공유 폴더로 ChatGPT와 Claude가 협업한다 — 파일 기반 이기종 AI 협업 체계로 AI 팀을 운영하는 법

- 공모 분야: 업무 생산성 개선을 위한 AI 활용
- 한 줄 소개: 공유 폴더를 작업 인계 장소로 삼아 ChatGPT(Codex)와 Claude(Claude Code)가 작업 지시서와 결과를 주고받으며 서로 검증하게 한, AI 팀 운영법입니다.

1. 문제: PM 한 사람이 팀의 일을 다 해야 할 때

저는 약 18년 동안 여러 산업의 IT 프로젝트에서 개발자로 시작해 분석·설계 PL과 PM을 맡아 왔고, 지금은 제조 현장 IT 시스템을 만드는 회사에 다니고 있습니다. 사람 팀을 이끌어 본 경험이 있기에 AI도 "팀"으로 꾸려 보자는 생각을 하게 됐습니다.

AI 챗봇 하나에게 "이것 좀 만들어 줘"라고 하면 빨리 나오기는 합니다. 그런데 세 가지 문제가 반복됐습니다.

1. AI가 스스로 틀린 것을 모릅니다. 그럴듯한 코드와 보고서가 나오지만, 실제로 돌려 보면 빠진 곳이 있습니다.
2. 방향이 조금씩 흘러갑니다. 대화가 길어지면 처음 요구와 다른 것을 만들고 있습니다.
3. 확인을 결국 제가 다 해야 합니다. 빨라진 만큼 검토 부담이 저에게 몰립니다.

그래서 생각을 바꿨습니다. AI 하나를 잘 쓰는 것이 아니라, AI 여럿을 팀으로 꾸리자.

이 글은 기관의 공식 사업을 소개하는 것이 아니라, 제가 개인 연구개발 과정에서 스스로 설계하고 적용해 온 AI 협업 운영 방식입니다.

2. 방법: AI에게 "직무"를 준다

사람 개발팀처럼 역할을 나눴습니다. 핵심은 같은 AI가 만들고 검사하지 않게 하는 것입니다.

역할	도구	업무
분석·독립 검토 AI	OpenAI GPT / Codex	요구사항과 현재 상태의 차이 정리(gap분석), 마지막 독립 검토
설계·통합 검증 AI	OpenAI GPT / Codex	상세 설계, 통합 결과 검증
작업 분해·게이트 AI	Anthropic Claude / Claude Code	설계를 작은 작업 지시서로 쪼개고, 결과를 통과·보완으로 판정
개발·자동 시험 AI	Anthropic Claude Code	지시서대로 구현, 자동 시험, 결과 보고
최종 결정	사람 PM	범위·금지선 결정, 예외 승인, 실제 화면 확인, 최종 판단

왜 "이기종"인가

의도적으로 분석·설계·최종 검토는 한 회사의 AI, 작업 분해·개발·검증은 다른 회사의 AI에게 맡겼습니다. 서로 다른 서비스와 모델은 작업 환경과 답변 경향이 달라, 같은 결과물을 다른 관점에서 보게 하는 데 도움이 됐습니다. 수행역할과 검증역할을 다르게 두면 한쪽이 놓친 것을 다른 쪽이 걸러 낼 가능성이 높아집니다.

이 사례에서도 그랬습니다. Claude 쪽 게이트를 통과한 결과물을, 개발 미참여 GPT 쪽 독립 검토가 다시 보자 결함 3건이 더 나왔습니다(6장).

