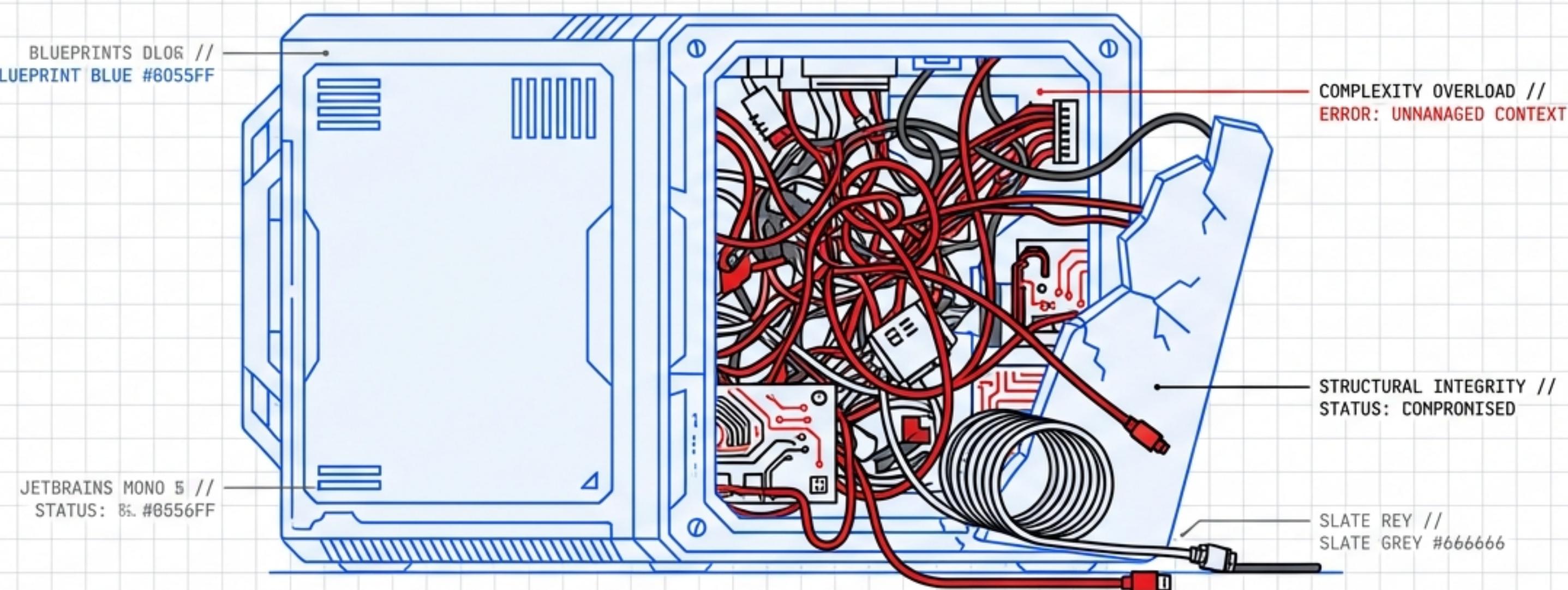
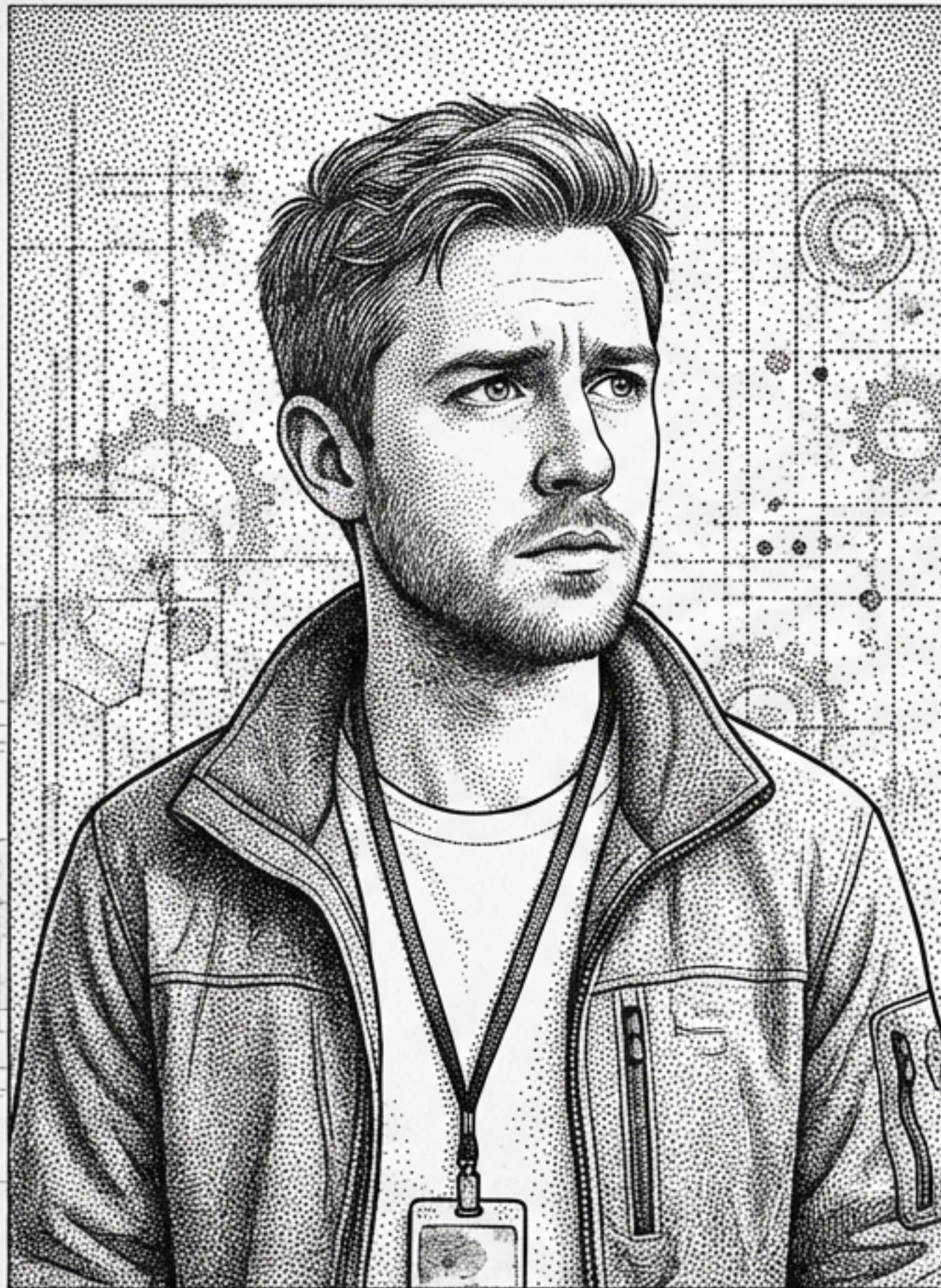


