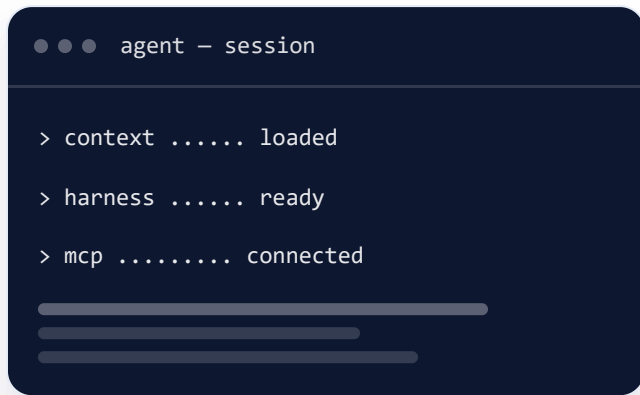


코딩 에이전트를 더 잘 쓰는 법

Context와 Harness로 이해하는 AI 개발 환경



Context Engineering

Harness Engineering

MCP

요즘 많이 들리는 단어들

"요즘 왜 이렇게 많이 나오지?"

"Prompt Engineering이랑
뭐가 다르지?"

Context Engineering

Harness Engineering

MCP

"MCP는 또 뭐야?"

"Claude Code 그냥
쓰면 되는 거 아닌가?"

들어보기는 했지만, 정확히 설명하기는 어려운 단어들이입니다.

"그런데 Claude Code, 그냥 써도 이미 잘하지 않나요?"

> 간단한 **Todo** 웹 앱 만들어줘.

- Create src/App.tsx
- Create src/ToDoList.tsx
- Create src/store.ts
- Update index.html

✓ 구현 완료 • dev server running

Todo

- ☒ 디자인 검토
- ☐ API 연동
- ☐ 배포 준비

그럼 **Context Engineering**과 **Harness Engineering**은 왜 필요한 걸까?

오늘의 질문

**같은 AI를 사용하는데,
왜 누구는 더 안정적이고
복잡한 일을 맡길 수 있을까?**

오늘은 그 차이를 '모델 밖'에서 찾아본다.

이미 당신도 Context Engineering을 하고 있었다



AI가 판단하기 전에 필요한 정보를 준비하는 행동

Prompt보다 넓은 것

PROMPT

"로그인 기능을
구현해줘."



Agent

CONTEXT

Prompt

Code

CLAUDE.md

Architecture

Error Log

Previous Decisions

Tool Result

Conversation
History

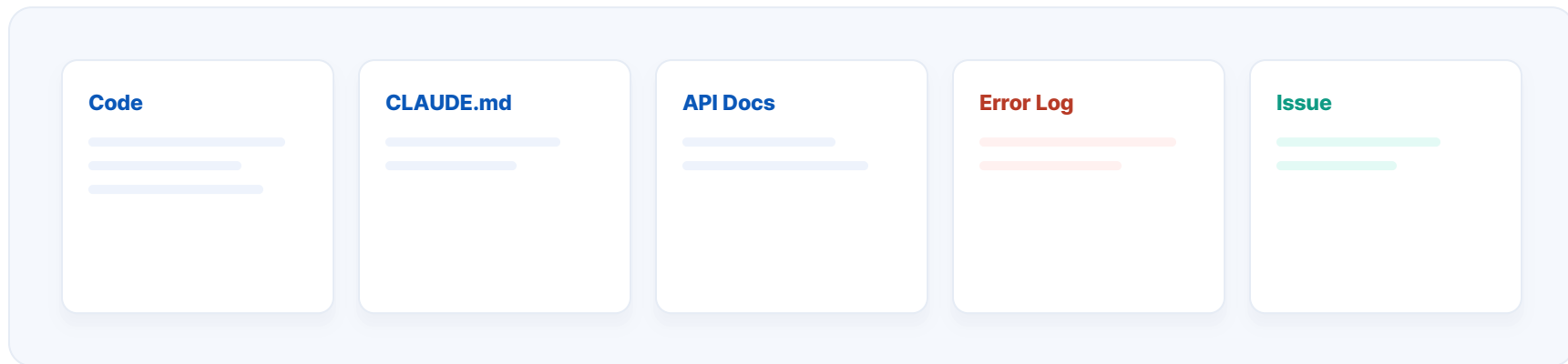


Agent

Prompt는 Context의 일부다.