작업은 "공유 폴더"로 주고받습니다

서로 다른 회사의 AI는 서로 소통할 방법이 없습니다. ChatGPT 쪽(Codex)과 Claude 쪽(Claude Code)은 각자의 프로그램 안에서 일합니다. 그래서 둘 다 볼 수 있는 폴더 하나를 "작업 인계 장소"로 정했습니다.

```
업무폴더/날짜/
  01_작업지시/    ← 작업 지시서를 놓는 곳
  02_결과/        ← 개발AI가 결과 보고서를 놓는 곳
  03_검토/        ← 검사AI가 검토서를 놓는 곳
  04_최종판정/    ← 통과 / 보완 판정을 놓는 곳
```

1. 분석AI가 분석문서를 생성합니다.
2. 설계AI가 분석문서를 검토하고 설계문서를 생성합니다.
3. 작업지시AI가 설계문서를 검토하고 01_작업지시에 작업 지시서 파일을 생성합니다.
4. 개발AI는 폴더를 모니터링하다가(폴더 감시) 새 파일이 감지되면 일을 시작합니다.
5. 끝나면 결과를 02_결과 폴더에 생성하고, 작업지시AI가 그것을 보고 검증합니다.
6. 이렇게 파일이 오가며 작업이 계속 이어집니다. 사람은 범위와 금지선을 정하고, 판단이 필요한 예외를 승인하며, 마지막에는 실제 화면을 직접 확인합니다.

처음엔 이렇게 폴더 감시로 작업을 주고받았고, 지금은 AI끼리 짧은 알림 메시지로 "몇 번 끝났다, 파일은 어디"만 주고받습니다. 어느 쪽이든 원칙은 "정본은 파일"입니다. 판단의 근거는 언제나 폴더에 작성된 문서입니다. 덕분에 AI가 기억을 잃거나 대화가 끊겨도 폴더만 보면 어디까지 왔는지 그대로 이어집니다. 그 자체가 작업 이력이 됩니다.

사용 도구: Claude 쪽은 Claude Desktop의 기존 작업 세션들이 Monitor 도구로 담당별 작업 지시 폴더를 감시하다가, 새 지시서가 생기면 작업을 시작했습니다. Codex 쪽은 Codex 앱의 자동화 기능이 5분마다 결과 폴더를 수집·검증하고, 다음 작업 지시서를 Markdown 파일로 다시 발행했습니다. 즉 도구 조합은 Claude Desktop Monitor, Codex 앱 자동화(5분 주기), 공유 폴더의 Markdown 파일 세 가지입니다. 작업마다 새 프로그램을 띄우지 않고 이미 실행 중인 작업 세션을 그대로 이어 썼으며, 작업은 사람이 정한 시작 신호를 주면 시작됩니다.

일하는 순서: 작업 지시서

AI끼리 일을 넘길 때는 대화가 아니라 문서(작업 지시서)로 인계합니다.

```
분석서 → (교차 검토) → 설계서 → (교차 검토, 개정) → 작업 지시서 1건
   → 개발 + 자동 시험 → 결과 보고서 → 게이트 검증 → 통과면 다음 지시서
   → 마지막에 독립 검토 → 통합 검증 → 사람 확인
```

작업 지시서 한 장에는 항상 네 가지가 들어갑니다.

- 범위: 고쳐도 되는 곳과 절대 건드리면 안 되는 곳
- 완료 조건: 무엇이 되면 끝인지(시험 항목 표)
- 검증 방법: 개발AI 말고 누가, 어떻게 다시 확인하는지
- 보고 형식: 결과를 어떤 숫자로 보고하는지



그림 1. 역할 분담과 협업 흐름도

3. 운영 기록: 두 달 반, 작업 지시서 370건

이 방식은 한 번 해 본 실험이 아닙니다. 공유 폴더에 남은 기록입니다.

항목	기록
운영 기간	2026년 8월 5일 ~ 10월 15일, 폴더가 남은 작업일 21일
작업 지시서	370건
결과 보고서	357건
폴더에 남은 문서	840건

집계 기준: 작업 지시서·결과 보고서는 각 폴더에 남은 문서 파일 수, 전체 문서는 작업 폴더 아래 문서 파일(Markdown) 수입니다.



그림 2. 운영 기록과 두 사례

4. 사례 1: 사람이 수면하는 9시간, AI 일곱이 일했다

제가 개인 연구개발로 만들고 있는 물류 운송 관리 시스템(TMS)의 공통 기반을 밤새 맡겨 본 실험입니다.

