AGENT KOREA · KEYNOTE 04

AI를 도구로 쓰다가,
운영체계를 만들었다

완료를 의심하고, 검증하고, 어긋나면 **다음 실행을 바꾸는 루프**를 설계할 수 있는가

HUE

ACT 01 / 05 · 질문

질문

THE QUESTION

좋은 구조는 좋은 결과를 **보장하는가**. 발표 전체를 끌고 가는 **하나의 질문**에서 시작한다.

앞의 세 발표가 만든 흐름

네 개의 키노트는 하나의 질문으로 이어진다

ALEX AI

도구보다 구조가 중요하다

이 발표가 받는 질문

좋은 구조가 있으면 결과가 보장되는가?

공냥이

구조는 사람과 조직의 언어로 번역돼야 한다

이 발표가 받는 질문

그 언어가 실행에서 지켜졌는지 어떻게 확인하나?

류주임

구조는 도메인 문제에서 사례가 된다

이 발표가 받는 질문

그 사례가 완료처럼 보이는 미완성인지 어떻게 검증하나?

이 발표

구조를 운영 루프로 닫는다

이 발표의 답

결과를 믿는 것이 아니라, 의심하고 검증하는 체계여야 한다

앞 발표가 **구조의 필요성**을 만들었다면, 이 발표는 **구조와 결과 사이의 검증 루프**를 닫는다.

연사 소개

이 루프를 직접 설계하고 운영하는 사람

Hue · Hugh Kim

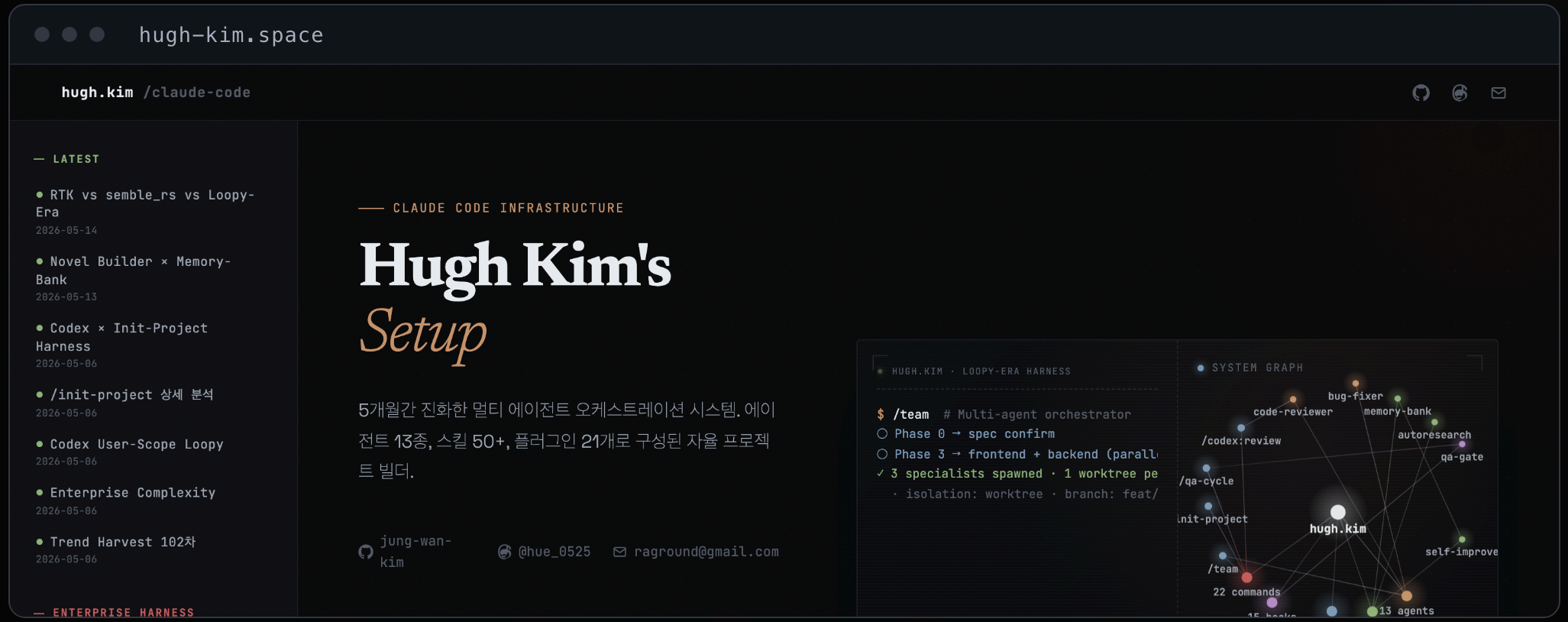
20년차 개발자

건설회사 **AX 프로젝트 테크리더** · **AX Harness System** 개발 담당

지금	현장의 AI 협업을 도구 묶음이 아니라 **검증되는 운영체계(harness)**로 설계·운영합니다
20년	오래 쌓은 엔지니어링 감각 위에, **AI를 운영하는 방법**을 다시 설계하고 있습니다

● 블로그 · HUGH-KIM.SPACE

Agentic Coding 블로그



hugh-kim.space — harness·Memory Bank·자가진화 루프 설계를 기록하는 블로그

github.com/jung-wan-kim

이 발표가 던지는 질문

모델이 발전하고, claude code 같은 범용 harness가 발전해도, 변하지 않는 것이 무엇인가?

이 발표는 그 하나의 답을 찾는 과정입니다.

우리는 이미 다 겪었다

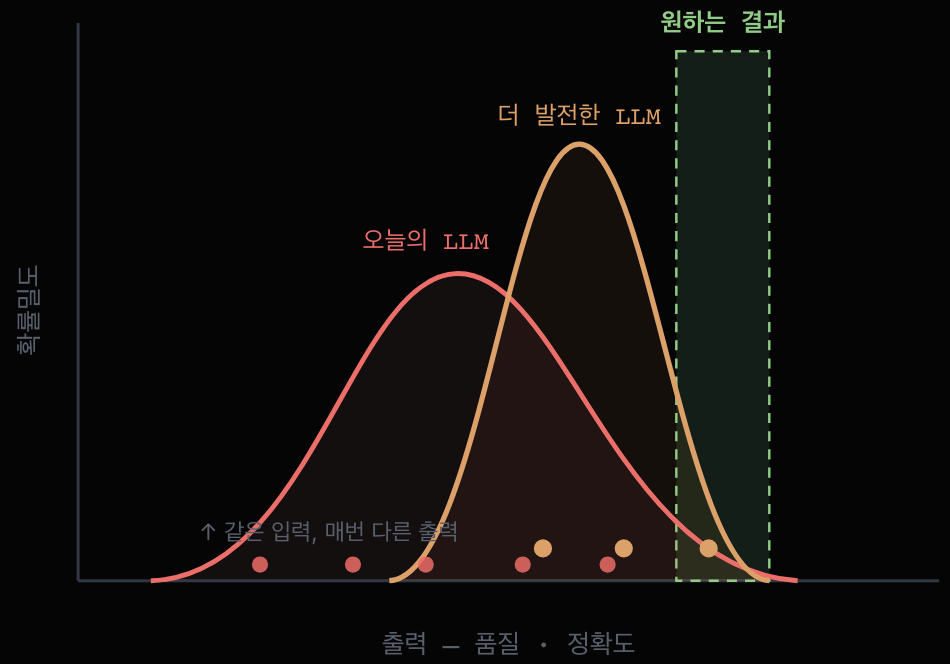
우리는 무엇이든 만들어주는 harness를 쓴다. 그런데 한 번의 프롬프트로 ~~내가 원하는 단 하나의 애플리케이션~~은 나오지 않는다.

