Harness Engineering

AI가 잘 일하는 환경을 설계하는 기술

Team Attention 이호연 · 2026.04.07

이호연

Builder, Team Attention

- 전) Contents Technologies, Tech Lead
- 전) Terminal X, Tech Lead
- 전) Socar, Senior Data Engineer
- 전) 컨텐츠 크리에이터 (인프런/클래스101 수강생 10K+)

Team Attention

대한민국 No.1 AI Native Team

01 매달 **AI 해커톤 및 행사** 기획·운영

02 SF · Seoul 동시 해커톤 **"랄프톤"** 개최
90팀+ 참여

03 **Show and Prove Club** 운영
실전 Harness, Claude Code as OS
등


Show and Prove · Seoul


Ralphton · SF

AGENDA

오늘 다룰 내용

01 **Harness Engineering이란**

02 **내 Harness 시연**

03 **구조(Scaffolding)** — 폴더, 도구, 경계

04 **맥락(Context Engineering)** — CLAUDE.md, 규칙, 점진적 노출

05 **계획(Planning)** — Plan → Spec → 승인

06 **실행(Orchestration)** — 혼자, 부하 파견, 팀

07 **검증(Verification)**

08 **개선(Compound)**

HARNESS ENGINEERING이란

프롬프트만으로는 부족하다

"말을 잘 하는 것" → **"일하는 방법을 설계해주는 것"**

HARNESS ENGINEERING이란

이미 빠른 테크 회사들은 Harness를 구성하고 있다

LangChain

+14%p

같은 모델, 하네스만 변경
30위 → Top 5

GPT-5.2-Codex 고정. 셀프 검증 루프 + 컨텍스트 자동 수집 + 둠 루프 탐지 추가. TerminalBench 52.8→66.5%

OpenAI

100만 줄

인간 코드 0줄
엔지니어 3~7명, 5개월

5개월을 쓴 곳: 코딩이 아니라 하네스 설계. 5가지 교훈 중 "더 좋은 모델 써라"는 없다. 전부 환경 설계.

Anthropic

$9 vs $200

$9는 비기능, $200은 완성
차이는 모델이 아니라 하네스

싱글 에이전트 20분/$9 → 실패. 3에이전트 하네스 6h/$200 → 완전 동작. 간소화 버전 $124로 품질 유지.

Stripe

1,000 PR/주

완전 무인 자동 머지
전문화된 소형 에이전트 다수

매 스텝 검증 게이트 + 정밀 컨텍스트 관리. 하네스 없이는 불가능한 규모.

모델 교체로 5% 개선하는 것보다, **하네스 설계로 15% 개선하는 것이 현실적**이다

HARNESS ENGINEERING이란

진화 흐름

LEVEL 1

Prompt Engineering

"이렇게 말해봐"

질문을 잘 하면 답이 좋아진다

→

LEVEL 2

Context Engineering

"이런 배경지식을 줘봐"

배경을 알려주면 맥락을 이해한다

→

LEVEL 3

Harness Engineering

"환경 자체를 설계하자"

작업 환경을 만들어주면 혼자서도 잘한다

정의

Harness Engineering

AI가 혼자서도 잘 일할 수 있는 **작업 환경을 만들어주는 것**

맥락
뭘 해야 하는지

제한
뭘 하면 안 되는지

HARNESS
작업 환경

작업 흐름
어떻게 일하는지

검증
잘 했는지 확인

OpenAI
"사람이 방향을 잡고, 에이전트가 실행한다."
openai.com/index/harness-engineering

Anthropic
"규율은 코드가 아니라
스캐폴딩에서 드러난다."
anthropic.com/engineering/harness-design

HARNESS ENGINEERING이란

Harness의 정의는 생각보다 넓다

좁은 의미의 Harness

가드레일 설정
CLAUDE.md에 DOs/DON'Ts + Hook으로 에이전트에 제한을 건다
예: "테스트 없이 커밋 금지" → Hook이 차단

목적별 툴셋
Skill + Agent 조합으로 특정 작업을 효율적으로 수행
예: /bugfix = 진단 → 분석 → 수정 → 검증

비개발 영역도 가능
SEO 감사, 콘텐츠 제작, 리서치 등 개발 외 워크플로우도 설계
예: /geo-audit = 기술+콘텐츠+스키마 병렬 분석

이 강의에서 말하는 Harness

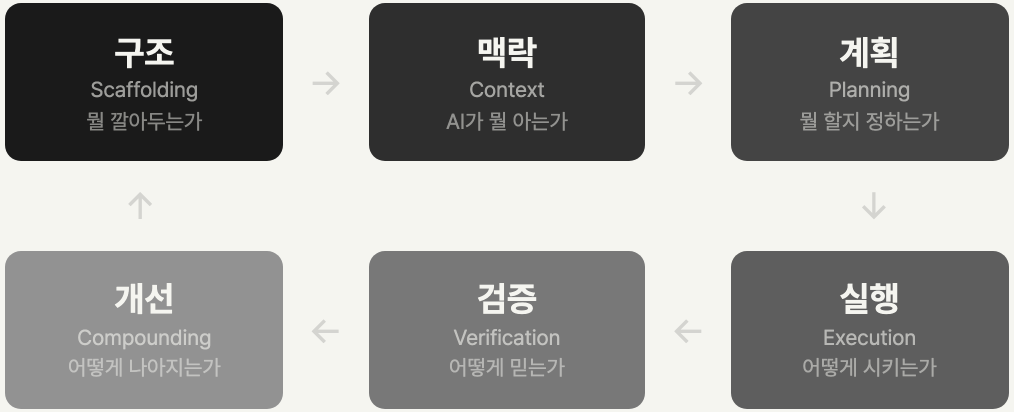
특정 스킬이나 에이전트 조합이 아니라,
AI 에이전트의 **작업 환경 자체를 설계**하는 것.

