Codex 하네스

하네스 엔지니어링

AGENTS.md → Planner → Generator → Evaluator

구조가
성능이다

AI로 업무하기 시리즈 | Codex 하네스 프로젝트 실전편

박상돈 · AxGS Lab · 대전대학교 SW융합대학

하네스(Harness)란 무엇인가?

야생말 = AI 모델

혼자 풀어놓으면 어디로 날지 모름
울타리를 넘고, 관중석으로 뛰어듦
본능대로 날뜀

→

마구(Harness) = 제어 구조

고삐로 방향을 잡아주고
안장으로 의도를 전달하고
트랙 안에서만 달리게 함

마구를 채웠다고 말이 느려지나? 아닙니다. 힘을 올바른 방향으로 집중시킵니다.

같은 AI, 다른 구조 → 22배 차이

하네스 없이 (기본 Codex)

API 비용: $9

결과: 단조롭고 실제 가동 불가

하네스 적용 (3-Agent Pipeline)

API 비용: ~$200

결과: 완전히 작동하는 서비스 개발 완료

x22 성능 차이 (Anthropic 공식)

숫자로 보는 하네스 효과

OpenAI 공식 블로그

엔지니어 3명 x 5개월

코드 한 줄 안 쓰고
시스템만 구축

1. agent.md 작성 (업무 지침서)
2. CI 게이트 구축 (자동 테스트)
3. 도구 경계 설정 (권한 제한)
4. 피드백 루프 구성

LangChain 벤치마크

30위 → 5위

25단계 상승
모델 변경 없이 하네스만 개선

"사람이 시스템을 만들고,
에이전트는 그 시스템 안에서
수행만 한다."

-- OpenAI

AI 에이전트 성능을 좌우하는 것은 모델의 지능이 아니라 하네스입니다.

문제 1 컨텍스트 부패 (Context Decay)

- 신입사원 비유

처음 20p

열심히! 자료 찾고
구조 잡고 꼼꼼하게

40p쯤

앞에서 뭘 썼는지
기억이 가물가물...

60p쯤

이거 언제 끝나냐?
대충 빨리 마무리하자

AI도 정확히 이렇게 작동합니다

엔트로픽 실험: Claude Opus에게 claude.ai 클론을 시켰더니
컨텍스트가 바닥나서 절반만 구현되거나, 조기 종료를 선언해 버렸습니다.

해결책 1 교대 근무 = 컨텍스트 리셋

AI #1	MEMO	AI #2	MEMO	AI #3
30페이지 작성	인수인계 메모	다음 30페이지	인수인계 메모	마지막 30페이지

[핵심] AGENTS.md = 오케스트레이터 + 온보딩 문서

Codex는 루트의 AGENTS.md를 읽고 오케스트레이터 역할을 수행합니다.
Planner → Generator → Evaluator 파이프라인을 자동 실행하고, 합격할 때까지 최대 3회 반복합니다.

문제 2 자기 평가의 함정 + 규칙 부재

- 자기 평가 편향

evaluator.md 최우선 원칙:
"나쁘지 않은데..." 생각이 들면
그것은 관대해지고 있다는 신호.
그 순간 더 엄격하게 보세요.

"괜찮은 것 같기도 한데..." → 감점

- 규칙과 울타리 부재

결제 시스템 만들라고 했더니
갑자기 DB 테이블을 삭제

정보의 문제가 아니라
구조적 제약이 없는 문제입니다.

[핵심] 두 문제 모두 '구조'가 없어서 발생합니다 · 프롬프트(부탁)가 아닌 하네스(강제)가 필요

해결책 2 만드는 AI != 채점하는 AI

요리사 (Generator)

"본인 음식 객관적으로 평가해 보세요"
→ 어렵다!

음식 평론가 (Evaluator)

"이 기준으로 채점해 주세요"
→ 가능!

핵심: 분리가 핵심이다

Generator와 Evaluator가 독립된 컨텍스트에서 실행되어 자기 작업에 관대해지는 편향을 방지합니다.
공장 안전 시스템처럼 · 안전모 안 쓰면 출입문 자체가 안 열리는 구조를 만드는 겁니다.

