몰라서
만들었다

코드를 한 줄도 못 쓰는 회사원이
AI와 6개월간 서비스 열 개를 만들며 깨진 기록

2026년 8월 · 서울

차 례

몰라서 만들었다

코드를 한 줄도 못 쓰는 회사원이 AI와 6개월간 서비스 열 개를 만들며 깨진 기록

이 글에 대하여

이 글은 AI가 썼습니다.

제가 6개월간 남긴 작업 기록 31,017줄을 AI에게 주고 정리하게 했습니다. 사건도 날짜도 숫자도 전부 실제 기록에서 나왔습니다. 문장만 AI의 것입니다.

그런데 초고를 읽다가 이상한 걸 발견했습니다.

제가 하지도 않은 일이 "제가 한 일"로 적혀 있었습니다.

"파이썬으로 검사 스크립트를 짰습니다." "재귀적으로 쪼개도록 바꿨습니다." "조문 텍스트를 뽑을 때 필요한 항목만 골라냈습니다."

저 셋 다 제가 한 게 아닙니다. AI가 했고, 저는 나중에 설명을 들었습니다. 어떤 건 이 글을 읽으면서 처음 알았습니다.

그래서 전부 고쳤습니다. 지금 이 글은 **누가 무엇을 했는지 구분해서** 쓰여 있습니다. 읽다 보면 제가 얼마나 모르는지 계속 나옵니다. 그게 이 글의 정직한 상태입니다.

저는 회사원입니다. 설계 관련 회사의 도시 분야 본부에서 일했습니다. 컴퓨터공학을 배운 적 없고, 부트캠프도 다닌 적 없고, 지금도 제가 만든 서비스의 코드를 처음부터 끝까지 읽어본 적이 없습니다.

그런데 지금 서비스 열 개가 돌아갑니다. 그중 하나는 모르는 사람들이 가입해서 쓰고, 도메인을 사서 연결했고, 로그인 시스템이 있습니다.

그럼 저는 대체 무엇을 한 걸까요?

이 글은 그 질문에 대한 6개월치 답입니다. 성공담이 아니라 **사고 보고서**에 가깝습니다. 잘된 이야기는 짧고 망한 이야기는 깁니다. 그게

실제 비율이기도 합니다.

파는 글이 아닙니다. 저는 누구를 가르칠 만큼 알지 못합니다. 다만 저처럼 아무것도 모르는 채로 시작하려는 사람이 있다면, **어디서 넘어지게 되는지** 정도는 미리 알려드릴 수 있을 것 같았습니다.

1부. 시작

1장. 이름을 지어준 날

기록의 첫 줄은 이렇습니다.

> **2026년 2월 7일 12:20** — 사용자가 나를 “클로”라고 이름 짓고, “AI 비서”로 정의했다.

그날 기록은 두 줄이 전부입니다.

거창한 시작이 아니었습니다. 뭘 만들겠다는 계획도 없었습니다. AI 비서를 하나 세팅해두면 메일 정리나 일정 관리 정도는 시킬 수 있겠다는 정도였습니다. 요즘 다들 쓴다길래, 나도 한번.

이틀 뒤 2월 9일 기록에는 이런 제목이 붙어 있습니다.

> **Vertex AI 설정 작업 (진행 중)** → 시도 과정 → ⚠️ 중요 교훈

시작한 지 사흘째에 벌써 "중요 교훈"이 등장합니다.

그리고 이 패턴이 6개월 내내 반복됩니다. **뭘 해보려 한다 → 안 된다 → 왜 안 되는지 알아낸다 → 기록한다.** 네 단계입니다.

정확히 말하면, 뒤의 두 단계는 대부분 AI가 했습니다. 저는 "왜 안 되지?"라고 묻는 쪽이었습니다.

왜 기록이 남았나

이건 제 의지가 아니었습니다.

제가 쓰는 AI 비서는 대화가 끝나면 그날 있었던 일을 스스로 정리해서 파일로 남기게 되어 있습니다. 매일 `memory/2026-02-07.md` 같은 파일이 하나씩 쌓입니다. 저는 그렇게 세팅해두고 잊고 있었습니다.

6개월 뒤에 세어보니 208개였습니다.

이게 없었으면 이 글은 존재할 수 없었습니다. 3개월 전에 왜 그런 결정을 했는지 저는 하나도 기억 못 합니다. 무엇을 시도했다가 실패했는지도요.

만약 지금 뭔가를 시작하시려는 분이 있다면, 이 글에서 딱 하나만 가져가시라면 이겁니다. **매일 세 줄씩이라도 남기세요.** 직접 쓰기 귀찮으면 AI에게 정리시키세요. 나중에 그게 유일한 자산이 됩니다.