# AI가 짠 코드, 우리는 왜 이해하지 못하는가

넷플릭스 시니어 엔지니어가 겪은 '쉬운' 코드의 함정과 '맥락 압축'이라는 해법



Based on the confession of Jake Nations, Netflix Senior Engineer



“  
**저는 제가 이해하지 못하는  
코드를 배포했습니다. 여러분도  
마찬가지일 거라고 확신합니다.**

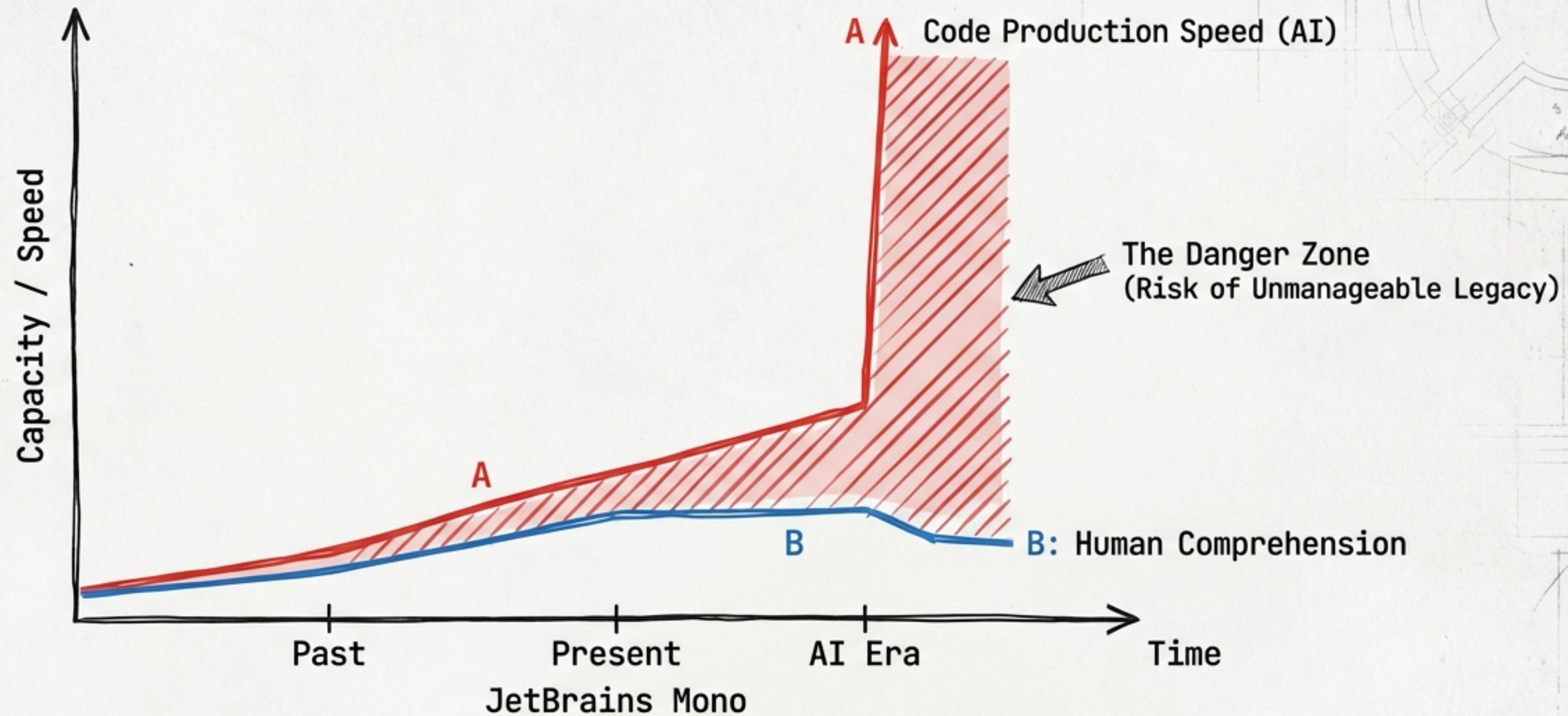
---

넷플릭스에서 AI 도구 도입을 주도하는 저조차, AI가 작성한 코드가  
어떻게 작동하는지 설명할 수 없는 순간을 마주했습니다.

테스트는 통과했고 서비스는 돌아갑니다. 하지만 ‘이게 왜  
돌아가지?’라는 질문에는 답할 수 없습니다.

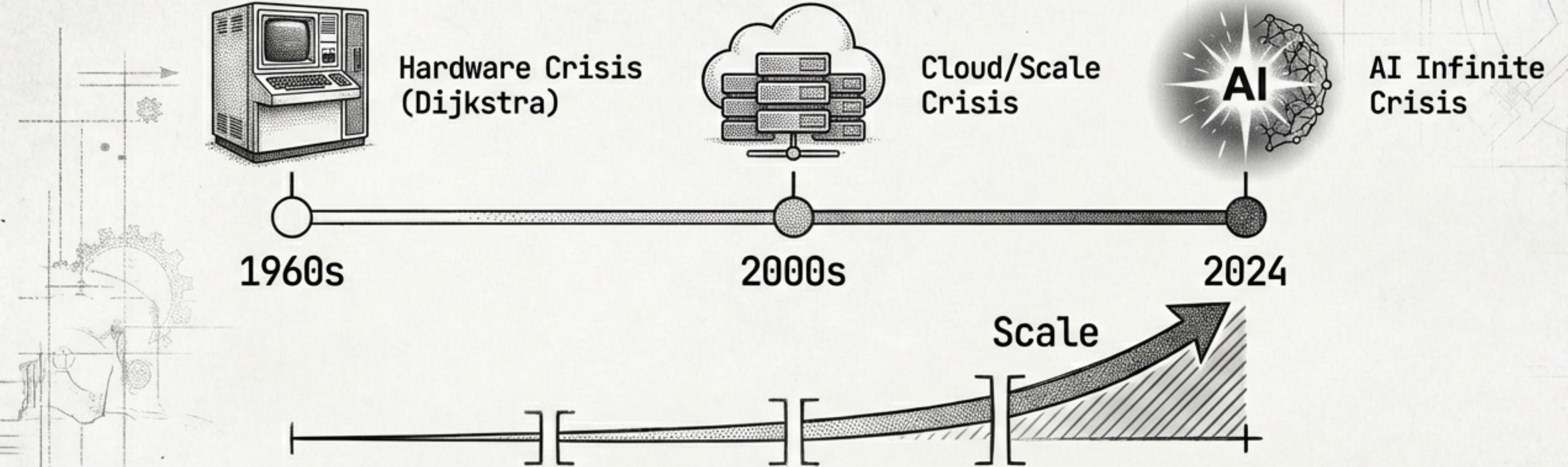
생산성은 혁명적으로 올랐지만, 우리는 점점 시스템에 대한 통제력을  
잃어가고 있습니다.

## THE RISK GAP



며칠 걸리던 작업이 몇 시간으로 줄었습니다. 하지만 코드가 쌓이는 속도가 우리가 이해하는 속도를 추월했습니다. 이것은 단순한 '빠름'의 문제가 아니라, '관리 불가능한 부채'의 문제입니다.