`Claude Code one-shot ✗` `Codex one-shot ✗` `그리고 그 다음 모델도 ✗`

> 도구가 약해서가 아니다. **한 번의 생성으로 끝나는 구조**여서다. 우리는 이미 수없이 다시 시켰고, 고쳐 시켰고, 또 어긋났다.

한 번으로 안 되는 진짜 이유

LLM은 매번 확률에 기인해 결정한다



아무리 발전해도 분포가 좁아질 뿐, 결정론이 되지는 않는다

① 같은 입력도 매 실행 **다른 출력** — 확률 분포에서 샘플링한다

② 모델이 좋아지면 분포가 **좁아질 뿐**, 목표에 매번 정확히 떨어지진 않는다

③ 일관된 결과·성능은 **모델의 몫이 아니라 바깥 구조의 몫**이다

> 그래서 일관성은 더 좋은 프롬프트가 아니라,
> 결과를 검증하고 교정하는 루프에서 나온다.

좋은 구조가 있어도 결과는 어긋난다

구조는 선언이고, 결과는 실행 후 검증해야 하는 상태다

선언된 것

STRUCTURE

좋은 구조 · 좋은 지침 · 좋은 계획

실제 일어난 것

EXECUTION

컨텍스트·우선순위·도구 흐름 속 실행

여기서 어긋난다

믿으면 안 되는 것

RESULT

완료 메시지 ≠ 검증된 결과

ACT 1 결론 *좋은 구조는 좋은 결과를 자동으로 보장하지 않는다.*

ACT 02 / 05 · 공감

공감

THE SYMPTOM

완료처럼 보이는 **미완성**. 실행에서 **반복된 문제들**을 함께 들여다본다.

우리가 LLM에게 당한 경험

이건 실패담이 아니라, 실행 중에 반복된 5가지 문제다

1	LLM은 **"완료했습니다"**라고 말하지만 실제로는 미완성일 수 있다	**그래서** 완료 메시지를 믿을 수 없다
2	강한 **MD 지침**을 적어도 반복 실행 중에 누락하거나 우회한다	**그래서** 규칙이 SOFT하게 무너진다
3	지적하면 **사과**하지만, 사과는 다음 실행 조건을 바꾸지 않는다	**그래서** 같은 실패가 반복된다
4	막히면 요구사항을 줄이고 기준을 낮추고 **UI만 보고 성공 판정**한다	**그래서** 미완성이 완료로 위장된다
∴	결과와 완료 메시지는 **기본적으로 의심하고 검증**해야 한다	**그래서** 의심이 기본값이 된다

"완료했습니다"는 증거가 아니다

LLM의 완료 메시지는 실제 동작 증거와 다르다

CLAIM · 모델이 말하는 것

"완료했습니다"

~ 자연어 선언

~ 다음 단계로 넘어가려는 신호

~ 스스로 검증하지 않은 상태

≠

EVIDENCE · 통과해야 하는 것

실제 동작 증거

✓ 빌드·타입체크 통과

✓ 실제 인터랙션 / DB 정합성

✓ 다른 검증 루프의 PASS

앵커 2

LLM의 완료 선언은 증거가 아닙니다.

완료처럼 보이는 미완성

보이는 것은 완료, 실제는 미완성 — 5가지 증상

보인다
UI가 그려진다
↓
하지만
기능이 연결되지 않았다

보인다
테스트 PASS
↓
하지만
실제 행동은 안 된다

보인다
API 200 OK
↓
하지만
DB에 데이터가 누락

보인다
막히면 완료
↓
하지만
요구사항을 몰래 줄였다

보인다
성공 판정
↓
하지만
UI만 보고 통과시켰다

그래서 결과와 완료 메시지는 **기본적으로 의심하고 검증해야 한다.**

왜 MD에 적어도 어길까

MD 지침은 SOFT하다 — 실행 흐름 속에서 갉아먹힌다

CLAUDE.MD · 강한 지침

절대 규칙

- ~~localStorage 금지~~
- ~~완료 전 QA 필수~~
- ~~테스트 없이 커밋 금지~~

→ 읽히지만, 실행 중 누락

SOFT 지침이 무너지는 3가지 이유

1 **컨텍스트**
긴 대화에서 앞쪽 지침이 윈도우 밖으로 밀려난다.

2 **우선순위**
당장의 태스크 완수가 배경 규칙보다 앞선다.

3 **도구 흐름**
여러 도구를 거치는 동안 규칙이 끼어들 지점이 없다.

왜 매번 사과만 반복할까

사과는 상태 변경이 아니다

01 틀렸다고 **지적한다**

02 모델은 **사과한다**

03 그러나 다음 실행 **조건은 그대로다**

04 그래서 **같은 실패가 반복**된다

∴ **실행 조건을 바꿔야** 루프가 끊긴다

ACT 2 결론 *LLM의 완료 선언은 출발점이지, 완료의 증거가 아니다.*

~~더 강한 프롬프트를 쓰면 되지 않을까?~~

↓

완료 선언을 검증하는 구조가 필요하다

ACT 03 / 05 · 설계

설계

THE DESIGN

보장 대신 **검증하는 구조**. 완료를 **증거로 통과**시키는 계약을 설계한다.

보장 대신 검증 계약

좋은 구조는 결과를 보장하지 않고, 결과가 증거를 통과하게 만든다

선언

DONE

모델이 "완료" 선언

→

제출해야 하는 것

EVIDENCE

빌드 · 인터랙션 · DB · 스크린샷

→

통과시켜야 하는 것

REVIEWER

독립 검증 루프가 PASS 판정

앵커 3

완료는 말이 아니라 계약입니다.

