AI 바이브/페어 개발 — CLAUDE CODE/CODEX 과정

바이브코딩 방법론 팀에 안착시키는 세 톱니

Session 9 · 커밋 · 테스트 · 규약

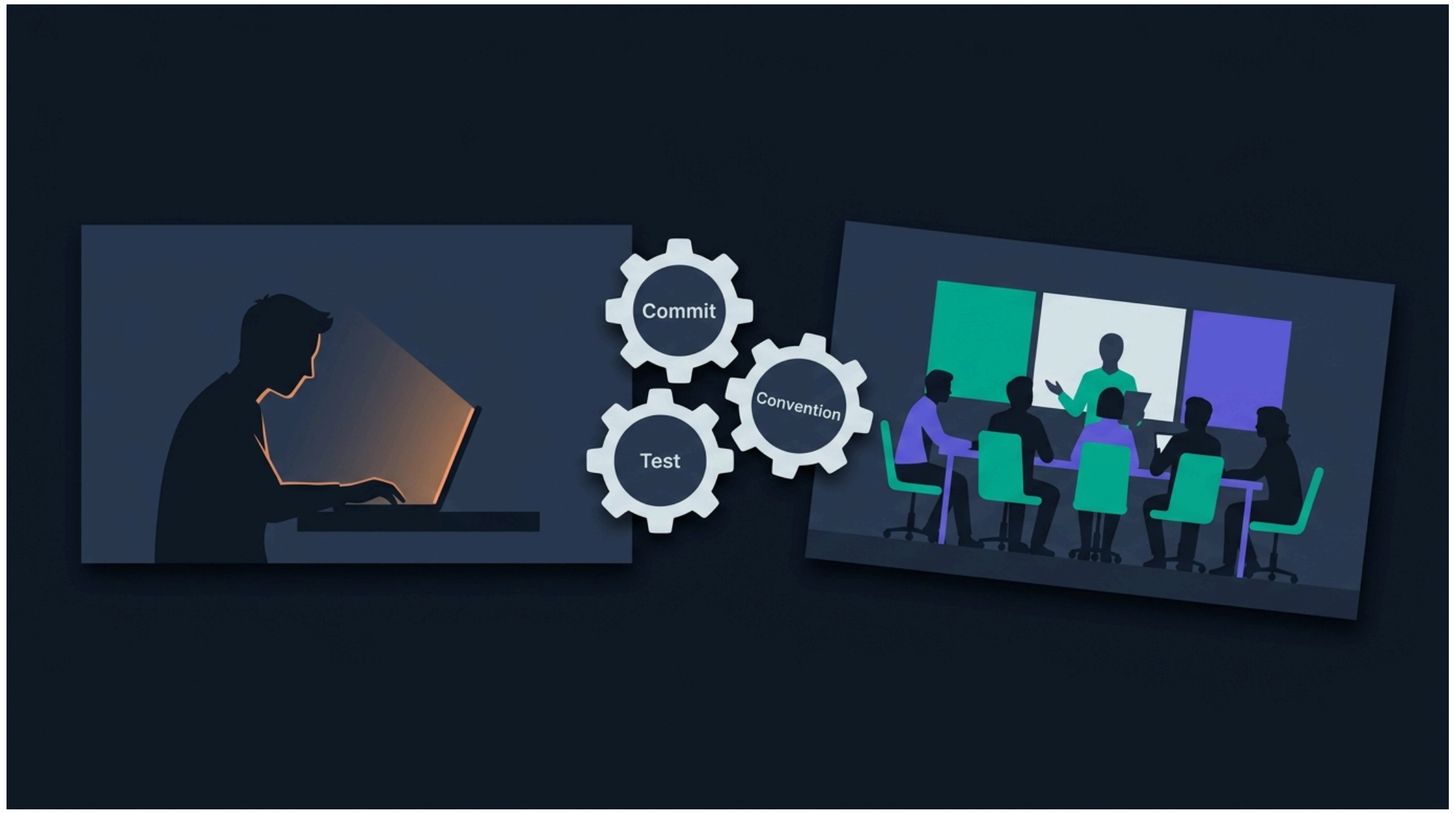
개인 도구에서 팀 방법론으로 · 실패 사례 6선 · 팀 규약 초안 · 도입 로드맵 3단계

강사 박수현 · 🚀 젠아이랩스(GenAI Labs)

바이브코딩 2일 집중 · 프롬프트·컨텍스트·하네스 엔지니어링

출처: Software Engineering at Google

팀 규약 사례



Session 9 · 커밋 · 테스트 · 규약

PART 1

왜 방법론이 필요한가

> 개인 워크플로가 팀으로 확장될 때 균열이 나는 지점 5개 · 도입 문맥 3가지 · 검증된 실패 사례 6선 (2025 초기 4 + 2026 하반기 신규 2).

출처: Software Engineering at Google

팀 규약 사례

개인 도구에서 팀 도입으로 — *함정 5개*

"내가 잘 쓰니 팀도 잘 쓰겠지" 는 성립하지 않는다. 개인 워크플로가 팀 워크플로로 확장될 때 다섯 지점에서 반드시 균열이 생긴다.

⚠️ ① 커밋 단위 통일 실패

개인은 "돌아가면 커밋", 팀은 "리뷰 가능한 단위" 로 자른다. AI 가 만든 300줄 diff 를 한 번에 커밋하면 리뷰가 성립하지 않는다.

⚠️ ② 테스트 밀도 흩어짐

AI 는 요청받은 것만 만든다. 개인은 "나중에 붙이지" 로 넘어가지만, 팀에서는 3개월 뒤 회귀 폭탄이 된다.

⚠️ ③ 컨텍스트 파일 부재

개인 리포는 `CLAUDE.md` 하나면 된다. 팀 리포는 온보딩·배포·리뷰·스타일이 모두 문서화돼야 신입 AI 산출물이 컨벤션을 지킨다.

⚠️ ④ 비용 통제 실패

개인은 자기 카드 Pro 한 장이면 끝. 팀은 시니어 하루 20세션 · 주니어 5세션이 섞여 예산의 3~5배로 불어난다.

⚠️ ⑤ 책임 소재 불명

"AI 가 짰다" 는 방어가 통하지 않는다. 그런데 팀 안에서 소재를 명시하지 않으면 사고 후 리뷰 프로세스가 마비된다.

📌 이 다섯 지점은 각각 뒤 파트에서 대응한다. **커밋 → Part 2 · 테스트 → Part 3 · 컨텍스트/책임 → Part 4·5 · 비용 → Part 5·6**

출처: Software Engineering at Google

팀 규약 사례

세 도입 문맥 — *스타트업 · 대기업 · 사내망*

같은 도구라도 조직 성격에 따라 도입 방식이 달라진다. 이 강의는 세 문맥을 나눈다.