- 구성: 지휘·검증 AI 1(ChatGPT 쪽) + 개발 AI 6(Claude 쪽). 개발 AI는 각자 자기 폴더를 지켜보다가 작업 지시서가 놓이면 일했습니다.
- 시간: 0시 32분 시작 → 9시 32분 측정 종료(정확히 9시간).
- 결과: 최종 자동 시험 534건 전부 통과, 시스템 검사 오류 0건. 다만 이 534건은 공통 기반과 경계 조건을 검증한 시험이고, 업무 전체가 완성됐다는 뜻은 아닙니다. 개발 AI 여섯 중 다섯은 통과, 하나는 보완 1건이 남아 기록했습니다.
- 검증이 한 일: 지휘 AI가 결과를 받을 때마다 "정상 경로는 통과했지만 잘못된 입력·경계값에서 뚫린다"는 것을 찾아 보완 지시서를 다시 발행했습니다. 모두 다섯 차례, 6건·5건·5건·3건·3건이었습니다.

솔직하게 남긴 실패사례도 있습니다. 이것이 오히려 이 방법의 핵심 교훈입니다.

- 개발 AI 다섯이 전날 폴더를 지켜보고 있어서 지시서를 못 받았습니다. 약 25분을 잃고, 각자 실제로 보고 있는 폴더에 지시서를 다시 놓아 이어 붙였습니다.
- 새벽 2시~6시 반에 검증 AI가 사용 한도(토큰)를 모두 써서 멈춰, 결과 6건이 쌓인 채 개발 AI 전원이 기다렸습니다(약 4시간 22분). 6시 52분부터 자동화가 끝난 9시 32분까지도 같은 이유로 다시 멈춰, 마지막 결과 6건은 한도가 풀린 뒤 오후에 사람이 다시 지시해 최종 판정했습니다(13시 14분). 결론은 "개발 AI를 늘리는 것보다, 검증 AI가 깨어 있게 하는 것이 진짜 병목"이었습니다.

5. 사례 2: 데이터 분석 도구를 하루 만에 다시 만들기

이번엔 분석·설계까지 이기종으로 나눈 사례입니다.

제가 개인 연구로 만들고 있는, 제조 품질 데이터를 올리면 기본 분석·논문 기법 비교·현장 지식 연결까지 해 주는 분석 도구를 다시 만든 사례입니다. 이전 버전은 결과를 보여 주기만 했고, 사용자가 자기 데이터를 올려 분석할 수는 없었습니다.

단계	업무	기록
분석	분석 AI가 현재·요구·차이를 정리 → 게이트 AI가 인용된 코드와 설정을 전부 열어 대조 검증	분석서 1건, 검증서 2건
설계	설계 AI가 상세 설계 → 분석 AI가 재검토 → 세 번 개정	설계서·검토서 5건
개발	작업 지시서 11건(8단계 + 보완 3건)을 순서대로 구현	결과 보고서 11건
시험	단계마다 자동 시험을 붙임	자동 시험 0건 → 296건
검증	독립 검토 → 통합 검증 296/296 통과 → 사람 최종검증	판정서 2건

분석서 작성부터 통합 검증 통과까지 약 하루(24시간 46분) 가 걸렸습니다(분석 시작 10월 12일 13시 40분 → 통합 검증 판정 10월 13일 14시 26분).

참고로, 같은 분량(시험 포함 코드 약 1만 5천 줄·자동 시험 296건·작업 단계 11개)을 사람 팀으로 했다면 분석 1명·설계 1명·개발 2명·시험 1명이 4~6주, 약 5~7.5인월은 걸렸으리라 봅니다. 실측이 아니라 PM 경험에 따른 추정입니다.

6. 이기종 교차 검증으로 잡아낸 것들

이 방법의 진짜 가치는 속도보다 AI가 혼자서는 못 찾는 실수를 다른 AI가 발견하는것에 있습니다.

