámbar completa. Una story no puede ser mitad verde y mitad roja — si cruza zonas, se parte antes de ejecutar.

Cada proyecto declara un **Project Zone Profile** en la constitución del repositorio: qué zonas admite. Green-only para portales internos y dashboards. Green + Amber para la mayoría de productos SaaS. Green + Amber + Red para plataformas con componentes de auth o compliance. Amber + Red para kernels de plataforma.

Una story con zona fuera del perfil declarado escala automáticamente como blocker en Health Reports. El equipo decide si cambia el perfil mediante Architecture Proposal, parte la story, o la saca a otro repo con perfil distinto. No hay excepciones silenciosas.

05
Los artefactos *críticos.*

AI-BOM, Health Reports, Context Package layered y Coverage Guarantees forman la cadena de trazabilidad que convierte generación asistida en práctica auditable.

AI-BOM granular por story.

Cada story incluye un AI Bill of Materials: modelos usados (nombre, versión, proveedor), tokens consumidos, modificaciones manuales post-generación (diff), prompts efectivos, cobertura de tests alcanzada. La granularidad por story permite atribuir tokens, modelos y edits a la unidad real de generación — la misma que el AI-BOM de release agrega.

Responde a la pregunta que toda auditoría hará: ¿quién o qué escribió este código y bajo qué criterios? Sin AI-BOM, no se mergea ni se cierra el feature.

Health Reports distribuidos.

Tres momentos: Step Health Check, Artifact Health Report y Spec Health Report. Misma estructura — Blockers, Warnings, Suggestions.

Sin detección distribuida, las contradicciones aparecen en producción. Con ella, el coste se mantiene donde se introdujo el problema.

Coverage Guarantees: lo que falta, no lo que está mal.

Health Reports atrapan lo que está *mal* en lo que escribimos. Coverage Guarantees atrapa lo que *falta* respecto al input que recibimos. Un artefacto puede ser coherente y aun así estar incompleto. Hay que validar las dos dimensiones.

- F0 → F1 — cada párrafo relevante del RFP aparece en Discovery brief o requisitos.
- F1 → F2 — cada feature del capability brief tiene su feature spec firmado.
- spec/design → stories — cada scenario y componente está cubierto por al menos una story.

Context Package layered.

La cadena de contexto fluye de lo más estable y general a lo más específico. Sin esta cadena completa, el agente alucina. La capa cara se compila una vez por feature; la capa fina, una por story.

La **capa cara** (`feature.context.yml`) se compila una vez por feature y se recompila solo si el feature cambia o si hay refresh de fuentes organizacionales. Es la inversión grande — organizacional + repo + golden spec + production context slice.

La **capa fina** (`story.context.yml`) referencia la capa cara como padre y añade únicamente el slice específico de componentes y scenarios de la story actual. El agente rota rápido entre stories sin recompilar nada caro.

El **Production Context** es un artefacto per-servicio que captura el comportamiento real del sistema en producción. No vive en el repo — vive en la plataforma de operación. Métricas runtime, historial de incidentes con causas raíz, comportamiento observado, deploys recientes. Es una *fuente* que alimenta el Context Package, no una compilación.

La distinción importa: confundir Production Context con Context Package lleva a no saber quién mantiene qué. El Platform Team y el SRE mantienen el Production Context Store; el AI Engineering Office mantiene los skills de compilación del Context Package.

06
Skill Execution *Model.*

V10 — MCP COMPATIBLE

La metodología se desacopla del *orquestador.*

Cualquier engine compatible con el Skill Contract puede ejecutar AI-SDLC. Claude Code, Cursor, Windsurf, un IDE plugin, o un agente custom en LangGraph o semantic-kernel — todos pueden invocar los skills como herramientas MCP sin necesidad de un runner propietario.

atomic

Transformación que produce una sección de un artefacto o un artefacto completo. La mayoría de prompts del pipeline.

validator

Verificación cross-artefacto. No produce contenido — produce Health Report. output: null, healthChecks: [...].

context

Compilación de Context Package. Cacheables por hash del input. Prerrequisito de cualquier skill de generación.

Cada prompt atómico expone un manifest YAML con Skill Contract: `skill-id`, `skill-version` (semver), `inputs` tipados, `outputs` con schema fijo `{output, healthChecks}`, `model-zone` y `prompt-hash` (SHA-256). El output schema fijo garantiza que cualquier orquestador puede consumir el resultado sin lógica custom.

Model routing por zona.

ZONA	MODELO RECOMENDADO	TEMPERATURA	USE CASES
Verde	Modelo rápido y barato	0.1	Extracciones, mapeos, derivaciones determinísticas
Ámbar	Modelo equilibrado	0.2	Síntesis, decisiones técnicas, validación cross-artefacto
Roja	Modelo potente	0.1	Razonamiento complejo, arquitectura, validators críticos

El `model-zone` declarado en el manifest se convierte en política del LLM Gateway corporativo. El Gateway decide qué modelo invocar según la zona — esta lógica era responsabilidad del runner en versiones anteriores; ahora es centralizada. Cualquier orquestador externo respeta la política sin replicar lógica.

Los **workflows** son composiciones declarativas de invocaciones de skills con `parallel-with` y `after` para orquestación. DAG estricto, referencias tipadas, barriers explícitos y política `on-health-blocker` declarativa. El workflow YAML es estándar — cualquier engine MCP compatible lo ejecuta.