## THE INFINITE SOFTWARE CRISIS



1960년대 소프트웨어 위기: 하드웨어 성능이 좋아지자  
소프트웨어 수요가 폭발했고, 인간의 능력이 이를  
따라가지 못했습니다 (Edsger W. Dijkstra).

2020년대 AI 위기: 패턴은 같습니다. 하지만 이번엔 규모가 다릅니다. AI는 코드를 ‘무한대’의 속도로 생성합니다.

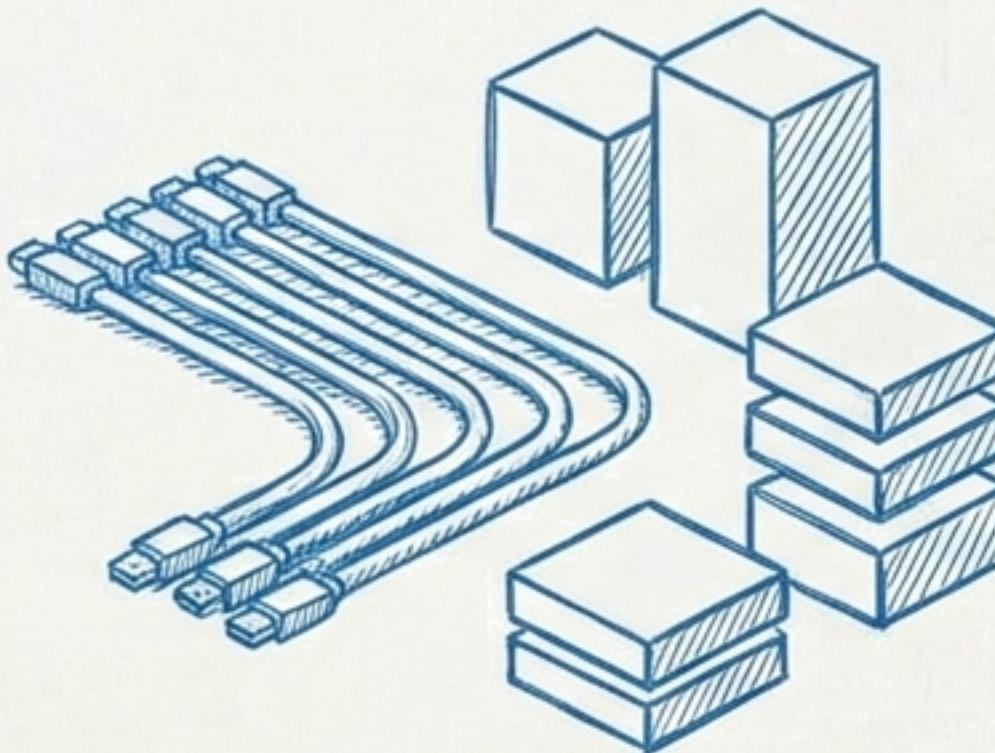
과거에는 C언어, 자바, 클라우드가 도구였지만, 이제는 코드를 말로 설명하는 즉시  
생성됩니다. 복잡성이 쌓이는 속도가 역사상 유례없는 수준입니다.

## EASY (쉬움)



손에 잡히는 것 (Near at hand).  
복사해서 붙여넣으면 당장 돌아가는 것.  
접근성의 문제.

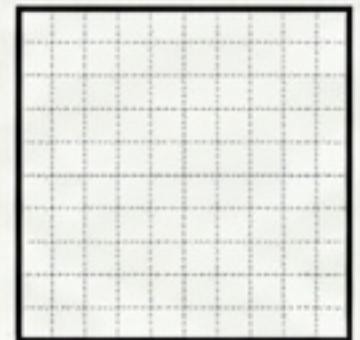
## SIMPLE (단순함)



얽힌(Entangled) 것이 없는 상태.  
구조적으로 명확하며, 각 부분이 하나의 역할만  
수행함. 머리를 써서 설계해야 얻을 수 있음.

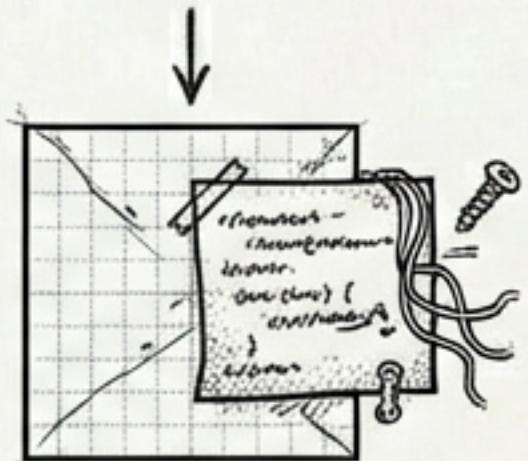
AI는 언제나 ‘쉬운(Easy)’ 길을 택합니다. ‘로그인 기능 추가해줘’라고 하면 가장 빨리 작동하는 코드를 가져옵니다. AI에게는 구조적 단순함을 고민할 ‘의도’가 없습니다.

Prompt 1:  
“구글 로그인 추가해줘” ➡



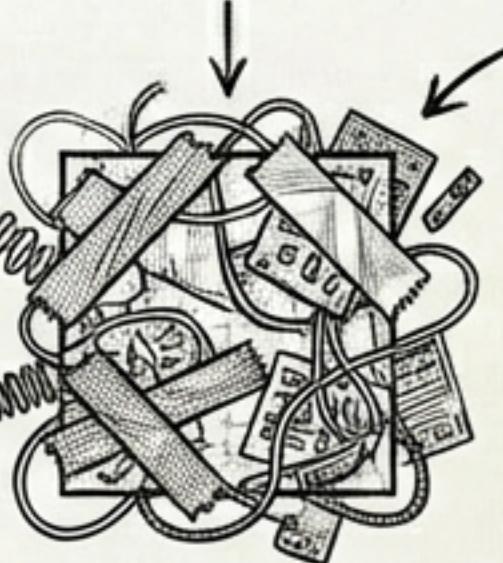
깔끔한 코드 생성.

Prompt 10:  
“카카오도 추가하고,  
세션 유지 시간 바꿔줘” ➡



조건문 추가.