무엇이 증거인가

완료를 통과시키려면 재현 가능한 증거가 필요하다

증거 01

빌드 · 타입 체크

구문·타입이 **깨지지 않았는지** 결정론적으로 차단

증거 02

실제 인터랙션

버튼이 그려짐 ≠ **동작함** — 클릭·입력·제출까지 확인

증거 03

DB 정합성

API 200 ≠ **데이터 저장** — 재조회로 정합성 확인

증거 04

스크린샷 시각 확인

DOM 존재 ≠ **사용자가 본 화면** — 렌더 결과를 눈으로 검증

증거 05

크로스 모델 리뷰

단일 모델의 **맹점**을 다른 모델(Codex)이 교차 검증

증거 06

회귀 테스트

새 변경이 **과거의 PASS**를 깨지 않았는지 재실행

증거는 "보이는 것"이 아니라 **재현·검증 가능한 것**이다.

SOFT 지시와 HARD GATE

중요한 기준은 읽히는 문장이 아니라 실행을 멈추는 조건이다

SOFT · 지시

"QA 없이 커밋하지 마세요"

```
# CLAUDE.md 한 줄
- 완료 전 QA 필수
# → 모델이 읽지만, 무시 가능
```

- × 문맥에 따라 누락된다
- × 위반해도 진행이 멈추지 않는다

HARD · GATE

EXIT 2

QA PASS 없으면 push 자체가 차단

```
# qa-gate-before-push.sh
[ -f .qa-cycle-passed ] || exit 2
# → 통과 못 하면 실행이 멈춘다
```

- ■ 우회 불가능한 실행 조건
- ■ exit code로 결정론적 차단

기억은 저장이 아니라 다음 실행 조건이다

과거 기록이 아니라 다음 행동을 바꾸는 장치

memory-bank

대화·결정·실패를 검색 가능한 사실로

+

rules / scaffold

반복 마찰을 규칙·hook으로 결정화

다음 실행 조건

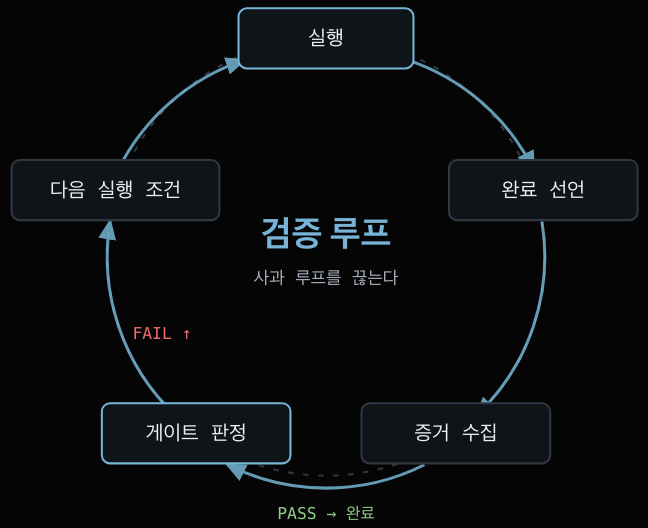
다음 세션의 행동을 사전에 바꾼다

앵커 7

기억은 저장이 아니라 다음 실행 조건입니다.

검증 루프 — 발표 전체의 한 그림

구조는 결과를 의심·검증·교정하는 닫힌 루프를 만든다



사과 루프	지적 → 사과 → **조건 그대로** → 같은 실패 (Act 2)
01 실행	실행하고 **완료를 선언**한다
02 검증	증거를 모아 **게이트가 PASS/FAIL**을 판정한다
03 교정	FAIL이면 **다음 실행 조건을 바꾼다**
04 기억	바뀐 조건이 **memory로 다음 실행에 주입**된다
∴	**사과가 아니라 조건이 바뀌어** 루프가 닫힌다

앵커 6

좋은 구조는 결과를 믿게 만드는 것이 아니라, 결과를 의심하고 검증하게 만듭니다.

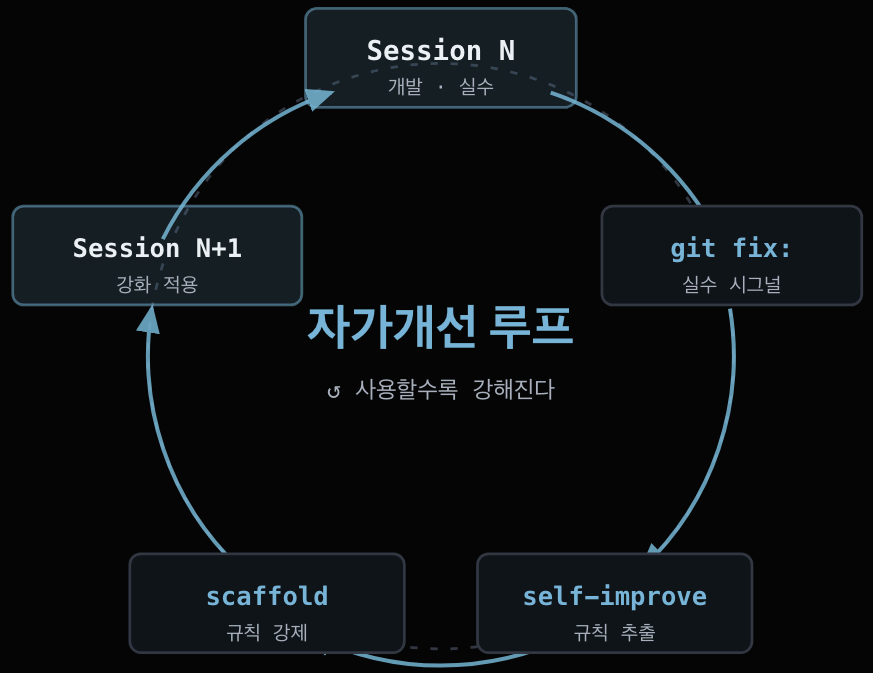
HARNESS는 개인 AI WORK OS다

도구의 묶음이 아니라 하나의 운영체계

ACT 3 결론 *좋은 구조는 결과를 믿게 만드는 것이 아니라, 결과를 검증하게 만든다.*

시스템 심화 · LOOPY-ERA 자가진화 파이프라인

실수가 다음 실행 조건을 바꾸는 세 개의 루프



왜 만들었나

전통 harness는 **엔트로피가 증가**하고 규칙이 노후화된다. Karpathy의 Loopy Era — 사람이 규칙을 쓰는 게 아니라 **AI가 자기 실수를 관찰해 규칙을 스스로 발견**하게 만든다.

