La IA no escribe buen software

El entorno sí

Adrian Ferrera · CommitConf 2026

LeanMind
GOOD [CODE] PEOPLE

¿Estamos generando mejor software o simplemente más software?

El problema no es nuevo

Durante años hemos repetido los mismos errores:

- Correr antes de entender
- Construir antes de validar
- Dejar los tests para después
- Aceptar deuda técnica como inevitable
- Confundir velocidad con progreso

> *Ahora podemos hacerlo a una **velocidad vertiginosa**.*

Antes de la IA ya teníamos un problema

Coste de mala calidad del software (CISQ, 2022)	**$2.41 billones**
Deuda técnica acumulada	**$1.52 billones**

GitClear, 2025	2021 (pre-IA)	2024 (post-IA)	Variación
Código duplicado (copy/paste)	8.3%	12.3%	**+48%**
Líneas movidas (refactoring)	25%	<10%	**-60%**
Code churn (reescrito en <2 semanas)	baseline	x2	**+100%**

Antes de la IA ya teníamos un problema

La IA no aparece en una industria sana.

Aparece en una industria que ya arrastraba **problemas estructurales**.

La IA no elimina ese problema

Lo acelera.

La nueva deuda

> *La IA no crea deuda técnica nueva.*
> ***Acelera la deuda técnica que ya sabíamos crear.***

> *Hemos reducido el coste de producir código, pero no necesariamente el coste de* ***entenderlo, validarlo y mantenerlo****.*

Verification debt

La IA genera código con apariencia correcta.

Pero ese código **debe ser verificado**.

> *El nuevo cuello de botella no siempre será escribir código.*
> *Será saber si ese código es **correcto**.*

La tesis

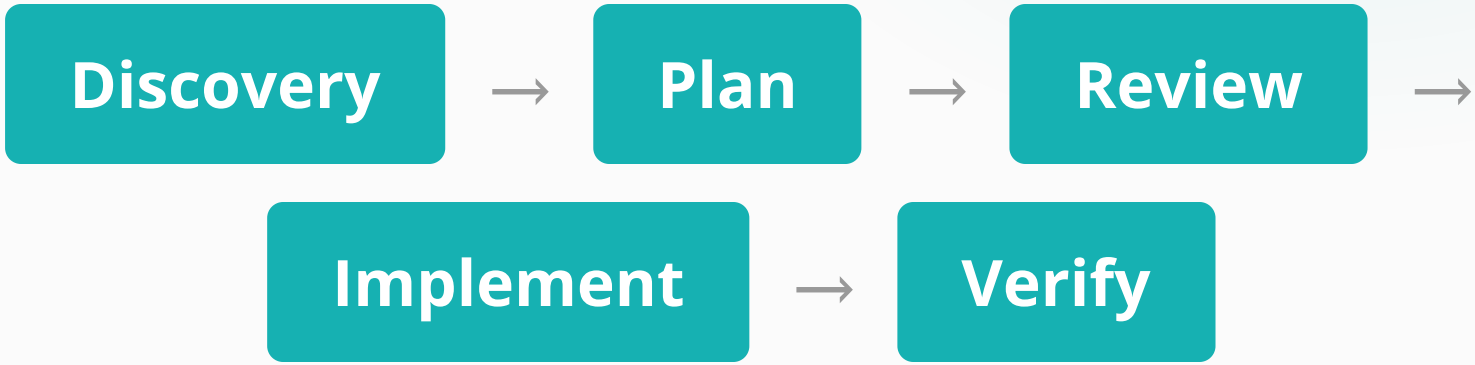
La IA no escribe buen software por sí sola.

El entorno sí.

Transición

> *Si la IA acelera nuestras malas prácticas, necesitamos una forma de introducir **criterio** antes, durante y después de la generación de código.*

El flujo



No es Scrum para agentes.
Es una **estructura mínima de control**.

Discovery

¿Estamos entendiendo bien el problema?

Antes de implementar, el sistema debe **pensar el problema** mediante un **debate socrático** con el humano:

- Explorar alternativas
- Identificar riesgos
- Explicitar supuestos
- Detectar ambigüedades
- Cuestionar la premisa inicial

Discovery — Debate socrático

explorer-agent.md

```
---
name: explorer-agent
role: Socratic exploration of the problem
model: claude-sonnet-4
visibility: hidden
---
## Instructions
- Challenge assumptions with questions
- Compare alternative approaches
- Surface risks and unknowns
- Ask "why" before accepting requirements
- Never propose solutions without debate
## Output
```

Plan

¿Sabemos qué vamos a cambiar y por qué?