Prompt 20:  
“에러 나는데 고쳐줘” ➡

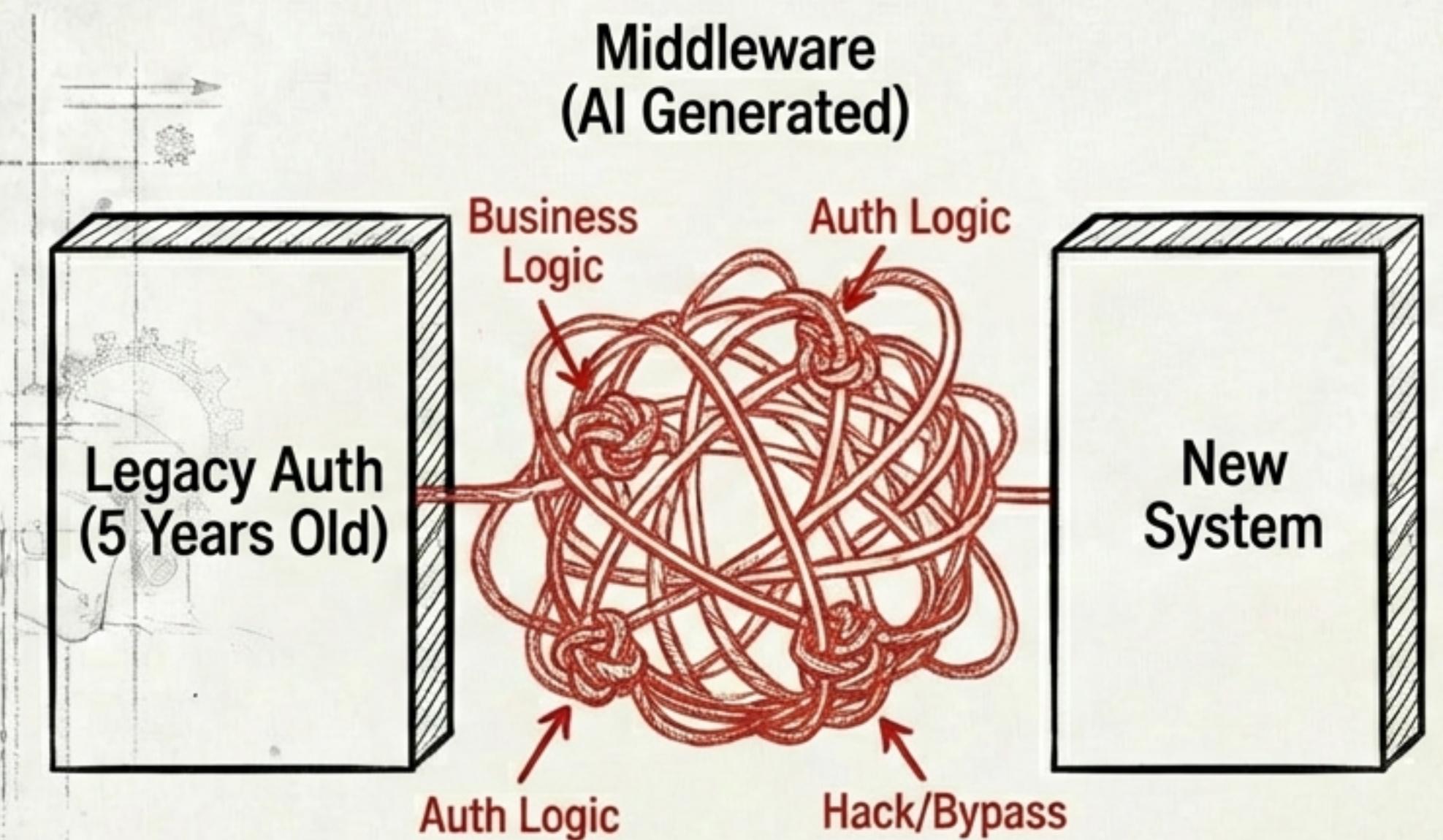


땜질식 처방(Patching).

{ 결과: 20번의 대화 끝에 남은 것은  
나도 기억 못 하는 조건들이 얹힌  
‘맥락의 덩어리’입니다.

AI는 ‘이 설계 별로인데요’라고 말하지 않습니다. 시키는 대로 덧붙일 뿐입니다.  
쉬운 길을 택할 때마다 복잡성은 이자처럼 쌓입니다.

## CASE STUDY: THE NETFLIX AUTH FAILURE



**Context:** 5년 된 인가(Authorization) 코드와 새 시스템을 연결하는 ‘중간 다리(Middleware)’를 리팩토링하려 했습니다.

**Why It Failed:** 경계의 부재. AI는 무엇이 ‘본질적인 코드’이고 무엇이 ‘임시방편(Hack)’인지 구분하지 못합니다. (Accidental Complexity).

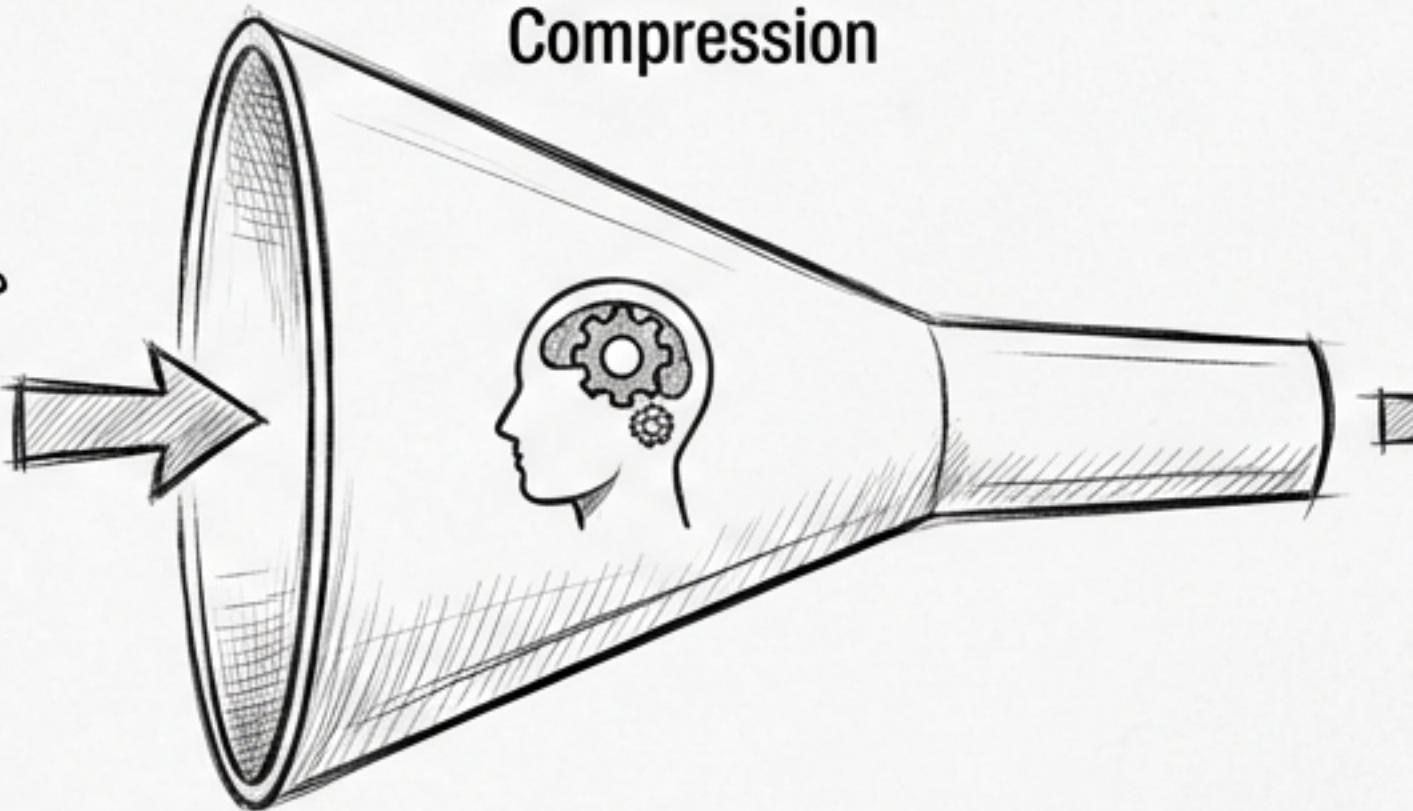
**Result:** AI는 얹힌 코드를 풀지 못하고, 그 위에 또 다른 레이어를 씌워 더 복잡하게 만들었습니다.