- "시험 통과"였는데 실제 화면에서 깨진 것 4건: 자동 시험이 전부 통과한 상태에서, 실제 데이터를 실제 화면에 올려 눌러 보니 결함 4건이 나왔습니다 (예: 목록에서 둘째 항목을 승인하면 화면이 첫째 항목으로 돌아가 다음 버튼이 엉뚱한 항목에 눌림). 항목이 하나뿐인 시험 데이터로는 드러나지 않는 문제였습니다.
- "완료" 판정 뒤에 다른 회사 AI가 찾은 것 3건: Claude 쪽 게이트가 통과시킨 뒤에도, 개발 미참여 GPT 쪽 독립 검토가 다시 보니 "다른 데이터의 결과가 남아 보이는 문제" 등 3건이 나왔습니다. 모두 고치고 재검증했습니다.
- 제조데이터 분석 점수가 너무 좋을 때 의심한 것: 한 단계에서 예측 점수가 높게 나오자 평가를 다시 따져 봤습니다. 평가 단위를 기계의 샷에서 생산 묶음으로 바꾸자 점수가 크게 내려갔고, 기본값을 생산 묶음으로 바로잡았습니다. 같은 생산 묶음의 기록이 학습과 평가에 섞이면 실제보다 좋아 보일 수 있기 때문입니다.
- 개발자가 설계자보다 나은 길을 찾았을 때: 작업 지시서대로 구현해 보니 엉뚱한 열까지 후보로 잡히는 것을 실측으로 확인하고, 근거를 붙여 게이트 AI에게 "설계 변경 요청"을 올려 승인받았습니다. 설계서 대로만 하지 않고, 개발하면서 설계가 잘못된걸 말하도록 했습니다.

7. 사람이 지켜야 할 윤리·책임

AI에게 많이 맡길수록 사람이 지켜야 할 원칙이 분명해야 했습니다.

1. 결정은 사람이 합니다. AI는 분석·정리·선택지 제시까지만 하고, 판단이 불분명한 것은 "관측만 하고 미판정된 건"으로 올리게 했습니다.
2. 틀린 것은 기록합니다. AI가 틀리면 "무엇이 틀렸고 → 왜 → 무엇을 고쳤고 → 다음엔 무엇부터 볼지"를 기억 파일에 남기게 했습니다. 같은 실수가 줄었습니다.
3. 운영 데이터와 원본은 건드리지 않습니다. 원본 자료와 운영 저장소는 읽기만 하고, 시험은 복사본에서만 하게 했습니다. 작업 전후로 파일 지문(해시)을 비교해 "원본이 수정되지 않았음"을 증명하게 했습니다.
4. 배포와 외부 전송은 사람이 직접 말할 때만. "배포해 줘"라는 말이 없으면 AI는 서버에 올리지 않습니다.
5. 검증의 크기는 변경의 크기에 맞춥니다. 문구 하나 고쳤는데 전체 시험을 다시 하지 않도록, 변경된 범위만 시험하게 했습니다.

외부 저작물 인용은 없으며, 이 글의 모든 수치는 본인 운영 기록입니다.

8. 누구나 따라 하는 3단계

작업 지시서 양식 (공개용 빈 양식)

필수 항목 4가지로 구성, 대화 대신 문서로 작업 인계

번호·제목	[번호]_[담당]_[한 줄 제목]
목적·근거	작업 목적 및 근거 문서(분석서·설계서·검토서) 위치
① 범위	변경 허용 범위 / **변경 금지 범위**(원본·운영 데이터·배포)
② 완료 조건	시험 항목: 번호 · 대상 · 기대 결과 · 통과 여부(개발 AI / 검증 AI)
③ 검증 방법	개발 AI 외 검증 주체 및 재확인 방법(명령·화면)
④ 보고 형식	결과 보고서 위치 및 회신 형식(시험 결과 · 변경 파일 · 잔여 사항)
사람 판단 사항	판단이 필요한 사항은 수정하지 않고 보류 사항으로 보고