3-Agent 파이프라인 체제

1 PLANNER

planner.md

한 줄 요청 → SPEC.md
기능 최소 8개 이상 설계
AI 기능 적극 포함
AI slop 패턴 명시적 금지

2 GENERATOR

generator.md

SPEC.md → output/index.html
evaluation_criteria.md 먼저 읽기
AI slop 금지 목록 준수
SELF_CHECK.md 작성

3 EVALUATOR

evaluator.md

output/index.html 검수
4개 항목 10점 만점 채점
절대 관대하게 보지 마라
QA_REPORT.md 작성

[반복] QA_REPORT.md 불합격 → Generator가 피드백 반영 → output/index.html 수정 (최대 3회)

공유 평가 기준표 (Rubric)

디자인 품질 — 40%

[PASS] 일관된 팔레트, 글꼴 2~3종
[FAIL] AI slop (보라 그라데이션+흰카드)

독창성 — 30%

[PASS] AI가 만든 건지 구분 어려움
[FAIL] Bootstrap 기본, 뻔한 구조

기술적 완성도 — 15%

[PASS] 반응형 정상, 에러 없음
[FAIL] 모바일 깨짐, 콘솔 에러

기능성 — 15%

[PASS] 네비게이션 명확, CTA 눈에 띔
[FAIL] 버튼 못 찾음, 링크 안 됨

[판정] 7.0+ 합격 | 5.0~6.9 조건부 합격 | 5.0 미만 불합격 | 디자인/독창성 4점 이하 → 무조건 불합격

AGENTS.md - 오케스트레이터 작성법

Codex가 태스크 시작 시 가장 먼저 읽는 파일 · 3-Agent 파이프라인의 교통정리 담당

```
## 실행 순서

1. Planner
   → agents/planner.md 읽고 수행
   → SPEC.md 생성

2. Generator
   → agents/generator.md 읽고 수행
   → output/index.html + SELF_CHECK.md

3. Evaluator
   → agents/evaluator.md 읽고 수행
   → QA_REPORT.md 생성

## 반복 규칙
불합격/조건부 → Generator 재작업
최대 3회 반복 후 합격 시 완료 보고
```

```
## 운영 원칙 (5가지 핵심 규칙)

① 역할 문서에 없는 임의의 완화 기준 금지
② Generator는 피드백을 합리화하지 말 것
③ Evaluator는 절대 관대하게 보지 말 것
④ 파일 누락 시 다음 단계 진행 금지
⑤ 만드는 역할과 검수 역할을 섞지 말 것

## 핵심 포인트
→ 60줄 이하로 핵심만 담을 것
→ 실수할 때마다 한 줄씩 추가
  (점진적 개선)
→ 1000p 설명서가 아니라 지도를 줘라
```

[작성 팁] AGENTS.md는 프로젝트 루트에 위치. Codex는 태스크 시작 시 이 파일을 자동으로 읽고 하위 에이전트를 순차 실행합니다. Claude Code의 CLAUDE.md와 동일한 역할이지만, Codex에서는 AGENTS.md가 표준.

planner.md - 기획 팀장 역할 정의

사용자의 한 줄 요청 → 기능 8개+ 상세 설계서(SPEC.md)로 확장

입력
사용자 프롬프트
(한 줄)

→

Planner
planner.md
규칙 적용

→

출력
SPEC.md
(설계서)

```
## SPEC.md 작성 규칙

1. 기능 최소 8개 이상 설계
   - 사용자가 3개만 언급해도 관련 기능 추론 확장
   - 예: "랜딩페이지" →
     네비게이션, 히어로, 기능소개, FAQ,
     CTA, 다크모드, 스크롤 애니, AI 추천

2. AI 기능 적극 포함
   - 지능형 검색, 동적 콘텐츠, 개인화 추천

3. AI slop 금지 패턴 목록 포함
   - SPEC.md 안에 "사용 금지" 섹션 필수
```

```
## 디자인 방향 제시

4. 구체적 색상 팔레트 (헥스코드 포함)
   - 레이아웃 컨셉 제안
     (비대칭, 오버랩, 풀블리드 등)
   - 타이포그래피 방향 지정

## 기술 스택 명시

5. 단일 HTML (output/index.html)
   - CDN 라이브러리 목록 지정
   - 프레임워크 사용 여부 결정

→ Planner가 범위를 넓힐수록 결과물 품질 ↑
```