① **자가개선 루프** — fix 커밋 → self-improve → scaffold 자동 추가

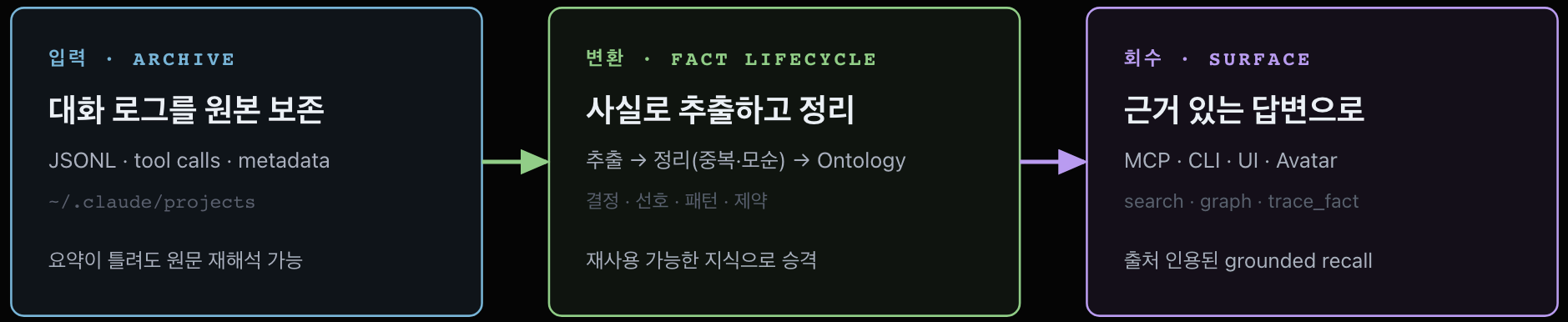
② **HARD 강제** — exit 2로 물리적 차단 (qa-gate·scaffold-violation)

③ **user-proxy QA** — 3-tool 교차 + Codex 이중 리뷰, 수렴까지 무정지

`43 rules` `9 HARD gates` `7 loops` `2,854 facts`

시스템 심화 · MEMORY-BANK 구조화된 장기 기억

대화를 저장하는 게 아니라, 다시 쓸 수 있는 지식으로 변환



왜 만들었나

단순 대화 검색을 넘어야 했다. **요약이 틀리면 원문 회수가 불가**하고, 프로젝트 의사결정의 변화를 추적할 수 없다 — 그래서 대화를 **재사용 가능한 지식 구조**로 변환한다.

- **Retrieval First** — 모든 기능은 과거 대화 재탐색 위에 성립
- **Fact Promotion** — 유의미한 세션을 reusable statement로 승격
- **Grounded Surface** — 생성 요약보다 출처 있는 회수 우선

`2,854 facts` `20 domains` `all-MiniLM-L6-v2`

MEMORY BANK · 7-STEP DATA FLOW

대화를 다시 쓸 수 있는 지식으로 — 7단계 수명주기

01–03 · ARCHIVE 원본 보존·회수 | 04–06 · FACT 지식으로 승격 | 07 · SURFACE 근거 회수

01 SYNC
동기화
src/sync.ts

대화 JSONL을 archive로 복사하고 증분 처리 대상을 만든다

→

02 INDEX
색인
src/indexer.ts

exchange를 SQLite·vector 테이블에 적재해 회수 기반을 만든다

→

03 SEARCH
검색
src/search.ts

semantic·text·hybrid로 과거 대화를 다시 찾는다

→

04 EXTRACT
추출
fact-extractor.ts

세션 종료 시 장기 보존 가치가 있는 사실을 뽑는다

→

05 CONSOLIDATE
정리
consolidator.ts

중복·충돌·진화 이력을 정리해 fact 품질을 지킨다

→

06 RELATE
연결
ontology-classifier.ts

domain·category를 붙이고 typed relation으로 연결한다

→

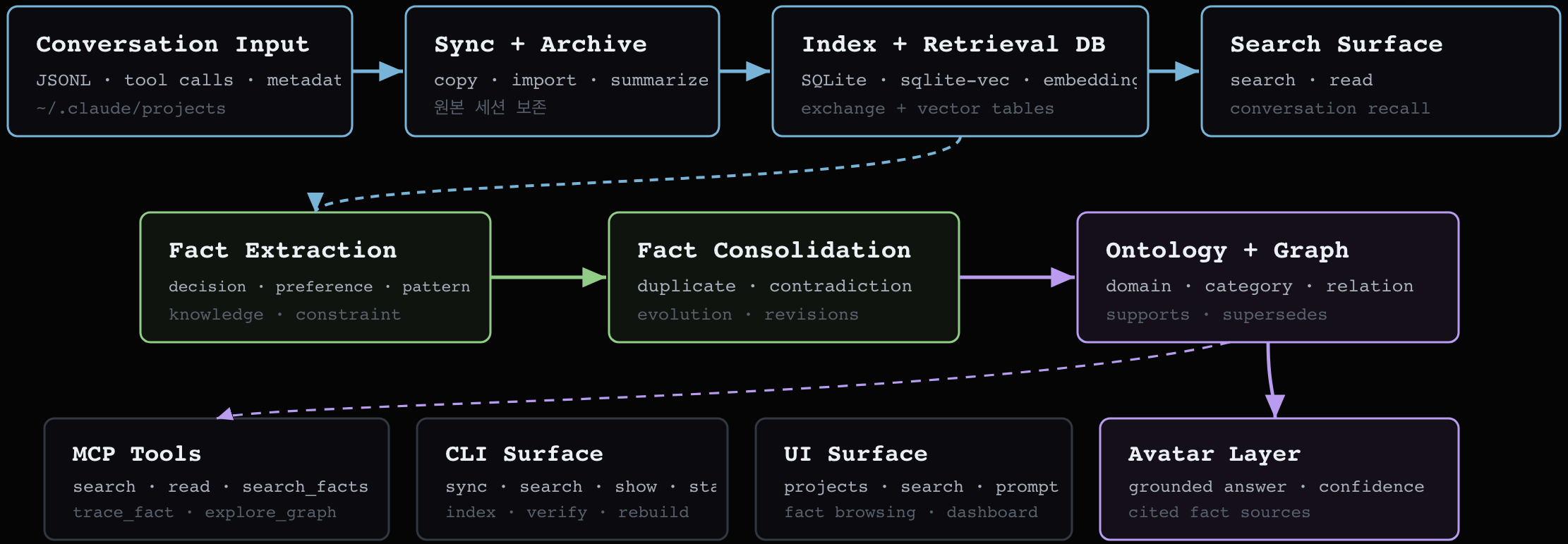
07 RECALL
회수
mcp-server.ts

MCP·CLI·UI·Avatar가 search·graph 인터페이스를 연다

context 관리는 저장이 아니라, **다음 실행에 주입할 지식의 수명주기 관리**다.