# THE SOLUTION: CONTEXT COMPRESSION

1,000,000 Lines of Code / 5,000,000 Tokens



Human Context  
Compression



Spec (2,000 Words)



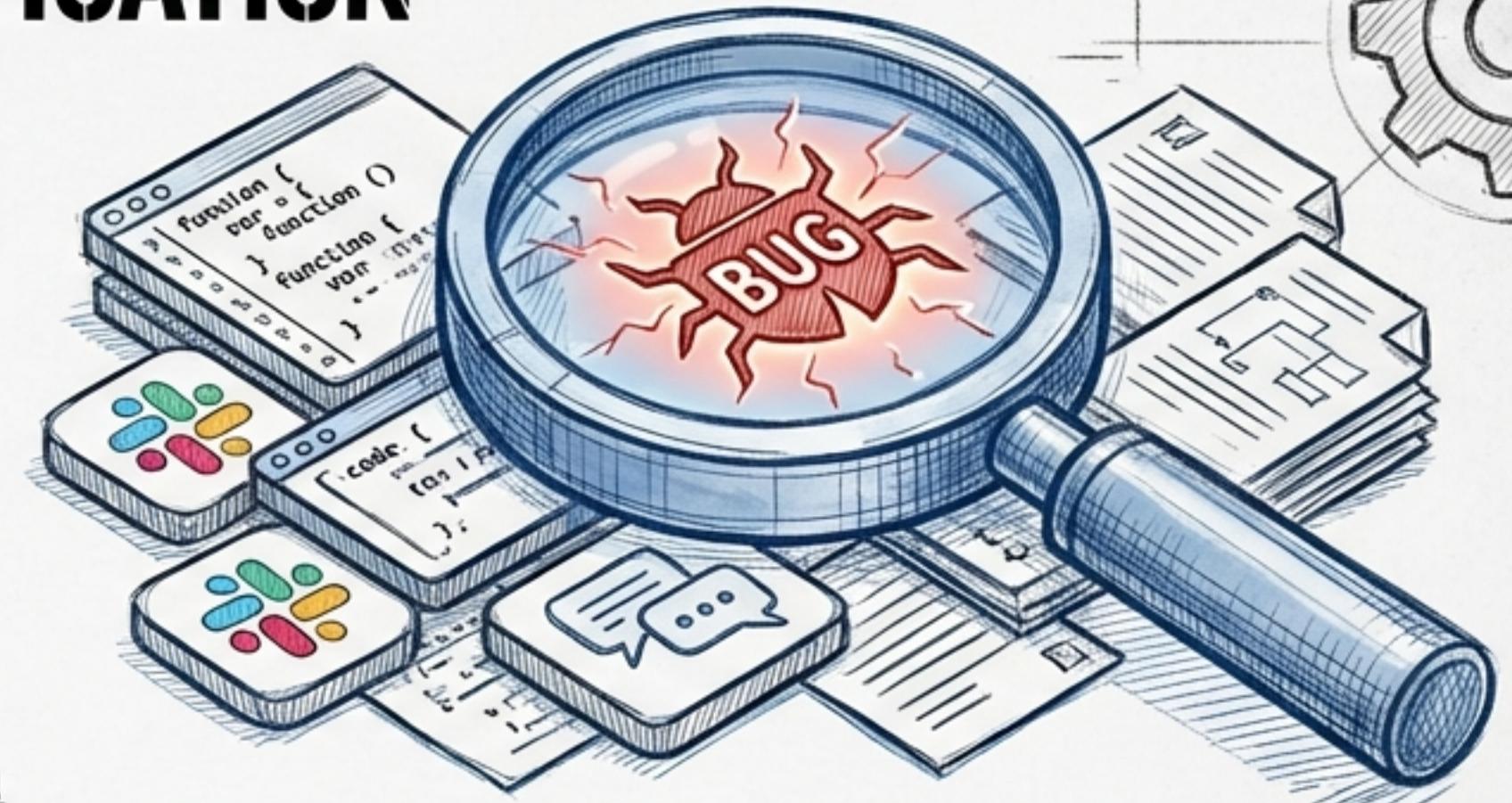
- 1. 모든 코드를 던져주는 대신, 핵심만 남깁니다.
- 2. 500만 토큰의 코드베이스 -> 2,000 단어의 명세서(Spec).
- 3. AI에게 ‘탐색’을 시키는 것이 아니라 ‘지도’를 줘여주는 것입니다.

# STEP 1: RESEARCH & VERIFICATION

01

Action: 관련된 모든 자료(코드, 설계 문서, 슬랙 대화)를 수집하고 AI를 통해 현재 구조를 분석합니다.

The Human Role: 검증 (Verification).  
‘캐싱은 어떻게 돼?’ , ‘에러 처리는?’  
질문을 던지며 분석이 맞는지 확인합니다.