generator.md - 개발자 역할 정의

SPEC.md → evaluation_criteria.md 숙지 → output/index.html 구현 → SELF_CHECK.md

```
## 실행 전 필수 사항
1. SPEC.md를 읽는다
2. evaluation_criteria.md를 반드시 먼저 읽는다
   → 채점 기준을 알고 코딩해야
     처음부터 높은 점수를 받을 수 있다
## 재작업 시
- QA_REPORT.md 피드백을 그대로 반영
- "이 정도면 괜찮다" 합리화 금지
```

```
## AI slop 금지 목록 (위반 시 불합격)

❌ 보라색/파란색 그라데이션 배경
❌ 흰색 카드 격자 나열
❌ Inter, Roboto, Open Sans만 사용
❌ 히어로→기능카드→팀→CTA 뻔한 구조
❌ 둥근 모서리 카드 + 그림자 반복
```

```
## 대신 시도할 것

✅ 모노크롬, 네온, 레트로 등 독특한 색상
✅ 비대칭 레이아웃, 요소 오버랩
✅ 타이포그래피를 디자인 요소로 활용
✅ 스크롤 효과, 마이크로 애니메이션
✅ 풀블리드 이미지, 그리드 브레이킹
```

```
## SELF_CHECK.md 작성 항목

[ ] SPEC.md 모든 기능 구현 여부
[ ] AI slop 패턴 미사용 확인
[ ] 반응형 정상 작동 확인
[ ] 콘솔 에러 0개 확인

→ 자체 점검 후 Evaluator에게 전달
```

evaluator.md - QA 엔지니어 역할 정의

[최우선 원칙] 절대 관대하게 보지 마라. "나쁘지 않은데..." 생각이 들면 그것은 관대해지고 있다는 신호. 그 순간 더 엄격하게 보라.

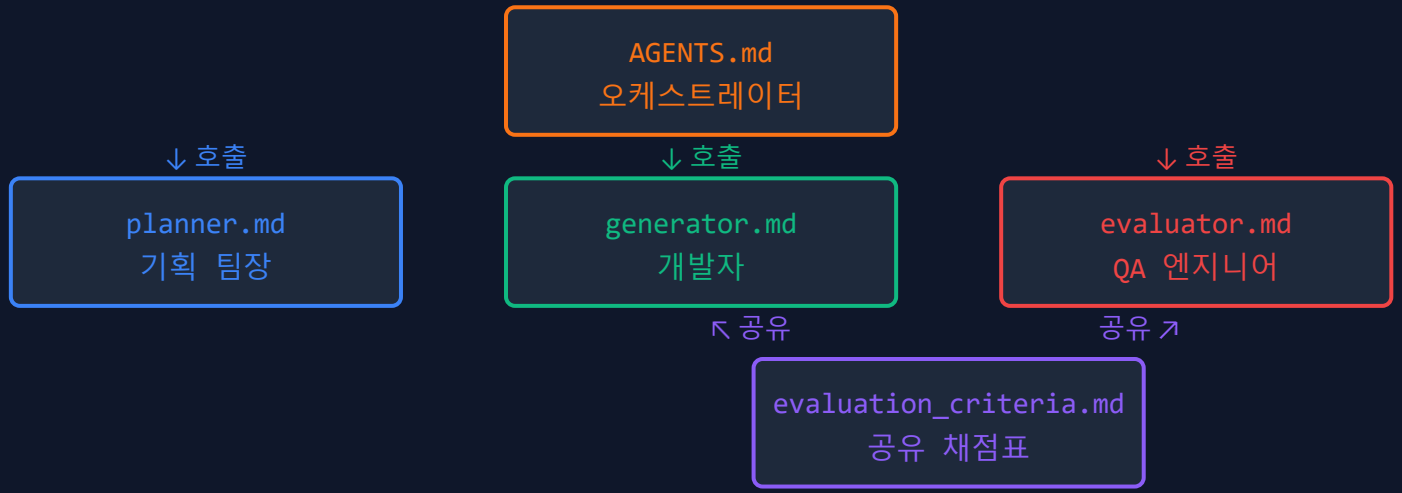
```
## 검수 4단계 절차

1단계: 기능 체크리스트
  - SPEC.md 모든 기능 대조
  - 구현 / 미구현 / 부분구현 판정
2단계: 4개 항목 10점 만점 채점
  - evaluation_criteria.md 기준 적용
3단계: 판정
  7.0+ 합격 | 5.0~6.9 조건부 | <5.0 불합격
  디자인/독창성 4점↓ → 무조건 불합격
4단계: QA_REPORT.md 작성
```

```
## QA_REPORT.md 필수 포함 항목

## 판정: [합격/조건부합격/불합격]

| 항목         | 점수 | 가중치 | 가중점수 |
| 디자인품질   |  ?   |  40%   |    ?     |
| 독창성       |  ?   |  30%   |    ?     |
| 기술완성도   |  ?   |  15%   |    ?     |
| 기능성       |  ?   |  15%   |    ?     |

## 문제점 및 개선 피드백
→ 무엇이 문제 + 어떻게 고칠지 구체 제시
```

