강남대학교 · 2026 2학기 특강

Agentic Engineering 실무

회차별 목차 · 5회 × 4교시 · 슬라이드 421장

회차	날짜(목)	주제	교시
1회차	2026-11-05	코드를 제안하던 AI가 일을 하기까지	1, 2, 3, 4교시
2회차	2026-11-12	에이전트가 딴길로 새지 않게	5, 6, 7, 8교시
3회차	2026-11-19	더 오래 맡기고, 방법론을 패키징하다	9, 10, 11, 12교시
4회차	2026-11-26	병렬 개발과 품질	13, 14, 15, 16교시
5회차	2026-12-03	자유 공유와 토론, 스터디로	17, 18, 19, 20교시

각 교시 아래 번호는 슬라이드 장 번호입니다. 흐리게 표시한 줄은 부 표지·질문 시간 같은 구분 장입니다. 실습 교시(4·8·12·16)는 표시를 붙였습니다.

장인하

1회차 2026-11-05

코드를 제안하던 AI가 일을 하기까지

1교시 AI는 어떻게 코딩을 맡게 되었나 38장

2 AI 코딩 5년은 제안·일·오래 일함·공장, 네 시대로 나뉜다
3 모든 장치는 '실수를 막는 축'과 '계속을 넘기는 축' 중 하나에 놓인다
4 '바이브 코딩'이라는 말이 나온 지 1년, 코딩 에이전트가 제대로 일하기 시작했다
5 제안에서 일로
6 자동완성은 함수명과 주석을 보고 다음 줄을 제안했다
7 IDE 채팅은 묻고 답을 받게 했지만, 옮기고 실행하는 일은 사람 몫이었다
8 터미널 에이전트는 읽고, 고치고, 실행하고, 커밋까지 스스로 한다
9 터미널 에이전트는 사용자와 같은 셸 권한으로 움직여서 루프를 혼자 돈다
10 승인 요청이 너무 잦아, 사용자는 요청의 93%를 그냥 눌렀다
11 딴길로 새지 않게
12 스펙과 Task 목록이 에이전트가 딴길로 새는 것을 붙잡는다
13 대화를 압축할 때마다 정보가 사라지고, 입력이 길수록 성능도 떨어진다
14 TDD(Test-Driven Development, 테스트 먼저 쓰기)를 AI에게 시키면 '테스트 통과'가 곧 완료 기준이 된다
15 도메인 경계로 범위를 묶고, 프롬프트에 개발 수준을 적는다
16 서브에이전트는 맥락을 나눠 써서 메인 창을 아낀다
17 워크트리로 작업마다 폴더를 나누면 병렬 에이전트가 서로 덮어쓰지 않는다
18 여기까지, 질문 받습니다
19 더 오래 맡기고, 패키징하다
20 자동 승인은 사람이 누르던 승인을 분류기 모델이 대신한다
21 채팅에서는 사람이, 에이전트에서는 에이전트가 '계속'
22 '계속'을 자동화해, 반복에서 목표 달성까지 도는 장치로 발전했다
23 코드는 언어 서버(LSP)로, 최신 문서는 문서 조회로 필요한 만큼만 가져온다
24 Claude의 창이 10만에서 100만 토큰으로 커져, 압축 없이 더 오래 일한다
25 '하네스'는 모델을 둘러싼 장치 전부를 가리키는 말이 되었다
26 스킬은 방법론을 폴더 하나로 묶어 필요할 때만 읽게 한다
27 과열을 막는 스킬은 코드를 쓰기 전에 '정말 필요한가'부터 묻는다
28 병렬 개발과 품질
29 한 개발자가 여러 에이전트를 병렬로 돌리는 '에이전트 팩토리'가 나왔다
30 코디네이터가 일을 나눠 주고, 주기적으로 깨어 점검하며 며칠간 완주한다
31 스펙은 금방 낡으니, 결정은 ADR에 할 일은 이슈에 남긴다
32 CI 비용 폭증의 원인은 단가가 아니라 실행량이 4배 넘게 늘어난 것이다
33 머지 큐는 PR을 줄 세워 묶어서 한 번에 검증해 CI를 줄인다
34 테스트가 틀릴 수 있으니, 스모크 테스트는 AI가 직접 눌러 확인한다
35 가상 고객이 된 에이전트가 화면을 눌러보고, 피드백으로 스스로 고친다
36 각 장치는 앞 단계의 실패를 풀려고 차례로 쌓였다
37 코드를 쓰던 사람에서, 장치를 설계하는 사람으로
38 5회차 일정: 회차마다 내 하네스를 한 번씩 키운다