그림 3. 작업 지시서 양식(공개용 빈 양식)

1. 만드는 AI와 검사하는 AI를 다른 회사 것으로 두십시오. 처음엔 둘이면 충분합니다. 예: 하나로 만들고, 다른 회사 AI에게 "틀린 곳을 찾아라"고 맡깁니다.
2. 대화 대신 공유 폴더에 작업 지시서 파일로 넘기십시오. 폴더 네 개(작업지시·결과·검토·판정)와 "범위·완료 조건·검증 방법·보고 형식" 네 줄짜리 양식 하나로 시작합니다.
3. 마지막 확인은 실제 화면에서, 사람이 하십시오. 자동 시험이 다 통과해도 실제 데이터를 넣고 직접 테스트합니다.

오늘 바로 해 보는 가장 작은 형태는 다음과 같습니다. ① 바탕화면에 폴더 하나를 만듭니다. ② 챗 창 A(만드는 AI)에 "결과를 이 폴더의 결과.md로 저장해 달라"고 요청합니다. ③ 다른 회사 AI의 챗 창 B에 "결과.md를 읽고 틀린 곳·빠진 곳만 검토.md로 정리해 달라"고 요청합니다. ④ 사람은 검토.md만 읽고 결정합니다. 프로그램 설치나 명령어는 필요하지 않습니다. 부록의 자동화 설정은 이 과정을 자동으로 돌리고 싶을 때 참고하십시오.

9. 효과와 한계

효과

- 혼자서도 분석→설계→개발→검증의 팀 공정을 약 하루(24시간 46분) 만에 실행했고, 작업 지시서 11건과 자동 시험 296건이 남았습니다.
- 별도 개발 플랫폼 없이 유료 AI 구독 2종과 공유 폴더만으로, 밤새 9시간 동안 AI 일곱이 일하게 할 수 있었습니다.
- 결과물마다 문서와 시험이 남아, 나중에 누가 봐도 왜 그렇게 만들었는지 추적됩니다.
- 사람은 "만드는 일"보다 "결정하는 일"에 시간을 씁니다.

한계

- AI 사용료가 듭니다. 2026년 10월 현재 제 구독은 ChatGPT 월 3만 원 안팎과 Claude 월 100달러 요금제로, 합쳐 월 17만 원 안팎입니다(환율·요금제에 따라 달라집니다). 사람 인건비와 비교하면 작지만 0원은 아닙니다.
- AI끼리의 검증도 완벽하지 않습니다. 자동 시험이 모두 통과해도 실제 화면에서 결함이 나왔듯, 마지막 확인은 반드시 사람의 몫입니다.

10. 향후 계획

수상하게 되면 상금은 공용 장비의 사용 대기 시간을 줄이기 위한 전용 로컬 AI 워크스테이션 구축에 쓸 계획입니다. 1단계로 24GB GPU 1장 환경에서 27B급 모델 상시 운영, 그래프 RAG 연구, 제조 데이터 분석, 민감 자료의 로컬 처리 비중 확대를 추진하고, 이후 예산과 실제 성능 검증 결과에 따라 GPU를 더해 양자화 70B급 모델 시험으로 넓히겠습니다.

부록. 재현 방법 — 2026년 8월 28일 야간 작업의 실제 설정

이 부록은 사례 1(야간 9시간 자율 개발)에서 실제로 사용한 설정과 지시문을 원문 그대로 싣습니다. 사람 이름, AI 별칭, 컴퓨터 경로, 대화·자동화 식별자는 [ ]로 가렸고, 내부 용어와 개인적인 표현은 일반 용어로 바꾸었습니다. 그 밖의 내용은 원문과 같습니다.