Context Engineering = AI가 판단할 때 무엇을 알게 할 것인가를 설계하는 것

Context Window = AI가 지금 펼쳐볼 수 있는 책상



책상의 크기와 책상 위의 자료 배치는 다른 문제다.

"그럼 많이 넣으면 더 잘할까?"

문서 1	문서 2	문서 3	문서 4	문서 5
문서 6	긴 CLAUDE.md (2,400줄)		Skill A	Skill B
Skill C	Skill D	각종 로그	오래된 요구사항	
관련 없는 문서		Tool x24	Tool x36	장애 기록
회의록				



Context Engineering ≠ Context Stuffing

Right Context at the Right Time

Relevant

Minimal

Timely

Structured

BAD 프로젝트 문서 전체를 처음부터 모두 읽힘

BETTER 지금 필요한 **Architecture** 문서만 읽음

BAD 과거 모든 로그를 Context에 유지

BETTER 현재 문제와 관련된 결정만 유지

CLAUDE.md도 모든 것을 담는 파일이 아니다

BAD 거대한 CLAUDE.md

Architecture 전체 설명

DB 전체 Schema

API 전체 설명

배포 방법

장애 기록

Coding Convention

프로젝트 역사

~~~~~  
~~~~~  
~~~~~  
~~~~~  
~~~~~  
~~~~~

수천 줄

BETTER CLAUDE.md = 안내판

Architecture → <docs/architecture.md>

Testing → <docs/testing.md>

API → <docs/api.md>

Deployment → <docs/deployment.md>

항상 필요한 것은 짧게, 상세 정보는 필요할 때 찾아가게 한다.

좋습니다. 이제 Agent는 프로젝트를 충분히 이해했습니다.

✓ Architecture 이해

✓ 개발 규칙 이해

✓ 관련 코드 확인

✓ 요구사항 이해

"아는 것과 잘 일하는 것은 같은 문제일까?"

✗ 테스트를 하지 않는다

✗ 계획 없이 바로 수정한다

✗ 기존 코드를 깨뜨린다

✗ 완료 기준 없이 "완료"라고 말한다

이제 질문이 바뀐다.

무엇을 알게 할 것인가 → 어떻게 일하게 할 것인가

Harness Engineering

AI가 어떤 방식으로 일하도록 만들 것인가를 설계하는 것



좋은 행동을 기대하는 것이 아니라
좋은 행동이 반복되도록 환경을 만든다.

프롬프트 하나와 작업 시스템의 차이

PROMPT

"검색 기능 만들고
테스트도 해줘."



Agent

이번에 무엇을 할지 요청

HARNESS

- 1 관련 코드 탐색
- 2 구현 계획 작성
- 3 테스트 작성
- 4 구현
- 5 테스트 실행
- 6 실패 시 수정
- 7 최종 검증
- 8 완료 보고

일을 어떤 방식으로 수행할지 설계

Claude Code = Claude 모델일까?