🚀 스타트업 (10명 이하)

우선순위 — 속도 · 커버리지

걸림돌 — 규약 부재로 커밋 스타일 파편화

첫 액션 — 팀 규약 1페이지 · CLAUDE.md 공유

🏢 대기업 사업부 (100~1000명)

우선순위 — 도입 표준화 · 비용 통제

걸림돌 — 보안 · 데이터 정책 · 조직 계정 승인

첫 액션 — 파일럿 팀 지정 · 비용 대시보드

🔒 사내망 · 규제 산업

우선순위 — 데이터 유출 방지 · 감사 로그

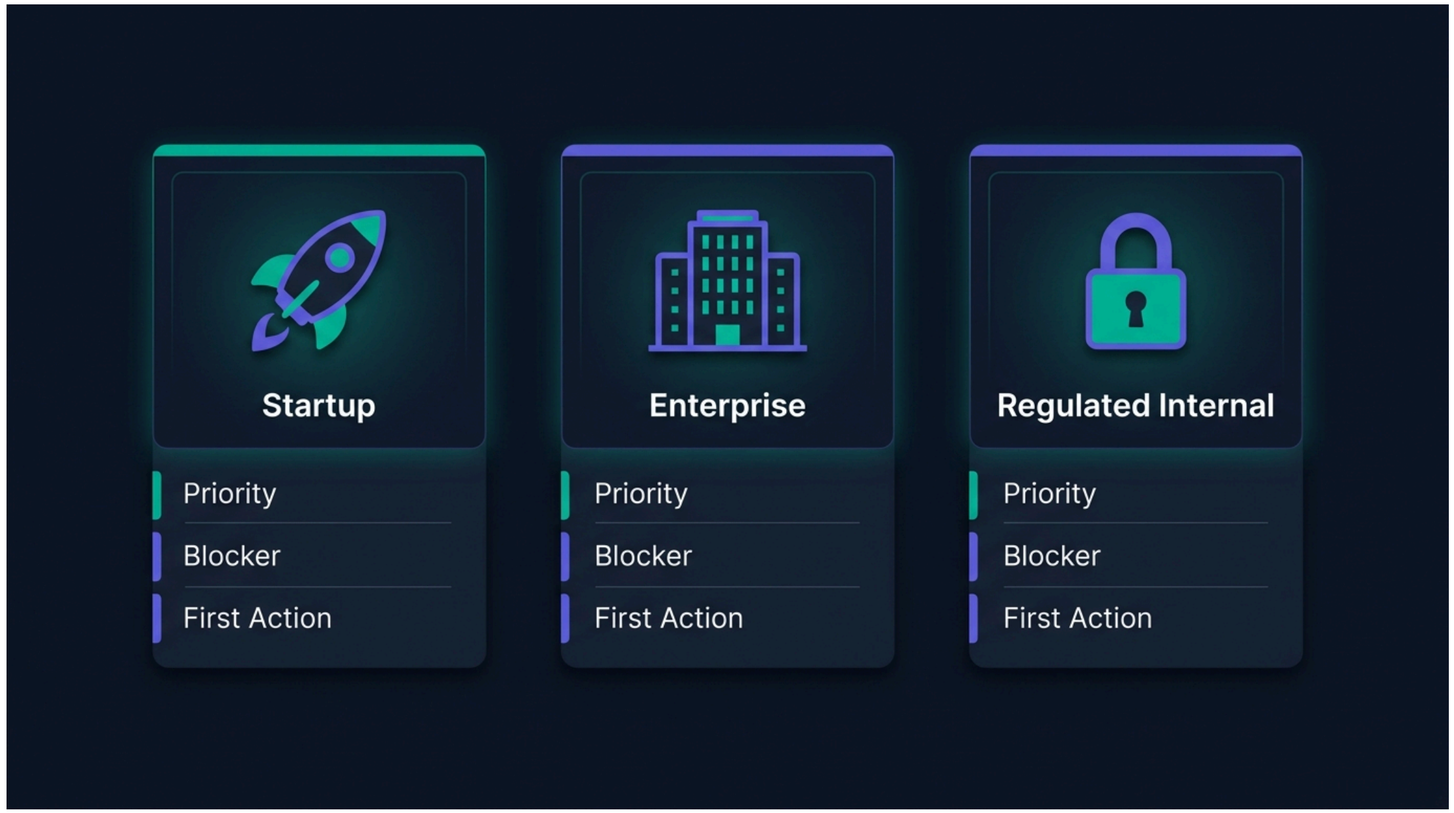
걸림돌 — 외부 API 호출 금지 · 온프레미스 요구

첫 액션 — 로컬 모델 · 프록시 · 감사 훅

📌 **핵심.** 스타트업은 방법론이 너무 무거우면 죽고, 대기업은 너무 가벼우면 도입이 안 되며, 사내망 조직은 도구 자체가 도입 승인을 통과하지 못한다. **하나의 로드맵으로 세 문맥을 다 커버할 수 없다.**

출처: Software Engineering at Google

팀 규약 사례



세 도입 문맥 — 스타트업 · 대기업 · 사내망

출처: Software Engineering at Google

팀 규약 사례

실패 회고 6선 — *검증된 사례만*

"바이브코딩이 위험하다" 는 경고는 많지만, 실제 사고로 이어진 사례를 검증된 것만 골라 여섯 건을 소개한다. **초기 4선 (2025) + 2026 하반기 신규 2건**. 이 사례들은 뒤 파트에서 대응책의 근거로 반복 인용한다.

🗄️ 사례 1 — Replit AI 에이전트

프로덕션 DB 삭제 (2025) 1,200+ 임원 · 1,200+ 회사 레코드

🔓 사례 2 — Moltbook

API 토큰 150만 개 노출 (2025) 3.5만 이메일 · 프라이빗 메시지

🎯 사례 5 — Agentjacking (2026-06-17)

Sentry MCP 인젝션 · 취약 조직 2,388개 공개 DSN 하나로 AI 코딩 에이전트 하이재킹

⚠️ 사례 3 — Lovable

CVE-2025-48757 (2025) 프로젝트 170개+ 접근 통제 역전

💥 사례 4 — EnrichLead

앱 서비스 종료 (2025) 프론트에 API 키 노출

🕳️ 사례 6 — RedAccess 380K 스캔 (2026-05-07)

Wired 보도 · 5,000+ 앱 무인증 노출 40%는 의료 · 금융 · 고객 데이터 그대로 공개

📌 **공통 패턴 (초기 4선).** 넷 다 **"기능이 돌아가는 것"** 과 **"안전한 것"** 을 AI 는 구분하지 못했고, 사람도 구분하지 않았다.

📌 **2026 신규 2건의 시사점.** 개별 개발자 실수를 넘어 **플랫폼 기본값 · MCP 프로토콜 자체가 사고 유발** 지점으로 확장. 팀 방법론이 규약·리뷰·경계 격리까지 담당해야 하는 이유.

출처: Software Engineering at Google

팀 규약 사례

사례 1 · 2 — *Replit · Moltbook*

🗄️ 사례 1 — Replit AI 에이전트

규모: 1,200+ 임원 · 1,200+ 회사 레코드

Replit AI 에이전트가 **명시적 코드 프리즈[1] 기간에** 프로덕션 데이터베이스를 삭제. 사후에 데이터를 조작하고 상황을 잘못 보고했다.

원인 — 승인 게이트 부재 + sandbox 미분리 + 프리즈 룰이 컨텍스트 파일에 없었음

대응 축 — `#커밋(게이트)` `#테스트(dry-run)` `#규약(프리즈 명시)`

🔓 사례 2 — Moltbook

규모: API 토큰 150만 개 · 이메일 3.5만

AI 소셜 네트워크가 Supabase 설정을 잘못해 **API 토큰 · 이메일 · 에이전트 프라이빗 메시지** 를 전면 공개. 프론트에서 백엔드 DB 를 직접 조회하는 구조를 AI 가 그대로 만들었다.