2교시 자동완성과 IDE 채팅 23장

2 이번 시간 순서: 제안 → 대화 → 남은 문제
3 예전 자동완성은 문법 규칙으로 이름 후보를 보여 줄 뿐이었다
4 생성형 자동완성은 이름이 아니라 함수 전체를 제안한다
5 생성형 자동완성은 코드 생성 모델을 에디터 안으로 가져왔다: 2021년 Copilot
6 2022년 Copilot은 월 10달러짜리 정식 개발 도구가 되었다
7 함수 이름과 주석이 곧 AI에게 주는 지시문이다
8 자동완성은 커서 앞뒤 코드를 모두 보고 가운데를 채운다
9 FIM은 논문과 실제 제품에서 효과가 확인됐다

10 자동완성은 열린 파일까지 보지만 실행 결과는 보지 못한다
11 Copilot을 쓴 쪽이 같은 과제를 55.8% 빨리 끝냈다
12 여기까지, 질문 받습니다
13 55.8%는 작은 과제 하나에서 나온, 범위가 넓은 값이다
14 ChatGPT는 대화를 열었지만 코드를 옮기는 일은 사람 몫이었다
15 IDE 채팅은 코드와 오류를 보고 답한다: 대표 예 Copilot Chat
16 자동완성은 코드 조각을, 채팅은 설명과 수정안을 돌려준다
17 AI 네이티브 IDE는 에디터 전체를 AI 채팅 중심으로 다시 짰다: 대표 도구 Cursor
18 AI 에디터의 채팅은 @로 고른 여러 파일을 맥락으로 삼는다(Cursor)
19 AI 에디터는 환각을 막으려고 맥락에 넣은 파일에만 변경을 적용했다(Cursor Composer)
20 자동완성도 채팅도 '계속'을 누르는 것은 사람이다
21 같은 버그도 자동완성과 채팅 모두 사람이 실행해 확인한다
22 오늘 본 것을 묶으면: 제안하는 AI의 구조와 한계
23 터미널 에이전트와 승인 피로

3교시 터미널 에이전트와 승인 피로

23장

2 채팅은 답만 주고, 루프는 사람이 돈다
3 Aider: 터미널에서 저장소를 고치고 커밋한다
4 터미널 에이전트를 주류로 만든 도구가 Claude Code다
5 터미널 에이전트는 내 셸 권한으로 움직인다
6 터미널이라서 파이프와 CI에도 들어간다
7 에이전트 루프: 모으고, 행동하고, 확인한다
8 에이전트는 도구 다섯 범주로 행동한다
9 루프 해부: 도구를 부르는 동안 계속 돈다
10 같은 버그를 맡기면 에이전트가 찾고 고치고 다시 시험한다
11 도구 호출 기록으로 에이전트를 감사한다
12 여기까지, 질문 받습니다
13 같은 시기에 Codex CLI와 Gemini CLI도 나왔다
14 사용자는 승인 요청의 93%를 그냥 승인한다
15 승인 피로의 해법: 덜 묻기·미리 허락·격리·대신 판단
16 안전한 명령은 미리 허용하고, 위험한 것은 금지해 둔다
17 프롬프트를 전부 끄면 번거로움은 없지만 보호도 없다
18 사례 1: 코드 동결 중에 운영 DB가 지워졌다
19 사례 2: 설치된 AI CLI가 악성코드의 도구가 되었다
20 권한은 deny → ask → allow 순으로 평가하고 첫 일치가 이긴다
21 샌드박스: 격리 안에서는 묻지 않는다
22 승인 비용·작은 창·미숙한 모델 때문에 짧은 일만 맡겼다
23 오늘 본 것을 묶으면: 맡기는 순간 권한이 설계 대상이 된다