맥락, 제한, 흐름, 검증 — 이 모든 것을 포괄하는 개념.

Anthropic — *"Harness의 모든 구성요소는 모델이 혼자서는 못 하는 것에 대한 가정을 담고 있다."*

큰 그림

Harness의 6개 축 — 순환 구조



내 Harness 라이브 데모

프로젝트 구조 → 설정 파일 → 실제 작업 흐름

LIVE DEMO

RESOURCES

참고 자료 & 리소스

Harness Engineering 구성 요소를 스킬로 남겨놨습니다 — 플러그인 설치하거나 git 주소로 분석 요청하세요

H

team-attention/harness

materials/slides 강의자료 · **harness-checklist.md** 기본 체크리스트

[Link]

Plugin Slides

@

team-attention/hoyeon

제 개인 Harness 전체 구성이 궁금하신 분은 이 레포를 참고하세요.

[Link]

Example

P

plugins-for-claude-natives

Team Attention에서 자주 쓰는 플러그인 모음. 실무에서 검증된 도구들.

[Link]

Plugins

> **플러그인 설치** 또는 Claude Code에 git 주소 넣고 **"분석해줘"** — 둘 다 됩니다

01

구조(Scaffolding)

프로젝트 구조, 도구 배치, 경계 설정 — 한 번 해두면 계속 쓰는 것

구조(SCAFFOLDING)

프로젝트 구조 설계

1

Monorepo로 묶기

소스코드, 문서, 테스트, 설정을 하나의 프로젝트에서 관리. AI가 전체 맥락을 한눈에 파악할 수 있다.

`src/ docs/ tests/ .claude/`

2

역할별 폴더링

목적이 명확한 폴더 구조. AI가 어디에 뭘 넣어야 하는지 바로 안다.

`docs/ tests/ .dev/ out/ .claude/`

3

아키텍처가 퀄리티를 결정

코드의 아키텍처가 잘 잡혀 있으면, 시간이 갈수록 AI 산출물의 퀄리티가 올라간다.

`clean arch → consistent output`

프로젝트 구조를 잘 설계하면, **AI는 그 구조에 맞춰 따라간다**

구조(SCAFFOLDING)

폴더링 & 아키텍처가 퀄리티를 결정한다

```
my-project/
├── src/ # 비즈니스 로직
├── docs/ # AI의 참고 문서
├── tests/ # 검증 인프라
├── .dev/ # 개발 도구/스크립트
├── .claude/ # AI 설정
├── out/ # 빌드 산출물
└── CLAUDE.md # 프로젝트 지도
```

왜 폴더링이 중요한가
AI가 "테스트 작성해줘"→tests/, "문서 업데이트"→docs/에 자동으로 찾아간다

docs/ = AI의 참고서
아키텍처 문서, 컨벤션, 스펙을 놓으면 AI가 작업 전에 참고한다

구조 없음

모든 게 src/ 하나에

파일 100개가 평면적으로 나열
비즈니스 로직과 유틸이 혼재
테스트가 소스 옆에 산재

시간이 갈수록 AI 품질 하락

클린 아키텍처

레이어별 명확한 책임

도메인 / 서비스 / 인프라 분리
의존성 방향이 일관적
새 기능 = 기존 패턴 복제

시간이 갈수록 AI 품질 상승

구조(SCAFFOLDING)

사람의 문서 vs AI의 문서 — 분리해서 관리

분리하지 않으면, 사람이 관리를 멈춘 순간 AI도 엉뚱한 맥락으로 일한다

사람이 관리 · 비즈니스의 진실

docs/

- 비즈니스 룰 · 도메인 정의
- 체크리스트 · 온보딩 가이드
- ADR · API 스펙 · 외부 연동 규격

AI가 남기는 기록 · 작업의 흔적

.dev/

- learnings · troubleshooting 기록
- 작업 로그 · 디버깅 히스토리
- 실험 결과 · 스크래치패드

아래 설정 파일들이 위 문서를 참조하고 지탱한다

CLAUDE.md
프로젝트 지도 (~100줄)
docs/와 rules/로 포인팅

.claude/rules/
코딩 규칙 · 테스트 컨벤션
glob 패턴으로 조건부 로드

.claude/skills/
반복 작업 레시피
/commit, /review 등

...
hooks, agents,
MCP, plugins 등

위: 콘텐츠(사람+AI) / 아래: 설정(AI 행동 규칙). **docs/는 사람의 책임**

구조(SCAFFOLDING)