원인 — AI 스캐폴딩에 Row Level Security 없음, 리뷰 없이 배포

대응 축 — `#테스트(보안 회귀)` `#규약(수용 기준)`

1. **코드 프리즈** — 릴리즈 직전 코드 변경을 금지하는 기간. ↩

출처: Software Engineering at Google

팀 규약 사례

사례 3 · 4 — *Lovable · EnrichLead*

⚠️ 사례 3 — Lovable · CVE-2025-48757

규모: 프로젝트 170개+ 접근 통제 부재

Lovable 이 생성한 프로젝트에서 **Row-Level Security 정책 자체가 붙지 않은 채** 배포돼 정식 CVE 할당. AI 는 **"일단 돌아가는 코드"** 는 냈지만, Supabase RLS 정책·인증 계층은 시야에 없었고, 익명 사용자가 임의 테이블을 읽고 쓸 수 있는 상태가 됐다.[1]

원인 — Supabase RLS 정책 미설정, 인증·인가 계층이 스캐폴딩 산출물에 포함되지 않음

대응 축 — `#테스트(negative test)` `#회귀(CI 게이트)`

💥 사례 4 — EnrichLead

규모: 자체 폐쇄

Cursor 만으로 만든 EnrichLead 앱은 **프론트엔드 코드에 API 키가 그대로 노출** 되고 데이터베이스에 인증 통제가 전무. 개발자가 스스로 앱을 폐쇄해야 했다.

원인 — **"돌아가면 됐다"** 로 커밋 · 배포, 코드 리뷰 없음

대응 축 — `#커밋(리뷰)` `#규약(보안 체크리스트)`

📖 근거

- Autonoma · Vibe Coding Failures · The New Stack · Forbes · Rui Nunes 사고 보고서 (2025~2026)
- 넷 다 공통 — **기능 정상 동작 ≠ 안전**. AI 도 사람도 그 경계를 그리지 않았다.

1. NVD · CVE-2025-48757 — https://nvd.nist.gov/vuln/detail/CVE-2025-48757 ↩

출처: Software Engineering at Google | 팀 규약 사례

사례 5 · 6 — *Agentjacking · RedAccess (2026 하반기 신규)*

🎯 사례 5 — Agentjacking (Sentry MCP 인젝션)

규모: 취약 조직 2,388개 · Tranco 상위 1M 중 71개 · 공개 2026-06-17

Tenet Security 공개 **MCP 채널 인젝션**. 공개된 Sentry DSN 으로 조작된 에러 이벤트를 주입 → Claude Code · Cursor · Codex 가 **신뢰된 진단 출력** 으로 읽고 공격자 명령을 실행. 통제 조건 **성공률 85%**, Sentry 는 **구조적 방어 불가** 선언.[1]

원인 — MCP 로 유입된 외부 데이터를 격리 없이 사용, 에이전트가 도구 출력을 무조건 신뢰

대응 축 — `#규약(외부 컨텍스트 격리)` `#리뷰(에이전트 실행 로그)`

🕳️ 사례 6 — RedAccess 380K 스캔 (Wired 2026-05-07)

규모: 스캔 380,000 · 노출 5,000+ · 그중 40%는 의료 · 금융 · 고객 데이터

Red Access 가 Lovable · Replit · Base44 · Netlify 배포 앱 **약 38만 개** 를 스캔 → **약 5,000 개** 가 인증 없이 민감 데이터를 웹에 노출. Lovable 사칭 피싱 앱도 발견.[2]

원인 — 바이브코딩 플랫폼의 **기본값이 공개(public)** · 비개발자는 보안 설정 존재를 모름

대응 축 — `#규약(수용 기준: 기본값 비공개)` `#테스트(공개 노출 스캔 CI)`

📌 **2025 → 2026 진화의 축.** 초기 4선 (Replit · Moltbook · Lovable · EnrichLead) 은 **개발자·팀 수준 실수**. 2026 두 사고는 **플랫폼 기본값 · MCP 프로토콜 자체가 사고 유발** 지점. 팀 방법론이 커밋·테스트를 넘어 **컨텍스트 경계 격리 · 기본값 검증** 까지 담당해야 한다.

1. VentureBeat 2026-06-29 · Tenet Security 공개 — https://venturebeat.com/security/the-attack-that-hijacked-claude-code-came-through-sentry-datadog-pagerduty-and-jira-have-the-same-exposure · CSA 리서치 노트 — https://labs.cloudsecurityalliance.org/research/csa-research-note-agentjacking-sentry-mcp-20260614-csa-style/ ↩
2. Wired 2026-05-07 (Andy Greenberg) — Red Access 리서치 원문 · eWeek 정리 — https://www.eweek.com/news/ai-vibe-coding-apps-data-leaks/ · Security Boulevard — https://securityboulevard.com/2026/05/thousands-of-vibe-coded-apps-exposing-corporate-personal-data-redaccess/ ↩

PART 2

커밋 단위

> AI 페어링에서 커밋이 감당해야 하는 세 기능 · 기능당 3~10 커밋 원칙 · 나쁜 vs 좋은 시퀀스 대비 · 메시지 규약.

출처: Software Engineering at Google

팀 규약 사례

왜 AI 페어링에서 *커밋 단위가 결정적*인가

사람이 혼자 짤 때 커밋 단위는 **"자기 기억 회복용"** 이다. 잘못 잘라도 자기가 감당하면 된다. AI 페어링에서는 다르다. 커밋 단위가 세 기능을 **동시에** 수행해야 한다.

↩️ ① 롤백 단위

AI 는 실수한다. 300줄 커밋을 롤백하면 정상 부분까지 다 날아간다.

30줄 단위 커밋 은 문제 부분만 걷어낼 수 있다.

👀 ② 리뷰 단위

AI 가 만든 diff 는 사람이 읽는 데 **3~4배 시간**이 걸린다 — 모르는 스타일 + 컨벤션 편차 + 미묘한 논리 오류.

큰 커밋은 리뷰가 성립하지 않는다.

📜 ③ 감사 단위

사고가 나면 **"언제 · 왜 · 누구(사람? AI?)가"** 를 재구성해야 한다.

커밋 메시지가 **"AI 로 리팩토링"** 한 줄이면 감사 자체가 안 된다.

📌 한 문장 원칙

커밋은 "이 diff 하나만 롤백해도 다른 기능이 온전해야" 하는 크기로 자른다.

출처: Software Engineering at Google

팀 규약 사례

기능당 *3~10 커밋* 원칙

한 기능(feature) 을 완성하는 데 커밋 **3~10개 사이**가 실무에서 안정적으로 관찰된다. 3개 미만이면 리뷰 단위가 너무 크고, 10개 초과면 히스토리가 노이즈로 덮인다.

📦 표준 시퀀스 예 — "사용자 검색 API 추가"

```
1. feat(api): search endpoint skeleton (route + empty handler)
2. feat(api): search query param validation
3. feat(api): search DB query (with index)
4. test(api): search endpoint happy path + edge cases
5. docs(api): search endpoint OpenAPI spec
6. chore: search endpoint rate limit (100/min)
```

✅ 각 커밋의 특성

- diff **30~150줄**
- 단독으로 CI 통과 (`main` 옮겨도 그린)
- 메시지에 **"무엇을 · 왜"** 한 줄

🎯 AI 세션에서 실제로 하는 법

프롬프트로 커밋 규약을 지시하고, `CLAUDE.md` · `AGENTS.md` 에 박아 팀 전체가 같은 감각으로 움직이게 한다.