El **LLM Gateway** es la capa más crítica de la plataforma. Punto único por el que pasan todas las llamadas a modelos de IA. Centraliza filtrado DLP, routing inteligente, auditoría completa (prompt hash, modelo, usuario, proyecto, timestamp que alimenta el AI-BOM), metering por equipo y proyecto, y política multi-proveedor para evitar vendor lock-in.

En entornos con datos sensibles, el Gateway se despliega en VPC o on-premise para soberanía de datos. Innegociable en empresas con compliance regulatorio europeo — EU AI Act, GDPR, sector financiero o sanitario.

07
Gobernanza *integrada.*

Los Quality Gates son condiciones de transición del state machine, no capas que se superponen al proceso. Cada artefacto tiene once estados canónicos posibles — solo `approved` e `implemented` permiten generar downstream.

F0 — Contexto documentado + platform-dependencies.md firmado

Sin contexto documentado y sin firma de Architect + PO, no se arranca el Capability brief. Bloqueo absoluto antes de invertir tiempo en Framing.

F2 — Artifact Health Report sin blockers + 5 firmas obligatorias

Blockers abiertos bloquean la firma del artefacto. Warnings sin firmar explícitamente escalan a blocker. Sin las cinco firmas de la Golden Spec, no hay stories. Sin stories, no hay código.

F3 — Spec Health Report cross-artefacto + gates deterministas automáticos

Lint, SAST, cobertura de tests. Build rojo → el agente itera o escala. Blockers cross-artefacto → F3 no arranca. El Invalidation Impact Report se firma si el commit toca artefactos approved con dependents.

F4 — Aprobación humana + AI-BOM completo

Sin aprobación humana con comprehension check, no se mergea. Sin AI-BOM por story y agregado por feature, no se cierra el feature. Ambos son innegociables.

F5 — Deploy Risk Report + canary health check

risk_level: critical bloquea el deploy hasta review humano. Violación de rollback conditions declaradas en .spec.md activa auto-rollback inmediato sin intervención manual.

08
Kanban-first y *métricas.*

La IA acelera la generación 5–10×. Los sprints fijos crean inventario shippable que no se mergea por falta de revisión. El cuello se desplaza al review humano, que es flow-limited, no time-limited. Por eso la planificación canónica es Kanban.

5–10×
ACELERACIÓN
en generación de código con IA respecto a autoría manual

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

WIP
POR ZONA
Verde: 5 gen / 3 review. Ámbar: 3/2. Rojo: 1/1.

2*h*
REVIEW MÁX.
Story mal dimensionada si su acceptance tarda más de 2h en validarse

El Story Board tiene swimlanes por zona. Cada zona tiene su propio WIP limit, calibrado por la capacidad de review humano correspondiente. Si una columna llega al WIP, no entra trabajo nuevo hasta que algo sale. Stories de distinta zona no se intercambian.

Las métricas de flujo core son cinco: **Lead time** (ready → merged), **Cycle time** (generating → merged), **Throughput** (stories/tiempo por zona), **WIP age** (tiempo en columna) y **Blocker rate**. Más dos específicas de AI-SDLC: **Health Report blocker rate** (% de stories que generan blocker) y **Comprehension debt index** (salud del entendimiento del equipo).

El roadmap de madurez tiene cuatro stages. Stage 1 (Semana 1): un equipo piloto puede hacer AI-SDLC con IDE agéntico + estructura canónica + constitución mínima. Stage 2 (Mes 1–3): múltiples equipos con LLM Gateway + DLP + AI-BOM automático. Stage 3 (Mes 3–9): toda la organización con delivery AI-augmented operativo. Stage 4 (Mes 9–18): plataforma autónoma con RAG, knowledge graph y Production Context con análisis predictivo.

Equipos que ya operan en Scrum pueden adaptar los sprints como timeboxes de replenishment, pero la planificación canónica es flow-based. El motivo es estructural: los sprints time-boxed se convierten en cuello de botella artificial cuando la generación se acelera 5–10×. No compensa forzar el flujo en cajas de tiempo diseñadas para velocidades de codificación manual.

Conclusión.

El nuevo ciclo de vida del software no empieza con un ticket y termina con un merge. Empieza con una intención estructurada — la spec — y termina cuando el sistema en producción retroalimenta la siguiente spec. El código que hay en medio es el subproducto de ese ciclo, no su protagonista.

Esto cambia lo que significa construir software. Cambia quién hace qué. Cambia cuándo las decisiones se toman y quién las toma. Cambia lo que un equipo debe entender de su propio sistema para poder operarlo. Cambia, en definitiva, qué tipo de organización tiene ventaja.

Hay una pregunta recurrente cuando presentamos este sistema por primera vez: ¿esto va a sustituir a nuestros ingenieros? La respuesta es la opuesta. Va a hacer que el ingeniero sea, por fin, ingeniero: alguien cuya autoridad se ejerce sobre la intención y el resultado, no sobre la sintaxis. Alguien cuyo conocimiento queda inscrito en artefactos legibles y versionados, no en commits que nadie recuerda haber escrito.

La v10 no es la respuesta final. Es la forma actual de una pregunta que no va a cerrar: cómo construir software cuando el código deja de ser el cuello de botella. Habrá v11, y v12, y reemplazos que aún no hemos imaginado. El ciclo seguirá acelerando. Lo que no cambia es el principio: **especifica con precisión, genera con agentes, valida con rigor, opera con trazabilidad, y cierra el ciclo.**

> *El código es el subproducto.*
> *La spec es el activo.*

thinkia

thinkia.com