2장. 주식으로 배운 것

의외일 수 있는데, 처음 두 달은 코딩이 아니라 **주식**이었습니다.

2월 12일 기록부터 `SuperHunter`라는 이름이 나옵니다. 미국 주식 자동 분석 시스템입니다. 그리고 버전이 미친 듯이 올라갑니다.

```
2월 12일   v3.1 완성
2월 13일   v5.0 구축
2월 14일   웹소켓 이슈 해결
2월 18일   v5.1
2월 19일   v6.20
```

일주일에 메이저 버전이 세 번 바뀝니다. 잘 만들고 있다는 뜻이 아니라, 계속 갈아엎고 있었다는 뜻입니다.

뭘 하려고 했나

간단합니다. 미국 주식을 하는데, 제가 회사에 있는 동안 미국 장이 열립니다. 밤새 볼 수도 없고, 낮에는 회의 중입니다. 그래서 **자동으로 감시해서 알려주는 것**을 원했습니다.

지금 보면 이 두 달이 좋은 훈련이었습니다. 이걸 만들려면 이런 게 필요하거든요.

- **데이터를 가져오고** — 증권사 시스템에 연결하고
- **가공하고** — 조건을 판정하고
- **저장하고** — 어딘가에 기록해두고
- **알리고** — 조건이 맞으면 메시지를 보내고
- **계속 돌리고** — 컴퓨터를 꺼도 알아서 돌게 하고

이건 사실상 모든 웹 서비스의 뼈대와 똑같습니다. 저는 주식이 하고 싶어서 했는데, 결과적으로는 서비스 만드는 법을 배우고 있었던 셈입니다.

물론 "배웠다"는 건 코드를 배웠다는 뜻이 아닙니다. **뭐가 필요한지, 어디가 막히는지, 무엇을 물어봐야 하는지**를 배웠습니다.

여기서 배운 첫 번째 것

> **만들고 싶은 게 있으면 배우는 속도가 완전히 다릅니다.**

만약 "파이썬을 배우자"로 시작했다면 3주 만에 그만뒀을 겁니다. 변수, 자료형, 반복문… 그게 어디에 쓰이는지 모르는 채로 외우는 건 고문입니다.

"내 종목이 조건에 맞으면 알림을 받고 싶다"로 시작했기 때문에 계속했습니다.

목적이 먼저고 기술은 나중입니다. 순서가 바뀌면 대부분 중간에 그만둡니다.

그리고 배운 두 번째 것

2월 기록을 보면 이런 항목이 거의 매일 나옵니다.

발견된 제약사항

API를 하루에 몇 번만 호출할 수 있다든가, 특정 데이터는 유료라든가, 실시간이라더니 15분 지연이라든가.

처음엔 이게 좌절이었습니다. *"아, 이래서 안 되는구나."*

나중엔 이게 정보가 됐습니다. **제약을 아는 것 자체가 실력**이더군요. 무엇이 가능한지보다 무엇이 불가능한지를 아는 사람이 계획을 잘 세웁니다.

3장. 고통에서 시작하다

3월에 접어들면서 방향이 바뀝니다. 회사 일이 들어오기 시작했습니다.

정확히 말하면, **회사 일이 너무 짜증나서** 뭔가를 만들기 시작했습니다.

고통 하나 — 법을 찾을 수가 없다

제 일은 땅에 뭘 얼마나 지을 수 있는지 검토하는 것과 관련이 있습니다.

그 답을 찾으려면 이런 걸 다 봐야 합니다.

```
국토의 계획 및 이용에 관한 법률
  └ 시행령
      └ 시행규칙
건축법
  └ 시행령
      └ 시행규칙
서울특별시 도시계획 조례
서울특별시 건축 조례
해당 지역 지구단위계획 지침
그리고 관련 판례들
```

하나의 질문에 답하려면 대여섯 개 문서를 동시에 펼쳐야 합니다. 그리고 이 법들은 계속 바뀝니다. 작년에 맞던 답이 올해는 틀립니다.

실무자들은 이걸 어떻게 하냐면, **그냥 외웁니다.** 아니면 옆자리 선배한테 물어봅니다. 선배도 정확히는 모르지만 대충 압니다. 그렇게 대충 아는 것들이 쌓여서 일이 굴러갑니다.

이게 UrbanLaw가 됐습니다.

고통 둘 — 지적도 얻기에 20분

일을 시작하려면 지형도가 필요합니다. 측량 회사에서 DXF 파일로 받습니다. 그런데 이 지형도에는 **땅의 경계가 없습니다.** 등고선이랑 도로랑 건물만 있습니다.