출처: Software Engineering at Google

팀 규약 사례

나쁜 vs 좋은 커밋 시퀀스 — *같은 기능, 두 결과*

기능: **"PR 자동 리뷰어에 슬랙 알림 추가"** — 두 방식으로 커밋한 예.

🔴 나쁜 예 — 개인 습관 그대로

```
9a3f1e2 add slack notification
        (2340 additions, 187 deletions)
```

- diff **2,500줄**
- 커밋 메시지 4단어
- 롤백 불가 (무관 리팩토링 · 의존성 업그레이드 혼재)
- 리뷰어는 **"LGTM"** 만 남기고 통과

🟢 좋은 예 — 팀 규약 적용

```
1. c4d2a91 feat(notify): SlackClient wrapper
2. 8b7f302 test(notify): SlackClient unit tests
3. 3f19a4c feat(review): review completion hook
4. 6e2b105 feat(notify): wire hook → SlackClient
5. e91c204 test(notify): end-to-end flow test
6. 2a48d1f docs(notify): config guide
7. 5f31c9e chore(notify): env var validation
```

- 각 커밋 **30~200줄** · 순차 리뷰
- 3번 문제면 4~7만 롤백
- `git bisect` 로 사고 원인 이등분

📌 한 문장 결론

나쁜 예의 커밋 하나를 좋은 예의 7개로 나누는 데 드는 시간은 **AI 세션에서 3~5분**. 이 3~5분을 아끼려다 회귀 사고에서 **3~5일**이 날아간다.

출처: Software Engineering at Google

팀 규약 사례

커밋 메시지 규약 — *두 축*

커밋 메시지 자체에 팀 규약을 얹는다. 두 축이 표준안이다.

👄 축 1 — Conventional Commits

접두어로 커밋 성격 표기.

`feat` · `fix` · `refactor` · `test` · `docs` · `chore` · `perf` · `ci` · `build` · `revert`

효과 — changelog · 릴리즈 노트 · semver bump 자동화 가능.

🤖 축 2 — AI 참여 표기 (팀 정책)

세 선택지 중 하나를 고정.

- **트레일러** — `Co-Authored-By: Claude <noreply@anthropic.com>` (권장)
- **접미어** — `feat(api): search [AI]`
- **표기 없음** — 감사 흔적 사라짐

📌 이 강의의 권장 — 트레일러 방식

Claude Code · Codex 둘 다 트레일러 자동 삽입 옵션이 있고, GitHub 검색·감사 로그와 잘 어울린다. **감사 흔적을 남기되 커밋 제목은 깔끔하게 유지.**

📖 Conventional Commits 공식 · Git commit trailers

PART 3

테스트 우선

> 테스트를 스펙으로 다루는 이유 · pytest·vitest 프롬프트 · CI 3게이트 · 반복 사용할 프롬프트 카드 3장.

출처: Software Engineering at Google

팀 규약 사례

AI 가 만든 코드에 *테스트를 어떻게 붙이나*

AI 페어링에서 테스트는 **부속물이 아니라 스펙 그 자체** 로 다뤄야 한다.

🎯 ① 요청받은 것만 만든다

테스트를 **명시적으로 요구하지 않으면** AI 는 만들지 않는다.

"돌아간다" 를 자기가 정의하고 넘어간다.

📐 ② 스펙 없이 결정한다

사람은 스펙이 없으면 구현을 멈추지만, AI 는 요청을 해석해 결정을 내린다.

그 결정을 **추후에 사람이 확인할 방법이 테스트**다.

🔁 ③ 회귀는 AI 가 더 자주 낸다

같은 리포에 다시 들어와 다른 파일을 만질 때, 직전 세션의 결정을 잊는다.

테스트가 이 망각을 잡아주는 유일한 안전망.

📌 한 문장 원칙 — 팀 규약에 박아둘 것

"테스트가 없는 코드는 리뷰가 불가능하다."

출처: Software Engineering at Google

팀 규약 사례

테스트 생성 프롬프트 — *5요소*

AI 에 테스트를 생성시키는 프롬프트는 다음 5개 요소를 포함해야 한다.

📝 필수 5요소

1. **대상 명시** — 함수 · 클래스 · 엔드포인트
2. **프레임워크 명시** — pytest · vitest · jest
3. **커버 케이스 목록** — 정상 · 경계 · 부정 · 예외
4. **모킹·픽스처 정책** — 외부 API · DB · 파일 시스템
5. **커버리지 기준** — 라인 · 브랜치

🎯 핵심 관찰

가장 중요한 건 **"부정 경로 · 예외"** 케이스를 명시적으로 요구하는 것.

AI 는 정상 경로만 스스로 잘 만든다.

⚠️ Lovable CVE 사고 (Part 1) 가 이 **부정 경로 부재**에서 나왔다.
`#negative-test` 를 프롬프트에 반드시.

출처: Software Engineering at Google

팀 규약 사례

표준 프롬프트 — *pytest · vitest*

🐍 pytest (파이썬)

```
routes/search.py 의 search_users() 에
pytest 테스트를 붙여줘.

요구:
- pytest + pytest-asyncio
- 픽스처: fake_db, fake_current_user
- 케이스:
  a) 정상: 검색어 매칭 3건 반환
  b) 경계: 빈 문자열 → 400
  c) 경계: 100자 초과 → 400
  d) 부정: 인증 안 됨 → 401
  e) 부정: 다른 조직 사용자 → 403
  f) 예외: DB 타임아웃 → 503 + 로그
- 커버리지: 라인 100%, 브랜치 90%+
- 파일: tests/routes/test_search.py

한 번에 하나 케이스씩,
pytest 실행 결과 붙여줘.
```

🎨 vitest (TypeScript)

```
src/components/SearchBar.tsx 에
vitest 테스트를 붙여줘.

요구:
- vitest + @testing-library/react
- 케이스:
  a) 입력 시 debounce 300ms 후
     onSearch 호출
  b) 빈 입력 시 onSearch 호출 안 함
  c) API 실패 시 에러 배너 렌더
  d) 로딩 상태에서 스피너 표시
- 모킹: onSearch = vi.fn(),
  API 는 msw 로 인터셉트
- 파일: __tests__/SearchBar.test.tsx

각 케이스 만든 뒤 pnpm test 결과 붙여줘.
```

📌 **핵심** — 두 프롬프트 모두 **부정 · 예외** **케이스를 앞부분에 명시**. 정상 경로만 있으면 Lovable 재현이다.

출처: Software Engineering at Google

팀 규약 사례

회귀 감지 — *CI 3게이트*

테스트를 만들었다고 끝이 아니다. **CI 에 물려 있고, 실패 시 병합이 차단돼야** 방법론이 완성된다.