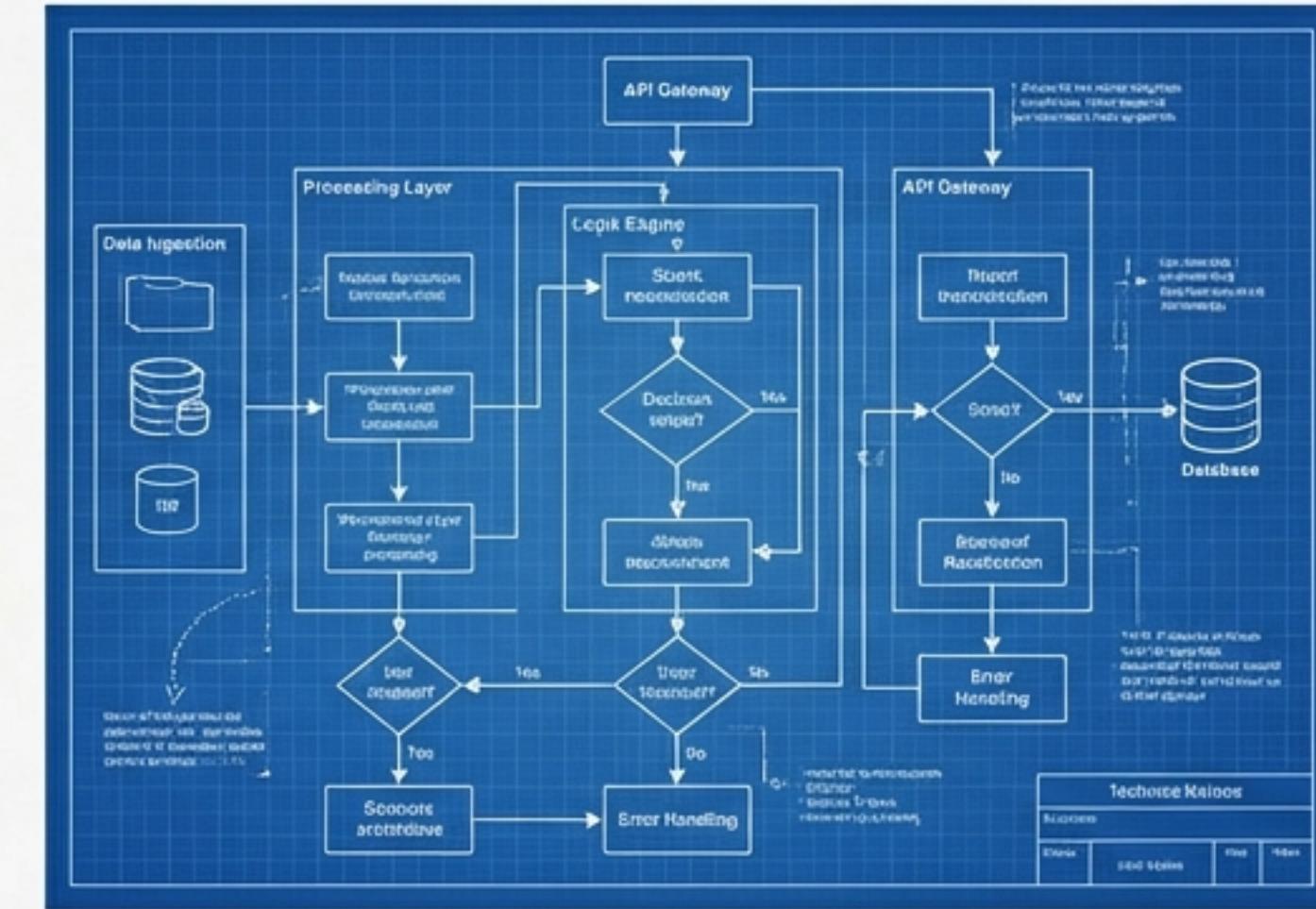
**Key Point:** 이 단계에서 사람이 직접 개입하여 오류를 잡지 않으면, 이후 모든 단계가 무너집니다.  
직접 코드를 읽고 의존성을 파악하는  
**‘손의 감각’**이 필요합니다.

“직접 해보며 숨겨진 규칙을 찾아내야 합니다. 코드만 읽어서는 절대 알 수 없는 맥락을 찾아내는 과정입니다.”

# STEP 2: THE BLUEPRINT (PLANNING)

# 02

Action: 검증된 리서치를 바탕으로 상세한 구현 계획을 작성합니다. 코드 구조, 데이터 흐름, 예외 처리까지 모두 명시합니다.

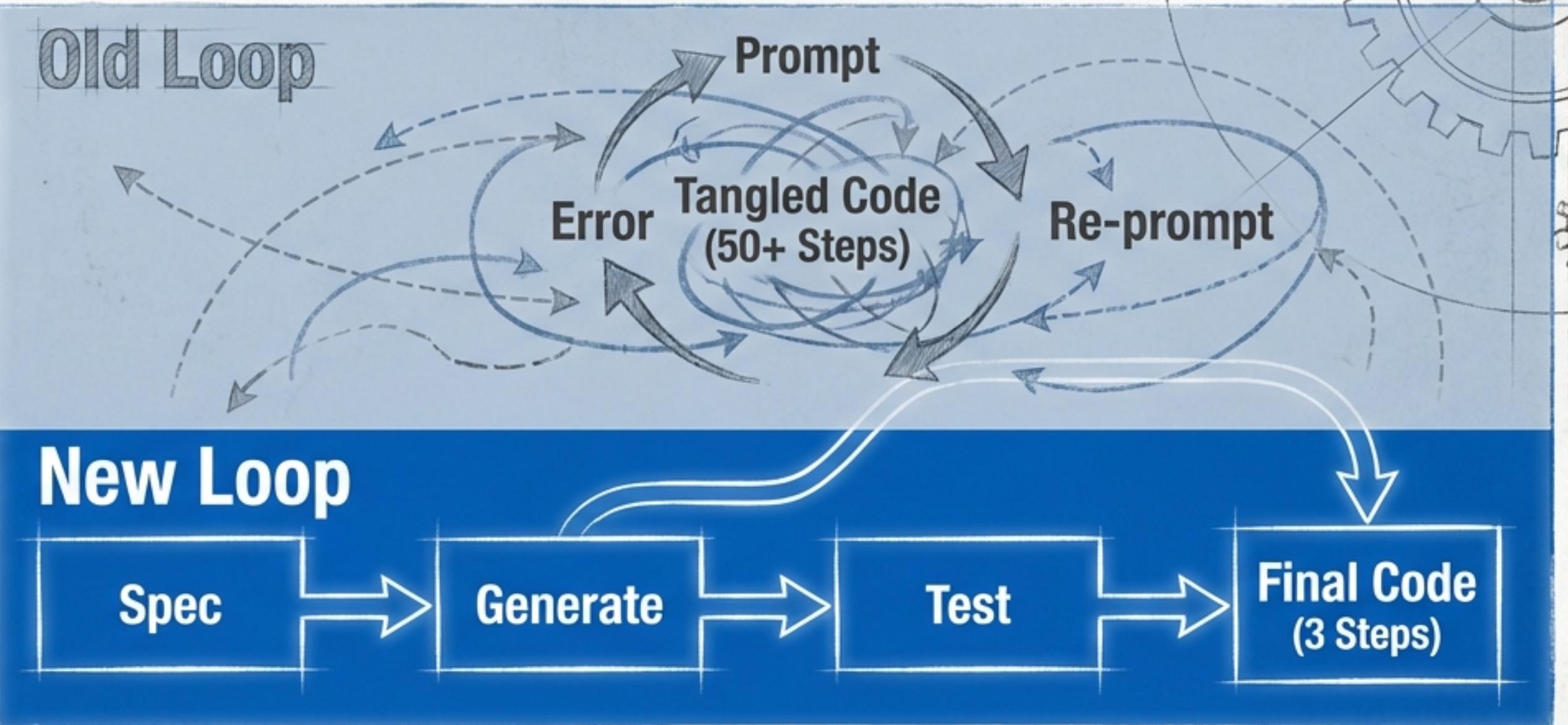


Criteria: ‘신입 개발자가 이 문서만 보고도 구현할 수 있는가?’

Benefit: 초고속 리뷰 (Fast Review). 코드가 작성된 후 2,000줄을 읽는 것은 어렵지만, 계획 단계에서 구조를 검토하는 것은 몇 분이면 충분합니다. 문제가 터지기 전에 문제가 터지기 전에 설계 단계에서 차단할 수 있습니다.