Convertir el razonamiento en pasos concretos:

- Qué se va a cambiar
- Qué ficheros se verán afectados
- Qué NO se va a tocar
- Qué pruebas hacen falta
- Qué riesgos hay que controlar

El humano puede: aceptar, corregir o rechazar el plan antes de que se modifique nada.

Plan — Ejemplo de agente

planner-agent.md

```
---
name: planner-agent
role: Create implementation plan
model: claude-sonnet-4
input: exploration_report
visibility: hidden
---
## Instructions
- Define scope and boundaries
- List files to modify
- Identify tests needed
- Flag risks and dependencies
## Output
```

Review

¿Tiene sentido la solución antes de construirla?

Revisar **antes** de implementar:

- La intención y el enfoque
- Los trade-offs
- El impacto en el sistema
- La coherencia con la arquitectura
- La mantenibilidad

> *Evita aceptar soluciones plausibles pero **mal orientadas**.*

Review — Ejemplo de agente

reviewer-agent.md

```
---
name: reviewer-agent
role: Review plan before implementation
model: claude-sonnet-4
input: implementation_plan
visibility: hidden
---
## Instructions
- Check architectural coherence
- Evaluate trade-offs
- Assess maintainability
- Verify scope boundaries
## Output
```

Implement

¿Está ejecutando el plan o improvisando?

La IA implementa **después** de:

1. Discovery
2. Planificación
3. Revisión

Deja de ser una máquina de generar código.
Pasa a ser un **ejecutor dentro de un marco de decisión**.

Implement — Ejemplo de agente

implementer-agent.md

```
---
name: implementer-agent
role: Execute the approved plan
model: claude-haiku-4
input: [implementation_plan, review_verdict]
visibility: hidden
---
## Instructions
- Follow the plan strictly
- No changes outside scope
- Respect architectural decisions
- Write tests alongside code
## Output
```

Verify

¿Qué señales tenemos de que esto es correcto?

No significa solo que compile. Significa reunir **señales suficientes**:

- Tests automatizados
- Tipado
- Linters
- Análisis estático
- Integración continua
- Revisión del diff
- Validación funcional
- Reglas de arquitectura
- Contratos de API
- Observabilidad
- Seguridad
- Documentación

Verify — Ejemplo de agente

verifier-agent.md

```
---
name: verifier-agent
role: Validate implementation correctness
model: claude-sonnet-4
input: code_changes
visibility: hidden
---
## Checks
- Run test suite
- Check type safety
- Run linters and static analysis
- Verify against spec
## Output
```

El orquestador

orchestrator.md

```
---
name: orchestrator
role: Coordinate the development flow
model: claude-sonnet-4
visibility: primary
---
## Phases
- explorer (humanGate: true)
- planner (humanGate: true)
- reviewer (humanGate: true)
- implementer (humanGate: false)
- verifier (humanGate: true)
```

orchestrator.md (cont.)

```
## Harnesses
tests, types, linters,
CI pipeline, architecture rules
## Principle
Structure + Feedback + Judgment
## Rules
- Never skip humanGate phases
- Verify before merge
- No improvisation in implement
```

Human in the Loop

No es revisar al final.

Es **decidir en el momento adecuado**.

El humano aporta lo que la IA no tiene de forma fiable:

Intención · Contexto · Criterio · Experiencia · Responsabilidad

Arneses

> *Un agente sin arneses es **velocidad sin dirección**.*

Los arneses son mecanismos que permiten a la IA operar con límites:

- Tests, tipos, linters
- Análisis estático, reglas de arquitectura
- CI/CD pipelines
- Revisión de seguridad
- Contratos de API, feature flags
- Validación humana

La ecuación

El **orquestador** aporta estructura.
Los **arneses** aportan feedback.
El **humano** aporta criterio.

Flexibilidad

Contexto	Discovery	Plan	Review	Verify
Bug pequeño	30 seg	Simple	Rápida	1 test
Cambio de arquitectura	Sesión completa	Varios pasos	Varias personas	Tests + seguridad + rendimiento
Spike	Exploratorio	Flexible	Decidir si continuar	Aprendizaje

> *La gracia del flujo es que no impone un proceso.*
> *Impone una **pregunta**.*

El mensaje

No queremos que la IA simplemente vaya más rápido.

Queremos que vaya más rápido en la dirección correcta.

Conclusión

> *¿Estamos generando mejor software o simplemente más software?*

La respuesta no depende de la IA.

Depende de nosotros.

La IA no nos libera de la responsabilidad técnica. La hace **más importante**.

La IA no escribe buen software por sí sola.

El entorno sí.

Y ese entorno lo diseñamos **nosotros**.

Adrian Ferrera · adrianferrera.dev · leanmind.es

LeanMind
GOOD [CODE] PEOPLE