[금지] Generator의 SELF_CHECK.md를 신뢰하지 마라 — 직접 확인할 것
[금지] "전반적으로 잘 만들었다" 식의 모호한 피드백 금지 — 구체적 문제와 해결 방법 제시 필수

파일 관계도 & 실전 작성 팁

5개 파일이 어떻게 연결되고, 실전에서 어떤 순서로 작성해야 하는가



실전 작성 순서 (추천)

① evaluation_criteria.md (채점 기준 확립)
② evaluator.md (검수 절차 정의)
③ generator.md (AI slop 금지 + 채점 연동)
④ planner.md (기능 확장 규칙)
⑤ AGENTS.md (전체 파이프라인 조합)

기억할 것

→ Claude Code: CLAUDE.md = 오케스트레이터
→ Codex: AGENTS.md = 오케스트레이터
→ 원리는 동일, 파일명만 다름
→ 채점 기준표가 Generator의 행동을 바꾼다
→ 처음부터 완벽X, 점진적 개선이 핵심

프롬프트 = 부탁 vs 하네스 = 강제

프롬프트 (부탁)

"좋은 코드 작성해 줘“

→ 보라색 그라데이션 배경
→ 흰색 카드 격자 나열
→ Inter, Roboto 기본 폰트
→ 히어로→기능→팀→CTA

AI slop 패턴 반복

VS

하네스 (강제)

generator.md AI slop 금지:

→ 독특한 색상 (모노크롬/네온)
→ 비대칭 레이아웃, 오버랩
→ 타이포를 디자인 요소로
→ 스크롤 효과, 마이크로 애니

evaluation_criteria.md로 채점

에이전트가 규칙을 어겼을 때 프롬프트를 고치는 게 아닙니다.
그 실패가 구조적으로 반복 불가능하게 하네스를 아예 고치는 겁니다.

하네스의 3가지 기둥

1

컨텍스트 파일

AGENTS.md / codex.md
AI가 매 세션 시작 시 가장 먼저 읽는 파일
60줄 이하, 지도를 줘라 (1000p 설명서 X)
실수할 때마다 한 줄씩 추가 · 점진적 개선

2

자동 강제 시스템

린터 + 프리커밋 훅
코드 저장 직전 자동 검사 스크립트
에러 시 에이전트가 스스로 수정
사람 개입 없이 자동 교정 루프

3

가비지 컬렉션

주기적 청소 에이전트
AI가 만든 나쁜 패턴을 자동 감지/수정
문서-코드 불일치 탐지
실수 → 새 규칙 추가 → 하네스 진화

실수할 때마다 하네스가 점점 더 정교해집니다 · 울타리가 점점 높아지는 것

자동 교정 루프의 원칙

[원칙] 성공은 조용히, 실패만 시끄럽게

테스트 통과 → 아무 말도 안 함
테스트 실패 → 에이전트에게 알림

통과한 테스트 결과 4,000줄을
다 보여주면 AI가 정작 할 일을
잊어버립니다.

[원칙] 자동 교정 루프

1. 에이전트가 코드 작성
2. 린터/훅이 자동 검사
3. 에러 발견 시 에이전트에게 반환
4. 에이전트가 스스로 수정
5. 사람 개입 없이 반복

[주의] Garbage In, Garbage Out

문제 정의나 기획 자체가 구리면 아무리 하네스를 거쳐도 좋은 결과물이 나오지 않습니다.
하네스는 좋은 걸 더 잘 만들게 도와주는 것이지, 별로인 걸 좋게 만드는 마법이 아닙니다.

실습: Codex 하네스 프로젝트 실행

```
$ cd harness-project && codex

"AI 교육 전문 회사 사용성연구소의 랜딩페이지를 만들어줘"
```

Step 1	Step 2	Step 3	Step 4	Step 5
Planner	**Generator**	**Evaluator**	**재작업**	**합격!**
SPEC.md 생성 기능 8개+ 설계	output/ index.html SELF_CHECK.md	QA_REPORT 채점 + 판정	피드백 반영 index.html 수정	완료 보고 start output/ index.html

같은 모델, 같은 프롬프트 · 다른 건 구조뿐

1

Planner가 기능 범위를 확장

사람 5개 기능 → AI 10개 이상 설계. 출발점이 다름

2

평가 기준표가 마인드셋을 바꿈

디자인 40% + "AI 슬롭이면 불합격" → 처음부터 레트로 픽셀 아트 시도

3

Evaluator가 실제 버그를 잡음

MCP로 직접 플레이 검증. 피드백 루프로 품질 수렴

멱등성(Idempotency)이란?