AI 도구 배치하기

S **Skills**
반복 작업의 레시피화. /commit, /review 같은 슬래시 명령

H **Hooks**
자동 안전장치. Pre(차단) / Post(검사) / Stop(일지) / Notification(알림)

A **Agents**
전문가 팀원. 서브에이전트 파견 또는 팀 구성

M **MCP**
외부 시스템 연결. DB, Slack, Linear 등 연동

P **Plugins**
위 컴포넌트를 하나의 패키지로 묶어서 배포/공유

구조(SCAFFOLDING)

경계 설정하기

AI가 뭘 알고, 뭘 할 수 있고, 뭘 하면 안 되는지를 정해주는 것

뭘 알려줄까	CLAUDE.md에 프로젝트 규칙, 코딩 스타일, 금기사항 작성	`CLAUDE.md · rules/`
어디까지 허용할까	Permission Mode로 자동 허용 범위 설정 (plan / auto / bypass)	`settings.json`
뭘 막을까	Hook으로 위험한 명령을 실행 전에 자동 차단	`.claude/hooks/`

예: CLAUDE.md에 "직접 main push 금지" 작성 + Hook으로 `git push --force` 차단 → 이중 안전장치

TRY IT 구조(Scaffolding)

구조 설계를 도와주는 스킬

처음 프로젝트를 셋업할 때, 그리고 기존 Harness를 점검할 때

/scaffold

SKILL · HARNESS PLUGIN

프로젝트 초기 구성을 자동으로 잡아주는 스킬. CLAUDE.md, rules, 폴더 구조까지 한 번에 생성.

[Link]

/check-harness

SKILL · HARNESS PLUGIN

현재 프로젝트의 Harness 상태를 체크리스트로 진단. 빠진 설정, 개선 포인트를 알려준다.

[Link]

/skill-creator

ANTHROPIC 공식 PLUGIN

스킬을 만들고 성능을 높이세요. Anthropic에서 제공하는 공식 스킬 생성 도구.

[Link]

> **Harness 플러그인**을 설치하면 /scaffold, /check-harness를 바로 쓸 수 있습니다

02

맥락

CLAUDE.md, 규칙, 점진적 노출 — AI가 뭘 알고 일하는가

맥락

설정 파일 한눈에 보기

아래로 갈수록 범위가 좁고, 우선순위가 높습니다 · 하위 파일은 상위를 자동 상속

~/.claude/CLAUDE.md — 나만 적용 · 모든 프로젝트

my-app/

- CLAUDE.md — 팀 공유 · Git 커밋
- .claude/
  - rules/ — 주제별 분리 · glob으로 조건부 적용
    - code-style.md — *.ts, *.tsx
    - testing.md — *.test.*, __tests__/**
    - security.md — **/auth/**, *.sql
- src/auth/
  - CLAUDE.md — 이 폴더 작업 시에만 자동 로드

맥락(CONTEXT ENGINEERING)

설정 파일 상속 구조 — 하위가 상위를 덮어쓴다

예: User에 "camelCase 사용"

→ 모든 프로젝트에 적용

예: Project에 "snake_case 사용"

→ 이 프로젝트만 `snake_case`

Project가 User를 오버라이드

예: auth/에 "JWT만 사용"

→ `auth` 폴더 작업 시만 적용

가장 좁은 범위가 우선

맥락

CLAUDE.md 실전 가이드

User	**내 작업 습관**, 선호하는 코딩 스타일, 공통 규칙
Project	**이 프로젝트의 기술 스택**, 컨벤션, 중요한 제약
Folder	**특정 모듈의 특수한 규칙**

팁: "최대 200줄 정도를 유지하며 계속 업데이트하라. 너무 길어지면 AI 성능이 급격히 저하된다."

맥락

맥락 관리의 핵심 원칙

Progressive Disclosure

필요한 것만 필요할 때 보여주기.
다음 슬라이드에서 자세히.

.claude/rules/

상황별 세분화된 규칙.
CLAUDE.md가 길어지면 분리.

Scope 계층

User / Project / Folder 각 레벨에 맞는 정보 배치.

"한꺼번에 다 주면 AI도 헷갈린다"

맥락

Progressive Disclosure

내용을 다 때려넣지 말고, "이런 상황에서는 이걸 참고해"라고 안내해서 필요한 것만 읽게 한다

SKILL.md 또는 CLAUDE.md 안에서

코드 작성 시 → **references/code-style.md** 참고
테스트 시 → **references/testing-guide.md** 참고
API 설계 시 → **references/api-convention.md** 참고
배포 시 → **references/deploy-checklist.md** 참고

핵심 아이디어

프롬프트에 **"이 상황에서는 이 문서를 읽어"**
라고 가이드를 주면 AI가 동적으로
필요한 문서만 읽는다

폴더 구조로 구성

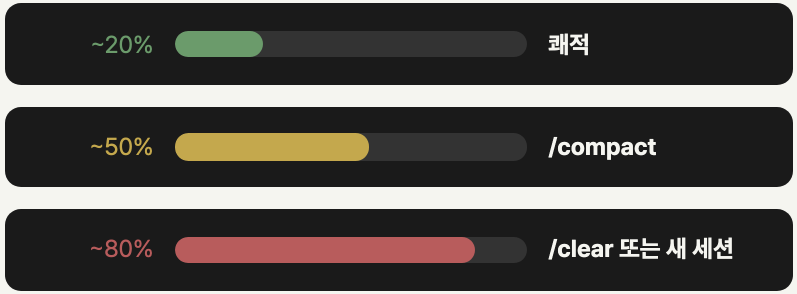
```
my-skill/
  SKILL.md
  references/
    code-style.md
    testing-guide.md
    api-convention.md
    deploy-checklist.md
```

```
CLAUDE.md (핵심만 30줄)
docs/
  architecture.md
  conventions.md
```