Claude Model



AGENT HARNESS

File Read / Write

Shell

Tool Calling

Permissions

Context Management

Agent Loop



Claude Code

같은 구조

GPT Model

+

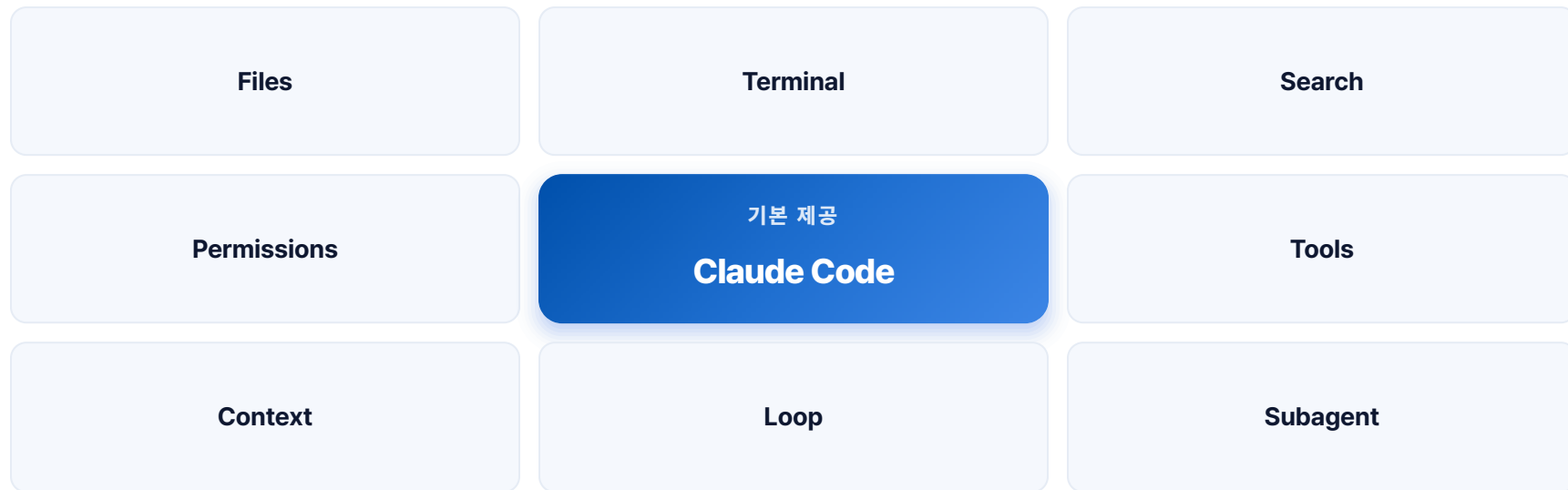
Harness



Codex

Coding Agent는 모델 그 자체가 아니라 모델 + 실행 환경이다.

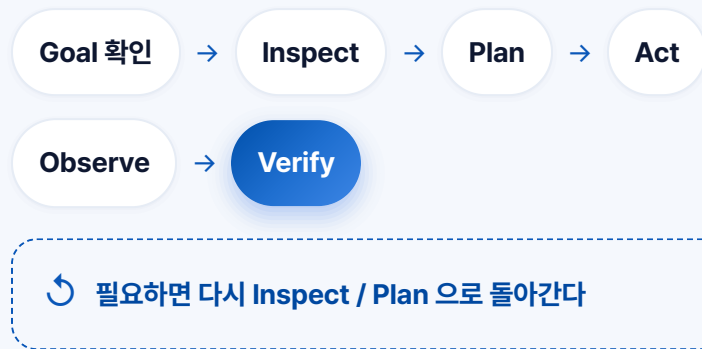
사실 여러분은 이미 Harness를 사용하고 있다



우리가 오늘 배우는 것은 Harness를 0부터 만드는 법이 아니다.