A. 전체 흐름

```
사람이 작업 시작을 알림
→ Claude Desktop의 기존 작업 세션들이 각자 담당 작업 지시 폴더(01_tasks) 감시 시작(5초 주기)
→ 새 작업 지시서가 생기면 수행하고, 결과 보고서를 자기 결과 폴더(02_results)에 저장
→ Codex 앱 자동화가 5분마다 결과 폴더를 확인
→ 설계 정본 · 실제 변경 파일 · 시험 결과를 대조해 통과(PASS) 또는 보완(REVISE) 판정
→ 같은 작업자의 작업 지시 폴더에 다음 작업 지시서 또는 보완 지시서를 새 파일로 발행
```

두 주기를 구분해야 합니다. **Codex 자동화는 5분 주기**로 결과를 모아 검증하고, **Claude 작업자는 5초 주기**로 자기 폴더에 새 지시서가 생겼는지 봅니다.

B. 준비물

준비물	용도
공유 폴더	날짜 폴더 아래 작업자별 폴더를 두고, 각자 01_tasks(작업 지시서)와 02_results(결과 보고서)를 가짐. 진행 현황 원장 파일 1개
Claude Desktop 작업 세션(작업자 수만큼)	이미 실행 중인 세션을 그대로 씀. 새 세션을 띄우지 않아 그동안 쌓인 맥락을 유지
Git Bash	Claude 작업자의 폴더 감시 명령을 실행하는 셸
Codex 앱 자동화 기능과 미리 열어 둔 대화	5분 주기 heartbeat가 이 대화에서 결과를 검증하고 다음 지시서를 발행

```
[작업 루트]/[날짜 8자리]/
  [작업자 1]/01_tasks/   [작업자 1]/02_results/
  …
  [작업자 6]/01_tasks/   [작업자 6]/02_results/
  진행 현황 원장.md
```

C. Codex 자동화 — 결과 검증과 다음 지시서 발행

C-1. 자동화 프롬프트(원문)

```
[공유 폴더]W20260828 아래 [작업자 1~6]의 02_results를 확인하고, 새 결과가 있으면 [설계 정본 폴더] 정본과 실제 변경 파일 · 시험결과를 근거로 검증하십시오. PASS면 다음 안전한 개발 작업 지시서를 해당 담당자 01_tasks에 발행하고, REVISE면 원인과 완료조건이 분명한 보완 작업 지시서를 발행하십시오. 이미 처리한 결과는 다시 처리하
```

```
지 말고, [공유 폴더]W20260828W[진행 현황 원장].md에 시작 · 완료 · PASS · REVISE · 소요시간 · 병목을 누적하십시오. 사용자 기존 미커밋 변경을 보존하고 push · 배포 · 운영 DB DDL/DML · 운영 외부송신은 하지 마십시오. 선택이 필요한 항목은 임의결정 목록에 남기고 되돌릴 수 있는 안전한 다음 작업을 계속하십시오.
```

C-2. 자동화 설정(원문)

```
version = 1
id = "[자동화 ID]"
kind = "heartbeat"
name = "[자동화 이름]"
status = "ACTIVE"
rrule = "RRULE:FREQ=MINUTELY;INTERVAL=5;UNTIL=20260828T003247Z"
target_thread_id = "[대화 ID]"
```

- 실행 주기: 5분
- 시작: 자동화 상태를 일시 정지(PAUSED)에서 활성(ACTIVE)으로 변경
- 종료 예정: 2026-08-28 00:32:47 UTC(한국 시각 09:32:47)
- 정리: 측정이 끝난 뒤 자동화를 삭제
- 새 결과가 없으면 지시서를 다시 내지 않고 다음 주기를 기다림

D. Claude 작업자 — 작업 지시 폴더 감시

D-1. 감시 명령(8월 28일 실행 원문, 작업자 1명분)

```
D="[작업 루트]/20260828/[작업자]/01_tasks"; mkdir -p "$D"; seen=$(ls -1 "$D" 2>/dev/null | sort); while true; do cur=$(ls -1 "$D" 2>/dev/null | sort); comm -13 <(echo "$seen") <(echo "$cur") | while read -r f; do [ -n "$f" ] && echo "[새 작업 지시서] $f"; done; seen="$cur"; sleep 5; done
```

다른 작업자는 담당 폴더 이름만 다릅니다.