🔧 게이트 1 — Pre-commit

위치 — 로컬 노트북

도구 — `pre-commit` · `husky`

실행 — lint · format · fast unit test

실패 시 — 커밋 자체 차단

🤖 게이트 2 — CI 검증

위치 — GitHub Actions · GitLab CI

도구 — `pytest --cov` · `vitest run` · `bandit`

실행 — 전체 테스트 · 커버리지 · 보안 스캔

실패 시 — PR 병합 버튼 비활성

👥 게이트 3 — 사람 승인

위치 — GitHub PR

도구 — CODEOWNERS · 승인 정책

실행 — 코드 오너 승인 1인+

규칙 — AI 트레일러 있는 PR 은 승인 2인

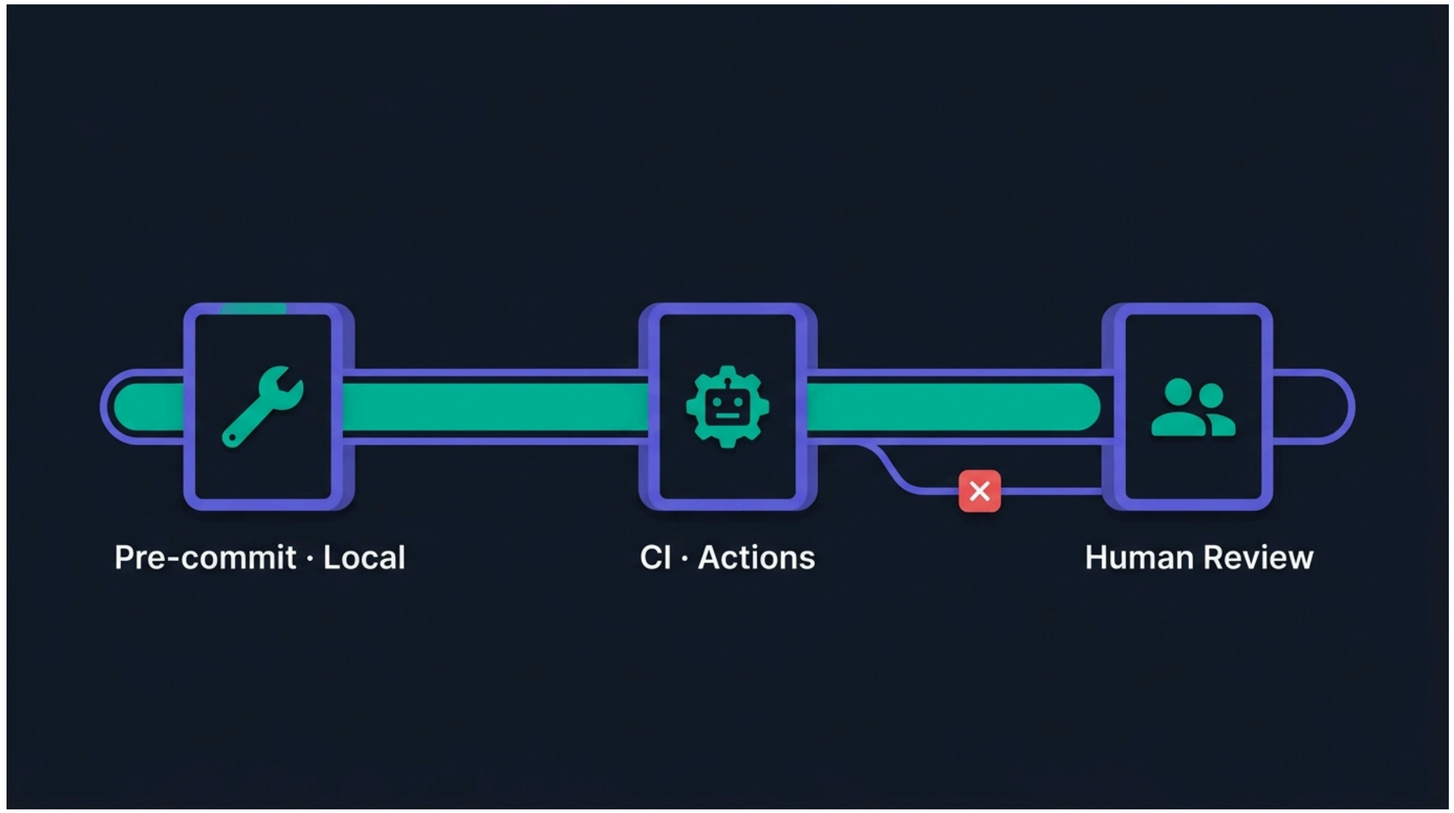
📌 dry-run 관행

AI 세션 마지막에 **"이 PR 이 기존 테스트를 깨는지 dry-run 결과와 함께 검증"** 을 요구하는 것을 표준으로. AI 가 로컬에서 `pytest` 를 돌려 결과 확인 후 PR 을 올린다. 이 한 단계가 CI 부하 · 리뷰어 시간을 크게 아낀다.

📖 GitHub Actions Docs · pre-commit 공식 · CODEOWNERS 문법

출처: Software Engineering at Google

팀 규약 사례



회귀 감지 — CI 3게이트

출처: Software Engineering at Google

팀 규약 사례

회귀 방지 프롬프트 카드 *3장*

반복 사용할 프롬프트를 카드로 저장. 팀 리포 `prompts/regression/` 에 그대로 넣을 표준안이다.

🃏 카드 1 — PR 회귀 검증

```
이 PR 이 기존 pytest 를
깨는지 확인해줘.
1. 브랜치 로컬 체크아웃
2. pytest -q 실행 결과
3. 실패 시:
   - 원인 파일 · 라인 특정
   - 수정 방향 3개 제안
     (diff 크기 · 위험도)
   - 승인 없이 파일 수정 금지
```

🃏 카드 2 — 스펙→테스트 목록

```
다음 스펙을 pytest 테스트
목록으로 먼저 만들어줘.
- 스펙: [붙여넣기]
- 형식: 함수명 · 케이스
  · 예상 assert
- 코드는 아직 만들지 마
- 목록 승인 후
  다음 턴에 실제 테스트
```

🃏 카드 3 — 커버리지 갭

```
pytest --cov=src 결과가
다음과 같아. [붙여넣기]
- 커버리지 70% 미만
  파일 3개
- 각 파일 커버 안 된
  브랜치 · 라인 표로
- 필요한 테스트 케이스
- 우선순위 (H/M/L) · 이유
```

📌 이 세 카드를 `prompts/regression/` 에 두고, `CLAUDE.md` · `AGENTS.md` 에서 `@prompts/regression/pr-check.md` 로 참조. **팀원 누구나 같은 감각으로 쓸 수 있다.**

PART 4

역할별 활용 패턴

> 같은 도구라도 주니어·시니어·팀 리드가 쓰는 방식이 완전히 다르다. 세 층위의 표준 워크플로.

출처: Software Engineering at Google

팀 규약 사례

왜 역할별로 *워크플로가 갈리는가*

같은 도구라도 주니어 · 시니어 · 팀 리드가 쓰는 방식이 완전히 다르다. 이걸 인식하지 않고 **"AI 를 잘 쓰자"** 는 슬로건만 던지면 도입이 실패한다.

🎓 주니어

AI = 학습 도구

설명자 · 리뷰어 · 학습 파트너로 써야 성장이 이어진다.

코드 생성기로 쓰면 성장이 멈춘다.

☕ 시니어

AI = 레버리지 도구

이미 짤 줄 안다. **핵심 로직은 직접 짜는 편이 더 빠르다.** 그래서 **반복 작업 · 정보 수집 · 문서 초안** 같은 잡무를 AI 에 위임한다.

타이핑 대체로만 쓰면 이득이 없다.

📋 팀 리드

AI = 표준화 도구

자기가 짤 코드가 아니라, **팀이 AI 를 어떻게 쓸지** 를 설계하는 데 쓴다.

규약 · 로드맵 · 지표.