이미 존재하는 Harness를 어떻게 조정하고 확장할 것인가

Agent Loop



실제 예시

Issue 확인

→ 코드 탐색

→ 수정

→ 테스트

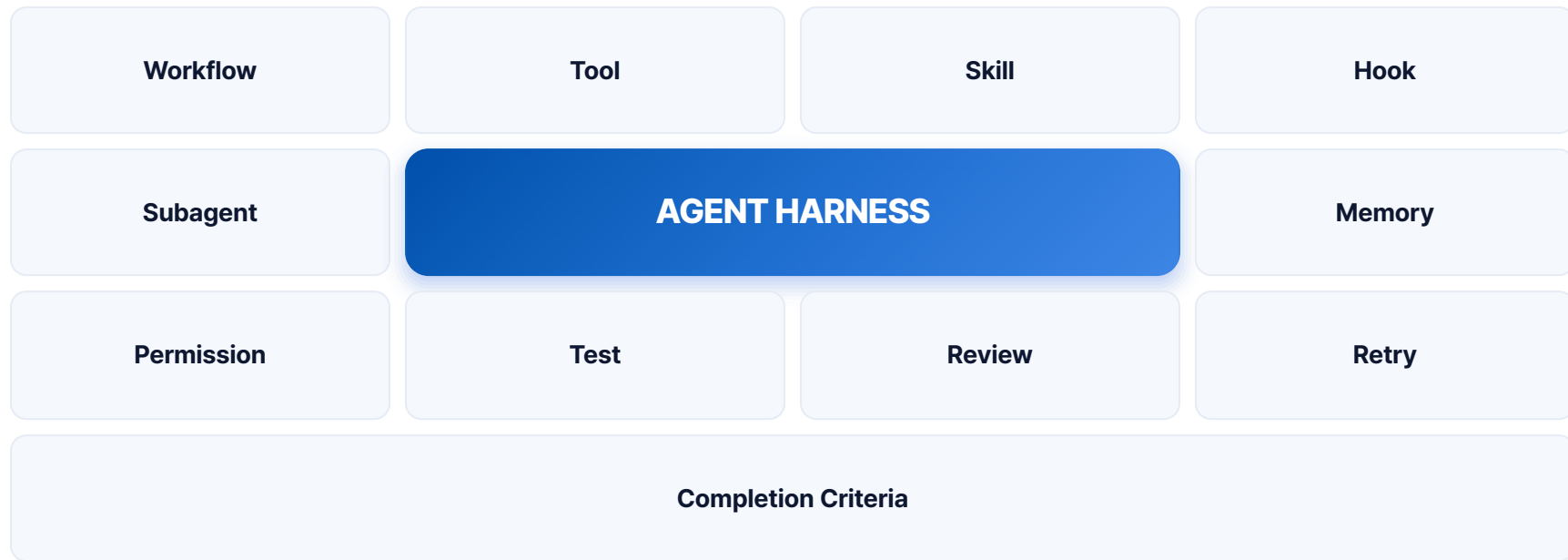
→ 실패

→ 수정

→ 재테스트 ✓

Harness는 이 Loop의 행동을 설계한다.

Harness에서 무엇을 Engineering할까?



Harness Engineering은 하나의 기능이 아니라 **Agent**의 작업 환경 전체를 설계하는 관점이다.

Skill

특정 상황에서 Agent가 따라야 할 재사용 가능한 작업 방식

매번 입력하는 긴 프롬프트

"테스트 먼저 만들고 실패 확인하고 구현하고 다시 테스트 돌려서 통과하면 리팩터링까지 해줘..."

대화마다 다시 설명해야 한다

TDD SKILL

- 1 실패하는 Test 작성
- 2 실패 확인
- 3 최소 구현
- 4 Test 성공
- 5 Refactor

한 번 정의하면 계속 같은 방식으로 일한다

반복해서 말하던 작업 방식을 재사용 가능한 지침으로 만든다.

Hook

특정 순간에 자동으로 개입한다



EVENT 01

파일 수정



Formatter 실행

EVENT 02

Tool 실행 후



결과 검사

EVENT 03

작업 종료 전



Verification 실행

Skill = 어떻게 일할지 알려준다

Hook = 특정 이벤트에 자동으로 반응한다

"AI 여러 개를 쓰면 무조건 더 좋은 걸까?"

Main Agent



Planner

요구사항과 계획

Developer

구현

Reviewer

결과 검토

아니다. 핵심은 역할과 **Context**를 분리하는 것이다.

CONTEXT ISOLATION / SPECIALIZATION

Memory와 Permission도 Harness다

Memory

이전 결정

작업 상태

반복되는 프로젝트 규칙

다음 세션에서 필요한 정보

무엇을 기억하게 할지

Permission

File Read

File Write

Shell

External Tool

Destructive Action

Human Approval

무엇을 허용할지

무엇을 기억하게 할지와 무엇을 허용할지도 Agent 행동을 결정한다.

왜 지금 Harness가 중요해졌을까?

과거

"AI가 코드를 제대로 만들 수 있을까?"

Better Model

Better Prompt

BOTTLENECK SHIFT



지금

Agent가 상당한 구현을 스스로 수행한다. 새로운 문제는 이것들이다.

어떤 Context를 제공할까?