4교시 내 하네스의 첫 버전: 에이전트에게 설계부터 시킨다 실습

13장

2 하네스도 에이전틱 엔지니어링 프로세스로 만든다: 분석 → 개요·ADR → 승인 → 구현 → 스모크
3 네 번의 실습은 하나의 에이전틱 엔지니어링 프로세스를 따라간다
4 "하네스 만들어 줘"라고만 하면, 에이전트가 빈칸을 가정으로 채운다
5 분석 대화는 1–7단계의 질문을 하나씩 묻고 끝난다
6 첫 프롬프트: 코드 전에 질문을 하나씩 하라고 못 박는다
7 분석 대화의 결과물은 짧은 개요 하나와 첫 ADR 하나다 (예)
8 저는 승인 전에 다섯 가지를 확인한다
9 승인한 뒤의 프롬프트: 원하는 것, 끝나는 조건, 끝나면 보여 줄 것
10 코드를 보는 대신, 실제로 시켜 보고 막히는지 확인한다
11 스모크 기록: 시나리오마다 기대와 실제를 나란히 적는다
12 다음 회차까지 해 볼 것, 다음 시간에 나누면 좋은 것
13 막힐 때: 대부분은 에이전트에게 주는 지시를 고치면 풀린다

2회차 2026-11-12

에이전트가 딴길로 새지 않게

5교시 SDD·Plan 모드와 압축의 한계 26장

2 승인을 정리해도, 에이전트는 딴길로 샌다
3 Plan 모드와 SDD는 같은 시기에 같은 문제를 푼다
4 SDD는 새 말이 아니다: 1987년 '설계 기술서'에서 2025년 '스펙 주도 개발'로
5 설계도를 먼저 그린다는 목적은 같다. 바뀐 것은 읽는 쪽과 스펙의 위상이다
6 SDD(스펙 주도 개발, Spec-Driven Development): 코드보다 스펙을 먼저 쓴다
7 SDD는 스펙을 얼마나 오래 쓰느냐로 세 수준이 나뉜다
8 Kiro는 요구사항, 설계, 작업 파일 세 개를 차례로 만든다
9 Kiro는 요구사항을 "WHEN…SHALL…" 문장 틀(EARS)로 쓴다
10 Spec Kit은 슬래시 명령으로 스펙에서 구현까지 단계를 밟는다
11 Spec Kit은 기능마다 spec.md와 tasks.md 뼈대를 채운다
12 constitution: 모든 변경에 적용하는 프로젝트 원칙
13 두 도구의 파일 흐름은 거의 같다
14 여기까지, 질문 받습니다
15 SDD가 항상 답은 아니다
16 Plan 모드: 읽고, 계획하고, 승인받고, 편집한다
17 Plan 모드는 승인 전에 소스를 안 고치고, 계획은 직접 고친다
18 Cursor도 계획을 파일로 남기고, 사람이 고친 뒤 실행한다
19 제가 쓰는 방식: 이슈 하나가 열두 단계를 지난다
20 세 번의 리뷰: 대상이 다르면 질문도 다르다
21 스펙을 써도, 일이 길어지면 창이 찬다
22 압축 뒤: 디스크의 파일은 남고, 대화는 요약된다
23 입력이 길어질수록 성능이 흔들린다: Context Rot
24 압축 대비: 진행 파일에 적어 둔다
25 세션 경계를 넘는 것은 파일뿐이다
26 스펙이 붙잡고, 파일이 남는다

6교시 TDD와 DDD 범위를 묶는 삼중 방어 24장

2 스펙이 있어도 "다 했다"는 말은 검증되지 않는다
3 스펙, 테스트, 경계가 서로 다른 실패를 막는다
4 TDD: 실패하는 테스트를 먼저 쓴다
5 사람이 분량 때문에 건너뛰던 TDD를 AI에게
6 완료는 에이전트의 말이 아니라 통과 결과
7 Red 확인이 먼저, Green은 그다음
8 에이전트는 테스트만 통과하는 지름길을 찾기도 한다
9 속임수 세 가지: 고치고, 건너뛰고, 박아 넣기
10 속임수에 대한 대응: 막고, 대조하고, 직접 본다
11 권한 deny: 테스트 파일은 수정 금지
12 여기까지, 질문 받습니다
13 훅: 수정 시도를 막고 이유를 돌려준다
14 리뷰어 서브에이전트: 스펙과 테스트를 대조한다
15 범위를 도메인 경계로
16 DDD: 업무 영역의 규칙을 코드 안의 모델로
17 유비쿼터스 언어: 사람과 코드가 같은 단어를 쓴다
18 바운디드 컨텍스트: 같은 단어, 영역마다 다른 뜻
19 예: 공고 · 지원 기록 · 알림, '상태'는 셋이 다르다
20 왜 에이전트에게 경계가 필요한가: 볼 범위 축소
21 클린 아키텍처: 의존은 안쪽으로만
22 프롬프트에 수준을 적는다: PoC냐, 엔터프라이즈냐
23 같은 기능도 PoC냐 클린 아키텍처냐에 따라 모양이 달라진다
24 오늘 본 것: 세 겹으로 범위를 묶는다