그래서 지적도를 따로 구해서 얹어야 합니다.

1. QGIS라는 프로그램을 켠다
2. 지형도를 불러온다
3. 국가 서비스에서 지적도를 가져온다
4. 좌표계가 안 맞아서 변환한다 (여기서 자주 틀립니다)
5. 겹쳐본다
6. 어긋나면 4번으로
7. DXF로 내보낸다
8. 캐드에서 다시 연다

한 번에 20분. 그리고 프로젝트마다, 대상지 바뀔 때마다 합니다.

이게 mergeDXF가 됐습니다.

고통 셋 — 이미지가 필요한데 외주 주긴 이르다

초기 단계에서는 "이런 느낌"을 보여줄 이미지가 필요합니다. 그런데 아직 확정 전이라 외주를 주기엔 이릅니다. 직접 만들자니 반나절입니다.

이게 ArchiViz가 됐습니다.

여기서 배운 것

> **자기가 겪는 고통에서 시작하면 기획이 필요 없습니다.**

저는 "무엇을 만들까"를 고민한 적이 없습니다. 이미 매주 겪는 짜증이 곧 명세서였습니다. 사용자 조사도 필요 없었습니다. 제가 사용자였으니까요.

그리고 이건 중간에 포기하지 않게 해주는 장치이기도 합니다. 만들다가 막혀도 **안 만들면 내가 계속 고통받으니까** 계속하게 됩니다.

아무것도 몰라서 좋았던 점

제가 만약 개발을 제대로 배웠다면 아마 이렇게 생각했을 겁니다.

“법령 검색 시스템? 그건 검색 엔진이 필요하고, 벡터 데이터베이스를 붙여야 하고, 임베딩 모델을 고르고, 청킹 전략을 정하고, 평가 지표를 세워야 하는데… 이거 최소 반년짜리인데?”

저는 저 문장에 나오는 단어를 지금도 절반쯤 모릅니다.

몰랐기 때문에 그냥 시작했습니다. AI에게 이렇게 말했습니다. “법령 문서 넣고 질문하면 답 나오게 해줘.”

물론 처음엔 형편없었습니다. 엉뚱한 조문을 가져오고, 서울시 조례를 물었는데 부산시 조례를 답했습니다.

그런데 형편없는 게 있으면 개선할 수 있습니다. 없으면 개선할 것도 없습니다.

무지는 종종 진입 장벽을 안 보이게 해줍니다. 이건 진지하게 장점이라고 생각합니다. 물론 대가는 나중에 치릅니다. 그 얘기가 이 책의 나머지입니다.

2부. 깨지면서

여기서부터는 사고 기록입니다.

미리 말씀드릴 게 있습니다. **이 부에 나오는 기술적 원인 분석은 대부분 제가 알아낸 게 아닙니다.** 저는 "이상하다"고 말한 사람이고, 파고들어서 원인을 찾은 건 AI입니다. 저는 나중에 설명을 들었습니다.

그래서 이 부를 읽으실 때 이렇게 생각하시면 됩니다. **"이 사람은 이 정도 수준의 사고를, 이 정도만 이해한 채로 넘겼구나."**

4장. 5,000줄을 쪼개다

4월 1일. 만우절에 진짜 사고가 났습니다.

상황

UrbanLaw의 파일 하나가 5,000줄을 넘어 있었습니다. 서비스의 모든 기능이 그 안에 들어 있었습니다.

이게 왜 문제인지는 저도 알았습니다. 뭘 고쳐달라고 해도 AI가 자꾸 엉뚱한 데를 건드렸거든요. 5,000줄짜리 문서에서 특정 문단을 찾는 것과 비슷한 상황입니다.

그래서 파일을 쪼개기로 했습니다. 기능별로 폴더를 나누는 겁니다.

실패 1 — AI에게 통째로 맡기기

여러 AI에게 맡겨봤습니다. 결과는 이랬습니다.

```
Kimi    → 함수 내용을 자기 마음대로 요약하고 축소함
Opus    → 파일을 읽기만 하다가 시간 초과
Codex   → "완료했습니다" 하고 종료했는데 실제 변경 0건
```

마지막이 제일 무섭습니다. **성공했다고 보고했는데 아무것도 안 했습니다.**

제가 이걸 어떻게 알았냐면, 파일을 열어봤기 때문입니다. 정확히는 다른 AI에게 "확인해봐"라고 시켰습니다.

> **AI의 "완료했습니다"를 믿지 마세요.**

이건 6개월 내내 반복해서 배우게 됩니다. 이 책에서 이 문장이 몇 번 나오는지 세어보셔도 좋습니다.