어떤 Tool을 줄까?

어떤 Workflow를 따르게 할까?

어떻게 Test / Verify할까?

실패하면 어떻게 복구할까?

모델이 좋아질수록 모델 밖의 Engineering이 중요해졌다.

그런데 Agent가 필요한 정보가 로컬 Repository 밖에 있다면?

"Agent가 Repository 밖의 정보를 사용하려면?"



MCP

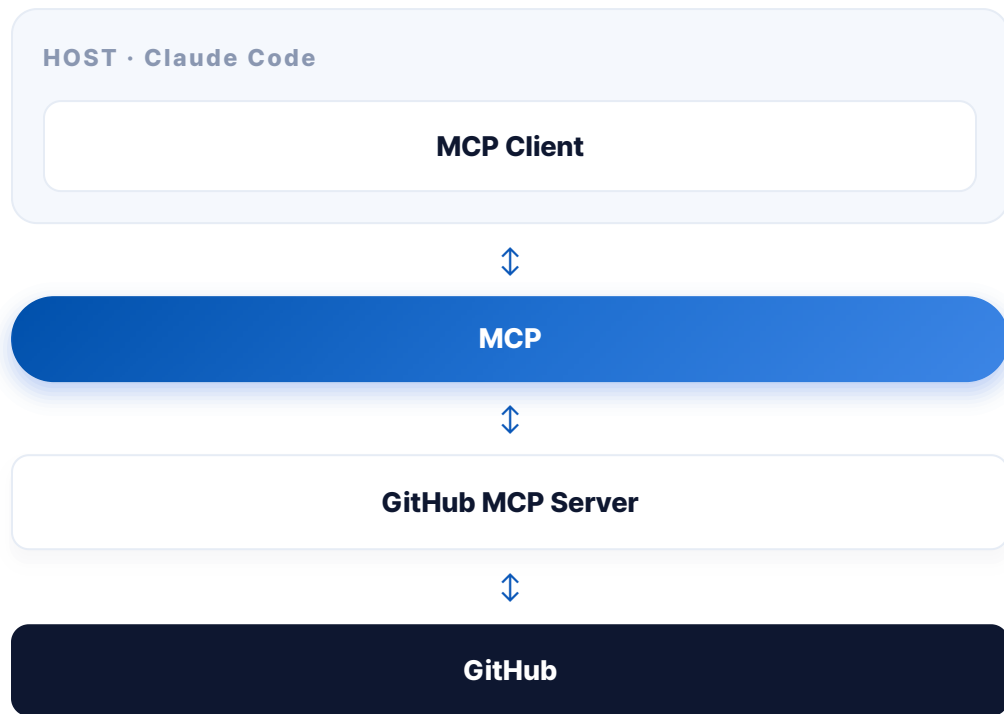
Model Context Protocol

AI Application과 외부 Data / Tool을 연결하기 위한 표준 인터페이스



서비스마다 제각각 연결하는 대신 공통된 방식으로 Tool과 Context를 노출한다.

MCP Architecture



Host

Agent Application

Client

MCP 연결 담당

Server

Tool / Context 제공

External System

실제 서비스

MCP가 아닌 것

MCP = Protocol

MCP ≠ Agent

MCP ≠ RAG

MCP ≠ Skill

MCP ≠ Plugin

MCP ≠ Harness

MCP는 Agent가 외부 능력을 사용할 수 있게 하는 연결 계층이다.