7교시 서브에이전트와 워크트리 23장

2 압축만으로는 창이 모자라서, 일을 나눠 맡기게 됐다
3 서브에이전트는 자기 창에서 일하고 요약만 돌려준다
4 탐색은 다른 창에서, 메인에는 결론만
5 기본 제공 서브에이전트 셋: 읽기 전용 둘, 만능 하나
6 서브에이전트는 마크다운 파일 한 장으로 정의한다
7 필드 몇 개로 정의하고, 같은 이름이면 위쪽 위치가 이긴다

8 서브에이전트는 자동 위임·말로·@-멘션··· agent로 부른다
9 독립된 조사는 동시에 시킨다
10 탐색·로그·리뷰는 맡기고, 되묻는 일은 메인에서
11 서브에이전트는 맥락을 아끼지만 사용량은 줄이지 않는다
12 여기까지, 질문 받습니다
13 한 폴더에서 에이전트 둘이 일하면 서로 덮어쓴다
14 git worktree로 한 저장소에서 브랜치마다 작업 폴더를 갖는다
15 저장소 기록은 하나, 작업 파일은 각자
16 Cursor는 2025년 5월 백그라운드 에이전트로 병렬 실행을 내놓았다
17 Claude Code는 --worktree 한 줄로 격리된 세션을 연다
18 Codex 앱은 워크트리를 기본 기능으로 넣었다
19 둘을 합치면 서브에이전트마다 자기 워크트리를 갖는다
20 합칠 때 충돌: 격리는 충돌을 미룰 뿐
21 이어 붙이면 한 흐름: 계획에서 합치기까지
22 병렬 방법은 여럿이고, 서브에이전트·워크트리·/batch는 역할이 다르다
23 오늘 본 것: 창을 나누고, 폴더를 나누고, 다음은 '얼마나 오래'

8교시 하네스를 한 번 더 키운다: 딴길을 막는 장치들 실습

14장

2 제 방식: 스펙 개요 → Task → 이슈를 만들고, 이슈 하나를 리뷰까지 끝낸다
3 첫 프롬프트(예): 질문을 먼저, 구현은 승인 뒤에
4 에이전트가 물어야 할 질문은 단계마다 하나씩이다
5 8·13단계의 결정은 한 줄씩이다: 번호 붙은 요구와, 끝 조건이 있는 Task (예)
6 14단계: 장치마다 이슈 한 건을 만든다 (#1~#4는 메뉴의 예)
7 구현에 들어가기 전에 저는 다섯 가지만 확인한다
8 1–6단계는 문서로 수렴시킨다: 분석, ADR 점검, 스펙, 계획 그리고 두 번의 리뷰
9 7단계: 테스트를 먼저 쓰고 실패를 보여 달라고 한 뒤, 막는 장치를 만들게 한다
10 구현은 테스트가 막고, 구현 리뷰는 별도의 리뷰어에게 맡긴다
11 예산이 다 되면 통과시키지 않고, 진행 파일에 적고 멈추게 한다
12 스모크: 이슈의 수용 조건을 일부러 어겨 보고, 막힌 것까지 기록한다
13 다음 회차까지 해 볼 것: 장치 하나를 골라 더하고, 어겨 보며 확인한다
14 막힐 때: 대부분은 하네스에게 주는 지시를 고치면 풀린다