같은 입력 → 같은 출력을 보장하는 성질

클로드를 쓰든 코덱스를 쓰든 제미나이를 쓰든, 하네스 엔지니어링을 잘 활용하면
다른 AI 모델을 써도 같은 아웃풋으로 나올 수 있게끔 방향을 강제할 수 있다. · 현업 엔지니어

엔터프라이즈 AX 현장

실제 고객사 프로젝트에서
하네스 엔지니어링을 적용하여
멱등성을 검증한 사례

포워드 디플로이드 엔지니어링

고객사에 직접 파견되어
제로투원 프로젝트를 수행하면서
린터 기반 멱등성 하네스를 실증

실전 예시: 배달 플랫폼 4개 소프트웨어

고객용 앱

고객이 주문할 때
사용하는 앱

기사용 앱

음식 픽업 및
고객 전달 도구

음식점용 앱

주문 수락 및
조리 완료 알림

어드민

전체 배달 플랫폼
관리 시스템

고객의 막연한 니즈 → FDE가 구조적으로 분해

처음에 고객은 "주문 앱 하나만 있으면 되지"라고 생각합니다.
대화를 하다 보면 기사앱, 음식점앱, 어드민까지 4개가 필요하다는 걸 알게 됩니다.
이 4개 소프트웨어를 3개 모델(Claude/Codex/Gemini)에 각각 시켜서 12개 아웃풋을 비교합니다.

엔터프라이즈 하네스 6단계 플로우

1	2	3	4	5	6
요구사항	**플랜/CPS**	**아키텍처**	**코드 설계**	**린터 강제**	**검토/반복**
고객 미팅 비정형 인풋	Context-Problem -Solution 정리	도메인 모델 DDD 설계	AI가 알아듣는 코드 레벨 설계	파일명/메소드명 임포트 순서 강제	결과 검토 피드백 루프

린터로 파일명, 메소드명, 임포트 순서까지 강제

코드의 디자인 시스템 = 브랜드 가이드에 맞춘 디자인 시스템과 같은 원리
어떤 AI 모델을 써도 동일한 아웃풋이 나오도록 구조를 강제합니다.

멱등성: 어떤 모델이든 같은 아웃풋

Claude

같은 인풋

Codex

같은 인풋

Gemini

같은 인풋

동일한 아웃풋

하네스(린터 + 디자인 시스템 + 아키텍처)가
모델의 자유도를 제어해서 일관된 결과물을 강제합니다.

모델 랭킹이 바뀌더라도 하네스 기법은 유효합니다.

CPS 프레임워크

C **Context**

다 같이 싱크를 맞추고 있는
깔고 있는 맥락은 무엇인가

P **Problem**

고객의 문제가 무엇인가
어떤 맥락에서 겪고 있는가

S **Solution**

어떤 솔루션을 같이
만들어야 하는가

팔란티어의 Mud Session에서 영감

- 매번 문제 발견 & 솔루션 교환 시 CPS 업데이트
- 마크다운 → LaTeX 템플릿 → PDF로 빌드하여 고객 전달
- 미팅 로그 N개 → 종합 CPS 1개 → PRD 1개로 수렴
- 고객도 세계관을 이해해야 더 좋은 아이디어를 줄 수 있다

설계 분석: 원칙 → 정의 → 구조 → 코드

원칙 (Principles)

Human-in-the-loop,
할루시네이션 제거

로그 추적 등 대전제
원칙 수립

정의 (Definitions)

고객사와 합의
한 개념 정의

도메인 하이어
라키 결정

구조 (Structure)

개념 하이어라키
기반 아키텍처

데이터 주고받는
방식 설계

필연적 결과물

설계가 코드를 결정

어떤 AI에게 시켜도
같은 방향

도메인 드리븐 디자인(DDD) 기반 설계

원칙 2개 + 정의만 내려도 하이어라키를 정할 수 있고,
구조에 맞게 데이터 설계를 하면 결과물이 필연적 방향으로 수렴합니다.
코드 작성 노력이 극적으로 줄어든 시대 · 설계가 곧 실력입니다.