D-2. Claude Desktop Monitor 등록

```
{
  "description": "[작업자] 작업 지시서 감시 ([작업자 폴더]/01_tasks, 20260828)",
  "persistent": true
}
```

- 위 명령을 Monitor 도구에 상주(persistent)로 등록합니다. 새 파일이 생길 때만 알림이 오고, 그때만 세션이 작업합니다.
- 알림을 받은 작업자는 작업 지시서를 읽고 수행한 뒤 결과 보고서를 자기 02_results에 저장합니다.
- 종료는 Monitor 작업을 정지하거나 해당 세션을 닫습니다.

E. 통과·보완 판정 기준

1. 제출 결과만 믿지 않고 설계 정본, 실제 변경 파일, 실제 시험 결과를 대조합니다.

2. 요구사항·금지 범위·시험·증거가 모두 충족되면 통과(PASS)로 판정합니다.
3. 누락, 정본 불일치, 경계값·보안 결함, 시험 실패, 증거 부족이 있으면 보완(REVISE)으로 판정합니다.
4. 통과이면 같은 작업자에게 다음 작업 지시서를 발행합니다.
5. 보완이면 원인과 완료 조건을 적은 보완 지시서를 발행합니다.
6. 발견 사항이 있는데 관성적으로 통과시키지 않습니다.
7. 이미 처리한 결과는 다시 처리하지 않습니다.
8. 선택이 필요한 사항은 임의로 확정하지 않고 임의 결정 목록에 남깁니다.
9. 푸시·배포·운영 DB 변경·운영 외부 송신은 하지 않습니다.
10. 시작·완료·통과·보완·소요 시간·병목을 진행 현황 원장에 누적합니다.

F. 알려진 함정과 대비(실제로 겪은 일)

함정	실제로 생긴 일	대비
날짜 폴더가 명령에 고정됨	작업자 다섯이 전날 폴더를 감시하고 있어 지시서를 받지 못함(약 25분 손실)	작업 날짜가 바뀌면 새 날짜 폴더로 감시를 다시 등록
새 파일 이름만 감지함	기존 지시서를 고쳐 쓴 경우 작업자가 알아채지 못함	보완 지시는 항상 새 번호의 새 파일로 발행하고 기존 파일은 고치지 않음
검증 쪽이 멈춤	새벽 2시~6시 반, 결과 6건이 쌓였는데 다음 지시서가 나오지 않아 작업자 전원 대기(약 4시간 22분)	결과가 쌓였는데 다음 지시서가 일정 시간 나오지 않으면 사람이 확인. 감시 루프는 새 파일이 없으면 조용히 기다리기만 하므로 별도 점검이 필요

G. 문서 양식

G-1. 작업 지시서(실제 구조)

```
# [번호] · [담당자] – [작업명]

상태: READY
난이도:
방식:

## 목적
## 먼저 읽기
## 할 일
## 결과 표
## 금지
## 결과

판정은 PASS/REVISE 중 하나다. 발견사항이 있으면 관성으로 PASS하지 않는다.
```

G-2. 결과 보고서(8월 28일 수행본의 구조)

```
# [번호] [담당] – [작업 제목] · 결과

- 담당: [담당자]
- 판정: [완료 또는 진행 차단]
- 시험: [시험 명령] → [건수와 결과]

## 변경 파일
| 파일 | 내용 |
|---|---|
| [신규 또는 수정 파일] | [변경 내용] |
- 금지 파일 무변경 확인
- 기존 미반영 변경 미수정, 패키지 추가 없음, 커밋과 배포 없음

## 구현 요약
## 시험명과 결과
## 임의 결정(이유 · 영향 · 되돌리는 법)
## 남은 연결(다음 단계, 이번 작업 범위 밖)
```

실제 문서 한 세트(지시서 → 결과 → 보완 지시 → 보완 결과)는 첨부 5 "운영 기록 증빙"에 원문으로 실었습니다.