3회차 2026-11-19

더 오래 맡기고, 방법론을 패키징하다

9교시 더 오래 맡기기: 자동 승인, '계속'의 자동화, 1M

26장

2 스펙과 테스트를 갖춰도, 사람이 남는 세 곳
3 수동 승인은 93%가 그냥 승인돼, 안전장치 구실을 못 한다
4 자동 승인(auto mode): 사람의 승인을 분류기 모델이 대신한다
5 분류기는 두 겹: 들어오는 글과 나가는 행동
6 호출은 세 단계로 걸러진다
7 분류기는 위험한 과잉 행동의 17%를 놓친다
8 auto mode는 배포·강제 푸시 같은 위험은 막고, 작업 폴더 안 일은 허용한다
9 Codex는 샌드박스 안은 자동, 밖으로 나갈 때만 승인으로 푼다
10 에이전트가 "다 했습니다"라고 멈추는 문제
11 반복 실행 루프의 대표 기법 Ralph: 같은 프롬프트를 셸 루프로 계속 넣는다
12 공식 플러그인: 바깥 루프를 세션 안의 훅으로
13 여기까지, 질문 받습니다
14 '계속'의 자리는 턴이 끝나는 순간의 훅
15 /loop: 정해진 간격으로 되풀이한다
16 Codex가 먼저: 목표를 주면 끝났다고 판단할 때까지
17 Claude Code /goal: 완료 조건을 주면 해제될 때까지 계속
18 /goal의 한 바퀴: 일하고, 평가하고, 이어 간다
19 /goal, /loop, Stop 훅: 다음 턴을 시작하는 쪽이 다르다
20 좋은 완료 조건은 네 조각으로 이루어진다
21 조건 문장 예: 끝 상태, 확인, 제약, 상한
22 오래 맡길수록 상한이 먼저다
23 100K에서 1M까지, 3년 만에 열 배
24 1M 정식 출시: 할증 없이, 기본 한도 그대로
25 창이 커도 채울수록 정확도는 떨어질 수 있다
26 승인·계속·창, 세 병목을 기계가 맡았다

10교시 하네스와 MCP, 주로 쓰는 MCP 도구

31장

2 오래 돌려 보니 남는 세 가지 문제
3 '하네스'는 Claude Code 출시 7개월 뒤에야 공식 글에 등장했다
4 첫 공식 사용: Claude Code를 움직이는 하네스
5 장기 실행 하네스: 세션이 끊겨도 이어 가게
6 세션 사이를 건너는 것은 파일과 git
7 OpenAI의 하네스 엔지니어링: 저장소가 기록 체계
8 에이전트 = 모델 + 하네스
9 MCP: AI와 도구를 잇는 하나의 표준
10 호스트, 클라이언트, 서버
11 claude mcp add 한 줄로 MCP 서버를 연결한다
12 보안 ①: 도구 설명에 숨긴 명령이 모델을 조종한다
13 보안 ②: 도구는 임의 코드 실행과 같다
14 도구가 많으면 일하기 전에 창이 찬다
15 여기까지, 질문 받습니다
16 도구 검색: 이름만 싣고, 필요할 때 불러온다
17 무엇을 하게 해 주는가로 고른다
18 코드를 아는 길은 언어 서버(LSP), 최신 API를 아는 길은 문서 조회다
19 문서 조회의 대표 도구 Context7: 버전에 맞는 문서만 가져온다
20 LSP의 대표 도구 Serena: 함수·클래스(심볼) 단위로 찾고 고친다
21 브라우저 자동화의 대표 도구 Playwright MCP: 접근성 트리 스냅샷으로 조작
22 브라우저 진단 도구 Chrome DevTools MCP: 콘솔과 성능까지 본다
23 GitHub MCP 서버: 이슈, PR, Actions를 에이전트가
24 Figma MCP: 디자인을 코드의 맥락으로
25 Supabase MCP: DB를 다루게 하되 가둔다
26 Sentry MCP: 운영 오류를 에이전트가 읽는다
27 Vercel MCP: 배포 상태와 로그를 묻는다
28 셸 출력 압축의 대표 도구 rtk: 명령 출력을 줄여서 에이전트에게 준다
29 정규식 재귀 검색 도구 ripgrep: 검색 결과를 걸러 주는 grep
30 창에 들어오는 것을 줄인다
31 하네스는 장치의 총합, MCP는 그중 연결