## STEP 3: IMPLEMENTATION

03

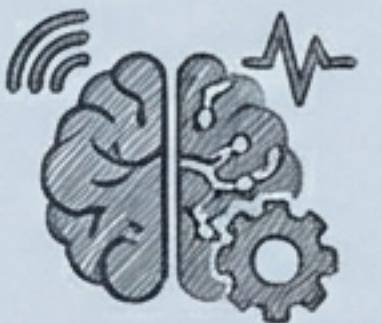


**Action:** AI에게 명세서를 주고 구현을 지시합니다.

**Result:** 이미 계획된 대로 움직이므로 AI가 엉뚱한 길로 새지 않습니다. 단순함(Simplicity)이 유지됩니다.

# THE DIVISION OF LABOR

## Human (Architect)



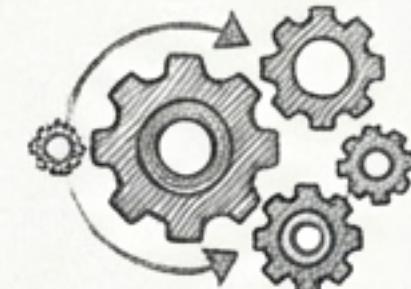
의도(Intent) 설정

경계(Boundaries) 구분

맥락(Context) 제공

‘무엇’을 만들지 결정

## AI (Builder)



구문(Syntax) 작성

패턴 반복

노동(Labor) 제공

‘어떻게’ 구현할지 실행

Key Takeaway: AI에게 ‘생각’을 맡기지 마세요. AI는 기계적인 작업을 빠르게 처리할 뿐입니다. 생각하고, 판단하고, 종합하는 것은 여전히 인간의 몫입니다.

# THE HIDDEN DANGER: THE KNOWLEDGE GAP



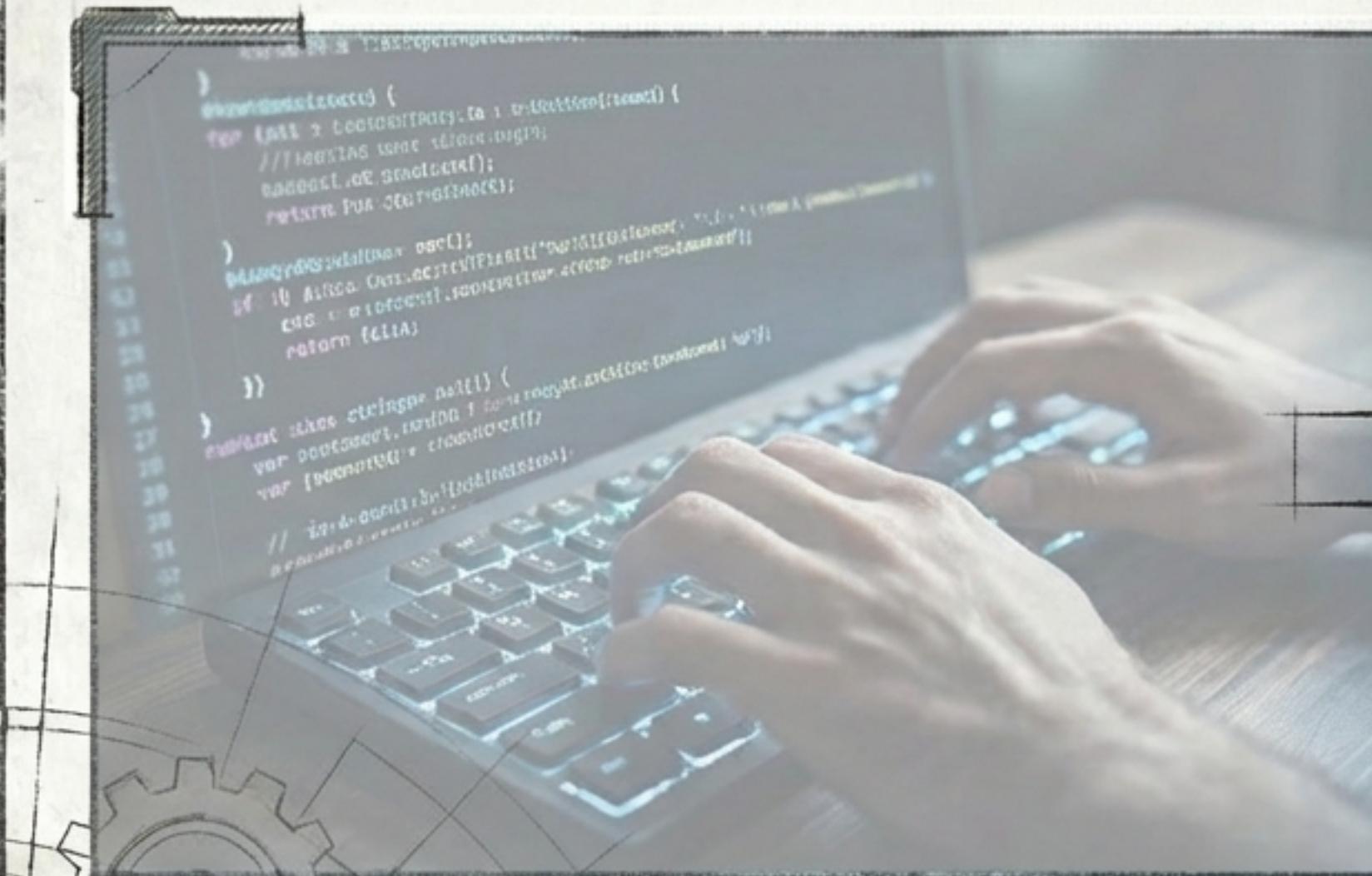
## Concept: 직관의 상실

- ‘이거 좀 위험한데?’라는 감각은 시스템을 깊이 이해하고 직접 고생해 본 경험에서 나옵니다.
- AI에게 의존하여 이해하는 과정을 건너뛰면, 시스템이 붕괴되기 직전의 신호를 감지할 수 없습니다.