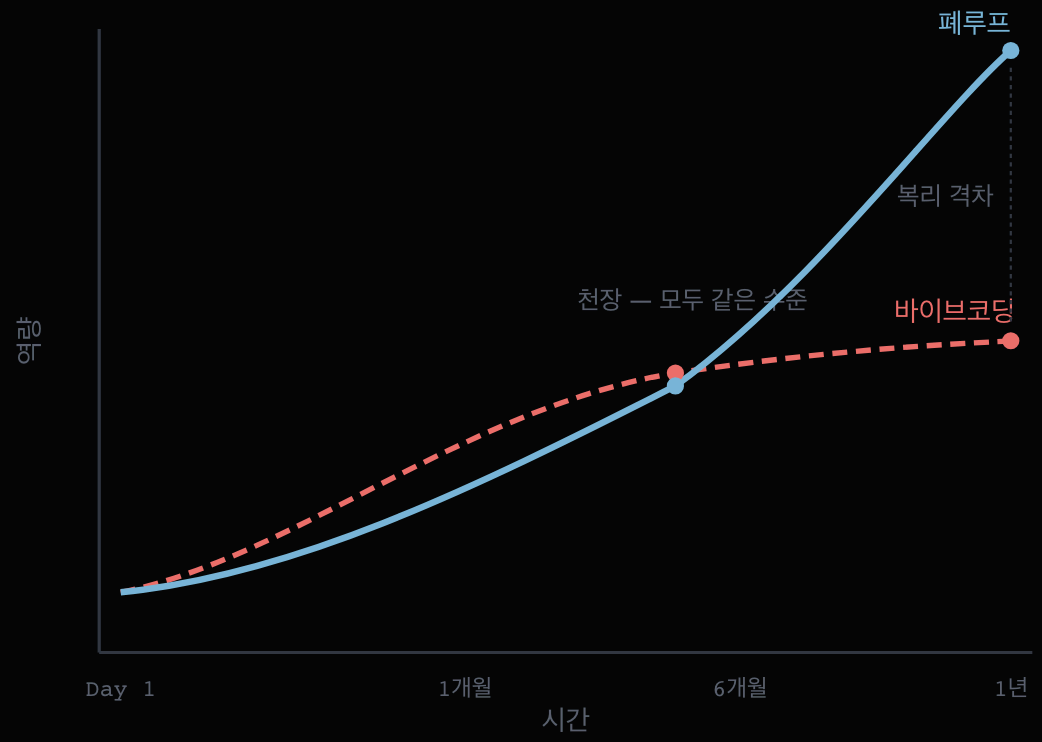
MEMORY BANK · ARCHITECTURE MAP

대화 로그에서 근거 있는 회수까지 — 전체 구조



시스템 심화 · SELF-EVOLVING 복리로 벌어지는 격차

프롬프트 한 줄 앱은 프롬프트 한 줄 앱과 경쟁할 수 없다



왜 만들었나

모두가 "AI한테 만들어달라"를 배운다. **프롬프트 한 줄로 만든 앱은 한 줄로 만든 다른 앱과 경쟁할 수 없다** — 둘 다 같은 수준이니까. 차이는 **폐루프**에서 난다.

① 실수 → **규칙 추출**

② HARD 강제(hook) → **효과 측정**

③ 강화 → 다음 세션에 **복리로 축적**

`33 hooks` `51 자동개선` `1년 후 NEVER DO 500`

ACT 04 / 05 · 증거

폐루프

THE CLOSED LOOP

검증·교정·기억이 맞물려 도는 **닫힌 루프**. harness와 hook이 그 루프를 **실제로 돌게** 만든다 — 그것이 증거다.

페루프 — CLOSED LOOP

한 번의 생성이 아니라 신호→수정→검증→확정이 돈다

SIGNAL

신호

완료 선언·사용자 수정·불만·실패가 신호로 잡힌다

→

PATCH

수정

코드를 고치고, 마찰을 규칙·hook으로 결정화한다

→

VERIFY

검증

증거를 모아 게이트가 PASS / FAIL을 판정한다

→

ACK / ROLLBACK

확정·되돌림

PASS는 확정, FAIL은 되돌리고 다음 실행 조건을 바꾼다

↺ FAIL이면 다시 — 사과가 아니라 **조건**이 바뀐다 — 세션이 끝나도 학습이 닫힌다 **(L6)**

레벨은 어디서 왔는가 · 계보

'harness 레벨'은 두 계보가 만나 만들어졌다

'레벨' 사고 SAE J3016 · 자율주행	자율주행의 **L0–L5 자율성 레벨**에서 출발 → AI 에이전트 자율성 레벨로 확장 (Swarmia · 학계 Feng·McDonald·Zhang, 5단계)
'harness' 개념 Martin Fowler · Birgitta Böckeler	ThoughtWorks가 **harness engineering**으로 명명 — 맥락·제약·피드백 루프의 설계. OpenAI도 같은 시기 독립적으로 같은 결론에 도달
핵심 명제 the harness, not the model	신뢰성은 모델 능력이 아니라 **그 둘레의 harness**가 만든다 — humans on the loop
이 발표의 확장 Hue · personal Work OS	그 계보 위에 **L6(페루프)·L7(Work OS)**를 개인 운영체계로 확장 — 다음 장이 그 레벨 정의다

HARNESS 레벨 — L0 → L7

어디까지 닫혔는가가 harness의 레벨을 정한다

L0–2	좋은 프롬프트 · 스킬 · 일부 자동화	세션 종료 후 학습이 안 닫힘
L3	완료 주장을 **evidence로 검증**	최소 기준
L4	Memory Bank 기반 **반복 실수 방지**	필수 substrate
L5	Soft rule을 **checkable · HARD**로 승격	핵심 운영 원칙
L6	**signal → patch → verify → ack/rollback** — 폐루프	**목표 최소선**
L7	**user · project · plugin · MCP · trend**까지 운영체계화 — Work OS	**지향점**

이 harness는 **L6(폐루프)를 넘어 L7(Work OS)**를 지향한다.

SOFT → HARD · HOOK PIPELINE

지침을 물리적으로 강제하는 hook 파이프라인

SOFT

자연어 지침

CLAUDE.md·SKILL.md — 무시·우회 가능

→

HARD

checkable hook

Edit·Write·Bash 이벤트에서 조건 검사

→

EXIT 2

BLOCK

위반 시 실행 자체를 차단 — 무시 불가

`qa-gate-before-push` `scaffold-violation-check` `code-quality-check` `no-localstorage` `no-env-commit` `portless-required`

`task-quality-gate` `SubagentStop · TaskCompleted`

SOFT → HARD

SOFT 체크가 2회 FAIL 하면 HARD로 자동 승격합니다. 규칙은 적는 게 아니라 강제되어야 합니다.

보여줄 것은 기능 시연이 아니라 페루프 증명이다

보여줄 장면의 기준이 다르다

✗

"만들었다"

멋진 기능을 시연한다. 한 번 잘 돌아가는 산출물을 보여준다. — 자랑이 된다.