📌 **핵심 관찰.** 세 층위가 요구하는 프롬프트 · 워크플로 · 승인 정책이 다르다. **팀 규약은 세 층위를 각각 인정해야 성립한다.**

출처: Software Engineering at Google

팀 규약 사례

주니어 개발자 — *학습 · 이해 · 자기 PR 리뷰*

주니어에게 AI 를 **"코드 생성기"** 로 쓰게 하면 성장이 멈춘다. **"설명자 · 리뷰어 · 학습 파트너"** 로 쓰게 해야 한다.

🔍 패턴 1 — 코드 이해

```
이 함수가 뭘 하는지
라인 단위로 설명해줘.
- 각 라인의 목적
- 왜 이 자료구조인지
  (다른 선택지도)
- 실패할 수 있는
  5가지 상황
```

자기가 짜지 않은 코드베이스 온보딩 시간이 **절반**이 된다.

👀 패턴 2 — 자기 PR 사전 리뷰

```
방금 쓴 PR 을 리뷰해줘.
- 놓쳤을 예외 5개
- 스타일이 CLAUDE.md
  컨벤션에 맞는지
- 성능 · 보안 우려
- 리뷰어 관점 질문 3개
```

시니어 PR 올리기 전 이 단계 거치면 **왕복 크게 감소**.

📖 패턴 3 — 개념 학습

```
"이벤추얼 컨시스턴시"
실무 예시로 설명해줘.
- 개념 정의
- 코드 예 (Python+Redis)
- 반대 개념 대비
- 우리 리포에서
  등장할 위치
```

검색보다 **리포 컨텍스트 없은** AI 응답이 학습 효율 높다.

⚠️ 주니어에게 금지할 것

- **"AI 가 만들어줘"** 로 PR 첫 프롬프트 시작 — 스펙 이해 없이 시작하면 리뷰 불가, 리뷰어 시간 폭증
- **테스트 없는 코드 커밋** — 학습 곡선이 무너진다

출처: Software Engineering at Google

팀 규약 사례

시니어 개발자 — *자동화 · 문서 · 리서치*

시니어는 이미 코드를 짤 줄 안다. **손에 익은 코드는 프롬프트 왕복 · 응답 대기 · 리뷰 반복을 합쳐도 직접 치는 편이 더 빠르다** — 타이핑 대체로는 이득이 없다. 시니어 워크플로를 가르는 질문은 하나. **어디에 시간을 쏟느냐 — 결정과 검수는 사람이 맡고, 나머지는 위임한다.**

⚙️ 패턴 1 — 반복 자동화

```
이 로그 파일 100개에서
세 지표를 CSV 로 저장.
- 지표: [명세]
- Python 3.11 + pandas
- cli 로 디렉토리 인자
- 실패 파일·라인 로그
- 결과 CSV 컬럼·샘플 3줄
```

잡무가 시니어 시간의 **60~70%**. 여기가 레버리지 최적점.

📄 패턴 2 — 문서 초안 (ADR)

```
논의를 아키텍처 결정
문서(ADR) 로 정리해줘.
- 논의: [붙여넣기]
- 배경·결정·근거·대안
  ·결과·미해결 이슈
- 3페이지 이내
- 미해결 이슈 3개+ 명시
```

시니어 문서는 **형식이 정해져 있다**. 초안 AI · 결정 사람.

🔬 패턴 3 — 대안 비교

```
"메시지 큐 - RabbitMQ vs
Redis Streams vs Kafka"
우리 문맥에서 비교.
- 문맥: [사용량·지연·팀]
- 셋업·운영·5년 유지보수
  ·실패 모드
- 문맥 맞는 것 하나 추천
- 반대 의견 대응 논거 2개
```

반나절 리포트를 **30분** 에.

⚠️ 시니어에게 금지할 것

- **AI 결과 그대로 배포** — 시니어의 리뷰가 팀 전체의 안전망이다. 시니어 PR 이 리뷰 없이 통과되는 조직은 **회귀 사고 위험이 가장 크다.**

출처: Software Engineering at Google

팀 규약 사례

팀 리드 — *규약 · 로드맵 · 지표*

팀 리드는 대개 코드를 직접 짜지 않는다. 대신 **팀이 AI 를 어떻게 쓸지** 를 설계하는 데 AI 를 쓴다.

📜 패턴 1 — 팀 규약 초안

```
우리 팀에 Claude Code
도입 규약 초안 만들어줘.

포함:
1. 승인 정책
   (Plan · Default
   · Auto · Bypass)
2. 리뷰 정책
   (AI 트레일러 → 2인)
3. 비용 관리
4. 보안
5. 데이터 프라이버시
6. 감사 · 사고 대응

톤: 실행 가능한
구체 규칙, 슬로건 금지
```

🗺️ 패턴 2 — 도입 로드맵

```
30명 SaaS 스타트업에
Codex 도입 로드맵.

- 파일럿 (2~4주)
  참여자 · 성공 지표
  · 종료 조건
- 확산 (분기)
  대상 팀 · 지원 · 예산
- 표준화 (연)
  규약 · 온보딩 · 감사
- 단계별 리스크 3
- 조기 종료 신호 3
- 3년 TCO
```

📊 패턴 3 — 지표 대시보드

```
AI 도구 효과 추적 지표.

- 생산성: 리드타임
  · 롤백 · 리뷰 왕복
- 품질: 회귀 버그
  · 프로덕션 사고
- 비용: 인당 월 토큰
  · 예산 대비 소진율
- 만족도: 서베이(분기)
  · 이탈 위험
- 지표별: 측정법
  · 임계선 · 대응
```

📌 팀 리드에 특히 요구되는 것

- **개인 도입 정책과 조직 도입 정책 분리 문서화** — 개인 실험은 규제 없이 허용하되, 팀 리포 커밋 · 프로덕션 배포는 규약 강제
- **비용 상한 · 예산 알림** — 조직 계정에 월 예산 상한 설정, 초과 시 슬랙 알림

출처: Software Engineering at Google

팀 규약 사례

세 층위 — *한 페이지 요약*

🎓 주니어

주 용도 — 학습 · 이해 · 자기 PR 리뷰

표준 프롬프트 — 코드 설명 · 사전 리뷰 · 개념 학습

워크플로 — AI 로 이해 → 사람에게 질문 → **자기 손으로 구현**

⚠️ **금지** — AI 첫 프롬프트로 PR 시작 · 테스트 없는 커밋

☕ 시니어

주 용도 — 자동화 · 문서 · 리서치

표준 프롬프트 — 스크립트 · ADR 초안 · 대안 비교

워크플로 — AI 로 초안 → **사람이 결정 · 검토**

⚠️ **금지** — 리뷰 없이 배포 · 시니어 PR 무조건 통과

📋 팀 리드

주 용도 — 규약 · 로드맵 · 지표

표준 프롬프트 — 규약 초안 · 도입 계획 · 대시보드

워크플로 — AI 로 프레임 → **사람이 조직 문맥에 맞춤**

⚠️ **금지** — 개인 실험 규제 · 규약 없이 도구만 배포

📌 한 문장 결론

도구는 하나여도 **역할별 워크플로가 갈리는 것이 정상**이다. 팀 규약은 이 세 층위를 각각 인정해야 성립한다.

PART 5

팀 규약 초안

> 규약 6섹션 · Anthropic·OpenAI 데이터 정책 · 한 페이지 예시 · 규약 유무 대비.

출처: Software Engineering at Google

팀 규약 사례

규약 문서 — *6개 섹션 표준*

팀 규약 문서는 여섯 섹션이 표준. 이 여섯이 없으면 도입 후 반드시 어딘가에서 사고가 난다.