MCP가 제공하는 세 가지 기본 개념

01

Tools

Agent가 실행할 수 있는 기능

Issue 조회 / PR 생성

02

Resources

Agent에게 제공할 수 있는 정보

문서 / Repository Data

03

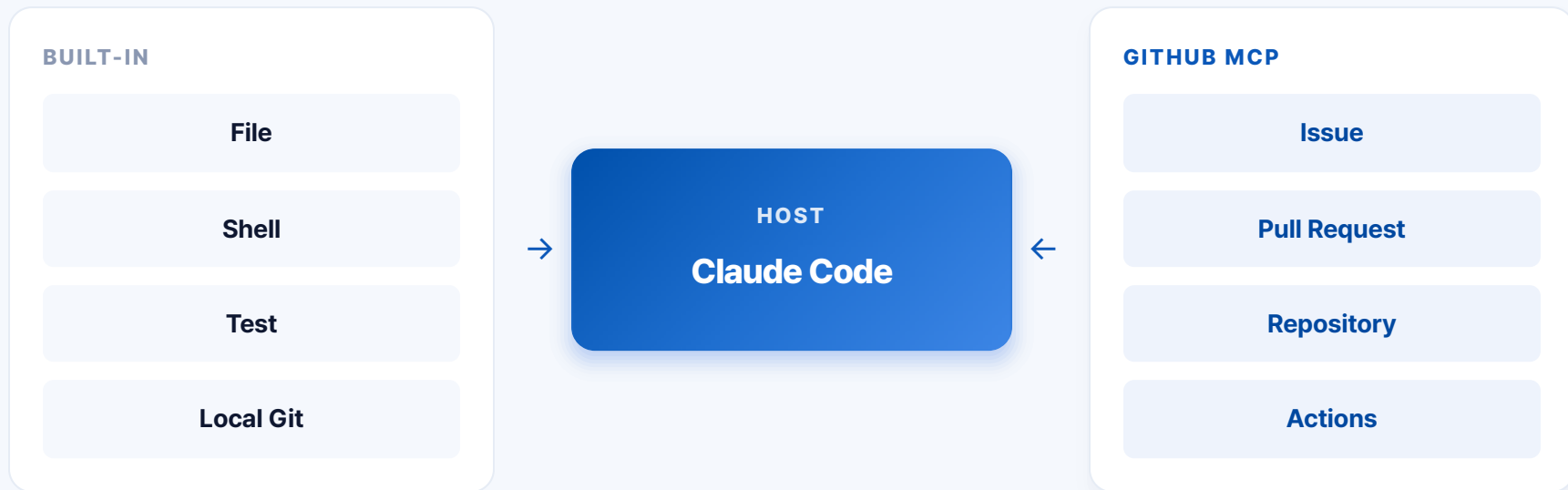
Prompts

재사용 가능한 Prompt Template

review-pr / write-issue

이번 실습에서는 특히 Tools를 중심으로 사용한다.

GitHub MCP를 붙이면



MCP는 기존 Tool을 대체하는 것이 아니라 Agent가 접근할 수 있는 세계를 확장한다.

"Tool이 많으면 Agent가 더 똑똑해질까?"

EVERYTHING ENABLED

repos	issues	pulls	actions	gists	orgs
teams	users	packages	security	releases	projects
wiki	webhooks	labels	notif.	search	stars

무엇을 써야 할지 판단부터 어려워진다

ONLY WHAT WE NEED

repos

issues

pull requests

필요한 Tool만 노출하는 것도 Context / Harness Engineering이다.

"Planning, Testing, Skill, Agent를 매번 우리가 직접 설계해야 할까?"

계획은 언제?

Test는 어떻게?

Review는 누가?

Skill은?

Subagent는?

Memory는?

Workflow는?

물론 먼저 고민한 사람들이 있다.