SKILL/CLAUDE.md에 다 넣는 게 아니라 **폴더로 분리하고 상황별로 참조**시킨다 → 컨텍스트 절약

맥락(CONTEXT ENGINEERING)

세션 맥락 관리 — 쌓이면 비워라



내 기준: 20~30% 되면 새로 시작

아예 다른 맥락의 작업을 하게 되면 /clear. 이어가야 하면 handoff로 맥락을 넘긴다.

`/clear`	컨텍스트 완전 초기화. 다른 주제로 전환할 때.
`/compact`	오래된 대화를 요약·압축. 같은 주제를 이어갈 때.
`handoff`	현재 세션의 맥락을 파일로 저장 → 새 세션에서 이어받기. 맥락 손실 없이 전환.

컨텍스트는 **채우는 것**만큼 **비우는 것**도 중요하다

맥락

.claude/rules/ 가이드

CLAUDE.md가 길어지면 주제별로 분리 + glob 패턴으로 자동 매칭

작동 방식

.claude/rules/ 폴더에 .md 파일을 놓으면 Claude Code가 자동으로 읽습니다. 파일명에 **glob 패턴**을 넣으면 해당 파일 작업 시에만 로드됩니다.

```
# .claude/rules/ 구조
code-style.md # 항상 로드
testing.md # 항상 로드
react-*.md # *.tsx 작업 시만
api-design.md # routes/ 작업 시만
```

규칙 파일 상단에 **glob 패턴**을 지정하면 조건부 로드가 됩니다.

예시

security.md

glob: **/*.sql, **/auth/**

.env 직접 수정 금지, SQL은 반드시 parameterized query

testing.md

glob: **/*.test.*, **/__tests__/**

커버리지 80%+, mock 최소화, 통합테스트 우선

code-style.md

(항상 로드)

함수 20줄 이내, camelCase, 주석은 영어

맥락(CONTEXT ENGINEERING)

주기적으로 점검하기

Skill · MCP는 Context를 먹는다

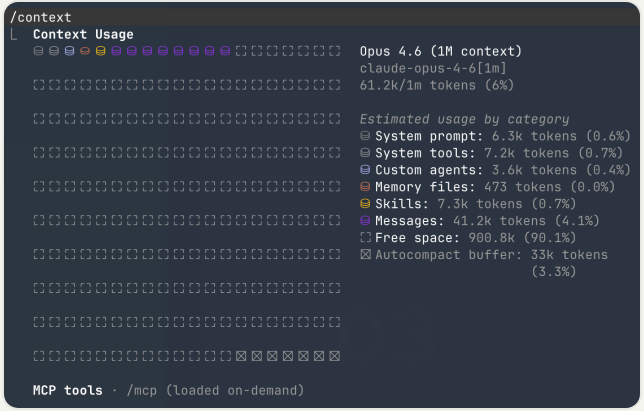
Skill, MCP는 생각보다 context를 많이 차지하고 pollution이 생길 수 있다. 안 쓰는 것들은 주기적으로 점검해서 정리하기.

```
/context 로 현재 사용량 확인
```

CLAUDE.md가 비대해질 때

관리 안 되면 중복·대치되는 내용이 쌓인다. AI한테 직접 점검을 시키자.

```
"CLAUDE.md, .claude/rules를 분석해서 논리적으로 문제가 없는지 + 중복되거나 대치되는 내용은 없는지 분석해줘"
```