✓

"의심하고 검증하고 다음 조건을 바꾼다"

완료 선언을 의심하고, 증거로 검증하고, 어긋나면 다음 실행 조건을 바꾼다. — 증거가 된다.

그래서 보여줄 장면은 **발표 질문에 답하는 증거 기준**으로 고른다.

ANDREJ KARPATHY · THE LOOPY ERA

프롬프트의 시대에서 루프의 시대로

지금까지 · PROMPT ERA

사람이 매번 더 좋은 프롬프트를 쓴다

출력이 어긋나면 **사람이** 다시 시키고 고쳐 시킨다. 루프를 사람이 돈다 — 학습은 사람 머릿속에만 남는다.

KARPATHY의 LOOPY ERA

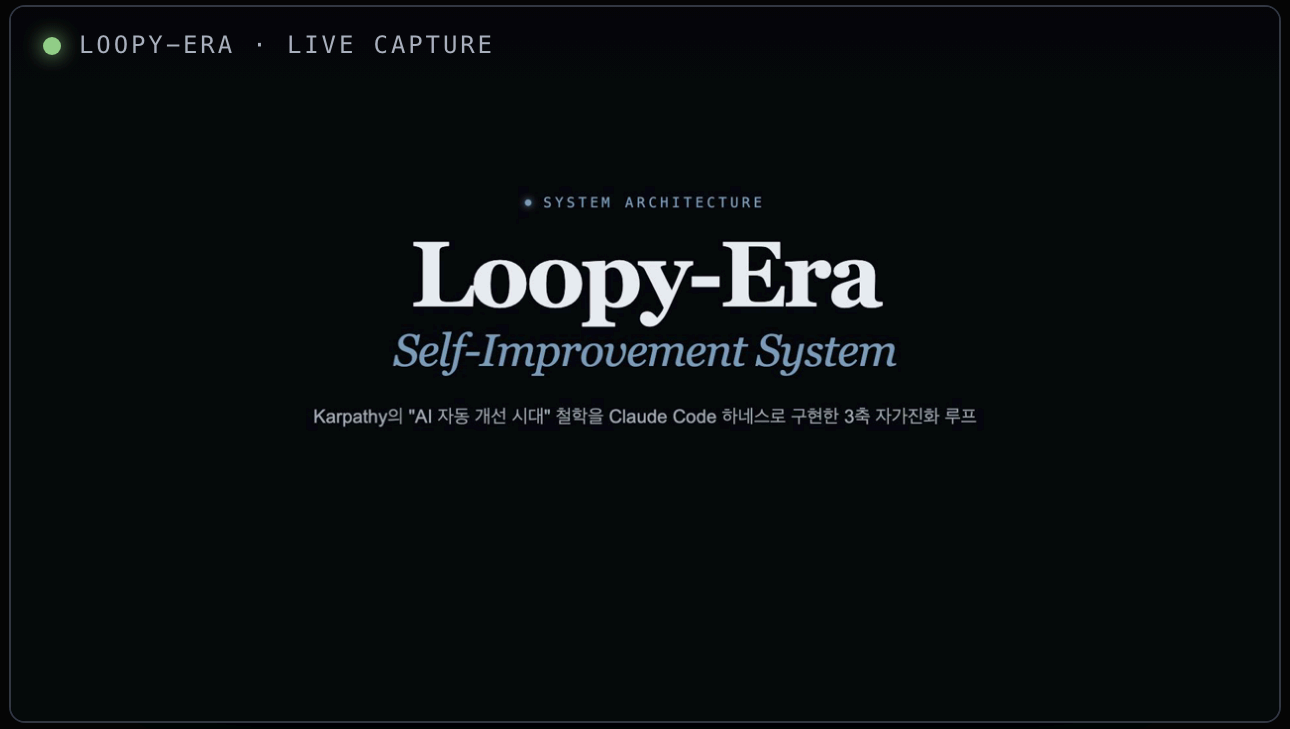
시스템이 자기 실수를 관찰해 스스로 강해진다

실패가 **신호**가 되어 규칙·스킬을 자동 발견하고 다음 실행을 강화한다. **루프를 시스템이 돈다** — 학습이 코드·메모리에 복리로 쌓인다.

그래서 차이는 모델 능력이 아니라 **루프(harness)가 복리로 쌓이는가**에서 난다 — 이 harness는 loopy-era의 개인 구현이고, 다음 장면들이 그 증거다.

증거 1 · LOOPY-ERA GATE

지침이 HARD gate로 바뀌면 AI가 우회할 수 없는 실행 조건이 된다



핵심 장면 · 발표 질문에 가장 직접 연결

SOFT 지침이 HARD gate가 되는 순간

1. SOFT: CLAUDE.md에 "QA 필수"라고 적는다
2. 위반 감지: hook이 .qa-cycle-passed 부재를 잡는다
3. HARD: `exit 2`로 push가 차단된다
4. AI는 우회할 수 없다 — 실행 조건이 바뀐다