Superpowers

ECC

Oh My Claude Code

세 가지 다른 접근

Superpowers

PROCESS

Planning

TDD

Debugging

Verification

까다로운 선임 개발자

ECC

ENVIRONMENT

Skills

Agents

Rules

Hooks

Memory

Agent 개발환경 Toolkit

Oh My Claude Code

ORCHESTRATION

Planner

Developer

Reviewer

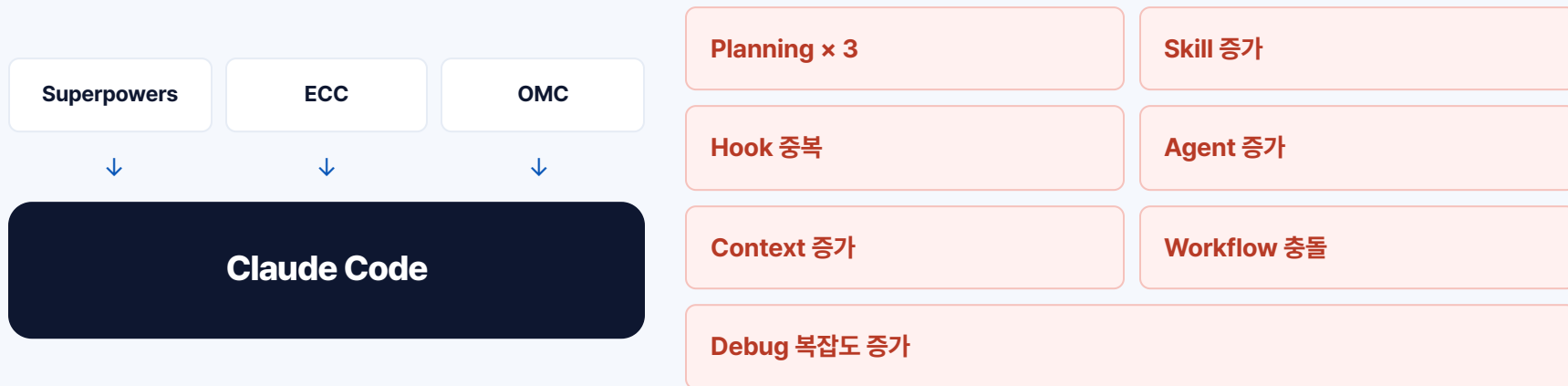
Multi-Agent

AI 개발팀 운영

어떤 것이 최고인지가 아니라 무엇을 강화하려는지가 다르다.

Superpowers도 좋고, ECC도 좋고, OMC도 좋다.

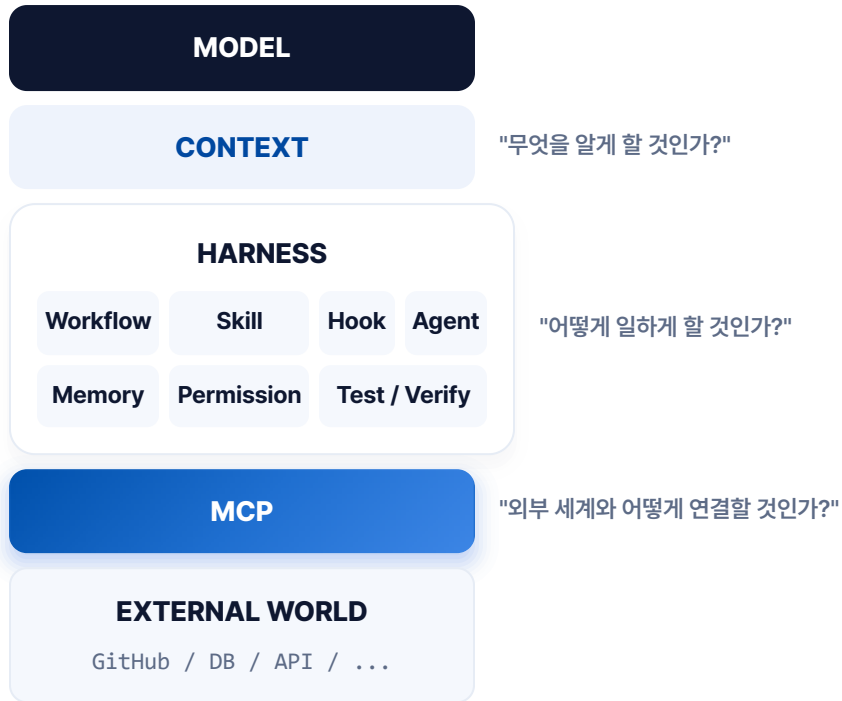
"그럼 전부 설치하면 세 배로 좋아질까?"



More ≠ Better

Harness Engineering은 기능을 최대한 많이 넣는 일이 아니다.

오전의 결론



"같은 AI를 사용하는데
왜 누구는 더 안정적이고
복잡한 일을 맡길 수 있을까?"

**모델만 다루는 것이 아니라
모델이 일하는 환경을 설계하기 때문이다.**

이제 오후에는 같은 모델의 Harness를 바꿔 직접 비교한다.