/context — 카테고리별 토큰 사용량 확인

쌓는 것만큼 **점검하고 비우는 것**도 Context Engineering이다

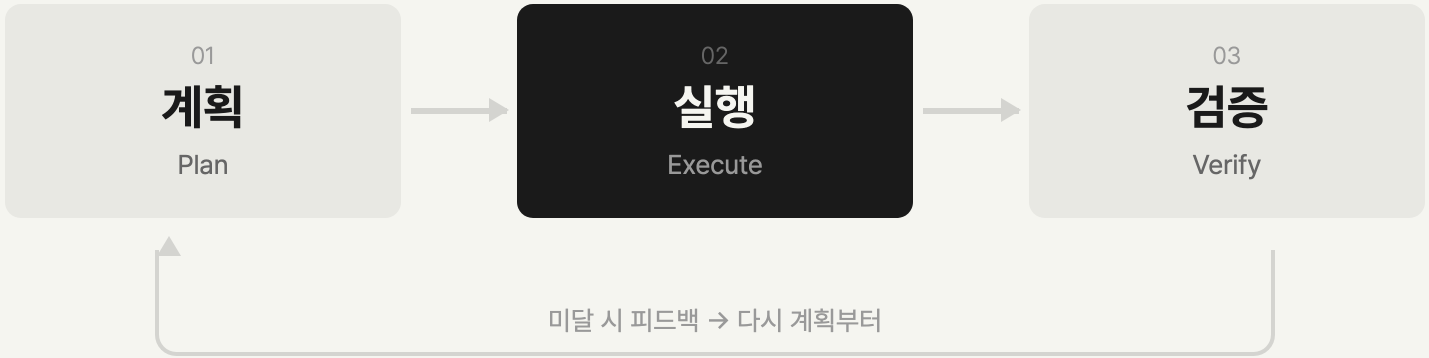
03

계획

AI에게 일을 시키기 전에, 무엇을 할지 함께 정하기

계획

핵심 흐름: 계획 → 실행 → 검증



한 번에 완벽하게가 아니라 **반복해서 수렴**하는 구조

계획

계획 세우게 하기 — "해줘"의 함정

안티패턴: "해줘"

"이거 만들어줘"

↓

AI가 알아서 만듦

↓

검수 → "아닌데..."

↓

다시 시킴 → 검수 → 다시...

↓

시간만 날림

같이 계획부터

"~~할 건데 같이 계획 세워보자"

↓

AI가 계획 작성

↓

사람이 검토 → 수정 → 승인

↓

승인된 계획대로 실행

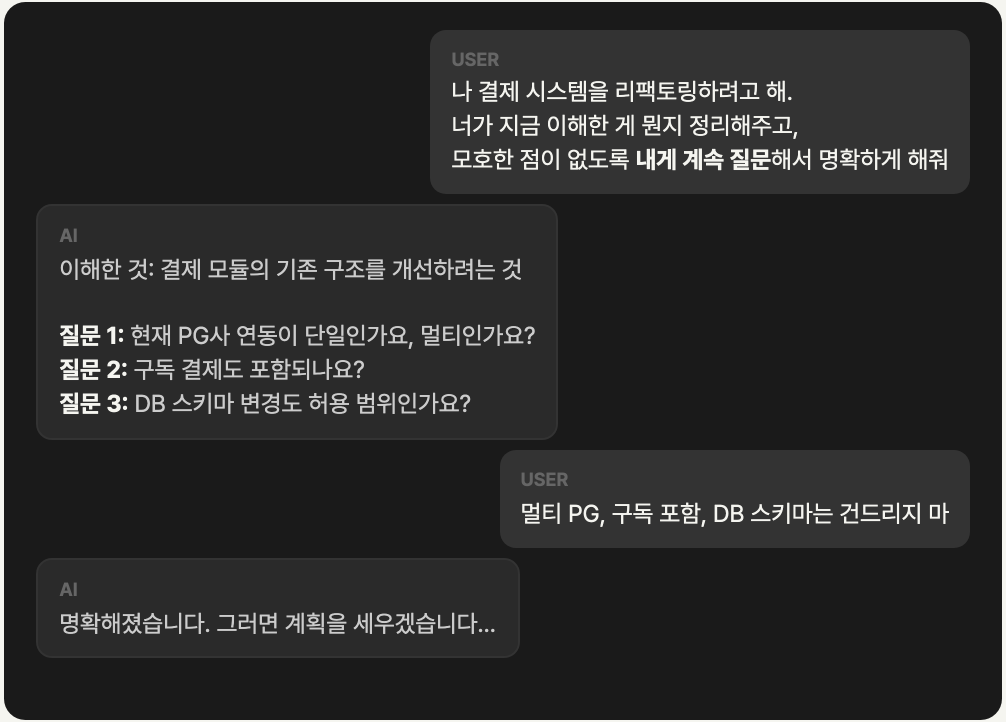
↓

높은 성공률

계획과 실행을 분리하면 검수 횟수가 줄고, 결과의 예측 가능성이 올라간다

계획(PLANNING)

나에게 질문하게 하기 — AskUserQuestion



왜 질문이 중요한가

사람은 자기 머릿속의 전제를 프롬프트에 다 담지 못한다. AI가 질문을 해야 빠진 맥락이 드러난다.

커스텀 Plan 스킬에서

인터뷰 단계를 넣으면 AI가 자동으로 AskUserQuestion을 사용해 모호한 점을 물어본다.

스킬 프롬프트에 이렇게 쓴다:

```
"이해한 것을 미러링하고,
모호한 점을 질문해서 명확하게 해줘"
```

"해줘"가 아니라 "물어봐" — AI가 스스로 맥락을 채우게 하기

계획

계획의 진화 — 커스텀 Plan 스킬

기본 Plan Mode의 한계

1 **Plan Mode**
계획과 실행을 분리. 하지만 사람이 모든 걸 프롬프트에 담을 수 없다.

↓ 한계를 느끼면

2 **커스텀 Plan 스킬**
인터뷰 + 요구사항 도출 + 플랜 파일까지 자동화하는 전용 스킬을 만든다.