증거 2 · MEMORY + SELF-IMPROVE

완료 후 수정과 불만이 다음 규칙과 기억으로 축적된다

핵심 장면 · THREADS 가치관과 가장 강하게 연결

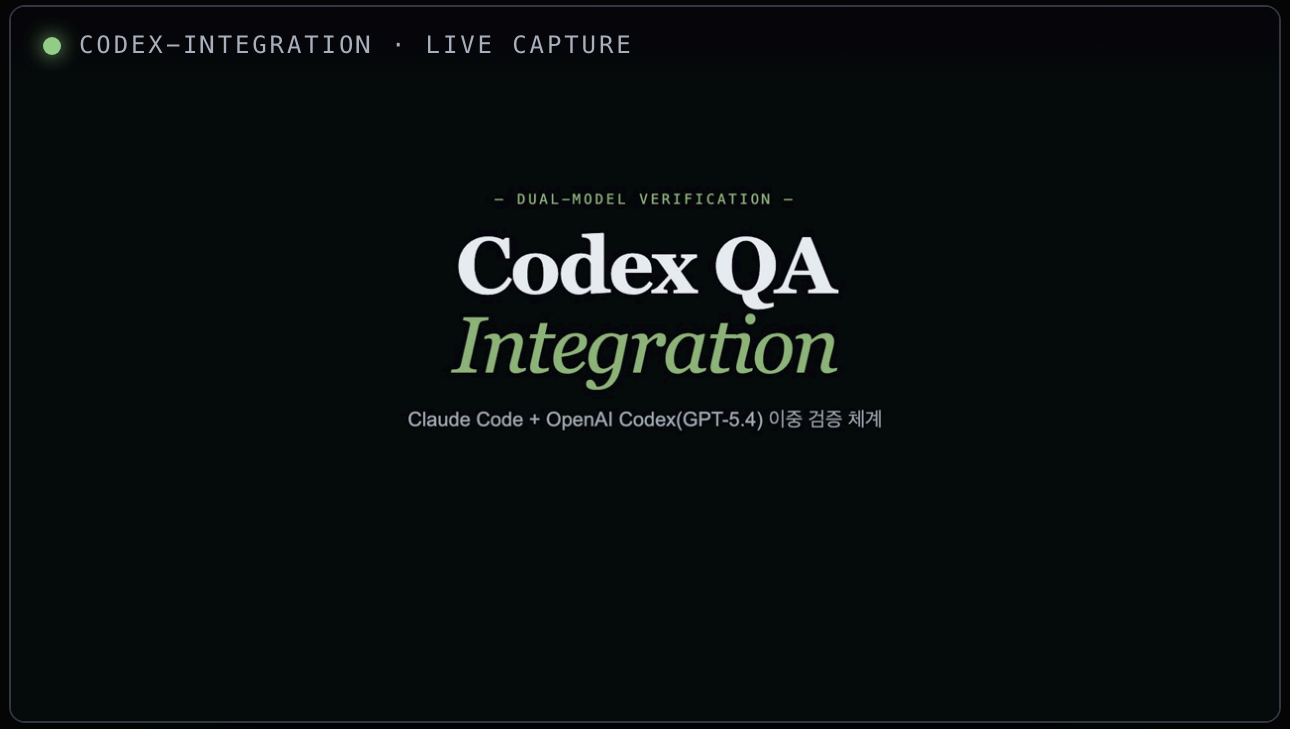
사과 대신 실행 조건이 바뀐다

1. 완료 후 사용자 수정·불만이 발생한다
2. self-improve가 마찰 패턴을 분석한다
3. rules / memory-bank로 결정화한다
4. 다음 세션의 실행 조건이 영구히 바뀐다

증거 3 · QA / CODEX CROSS-CHECK

단일 모델의 완료 선언을 다른 검증 루프가 의심하고 막는다

CODEX-INTEGRATION · LIVE CAPTURE



선택 장면 · 시간 7분 이상 확보될 때

완료를 다른 모델이 교차 검증한다

1. Claude가 "완료" 선언 + 코드 변경
2. Codex가 독립적으로 cross-review
3. CRITICAL 발견 시 PASS를 거부한다
4. 단일 모델의 맹점을 다른 루프가 차단

세 장면은 한 질문에 답한다

보여준 것은 자랑이 아니라 검증 루프의 증거다

증거 1
Loopy-Era Gate
→
루프 단계 · GATE
SOFT 지침이 우회 불가능한 HARD gate로 바뀐다

증거 2
Memory + Self-Improve
→
루프 단계 · MEMORY → 다음 조건
사과 대신 다음 실행 조건이 바뀐다

증거 3
QA / Codex Cross-Check
→
루프 단계 · EVIDENCE · REVIEWER
다른 검증 루프가 완료 선언을 의심하고 막는다

세 장면을 합치면 검증 루프 한 바퀴가 완성된다

● 그래서 — 완성된 HARNESS의 전체 구조

세 루프가 맞물린 자가진화 파이프라인

① SELF-IMPROVEMENT LOOP

Session N — 개발 · 실수

→ **git fix:** — 실수 신호

→ **Stop Hook** — self-improve-trigger

→ **pending/** — 큐 적재

→ **Session N+1** — 새 요청 · 주입

→ **/self-improve** — 마찰 → 규칙 도출

→ **Scaffold · Rules** — 43 rules · 2,854 facts

↺ *강화된 scaffold가 다음 세션에 반영 — 사용할수록 강해진다*

② HARD ENFORCEMENT GATES

Edit · Write · Bash hooks · exit 2 = BLOCK (무시 불가)

- **qa-gate-before-push** — push 차단 100%
- **scaffold-violation** — NEVER DO 감지
- **code-quality-check** — as any · @ts-ignore
- **no-localstorage** — 즉시 감지
- **no-env-commit** — 민감파일 차단
- **task-quality-gate** — UI 브라우저 증거
- **portless-required** — 경고 (WARN)

fix → 다음 루프의 신호

③ USER-PROXY AUTO QA LOOP

구현 완료 — 직접 · /team · /auto-issue

→ **Build · Type** — build · tsc --noEmit

→ **L0–L5 QA · 3-tool** — web-qa · agent-browser · screenshot

→ **Codex Cross** — GPT 이중 리뷰

→ **⟳ PASS?** — 양쪽 통과

PASS → **.qa-cycle-passed** — commit · push · Telegram ✅

재QA

FAIL

bug-fixer ×4 Rotation — 직접 → 구조 → codex:rescue → revert · 진전 있는 한 반복

3회 수렴 → Telegram 에스컬레이션

좋은 결과의 정의

좋은 결과는 한 번의 산출물이 아니라 가능한 상태다

1	반복 가능성	같은 조건이면 같은 결과를 다시 만든다
2	검증 가능성	완료를 증거로 확인할 수 있다
3	회복 가능성	어긋났을 때 되돌리고 고칠 수 있다
4	기억 가능성	실패가 다음 실행 조건으로 축적된다
5	확장 가능성	개인을 넘어 조직 기준으로 넓힌다

ACT 4 결론 *검증과 기억이 다음 실행 조건을 바꿀 때 구조는 실제 운영체계가 된다.*

ACT 05 / 05 · 확장

확장

THE SCALE

개인 harness를 **조직 운영체계**로. 복제되는 것은 도구가 아니라 **PASS의 기준**이다.

조직 확장의 진짜 장벽

모두에게 같은 harness는 만들 수 없다

IF 동일한 harness로 충분했다면, **claude code 하나로** 대부분 끝났을 것이다.