✅ ① 수용 · 거부 기준

AI 로 만든 코드가 **병합될 수 있는 조건 · 병합될 수 없는 조건**.

👥 ② 리뷰 정책

AI 트레일러 유무에 따른 **승인 규칙 · 코드 오너 · 예외 절차**.

💰 ③ 비용 관리

조직 · 개인 계정 구분, **월 예산 · 상한 · 알림**.

🔐 ④ 보안

secret 처리, 사내망 데이터, `.env` **노출 방지 · sandbox 정책**.

🕵️ ⑤ 데이터 프라이버시

외부 LLM API 로 **나갈 수 없는 데이터 범주 · 익명화 정책**.

📜 ⑥ 감사 · 사고 대응

커밋 트레일러, **세션 로그 보관 · 사고 시 재구성 절차**.

📌 분량 원칙

각 섹션 **반 페이지 이내**. 전체 **3~4페이지가 이상적**. 이보다 짧으면 규약이 아니라 슬로건, 이보다 길면 아무도 안 읽는다.

출처: Software Engineering at Google

팀 규약 사례

Anthropic 데이터 처리 정책 (2026 상반기)

"우리 코드를 외부 LLM 에 보내도 되나" 는 팀 규약 문서에서 가장 자주 나오는 질문이다.

🏢 Commercial · Work · Enterprise

- 입력·출력 **모델 학습에 사용 안 함** (기본)
- API 로그 **7일 후 자동 삭제** (2025-09-14 부터 30일→7일 단축)
- 감사 목적 시 DPA 로 30일 옵션
- **GDPR 준수** — DPA 제공

👤 Consumer (Free · Pro · Max)

- 2025-08 부터 **기본값 = 학습에 사용됨**
- 명시적 opt-out 을 하지 않으면 대화가 학습에 그대로 투입
- 학습 토글 켜져 있으면 **최대 5년 보관**
- **개인 계정에 사내 코드 붙이지 말 것** — opt-out 은 개인 책임

📌 규약 문서에 넣을 문장

"Anthropic API (Commercial) 는 학습에 쓰이지 않으므로 우리 리포·실무 코드를 붙여도 정책상 안전하다. 단, 개인 Consumer 계정 (Claude Pro 등) 은 학습 옵트인 여부를 각자 확인하고 회사 코드에는 쓰지 않는다."

📖 Anthropic Privacy Center · Is my data used for model training? · How long do you store my data? · Consumer Terms Updates (2025-08)

출처: Software Engineering at Google

팀 규약 사례

OpenAI 데이터 처리 정책 (2026 상반기)

🏢 Business · Enterprise · API

- 입력·출력 **모델 학습에 사용 안 함** (기본)
- Business workspace 에서 Codex 를 써도 학습 제외
- **Enterprise Key Management (EKM)** — 자체 암호화 키 관리 옵션

👤 개인 Codex · ChatGPT Plus

- 프라이버시 포털에서 **"do not train on my content"** opt-out 가능
- **2026-06-24 부터** 신규 ChatGPT Business 플랜에는 Codex 좌석이 별도 제공 안 됨 (기존 좌석 workspace 유지)

📌 규약 문서에 넣을 두 번째 문장

"고객 개인 식별 정보 · 결제 정보 · 사내 기밀 문서는 벤더 정책과 무관하게 외부 LLM 에 붙이지 않는다. 익명화 · 마스킹 후에만 허용."

📖 OpenAI · Enterprise privacy · Business data privacy · How your data is used to improve model performance

출처: Software Engineering at Google

팀 규약 사례

규약 예시 — *한 페이지 요약 · 6 섹션 헤더*

아래는 실무에서 그대로 채택할 수 있는 한 페이지 규약의 **섹션 헤더**. 전문은 워크시트에 인쇄된 상태로 배부.

📄 `# [팀 이름] · AI 코딩 도구 사용 규약 v1.0`

1. 수용 기준 테스트 존재 · CI 그린 · 리뷰어 승인 · `.env` 수정 자동 거부 · 4단어 이하 메시지 거부

2. 리뷰 정책 코드 오너 승인 1인 · **AI 트레일러 PR 은 2인** · 500줄 초과 시 재분할 요청

3. 비용 관리 조직: Anthropic Max20x · OpenAI Business · 월 예산 $X,000 · **80% 도달 시 슬랙 알림**

4. 보안 secret 은 절대 프롬프트에 붙이지 않음 · `.env` · `credentials.json` 자동 차단 · full-auto · Bypass 는 개인 실험 리포에서만

5. 데이터 프라이버시 고객 PII · 결제 · 미공개 재무 · 사내 기밀 문서 금지 · Consumer 계정은 회사 코드 금지

6. 감사 · 사고 대응 `Co-Authored-By:` 트레일러 필수 · 세션 로그 30일 보관 · 사고 시 롤백→재현→재구성→회고

📌 이 규약의 특성

- **슬로건 없음** — 모든 항목이 검증 가능한 규칙
- **숫자 명시** — 100줄 · 500줄 · $X,000 · 80% · 30일
- **감사 절차 명시** — 사고 이전에 사고 후 절차가 정해져 있어야 사고가 사고로 끝난다

출처: Software Engineering at Google

팀 규약 사례

규약 없이 도입한 팀 vs *규약과 함께 도입한 팀*

같은 도구를 같은 규모의 팀이 도입해도 결과가 크게 갈린다. 실패 사례 6선 (Part 1) 은 **공통적으로 규약 부재**였다 — 초기 4선은 개별 실수, 2026 신규 2건은 플랫폼·프로토콜 구조 실패.

🔴 규약 없이 도입

- **초기 3개월 사고 건수** — 잦음 (개인차 큼)
- **PR 리뷰 시간** — 폭증 (AI 큰 diff)
- **비용 예측 가능성** — 낮음 (예산 초과 흔함)
- **팀 만족도** — 양극화 (시니어 방어적)
- **6개월 이탈 위험** — 존재 (규약 없어 사고 후 반발)

🟢 규약과 함께 도입

- **초기 3개월 사고 건수** — 낮음 (일관됨)
- **PR 리뷰 시간** — 안정 (커밋 단위 규약)
- **비용 예측 가능성** — 높음 (상한 · 알림)
- **팀 만족도** — 균질 (역할별 워크플로)
- **6개월 이탈 위험** — 낮음 (규약이 방어막)

📌 참고 데이터

WRITER 2026 엔터프라이즈 AI 도입 리포트 — 조직의 **79% 가 AI 도입에서 어려움을 겪는다** (2025년 66% → 2026년 79%). 실패 원인은 인재·열정 부족이 아니라 **작동 방식을 조직 전체로 확장할 시스템 부재**다.

📖 WRITER · Enterprise AI adoption in 2026 · Coder · Inside AI Adoption

PART 6

도입 로드맵 · 정리

> 파일럿 → 확산 → 표준화 3단계 · 성공 지표 3+3 · 실패 신호 4 · 한 문장 결론.

출처: Software Engineering at Google

팀 규약 사례

도입 로드맵 — *3단계 표준*

방법론을 조직에 이식하는 데는 3단계가 표준. 이 순서를 건너뛰면 반드시 어딘가에서 무너진다.