커스텀 Plan 스킬 예시: /specify

1 **목표 확인** — 사용자의 의도를 미러링

↓

2 **인터뷰** — 모호한 지점을 질문으로 끌어내기

↓

3 **요구사항 + 태스크 도출**

↓

4 **플랜 파일로 떨구기** → /execute로 실행

`implicit(머릿속) → explicit(플랜 파일)`

기본 Plan으로 부족하면 프로젝트에 맞는 계획 스킬을 만든다

TRY IT 계획(Planning)

계획을 더 잘 세우기 위한 스킬

스펙을 명확하게, 모호함을 줄이고, unknown-unknown을 잡아내기

/specify

SKILL · HARNESS PLUGIN

스펙 파일을 더 잘 얻기 위한 스킬. 인터뷰 → 요구사항 도출 → 플랜 파일까지 자동화.

[Link]

/deep-interview

SKILL · HARNESS PLUGIN

깊이 있는 인터뷰로 unknown-unknown을 줄인다. 미처 생각 못한 엣지 케이스까지 끌어낸다.

[Link]

/clarify

PLUGIN · PLUGINS-FOR-CLAUDE-NATIVES

요구사항을 명확하게 하고 싶을 때. 모호한 지시를 구체적인 스펙으로 변환.

[Link]

> 계획 단계에서 **모호함을 줄이는 것**이 실행 품질을 결정한다

04

실행

혼자, 부하 파견, 팀 — 상황에 맞는 실행 패턴

실행

실행 패턴: 혼자 vs 부하 파견 vs 팀

혼자 · Single

단순 작업
대부분의 일상

부하 파견 · Subagent

병렬/전문화
위임 → 종합

팀 협업 · Team Mode

다관점 · 복잡
에이전트 간 소통

90%
단일/서브에이전트로 충분

~7x
팀 모드 토큰 비용

실행

상황별 오케스트레이션 패턴

순차 파이프라인

상황: 블로그 글을 쓰려는데 조사 → 초안 → 퇴고 → 발행 순서가 중요하다

"AI 에이전트 트렌드 조사해서 → 초안 작성하고 → 퇴고까지 순서대로 진행해줘"

동작: TaskCreate로 체크박스 생성, 하나씩 직렬 수행

병렬 Subagent

상황: 경쟁사 3곳의 랜딩 페이지를 동시에 분석하고 싶다. 서로 독립적.

"A사, B사, C사 랜딩 페이지를 각각 에이전트 파견해서 동시에 분석해줘. 끝나면 비교표 만들어"

동작: Agent 3개 spawn, 병렬 분석 후 메인이 취합

Team Mode

상황: 새 기능을 설계하면서 동시에 구현하고 리뷰도 받아야 한다. 에이전트끼리 소통이 필요.

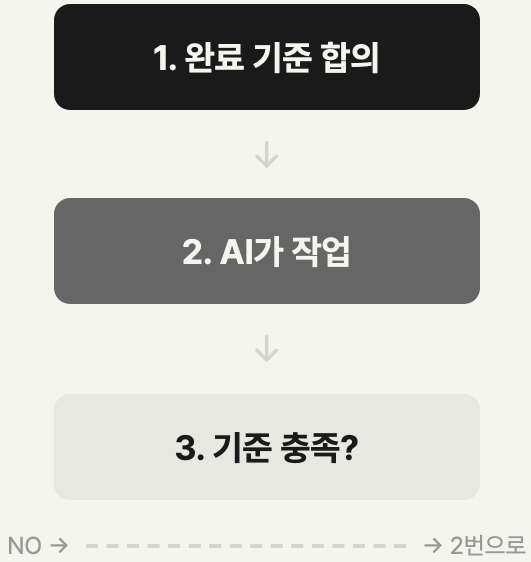
"Team 켜서 설계자, 구현자, 리뷰어 3명으로 팀 꾸려줘. 설계 나오면 구현자가 바로 시작해"

동작: TeamCreate로 팀 생성, 에이전트 간 직접 소통

실행

Ralph Loop — 될 때까지 반복

완료 기준을 정하고, 충족할 때까지 AI가 알아서 돈다



예시: "랜딩 페이지 만들어줘"

완료 기준: 모바일 반응형 + Lighthouse 90점 이상 + 카피 3번 이상 퇴고

1. 페이지 완성, Lighthouse 72점
2. 92점, 모바일 레이아웃 깨짐
3. 레이아웃 수정, 카피 퇴고 2회만
4. 전항목 충족 — PASS

사람이 한 일: 기준 정하기 1번. 나머진 AI가 알아서.

"뭐가 되면 끝인지"만 정해주면 AI가 될 때까지 계속 돈다

실행

Auto Research — 자율 실험 루프

사람은 방향만 정하고, AI가 밤새 실험을 돌린다

karpathy/autoresearch

program.md에 연구 방향 작성 (스킬과 같은 역할)

AI가 **train.py만 수정** — 아키텍처, 하이퍼파라미터 변경

시간당 **~12개 실험** 자율 수행, 밤새 무인 운영

성능 개선되면 유지, 아니면 폐기 → **다음 실험**

`github.com/karpathy/autoresearch`

핵심 패턴: 수정 → 실행 → 평가 → 반복 — 사람은 방향(program.md)만 잡아주면 된다

TRY IT 실행(Orchestration)

실행을 도와주는 도구들

오케스트레이션 패턴 적용, 공식 검증 플러그인, 자율 실험

/agent-orchestrate

SKILL · HARNESS PLUGIN

현재 문제에 최적화된 오케스트레이션 패턴을 자동 적용. 단일/병렬/파이프라인 등 상황에 맞게 선택.