하지만 그것만으로는 부족하다 — claude code에는 이 세 가지가 없다

01 · DOMAIN

도메인 지식

현장·산업의 맥락은 범용 모델 안에 들어있지 않다

02 · GOAL

조직·팀별 목표

무엇을 PASS로 볼지가 조직마다·팀마다 다르다

03 · WORKFLOW

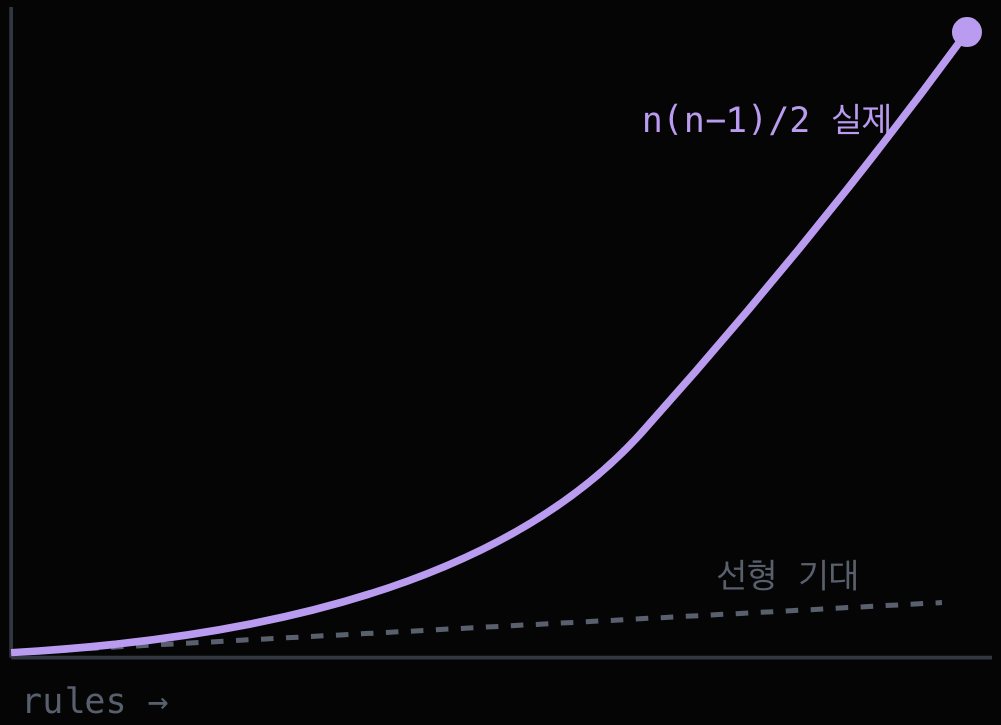
workflow

일하는 순서·기준·검증 루프가 팀마다 제각각이다

그래서 확장은 harness 복제가 아니라, **각 조직·팀의 도메인·목표·workflow를 얹는 문제**다.

개인 HARNESS도 복잡도 문제가 된다

규칙이 늘수록 상호작용과 충돌도 늘어난다



50 rules

개인 harness가 축적한 규칙 수

1,225 interactions

n(n-1)/2 — 규칙 간 잠재 상호작용·충돌

자동화의 가치는 많이 돌리는 데 있지 않고, **잘못 돌았을 때 배울 수 있는 데** 있다.

확장이 본질적으로 어려운 이유 · ENTERPRISE-HARNESS-COMPLEXITY

개인 최적화 → 조직 확장은 5가지 구조적 장벽에 막힌다

5.1

암묵적 지식 → 명시적 전파

개인

규칙의 "왜"를 본인만 안다

기업

모든 팀원이 이해해야 한다

50 rules × 평균 3 근거 = **150개 사건 컨텍스트** 전달

Difficulty 5/5

5.2

즉각 → 비동기 피드백

개인

실패하면 본인이 바로 수정

기업

실패가 다른 Worker에서 발생

분산 디버깅은 로컬의 **10배** 어렵다

Difficulty 4/5

5.3

단일 → 다중 진화 경로

개인

self-improve가 1방향 수렴

기업

각 harness가 독립 진화

공유 KG **schema governance** 미해결

Difficulty 5/5

5.4

통신비용 0 → N²

개인

모든 결정이 내 머릿속

기업

Worker 간 교차·합의

Brook's Law — 100대 → **4,950 경로**

Difficulty 4/5

5.5

무제한 → 제한된 실험

개인

실패해도 나만 손해

기업

남의 작업에 영향

rollback = **분산 트랜잭션** 문제

Difficulty 3/5

가장 핵심적인 병목 · THE CORE BOTTLENECK

user-proxy는 확장되지 않는다

PERSONAL

한 사람의 패턴을 학습한다

내 기준으로 QA PASS를 판정 — **즉각적이고 일관적**이다

→

ENTERPRISE

"누구의 기준으로 PASS인가?"

다수의 판단 기준 통일은 기술이 아니라 **조직 정치** — 기준의 소유자가 모호하다

결론 · THE OPEN PROBLEM

*$O(n^2)$ 복잡도는 기술이 아니라 **조직**의 문제다. "개인 최적화 → 조직 확장"은 아직 풀리지 않은 **열린 문제**다.*

개인에서 조직으로 — 확장 경로

확장은 도구 복제가 아니라 검증 기준의 단계적 이전이다

STAGE 1

Personal

내 harness · 내 규칙 · 내 게이트

PASS 소유자
나 자신

→

STAGE 2

Peer

동료와 기준 공유 · 교차 리뷰로 PASS를 검증

PASS 소유자
나 + 동료

→

STAGE 3

Team

팀 공통 hook · 공유 QA 게이트로 합의된 기준

PASS 소유자
팀 합의

→

STAGE 4

Org QA

조직 PASS 기준 · 책임자가 명확한 거버넌스

PASS 소유자
기준 오너

→

STAGE 5

Operating OS

기준이 곧 운영체계 · 책임과 기준을 상시 운영

PASS 소유자
운영 체계

복제되는 것은 도구가 아니라 **"PASS의 기준"**이다.

조직 확장은 QA 기준의 문제다

핵심은 도구 설치가 아니라 "누구의 기준으로 PASS인가"다

도구 설치 — 흔한 오해

"같은 툴을 깔면 같은 결과가 나온다"

WHAT	에이전트·MCP·hook을 동일하게 설치
결과	기준이 사람마다 달라 PASS가 제각각

기준 운영 — 실제 확장

"누구의 기준으로 PASS인가"를 정의·운영

WHO	QA PASS 기준의 소유자가 명확하다
HOW	거버넌스가 책임과 기준을 운영한다

ACT 5 결론 *조직 확장은 자동화 확장이 아니라 QA 기준과 책임의 확장이다.*

마무리

도구보다 구조,
구조보다 검증 루프

AI를 운영한다는 것은 결과를 믿는 것이 아니라,
결과를 의심하고 다음 실행 조건을 바꾸는 것이다.

최종 앵커 9

도구보다 구조, 구조보다 검증 루프입니다.

1. 좋은 구조는 좋은 결과를 **보장하지 않는다**
2. 완료 선언은 **증거가 아니다**
3. 완료는 말이 아니라 **계약**
4. 사과는 **상태 변경이 아니다**
5. 프롬프트는 이번 답, harness는 **다음 실행 조건**
6. 구조는 결과를 **검증하게** 만든다
7. 기억은 저장이 아니라 **다음 실행 조건**
8. 가치는 잘못 돌았을 때 **배우는 데** 있다
9. 도구보다 구조, **구조보다 검증 루프**

마지막 한 문장

절대로 LLM을 믿지 마세요.
늘 의심하고, 증거 없는 결과가 통과되지 않는 구조를 요구하세요.

감사합니다