11교시 스킬: Superpowers에서 Ponytail까지 29장

2 CLAUDE.md는 늘 읽히고, 스킬 본문은 쓸 때만 읽힌다
3 스킬은 SKILL.md가 든 폴더 하나다
4 SKILL.md: 위는 이름표, 아래는 지침
5 스킬은 필요한 만큼만 읽혀 컨텍스트를 아낀다
6 스킬은 세 곳에 두고, 사람도 Claude도 부를 수 있다
7 스킬은 Claude 기능에서 여러 에이전트가 읽는 표준이 됐다
8 플러그인은 스킬·명령·훅을 한 묶음으로 나눠 준다
9 개발 방법론 전체를 스킬 묶음으로: 대표 사례 Superpowers(스킬 15개)
10 Superpowers는 설계를 승인받은 뒤 Task마다 새 에이전트가 구현한다
11 brainstorming은 만들기 전에 설계를 묻고 승인받게 한다
12 계획은 잘게 쪼개고, 실행은 Task마다 새 에이전트가 한다
13 TDD(테스트 먼저 쓰기) 스킬은 어긴 코드를 지우고 다시 쓰게 한다
14 skill-creator는 스킬을 쓰고 시험하고 고치는 일을 돕는다
15 여기까지, 질문 받습니다
16 frontend-design은 뻔한 화면을 피하라고 지침으로 눌러 둔다
17 문서 스킬 네 개가 Word · PDF · PPT · Excel 작업을 맡는다
18 webapp-testing은 에이전트가 브라우저로 직접 눌러 보게 한다
19 계획을 끝까지 캐묻는 인터뷰형 스킬 grill-me
20 Karpathy guidelines: 나쁜 버릇을 막는 네 원칙
21 Ponytail은 코드를 쓰기 전에 7단 사다리를 먼저 오르게 한다
22 스킬은 성격이 갈리니, 작업 시점에 맞는 것만 고른다
23 과열 방지: 에이전트는 시킨 것보다 더 만든다
24 제 스킬 지도: 작업 시점마다 하나씩
25 제 스킬 패키지: kdev 여덟 개
26 kdev-start는 범위부터 읽고 과한 절차를 얹지 않는다
27 description 한 문장이 스킬을 부른다
28 나만의 스킬: 쓰고, 시험하고, 고친다
29 정리: 방법론은 폴더로 옮겨 다닌다

12교시 하네스를 한 번 더 키운다: 스킬 · MCP · /goal 중 필요한 것 실습 15장

2 제 방식은 늘 같다: 질문 → 설계 승인 → 구현 → 스모크, 그리고 /goal로 이슈 하나
3 지난 시간에 매번 말로 시킨 순서가 스킬 후보다
4 첫 프롬프트(예): 코드 전에 질문을 하나씩 하고, 설계는 ADR로 남기라고 한다
5 에이전트가 물을 질문은 격리, 리뷰, 스킬, MCP, 완료 조건에 걸쳐 있다
6 15단계의 결과는 ADR 한 건의 결정 요약이다
7 승인 전에 저는 다섯 가지만 확인한다
8 스킬은 하네스에게 시키고, 실제로 불리는지로 확인한다
9 MCP도 하네스에게 시키고, 붙었는지 직접 불러 보게 한다
10 이슈 하나를 맡기는 /goal은 끝 상태·증명·상한을 말로 적는다
11 /goal이 없는 도구라면 상한이 있는 반복 실행 루프를 시킨다
12 스모크: 스킬이 불리는지 보고, 일부러 어기게 시켜 본다
13 이슈 하나를 /goal로 맡기고, 끝나면 실행 기록을 보여 달라고 한다
14 다음 회차까지 해 볼 것: 고른 장치를 더하고, 돌려 보고, 기록해 둔다
15 막힐 때: 하네스에게 이렇게 다시 말한다

4회차 2026-11-26

병렬 개발과 품질

13교시 에이전트 팩토리, 병렬 에이전트 관리, 코디네이터-배치

27장

2 하네스를 오래 돌리게 되자, 병목은 '하나뿐'이 되었다
3 같은 회사 모델도 입력 단가가 최대 100배 차이 난다
4 한 사람이 큰 프로젝트를 맡으려면 하네스가 여러 개여야 한다
5 여섯 달 만에 모든 도구가 같은 방향으로 갔다
6 GitHub는 여러 회사의 에이전트를 한곳에서 관리하겠다고 밝혔다
7 Codex 앱은 에이전트마다 저장소 사본을 따로 준다
8 agent teams: 팀 리드가 독립된 세션의 팀원들을 이끈다
9 Cursor 3는 에이전트 여러 개를 병렬로 돌리는 새 창을 내놓았다
10 병렬 에이전트 관리의 대표 도구, Orca
11 Orca: 작업 하나 = 워크트리 하나 = 에이전트 하나
12 Orca는 터미널에서 도는 에이전트면 모두 돌릴 수 있다
13 Orca 한 화면에서 열 개 안팎의 작업을 감시한다
14 여기까지, 질문 받습니다
15 Orca: 같은 프롬프트로 세 에이전트를 경주시키고 하나만 고른다
16 Orca: 에이전트가 orca 명령으로 작업과 워커를 직접 만든다
17 Orca는 빨리 바뀌고, 기본값이 권한 우회라 주의해야 한다
18 코디네이터-배치 패턴: 코딩하지 않는 세션이 일을 나눈다
19 코디네이터는 읽고 보내고, 워커는 코딩한다
20 /batch: 큰 변경을 독립 단위로 쪼개 워크트리에 뿌린다
21 /loop: 간격을 생략하면 Claude가 스스로 깨울 시각을 정한다
22 한 사이클: 깨어나서 읽고, 판단하고, 보내고, 다시 잔다
23 며칠에 걸친 완주: 배치 → 점검 → 재배정 → 마무리
24 코디네이터는 깨어나 PR·이슈를 읽고 실패한 일을 재배정한다
25 이렇게 망가지고, 이렇게 막는다
26 안전장치는 도구가 이미 갖고 있다
27 묶으면: 병렬은 쉽고, 나눠 주고 점검하는 일이 어렵다