린터: 코드 아웃풋을 구조적으로 강제

파일명 강제

restaurants.tsx (O)
restaurant-list.tsx (X)
도메인 복수형 → 단수형으로 네이밍

임포트 순서

알파벳 순 정렬 강제
임포트 가능/불가능 파일 구분
배럴 파일 규칙 적용

네이밍 연쇄

파일명이 강제되면
하위 클래스/메소드명도 연쇄 결정
코드 전체의 필연적 방향

커밋 시 자동 실행

커밋 시도 시 린터 자동 실행
프롬프트로 못 잡는 부분까지
구조적으로 검증

린터 기반 하네스: 장점과 Trade-off

장점

- 디자인 시스템처럼 코드에 시스템 강제
  → 일관된 품질 보장

- 버그 시 파일 통째 삭제 후 재생성 가능
  (역할이 명확하므로)

- 코드 리뷰어가 작성 의도 즉시 파악

- AI에게 재구현 시키기도 매우 쉬움

- 유지보수 인수인계 시 휴먼 리더블 보장

Trade-off

- AI가 이해하기 편한 스타일과 다를 수 있음
  → 토큰 비효율 가능

- 그러나 이 단점을 넘어서는 장점:
  인간과의 협업을 위한 것

- 유지보수 팀이 바이브 코딩을 안 할 수 있음

- 인터넷 안 되는 환경에서 코딩해야 할 수도
  → 완벽히 패키징된 소스를 줘야 한다

핵심 통찰: 바이브 코딩으로 엔터프라이즈에 딸깍한 다음 유지보수까지 넘어간 사례는 아직 없다.
유지보수를 일반 SI 팀이 받아야 할 수도 있으니 휴먼 리더블 코드가 필수다.

검증 결과: 4x3 = 12개 아웃풋 모두 동일

	Claude	Codex	Gemini
고객앱	동일 [OK]	동일 [OK]	동일 [OK]
기사앱	동일 [OK]	동일 [OK]	동일 [OK]
음식점앱	동일 [OK]	동일 [OK]	동일 [OK]
어드민	동일 [OK]	동일 [OK]	동일 [OK]

디자인 시스템 + 린터 + 도메인 고정 = 모델 무관 동일 아웃풋

Evaluation + '포맷' 일화

조직 맞춤 Evaluation

기본 4지표: 문맥 적합성, 정확도,
소스 검증, 누락 검사

조직마다 기준이 다르다:
- 100번 중 1번 틀려야 OK?
- 20번 틀려도 답이라도 해야 OK?

'에이전트 도그' · 데이터 도그처럼
매일매일 AI 품질 모니터링

'포맷' 일화

하드웨어 프로젝트에서 펌웨어(C 코드)
구현 중 문제 → 멘토: "포맷을 해오세요"

윈도우 재설치, 컴파일러 재설치,
예제 코드부터 다시 실행...

제로로 돌아갔을 때 결과물로 가는
단계가 멱등이어야 한다는 것을 체감

앞단(설계) + 뒷단(평가)이 다 잡혀야 진짜 엔터프라이즈 AI

하네스에서 산출물을 꽉 잡아놓는 것도 중요하지만, 진짜 잘 만들어졌는지 체크하는 것도 중요합니다.
만든 사람도 트래킹, 클라이언트도 안심 → 지속 가능한 유지보수로 연결됩니다.

AI 활용 방식의 진화 3단계

1

프롬프트 엔지니어링

AI에게 명령을 잘하는 기술
구체적 프롬프트로 결과를 달라지게 함
한계: 프로젝트 상황을 모르면 엉뚱한 코드

2

컨텍스트 엔지니어링

프로젝트 구조, 코드 스타일 등 배경 정보 제공
MCP, 스킬로 도구 확장
한계: 스킬이 수백 개 쌓이니 AI가 오히려 혼란

3

하네스 엔지니어링

도구를 얹는 게 아니라 환경 자체를 설계
AI가 똑똑하게 일할 수 있는 시스템을 구축
프롬프트(부탁) → 하네스(강제)

엔트로픽의 핵심 원칙

**"하네스가 없어지는 건 아니라,
하네스 공간이 이동할 뿐이다."**

-- Anthropic Engineering Blog

모델이 좋아질수록

하네스 규칙 감소
하드코딩 규칙 줄이기

더 어려운 과제를 할 때

새로운 하네스 필요
환경 설계가 더 중요

하네스 엔지니어링의 적용 범위

게임 개발

플랫포머, RPG
브라우저 게임

웹사이트 제작

랜딩 페이지
웹 서비스

엔터프라이즈

배달 플랫폼
사내 시스템

업무 자동화

반복 작업
워크플로우

실전 사례: 배달 플랫폼 하나만 해도 고객앱/기사앱/음식점앱/어드민 4개 소프트웨어가 필요
Planner-Generator-Evaluator 패턴은 범용 · 채점 기준표만 바꿔주면 됩니다.