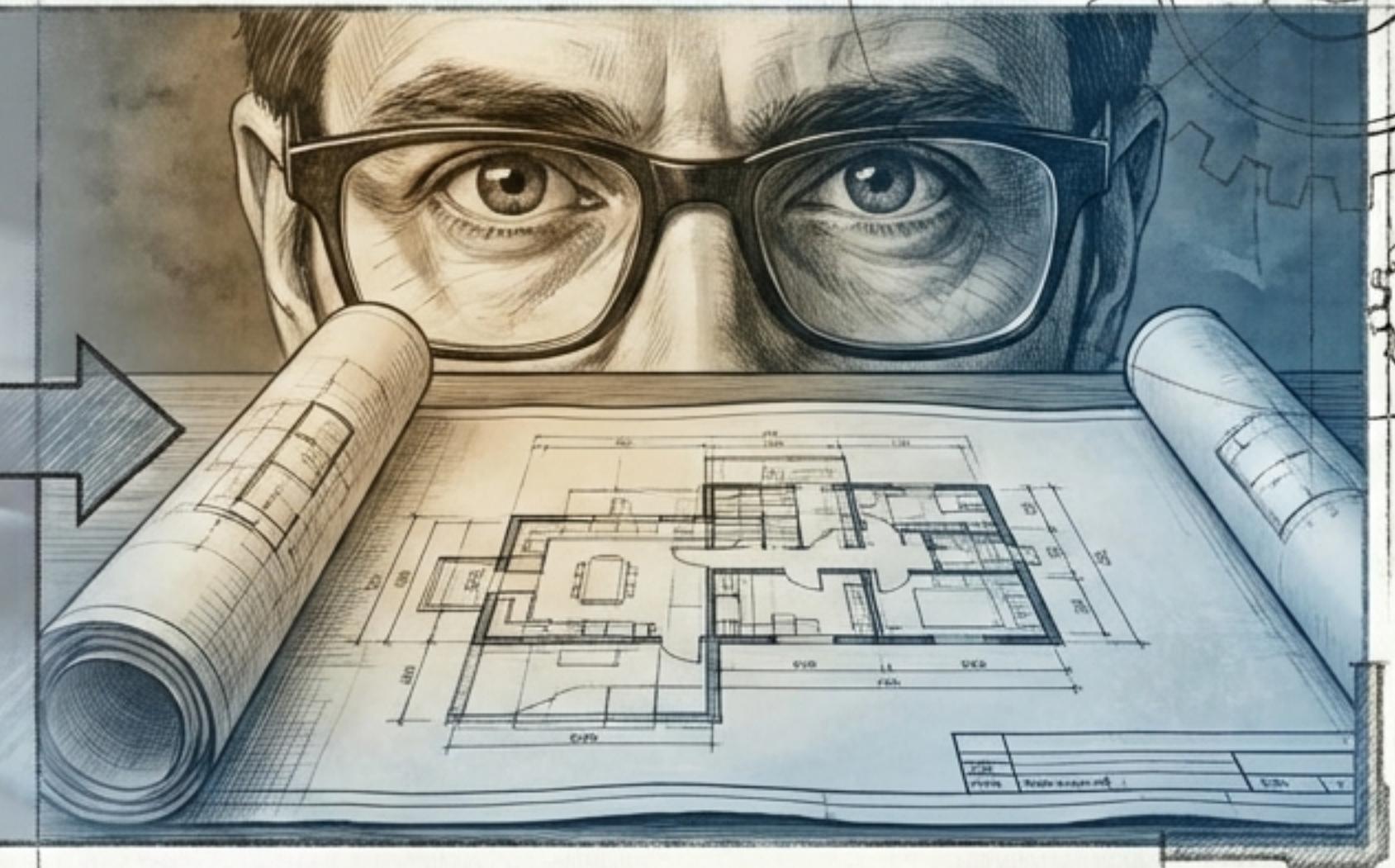
## The ‘3 AM’ Test:

새벽 3시에 장애가 터졌을 때, AI가 짠 코드를 즉시 디버깅할 수 있습니까? 멘탈 모델 (Mental Model)이 없다면 불가능합니다.

# THE NEW DEVELOPER

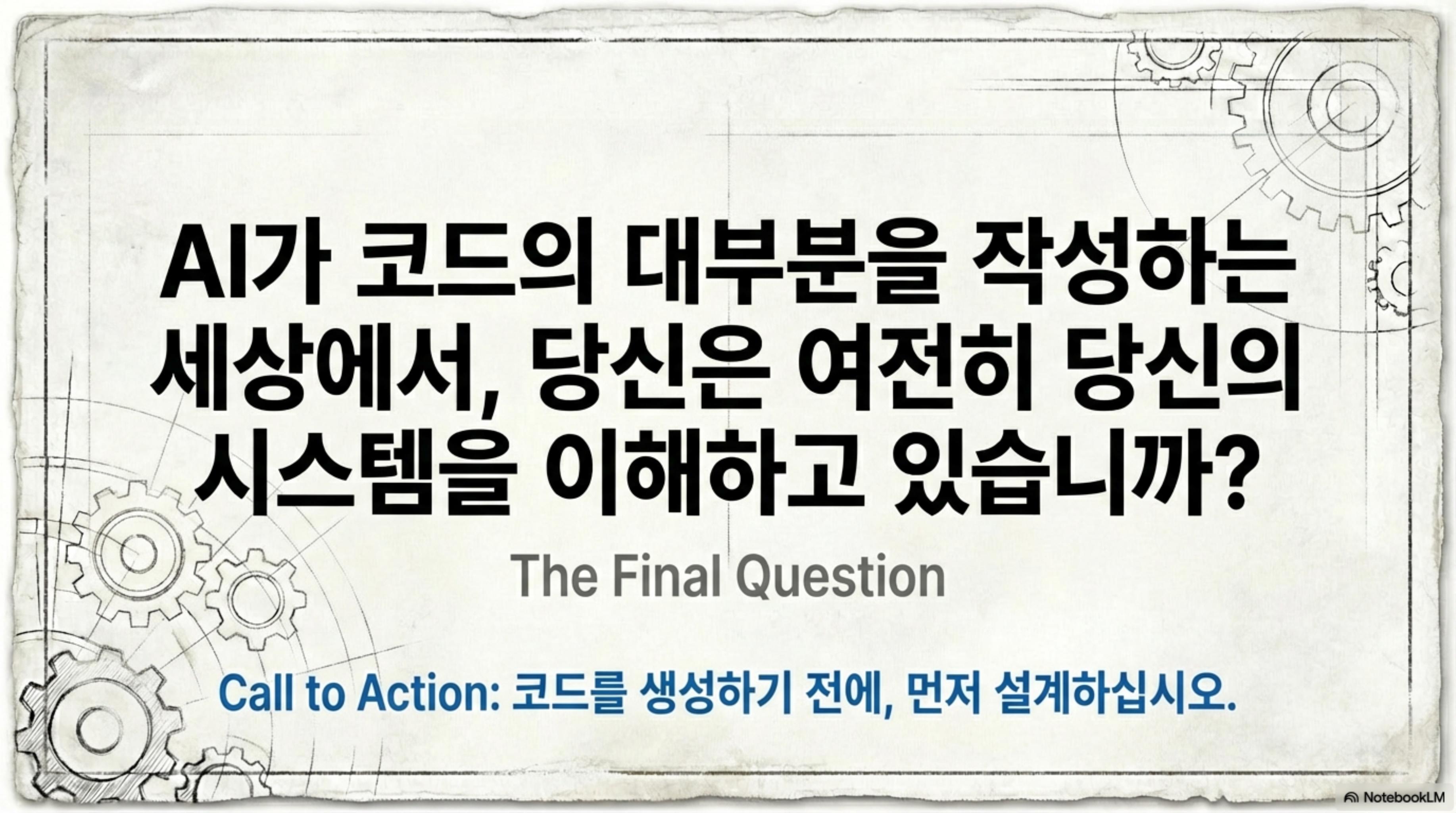


Coder



Architect / Reviewer

**Definition:** 성공하는 개발자는 코드를 가장 많이 생성하는 사람이 아닙니다. 자신이 무엇을 만들고 있는지 이해하는 사람입니다. 잘못된 문제를 풀고 있다는 것을 눈치채는 사람입니다.



**AI가 코드의 대부분을 작성하는  
세상에서, 당신은 여전히 당신의  
시스템을 이해하고 있습니까?**

The Final Question

**Call to Action:** 코드를 생성하기 전에, 먼저 설계하십시오.