14교시 SDD 이후: ADR과 이슈 원장

28장

2 SDD: 스펙이 사람과 AI의 기준이 된다
3 문제 1: 스펙은 길어지고, 길어지면 낡는다
4 문제 2: 에이전트는 낡은 스펙도 믿는다
5 문제 3: 병렬로 일하면 한 스펙에서 부딪힌다
6 그래서 스펙 하나를 성격별로 나눴습니다
7 결정과 근거를 남긴다는 생각은 1992년부터, 'ADR'이라는 이름은 2011년부터
8 근거를 남긴다는 목적은 같다. 바뀐 것은 크기, 놓이는 곳, 읽는 쪽이다
9 ADR: 결정 하나와 그 이유를 적은 기록
10 Nygard의 ADR 원형은 다섯 칸, 한두 쪽이면 끝난다
11 상태가 바뀌어도 기록은 지우지 않는다
12 MADR: 마크다운으로 가볍게 쓰는 ADR
13 세 서식, 같은 뼈대와 다른 칸
14 제 서식: 여덟 칸, 둘이 새로 붙었다
15 여기까지, 질문 받습니다
16 예시 ADR (1/2): 배경과 대안
17 예시 ADR (2/2): 선택, 영향, 다시 판단할 조건
18 '다시 판단할 조건'이 낡음을 알려 준다
19 왜 에이전트에게 ADR이 좋은가
20 모든 결정에 쓰지는 않습니다
21 이슈 원장: 설계도가 곧 작업 목록
22 프로젝트: 이슈와 PR이 표·보드·로드맵이 된다
23 이슈 하나 = 작업 하나, 큰 일은 하위 이슈로
24 PR 폭증은 이슈를 합쳐 줄인다
25 이슈를 닫는 규칙: 키워드는 이슈마다 하나씩
26 운영 기준 18단계: ADR은 7단계
27 이슈 하나가 지나가는 열두 칸
28 묶으면: 짧은 결정 기록과 한 장씩 닫는 이슈

15교시 초록불이 쏟아질 때, 그 초록불을 믿을 수 있나 26장

2 PR이 늘면, 병목은 CI와 검증으로 옮겨 간다
3 단가는 내렸다: 호스티드 러너 최대 39% 인하
4 터진 것은 실행량이다: 한 달 32.6억 회, 1년 전의 4배
5 하루 머지 30,401건: Anthropic은 소스 관리를 새로 만들었다
6 단가는 못 건드리니, 횟수와 시간을 줄인다
7 1. 낡은 실행은 취소한다: concurrency
8 2. 관계없는 변경은 거른다: 경로 필터
9 3. 받은 것은 다시 받지 않는다: 캐시
10 폴더 필터를 넘어: 영향받는 검사만 고른다
11 4. 묶어서 검증한다: 머지 큐
12 큐는 '누적 후보'를 만들어 검사한다
13 여기까지, 질문 받습니다
14 큐를 쓰려면 merge_group도 받아야 한다
15 GitLab에서는 머지 큐를 '머지 트레인'이라 부른다
16 병렬화는 기다림을, 범위 축소는 러너·분을 줄인다
17 테스트가 틀리면, 통과는 아무것도 증명하지 못한다
18 AI가 사람처럼 화면을 보고 눌러 볼 수 있게 됐다: computer use
19 브라우저 자동화(접근성 스냅샷)의 대표 도구, Playwright MCP
20 콘솔과 네트워크를 직접 읽게 하는 도구, Chrome DevTools MCP
21 내 로그인 상태 그대로 눌러 보는 확장, Claude in Chrome
22 무엇을 확인하느냐로 도구를 고른다
23 시나리오 한 개는 다섯 칸이면 된다
24 시나리오는 저장소 안의 마크다운 파일이다
25 "통과했습니다"가 아니라 증거 파일을 남긴다
26 CI는 줄이고, 초록불은 직접 확인한다