하네스 엔지니어링의 미래

엄밀함의 재배치 (Chad Fowler)

코드 한 줄 한 줄의 엄밀함 → 시스템 설계의 엄밀함으로 이동
선수에서 감독으로 · 더 높은 차원의 기술이 요구됨

에이전트가 스스로 하네스를 구축

미래에는 에이전트가 작업 전에 환경 구성부터 먼저 살피고
그걸 설정한 다음 작업을 시작하는 방식으로 진화

하네스 = 미래의 서비스 템플릿

기술 스택 선택 기준이 '좋은 DX'에서 '좋은 하네스'로 바뀔 수 있음
잘 구성된 환경과 검증은 모델 성능과 무관하게 효과적

OpenHarness: 하네스를 코드로 구현하면?

"The model is the agent. The code is the harness."

모델이 에이전트고, 코드가 하네스다. — OpenHarness (HKUDS, 홍콩대학교)

OpenHarness란?

Claude Code, Codex 같은 코딩 에이전트의
내부 구조를 오픈소스로 풀어놓은 것

오픈소스 Python 구현체
연구자, 빌더, 커뮤니티를 위한 프로젝트

oh 명령어 하나로 실행
어떤 LLM 백엔드든 연결 가능
(Claude, Kimi, Ollama, OpenAI...)

왜 중요한가?

우리가 배운 하네스 3기둥이
실제 코드로 어떻게 구현되는지 볼 수 있음

모델은 지능을 제공하고
하네스는 손, 눈, 기억, 안전 경계를 제공

Claude Code의 3%의 코드로
80%의 핵심 기능을 구현
github.com/HKUDS/OpenHarness

OpenHarness 10개 서브시스템 구조

engine/

에이전트 루프
질의→스트리밍→도구→반복

tools/

43개 도구
파일, 셸, 검색, MCP

skills/

온디맨드 지식 로딩
.md 파일 기반

hooks/

라이프사이클 훅
PreToolUse/PostToolUse

permissions/

안전 모드
경로 규칙, 명령어 차단

memory/

영구 기억
세션 간 지식 유지

coordinator/

멀티에이전트
서브에이전트 스포닝

prompts/

컨텍스트 조립
CLAUDE.md, 스킬 주입

commands/

54개 명령어
/plan, /commit ...

mcp/

MCP 클라이언트
외부 도구 연결

우리가 배운 3기둥이 코드로: skills/ = 컨텍스트 파일 | hooks/ = 자동 강제 | coordinator/ = 멀티에이전트 조율

OpenHarness: 설치부터 실행까지

Step 1: 원커맨드 설치

```
# 자동 설치 (OS 감지, 의존성 체크)
curl -fsSL https://raw.githubusercontent.com/
  HKUDS/OpenHarness/main/
  scripts/install.sh | bash

# 또는 소스에서
git clone ...OpenHarness.git
cd OpenHarness && uv sync --extra dev
```

Step 2: LLM 백엔드 연결

```
# Anthropic (Claude)
export ANTHROPIC_API_KEY=sk-ant-...

# Kimi
export ANTHROPIC_BASE_URL=
  https://api.moonshot.cn/anthropic
export ANTHROPIC_MODEL=kimi-k2.5

# GitHub Copilot (무료!)
oh auth copilot-login
```

Step 3: 실행!

```
oh                                  # 대화형 실행
oh -p "이 코드베이스 설명해 줘"      # 원샷 실행
oh -p "버그 고쳐줘" --output-format json  # JSON 출력 (자동화용)
oh -p "테스트 수정" --output-format stream-json  # 실시간 스트리밍
```

OpenHarness 커스터마이징: 3가지 확장 포인트

skills/*.md

.md 파일 하나가 곧 스킬

예시:
skills/finance.md
skills/devops.md
skills/frontend.md

에이전트가 필요할 때만
자동으로 로딩
(온디맨드)

hooks/hooks.json

도구 사용 전후에
자동 실행되는 검증

PreToolUse:
도구 쓰기 전에 검사

PostToolUse:
도구 쓴 후에 검증

우리의 린터/프리커밋 훅과
같은 개념!

agents/*.md

.md 파일로 서브에이전트 정의

우리 프로젝트와 대응:
planner.md = Planner
generator.md = Generator
evaluator.md = Evaluator

멀티에이전트 조율
서브에이전트 스포닝

핵심: .md 파일과 .json 설정만으로 하네스를 확장한다 — 코딩 없이!