[Link]

ralph

CLAUDE CODE 공식 PLUGIN

실행 결과를 자동 검증하는 공식 플러그인. Claude Code에서 바로 사용 가능.

[Link]

autoresearch

KARPATHY · 자율 실험

사람은 방향만 정하고 AI가 밤새 실험. 수정→실행→평가→반복 루프를 자율적으로 돌린다.

[Link]

> 실행은 **패턴 선택 + 자율성 범위 설정**이 핵심이다

05

검증

결과물을 어떻게 믿을 것인가

검증 원칙

01

기준이 있어야 검증이 가능하다

Sprint Contract

작업 전에 "뭘 만들고 어떻게 검증할지" 합의한다. 기준 없이 시키면 AI가 끝없이 하거나 대충 끝냈다고 한다.

Ralph Loop — 기준 달성까지 반복시킨다. 기준이 명확하면 자동화가 가능.

1 **완료 조건을 먼저 정한다**
"이 3가지가 되면 끝" — 시작 전에 합의

2 **조건을 측정 가능하게 쓴다**
"잘 되게" 말고 "테스트 통과 + 빌드 성공"

3 **미달이면 다시 돌린다**
기준이 있으니 자동 반복이 가능해진다

"뭐가 되면 끝인가"를 작업 시작 전에 정한다

검증 원칙

02

컨텍스트를 나누고, 관점을 분리한다

Anthropic — Harness design for long-running application development

Generator
결과물을 만드는 AI

분리

Evaluator
결과물을 평가하는 AI

> *"자기 작업을 평가하면* **quality가 mediocre해도 자신있게 칭찬***한다"*

> *"만든 AI와 확인하는 AI를 분리하는 것이* **가장 강력한 레버***"*

> *"Evaluator를 회의적으로 튜닝하는 게, Generator를 자기비판적으로 만드는 것보다* **훨씬 쉽다***"*

핵심: 같은 컨텍스트에서 만들고 평가하면 안 된다 — 관점을 물리적으로 분리해야 한다

검증 원칙

03

모델도 나누고, 역할도 나눈다

Codex
코드 리뷰

로직 오류, 보안 취약점,
테스트 누락 검출

Gemini
문서 리뷰

일관성, 정확성,
구조 검증

Opus / Sonnet
성능별 분업

Opus: 복잡한 판단, 아키텍처
Sonnet: 빠른 확인, 반복 검증

> *"Out of the box, Claude is a poor QA agent."* — **검증 에이전트도 튜닝이 필요하다.** *기준을 구체적으로, 회의적으로 설정해야 쓸 수 있다.*

다른 모델, 다른 역할로 교차 검증 — 한 모델의 맹점을 다른 모델이 잡는다

검증

04

에이전트에게 눈을 달아주기

코드만 검증하는 게 아니라, 화면도 직접 보고 확인할 수 있어야 한다

Browser Agent

CHROME-CDP / AGENT-BROWSER

실제 Chrome 브라우저를 제어. DOM 탐색, 클릭, 스크린샷, 네비게이션. 웹앱의 UX를 직접 검증.

```
chrome-cdp • agent-browser
```

Computer Use

BUILT-IN MCP

스크린샷 + 마우스/키보드로 모든 앱 제어. 웹이 아닌 네이티브 앱, 디자인 도구도 검증 가능.

```
computer-use MCP (built-in)
```

시각 검증 루프

PATTERN

만든다 → 스크린샷 찍는다 → 보고 판단한다 → 수정한다. 사람이 눈으로 하는 것을 AI가 대신.

```
generate → screenshot → evaluate
```

검증 범위를 코드에서 화면까지 확장하면 사람이 끼어들 일이 줄어든다

TRY IT 검증(Verification)

검증을 도와주는 도구들

브라우저·컴퓨터 사용 도구로 QA를 자동화하고, 검증 전략을 참고하세요

/qa

SKILL · HARNESS PLUGIN

Browser Agent, Computer Use 도구를 활용해서 QA를 자동화하는 스킬. 화면을 직접 보고, 클릭하고, 검증한다.

[Link]

verify 레퍼런스

REFERENCE · TEAM-ATTENTION/HOYEON

검증 스킬을 더 깊이 이해하고 싶다면 레퍼런스를 참고하세요. 검증 전략, 패턴, 실제 적용 사례가 정리되어 있습니다.

[Link]

> **검증을 자동화**하면 사람은 판단에만 집중할 수 있다

검증

안전장치

되돌릴 수 있는 환경

브랜치/Worktree 격리에서 작업. 실수해도 메인은 안전.

`git worktree add`

위험한 건 사람이 확인

삭제, 배포, 외부 발송 같은 작업은 승인 후 실행.

`Runtime Gate`

Dry-run 먼저

"이렇게 할 건데 맞나?" 미리보기 후 실행.

`--dry-run`

"실수해도 괜찮은 구조"를 만드는 것이 핵심

06

개선

어떻게 계속 나아질 것인가

개선

관측하고 개선하기

관측하기

세션 분석
프롬프트 패턴 + Skill/Agent 호출 빈도를 분석. 어디서 시간을 쓰는지, 어디서 실패하는지.