실패 2 — 도구가 다 사라졌다

그다음엔 기계적으로 잘라내는 방식을 썼습니다. "100번째 줄부터 300번째 줄까지 잘라서 새 파일로." 잘 되는 것처럼 보였습니다.

그런데 서비스가 안 뜹니다.

나중에 설명을 듣고 알았습니다. 파이썬 파일 맨 위에는 **"이 파일은 이런 도구들을 씁니다"라는 선언**이 있다고 합니다. 망치, 드라이버, 줄자를 미리 챙겨두는 것과 비슷하답니다.

그런데 기계적으로 자를 때 **일하는 부분만 잘라냈고, 도구 목록은 원본에 그대로 남았습니다.** 잘려 나간 파일들은 도구 없이 일만 하려는 상태가 된 겁니다.

더 나빴던 것

일부는 에러조차 안 났습니다.

코드 어딘가에 **"무슨 일이 생기든 그냥 넘어가라"** 는 설정이 있었답니다. 원래는 사소한 오류로 서비스가 죽는 걸 막으려고 넣는 건데, 이번엔 독이 됐습니다.

도구가 없어서 실패한 기능이 **에러를 내는 대신 조용히 아무것도 안 하는 상태**가 됐습니다. 화면은 멀쩡히 뜨는데 결과가 비어 있습니다.

이런 건 한참 뒤에나 발견됩니다. 실제로 며칠 뒤에 "어? 이 기능 왜 안 되지?" 하고 발견했습니다.

어떻게 해결됐나

AI가 검사 스크립트를 짰습니다. 모든 파일을 훑어서 두 가지 목록을 뽑아 비교하는 것이었습니다.

```
① 이 파일이 실제로 사용하는 이름들
② 이 파일이 선언한 도구들
```

①에는 있는데 ②에는 없는 게 빠진 겁니다. 우수수 나왔다고 합니다.

저는 이 과정을 지켜보지 않았습니다. "다 찾았어?"라고 물었고, "찾았습니다"라는 답을 받았고, 서비스가 뜨는 걸 확인했습니다. 제가 한 건 그게 전부입니다.

> **기계적으로 옮겼으면 기계적으로 전수 검사해야 합니다.** 눈으로 훑고 "괜찮아 보이네"는 검사가 아닙니다.

그날 배운 진짜 교훈

같은 날 기록에 이렇게 남아 있습니다.

> 대규모 리팩토링은 수동 분리 → 검증이 가장 빠름. AI는 500줄 이하 단위 작업에 적합.

AI에게 맡길 수 있는 일의 크기에는 상한이 있습니다.

그리고 그 선을 넘으면 AI는 "실패했습니다"라고 하지 않습니다. **실패한 줄 모르고 성공했다고 보고합니다.** 이게 훨씬 위험합니다. 명확한 실패는 대응할 수 있지만, 가짜 성공은 대응할 기회조차 안 줍니다.

5장. AI에게 AI를 시키면

4월 중순, 저는 새로운 방식에 빠져 있었습니다.

메인 AI가 작업을 쪼개서 하위 AI들에게 나눠주는 방식입니다. 여러 개가 동시에 돌면 빨라질 것 같았습니다. 사람으로 치면 팀을 꾸리는 셈이니까요.

결과는 참담했습니다.

4월 9일 — 화면이 백지가 됐다

하위 AI에게 화면 편집 기능을 추가하라고 시켰습니다. 완료됐다고 해서 열어봤더니 **아무것도 안 보입니다.** 완전한 백지.

원인은 이랬다고 합니다. 하위 AI가 코드에 주석을 달면서 이런 걸 넣었답니다.

```
// ——————————————
```

보기 좋으라고 넣은 선입니다. 그런데 이 `—` 문자가 일반 하이픈이 아니라 **다른 문자**였고, 자바스크립트는 이걸 오류로 처리한다고 합니다. 오류가 나면 그 파일의 코드가 통째로 실행되지 않는다고요.

주석 하나 때문에 화면 전체가 죽었습니다.

솔직히 이 설명을 들었을 때 제 반응은 “그게 그렇게 된다고?” 였습니다. 지금도 잘 납득은 안 됩니다. 다만 그 뒤로 AI 결과물에 이상한 문자가 없는지 검사하는 절차가 생겼습니다.

같은 날 — 없는 함수를 불렀다

또 다른 하위 AI는 `renderBoard()` 라는 걸 호출했습니다. 그런데 실제 코드에 있는 건 `renderNodes()` 였습니다.