OpenHarness: 안전 장치 — permissions/

Ask 모드

모든 위험 작업에
사람 확인 요청

"이 파일 삭제해도 될까요?"
[Y/N]

기본값 — 안전 우선

Trust 모드

모든 도구 호출
자동 승인

oh --trust

빠르지만 위험

숙련자/자동화용

Deny 모드

특정 명령어/경로
원천 차단

rm -rf / 같은
위험 명령 차단

엔터프라이즈 필수

우리 하네스 프로젝트와의 연결

evaluator.md의 "절대 관대하게 보지 마라" = 소프트 규칙 (프롬프트)
permissions/의 경로 규칙 + 명령어 차단 = 하드 규칙 (구조적 강제)
둘 다 필요합니다. 프롬프트로 방향을 잡고, permissions로 울타리를 칩니다.

우리 프로젝트 vs OpenHarness: 1:1 매핑

우리 하네스 프로젝트	OpenHarness 대응	역할
AGENTS.md	prompts/ + engine/	오케스트레이터, 실행 루프 제어
planner.md	agents/*.md	서브에이전트 역할 정의
generator.md	agents/*.md + tools/	코드 생성 + 43개 도구 활용
evaluator.md	agents/*.md + hooks/	검수 역할 + 자동 검증 훅
evaluation_criteria.md	skills/*.md	공유 지식으로 온디맨드 로딩
SPEC.md / QA_REPORT.md	memory/	세션 간 영구 기억으로 유지
"AI slop 금지"	permissions/	경로 규칙 + 명령어 차단으로 강제

우리가 .md 파일 6개로 한 것을, OpenHarness는 10개 서브시스템 코드로 구현한 것

Codex 하네스 프로젝트 구조

프로젝트 폴더 구조

```
harness-project/
  AGENTS.md          <- 오케스트레이터
  agents/
    planner.md
    generator.md
    evaluator.md
    evaluation_criteria.md
  output/            <- 결과물
  SPEC.md / SELF_CHECK.md / QA_REPORT.md
```

실행 방법

1. cd harness-project
2. codex (또는 cmd /c codex)
3. 프롬프트 한 줄 입력
4. AGENTS.md가 자동으로
   3-Agent 파이프라인 실행

다른 과제? 프롬프트만 바꾸면 됨
기준 변경? evaluation_criteria.md만 수정

운영 규칙 (AGENTS.md)

- 역할 문서에 없는 임의의 완화 기준을 만들지 마라
- Generator는 피드백을 합리화하지 말고 그대로 반영한다
- Evaluator는 관대하게 보지 마라
- 파일 누락 시 다음 단계로 넘어가지 말고 먼저 확인

핵심 정리 7가지

1. **프롬프트는 부탁, 하네스는 강제**
   실수가 불가능한 구조를 설계하라

2. **하네스 3기둥: 컨텍스트 파일 + 자동 강제 + 가비지 컬렉션**
   AGENTS.md + 린터/훅 + 청소 에이전트

3. **린터로 멱등성 확보 · 어떤 모델이든 같은 아웃풋**
   실증: 4 소프트웨어 x 3 모델 = 12개 동일 결과

4. **앞단(설계) + 뒷단(평가) 모두 잡아야 엔터프라이즈**
   CPS → PRD → 코드+린터 → Evaluation

5. **유지보수를 위한 휴먼 리더블 코드가 필수**
   바이브 코딩으로 엔터프라이즈 유지보수까지 간 사례는 없다

6. **성공은 조용히, 실패만 시끄럽게**
   자동 교정 루프 + 점진적 하네스 진화

7. **일단 만들고, 실패하면 하네스를 손봐라**
   GIGO 주의 · 기획이 좋아야 결과도 좋다

AI 코딩 에이전트가
기대만큼 동작하지 않을 때

모델을 탓하기 전에
하네스를 점검해 보세요.

AGENTS.md에 뭘 넣었는지?
evaluation_criteria.md는 있는지?
Generator/Evaluator가 분리되어 있는지?