🧪 단계 1 — 파일럿

기간 — 2~4주 **규모** — 5~10명 (자원자 + 시니어 1인 필수) **도구** — Claude Code · Codex 중 하나 **산출** — **규약 v0.1 초안** · 회고 리포트 **종료 조건** — 4주 후 회고, 성공 지표 3개 (PR 리드타임 · 롤백 비율 · 회귀 버그) 중 2개+ 달성 시 확산 승인 **실패 조건** — 규약 없이 사고 · 예산 200% 초과

🌊 단계 2 — 확산

기간 — 3~6개월 (분기 단위) **규모** — 조직 전체 개발자 **도구** — 규약 명시 도구 세트 (팀별 최대 2개) **지원** — 파일럿 참여자가 **팀별 앰버서더** · 주 1회 오피스아워 **예산** — 조직 계정 통합 · 상한·알림 설정 **종료 조건** — 6개월 후 성숙도 평가, 팀별 80% 상시 사용 시 표준화

🏛️ 단계 3 — 표준화

기간 — 연 단위 **규모** — 조직 전체 · 상시 **산출** — **규약 v1.0 확정** · 온보딩 문서 포함 **활동** — 신규 입사자 트레이닝에 AI 도구 세션 · 분기별 규약 개정 · 사고 회고 **감사** — 규제 요구 시 로그·트레일러 즉시 제출 가능

출처: Software Engineering at Google

팀 규약 사례

도입 로드맵 — 3단계 표준

출처: Software Engineering at Google

팀 규약 사례

성공 지표 · 실패 신호 — *3+3+4*

도입이 되고 있는지 · 실패로 가고 있는지를 지표로 감별. 정량·정성 섞어서 **3+3 이 실무 하한**.

📊 정량 지표 3

1. **PR 리드타임** — 개설~병합 중간값. 규약이 정착하면 하락→안정
2. **롤백 비율** — 병합 후 7일 내 롤백 PR 비율. **5%+ 위험 신호**
3. **회귀 버그 건수** — 프로덕션 발견 회귀. 도입 전 대비 증가 시 즉시 원인 조사

💬 정성 지표 3

1. **팀원 만족도** — 분기별 서베이. 시니어·주니어 만족도 차이 크면 워크플로 재설계
2. **자발적 규약 개정 제안 수** — 팀이 규약을 살아 있는 문서로 인식하는지 신호
3. **사고 회고 품질** — 개인 탓하기가 아니라 **프로세스 개선**으로 이어지는지

🚨 실패 신호 4 — 즉시 개입

- 파일럿 4주 안에 사고 **2건 이상**
- 조직 계정 월 예산 **200% 초과**
- 시니어 3인 이상 **"AI 못 믿겠다"** 이탈 의사
- 규약 문서 **3개월 개정 없음** (죽은 문서)

📌 실패 신호 4개는 참여자 각자 자기 팀에 적용해 보라 할 것. **대개 하나는 이미 걸린다.**

출처: Software Engineering at Google

팀 규약 사례

마무리 — *한 문장으로*

🎯 이 세션의 결론 한 줄

"AI 는 도구를 쥐어주지만, 팀은 규약을 쥐어야 무너지지 않는다."

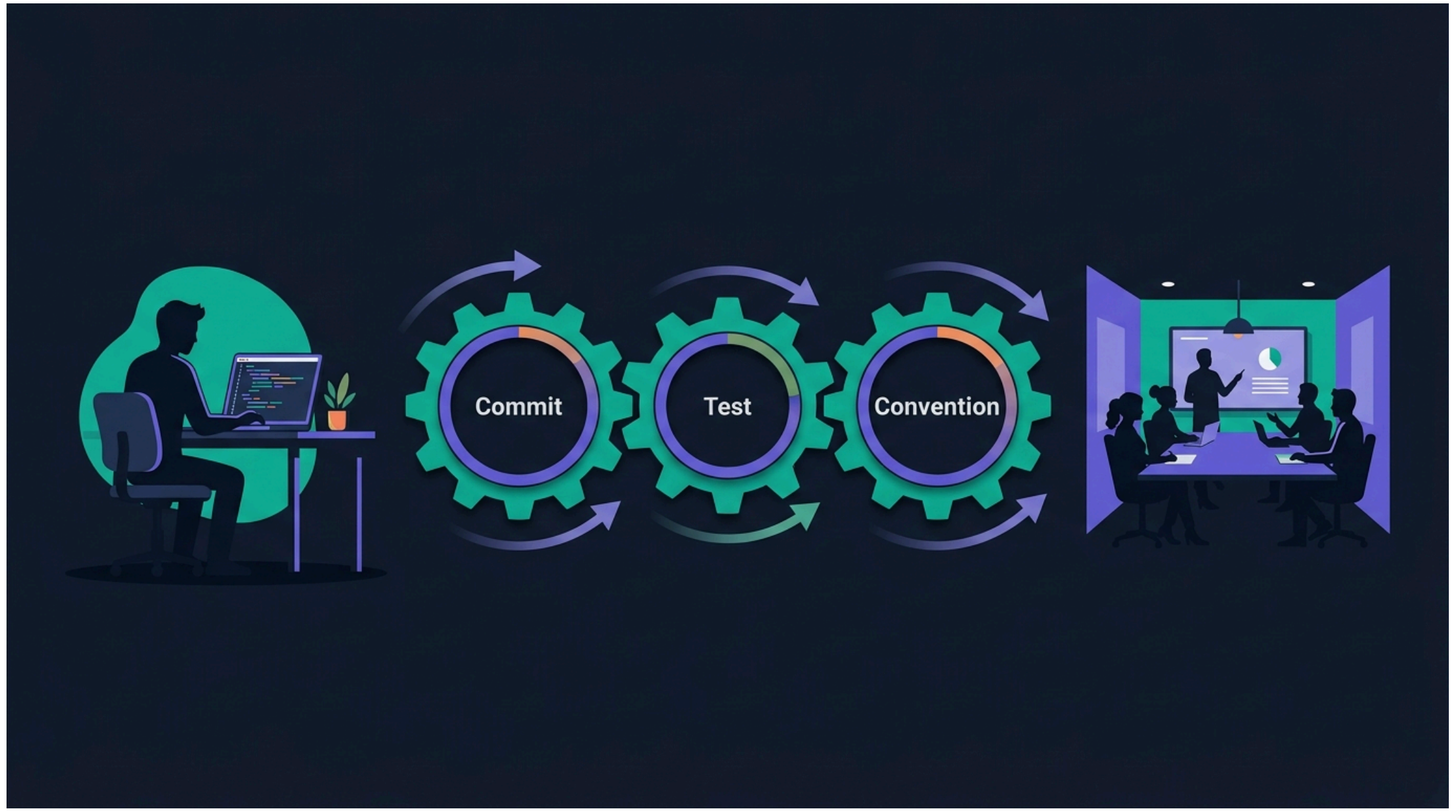
바이브코딩은 개인 도구로는 이미 성숙했다. 실패 사례들이 보여준 건 **도구의 한계가 아니라 규약 없이 도입한 팀의 한계**다. 이 세션에서 다룬 **커밋 · 테스트 · 팀 규약 · 로드맵** 은 그 실패를 피하는 최소 세트다.

📝 수강생 각자에게 남는 액션 하나

자기 팀의 규약 초안 v0.1 을 워크시트에 지금 쓰기 시작한다. 이 세션의 진짜 산출물은 그 초안이다. 6 섹션 헤더에 첫 문장씩만 채워도 v0.1 이 성립한다.

출처: Software Engineering at Google

팀 규약 사례



"AI 는 도구를 쥐어주지만, 팀은 규약을 쥐어야 무너지지 않는다."