16교시 페르소나 평가, REA, 마지막 실습 실습 30장

2 스모크가 통과해도 남는 질문: 쓸 만한가
3 페르소나 평가: 고객이 되어 눌러보고, 고친다
4 LLM 가상 사용자 시뮬레이션의 연구 사례, UXAgent: 사용성 시험을 미리 돌려 본다
5 LLM 페르소나 시뮬레이션 라이브러리, TinyTroupe: 성격과 목표를 준 가상 인물
6 가상 사용자는 사전 점검이다, 최종 판정이 아니다
7 페르소나는 지어내지 않고, 실제 고객에서 뽑는다
8 페르소나 파일: 목표와 포기 조건까지 적는다
9 실행: 과업은 한 줄, 증거는 파일로
10 관찰 항목: 판정보다 증거 한 줄
11 피드백은 이슈로: 재현 경로와 증거를 붙인다
12 자기 개선 루프: 평가가 다음 배치를 만든다
13 여기까지, 질문 받습니다
14 제 서비스에서 돌리는 모습
15 역공학(리버스 엔지니어링) MCP의 예, REA: 소스 없이 앱의 동작을 조사한다
16 REA 쓰는 법: 에이전트에게 묻고, 근거를 받는다
17 REA는 앱 안을 넓게 들여다보되, 허가받은 소프트웨어에만 쓴다
18 이슈 하나를 PR에서 스모크까지 보내고, 결과는 코드 대신 스모크로 확인한다
19 하네스는 PR 전에 여섯 가지를 하나씩 묻는다
20 첫 프롬프트로 질문부터 시키고, 다섯 가지를 보고 승인한다
21 PR과 CI는 만들게 시키고, 결과 요약을 보여 달라고 한다
22 머지하고 나면, 합쳐진 앱을 눌러 보고 표로 남긴다
23 이슈 하나를 실제로 보내 보고, 규칙을 일부러 어겨 본다
24 머지한 앱에 페르소나 평가를 돌리고, 피드백을 이슈로 남긴다
25 피드백은 다음 계획이 되고, 새 이슈로 처음부터 이어진다
26 내 하네스를 한 번 더 키운다: 무엇을 하는 하네스인지는 내가 정한다
27 고를 수 있는 장치 여섯 가지: 필요한 것만 골라 하네스에게 시킨다
28 다음 회차까지 해 볼 것과, 그때 나누면 좋은 것
29 자유롭게 나누고 이야기합니다: 돌릴 때마다 한 줄 남겨 두면 이야깃거리가 됩니다
30 막힐 때: 하네스에게 이렇게 다시 말한다

5회차 2026-12-03

자유 공유와 토론, 스터디로

17교시 각자 만든 하네스를 나누고 이야기하는 날 4장

2 형식은 자유입니다. 원하는 사람부터 나눕니다
3 이야기가 막히면 이런 질문에서 시작해 볼 수 있습니다
4 오늘 나온 것을 모아 두면 스터디로 이어 가기 쉽습니다

18교시 나누기와 토론 1부 4장

2 원하는 사람부터 나누고, 이야기가 이어지면 이어 갑니다
3 이야기가 막히면 이 카드 중 하나에서 시작해 봅니다
4 10분 휴식

19교시 나누기와 토론 2부 4장

2 원하는 사람부터 나누고, 이야기가 이어지면 이어 갑니다
3 이야기가 막히면 이 카드 중 하나에서 시작해 봅니다
4 10분 휴식

20교시 마무리와 앞으로 볼 것 13장

2 1교시의 연표: 네 시대를 지나왔습니다
3 '막는 장치'와 '계속' 두 축에서 내 하네스의 빈 곳을 찾습니다
4 장치는 앞 단계의 실패에서 나왔습니다
5 병목은 타이핑에서 결과 판정으로 옮겨 갔습니다
6 코드를 쓰던 사람에서, 장치를 설계하는 사람으로
7 하네스는 이 루프로 계속 자랍니다
8 CHANGELOG는 새 기능을 가장 먼저 알려 주지만, 이유는 말해 주지 않습니다
9 기능을 어떻게 쓰는지는 공식 문서에서 읽습니다
10 설계 이유와 실패담은 엔지니어링 블로그에 있지만, 최신 기능표는 아닙니다
11 GitHub Changelog로 Copilot·Actions의 변경을 날짜별로 비교해 봅니다
12 출처는 가까운 쪽부터 믿습니다
13 수고하셨습니다