AI Slop 감지
작업하다 보면 불필요한 코드, 중복 설정, 안 쓰는 규칙이 쌓인다. 이게 AI slop.

개선하기

3번 반복 → Skill로
같은 작업을 3번 반복하면 자동화할 타이밍. 스킬로 만들어서 재사용.

3번 틀리면 → Rule 또는 CLAUDE.md에
같은 실수 반복 → .claude/rules/ 에 규칙 추가하거나 CLAUDE.md에 명시.

Skill 개선 루프
만들기 → 사용 → 세션 분석 → 병목 → 개선. 스킬도 계속 다듬는다.

작업 → 관측(세션+사용 패턴) → **패턴 발견** → Skill 또는 Rule → **더 나은 작업**

개선

단순화하기

-1 **안 쓰는 건 치운다** — 필요 없어진 Skill, MCP, Rule은 바로 삭제. 쌓이면 AI slop.

↑ **모델이 좋아지면 Harness를 재평가** — 예전에 필요했던 가드레일이 지금은 불필요할 수 있다.

! **과설계 신호를 인식하기** — 설정이 너무 복잡하면 뭔가 잘못된 것. 점점 단순해져야 정상.

Anthropic — *"Harness의 공간은 모델이 좋아져도 줄어들지 않는다. 이동할 뿐이다."*

개선

잘 가고 있나? — 자가 진단

잘 가고 있다는 신호

+ **같은 말을 두 번 하지 않는다**
  맥락 전달이 잘 되고 있다
+ **실수가 규칙이 된다**
  개선 루프가 돌고 있다
+ **차단 장치가 뭔가를 막고 있다**
  사고가 구조적으로 예방되고 있다
+ **불필요한 것이 줄어든다**
  복잡해지는 게 아니라 단순해지고 있다

실패하고 있다는 징후

- **검수에 시간이 더 오래 걸린다**
  검증 자동화가 필요하다는 신호
- **시켰는데 원하는 결과가 안 나온다**
  맥락 전달 또는 계획 단계가 부족
- **스킬·에이전트가 많은데 잘 안 쓴다**
  context pollution + 제대로 동작 안 함
- **가이드 파일이 길어지고 관리 안 된다**
  CLAUDE.md·docs 분리/정리 없이 쌓임

좋은 Harness는 점점 단순해진다. 복잡해지고 있다면 뭔가 잘못된 것.

TRY IT 개선(Compound)

세션에서 인사이트 추출하기

작업 세션을 분석해서 패턴, 실수, 개선점을 자동으로 뽑아내기

session-wrap

PLUGIN · PLUGINS-FOR-CLAUDE-NATIVES

Claude Code 세션이 끝난 뒤, 세션 내용을 분석해서 인사이트를 추출하는 플러그인. 반복 패턴, 실수, 자동화 기회를 찾아냅니다.

패턴 발견
반복 작업 패턴을 감지해서 Skill 후보로 제안

실수 추출
세션 중 실수를 정리해서 Rule 후보로 제안

문서 업데이트
CLAUDE.md, docs 업데이트 필요 항목 감지

[Link]

세션 분석 → 패턴 발견 → Skill/Rule 추가 — 이것이 개선 루프의 핵심

마무리

그럼 사람은 앞으로 뭘 해야 할까요?

구조(Scaffold)

환경을 설계하고 유지하기

- 비즈니스 요구사항이 바뀔 때마다 코드베이스 구조 발전시키기
- 새 도구/서비스 도입 시 폴더링과 경계 재설계
- AI가 따라올 수 있는 아키텍처 유지

맥락(Context)

AI가 아는 것을 최신으로 유지

- CLAUDE.md와 docs/를 코드와 함께 갱신
- 새 컨벤션이 생기면 rules/에 추가
- 오래된 규칙은 정리 — 가비지 컬렉션

개선(Compound)

AI Slop이 나오지 않도록 점검

- AI 산출물의 품질을 지속적으로 모니터링
- 반복되는 실수 → 규칙으로, 반복 작업 → 스킬로
- 안 쓰는 건 치우고, 쓰는 건 날카롭게

AI가 코드를 쓰는 시대, 사람의 역할은 **"잘 짜기"에서 "잘 일하는 환경을 만들기"**로 바뀐다

마무리

지금 할 수 있는 것

1

이미 잘 만들어진 도구부터 써보기

- gstack — github.com/garrytan/gstack
- superpowers — github.com/obra/superpowers
- oh-my-claudecode — github.com/Yeachan-Heo/oh-my-claudecode

2

내 현재 상태 진단하기

본인이 이해가능한 형태로 Harness를 사용하는 것이 중요함.

3

개선하고 점검하기 반복

한꺼번에 다 바꾸려 하지 말 것. 불편한 지점만 하나씩.

> **⚠ 조심할 점**
>
> Harness를 설계할 때 너무 옥죄려고 하지 말 것. 자유도를 높게 주고,
> 안 되는 것만 제한하기. 생각보다 Agent는 예측 가능하게 잘 움직인다.

감사합니다

Q&A