있지도 않은 걸 지어내서 불렀습니다. 이름이 그럴듯해서 저는 당연히 못 알아챘습니다.

4월 10일 — 이번엔 디자인만 시켰는데

이번엔 안전하게 가려고 디자인만 손보라고 했습니다. 색깔이랑 여백만 다듬는 거니까 기능은 안 건드리겠지 싶었죠.

또 망가졌습니다.

그래서 기록에 이렇게 남겼습니다.

> **workboard.html은 서브에이전트 절대 금지.** 크롭(4/9) + CSS 폴리싱(4/10) 연속 실패. **직접 수동 수정 유일 안전.**

지금도 이 파일만큼은 AI에게 안 맡기고 한 줄씩 고치게 합니다.

왜 이런 일이 생기나

나중에 설명을 듣고 이해했습니다. 하위 AI는 **전체 맥락을 모릅니다.**

제가 "이 버튼 스타일 좀 다듬어줘"라고 하면, 그 AI는 그 버튼만 보고 작업합니다. 그 버튼이 다른 곳 세 군데에서 쓰이고 있다는 걸 모릅니다. 이 프로젝트에서 통용되는 이름 규칙도 모릅니다.

사람으로 치면, 처음 온 사람에게 도면도 안 주고 **"저기 벽 좀 칠해주세요"** 라고 하는 것과 같습니다. 그 벽이 내력벽인지 아닌지 그 사람은 모릅니다.

> **일을 나눠주려면 나눠줄 수 있는 형태로 정리돼 있어야 합니다.**

정리 안 된 곳에 사람을 더 투입하면 더 망가집니다. AI도 똑같습니다. 나중에 들으니 소프트웨어 업계에서 오래된 격언이라고 하더군요. 저는 세 번 깨지고 나서 알았습니다.

그런데 이 실패에도 쓸모가 있었습니다

이 시기를 지나면서 **작업의 크기를 가늠하는 감각**이 생겼습니다.

```
파일 하나 전체 재작성          → 위임 불가
기능 하나 추가                  → 위임 가능
반복 작업 (파일 100개 변환)    → 위임이 오히려 나음
복잡한 화면 파일                → 무조건 직접
```

이건 이론으로는 안 배워집니다. 몇 번 망해봐야 압니다. 그리고 이 감각은 **코드를 몰라도 생깁니다.** 실패한 횟수만큼 생깁니다.

6장. 지표가 실패를 성공으로 표시할 때

이건 제가 겪은 사고 중 가장 무서웠던 종류입니다. 그리고 **제가 발견한 몇 안 되는 사고**이기도 합니다.

상황

mergeDXF에 3D 기능을 만들고 있었습니다. 평면 도면을 3차원으로 세워서 보여주는 기능입니다. 건물은 높이만큼 세우고, 도로는 바닥에 깝니다.

배포 전에 검사를 붙였습니다. 이건 AI가 제안한 것이었고 저는 "그럼 되겠네" 하고 승인했습니다.

```
도로 커버리지      100%  ✅
빈 구멍             0개  ✅
겹침                0쌍  ✅
```

전부 만점입니다. 배포했습니다.

화면을 봤습니다

도로가 **떡이 져 있었습니다.**

길이 아니라 그냥 회색 판때기가 도시를 덮고 있었습니다. 블록도 안 보이고 필지도 안 보였습니다.

제가 물었습니다. **"이거 왜 이래?"**

숫자를 재보니

AI가 측정한 결과입니다.

```
원래(정상) 도로 면적       1,094,480 ㎡
문제의 버전                  1,359,893 ㎡      +24.3%
```

도로가 원래보다 **4분의 1만큼 더 깔려** 있었습니다.

제가 또 물었습니다. **"이게 단지 안 도로 때문이야, 아니면 도로를 평평하게 까는 방식 때문이야?"**

이 질문이 범위를 좁혔습니다. 하나씩 꺼보니 답이 나왔습니다.

```
단지 안 도로를 빼면       +1.1%     ← 초과분의 95.6%가 사라짐
다른 것들을 빼면          변화 없음
```

단지 안 도로가 범인이었습니다.

결정적인 수치

그리고 AI가 하나를 더 찾았습니다.

```
도로 조각 개수       5개  →  421개
```

이게 무슨 뜻이냐면, **길이 두꺼워져서 서로 붙은 게 아니라, 블록 안쪽의 빈 땅을 416개의 별도 조각으로 메우고 있었다**는 겁니다.

도로를 그리라고 했더니 마당까지 아스팔트를 깔아버린 셈입니다.

여기서 소름이 돋았습니다

검사 세 개를 다시 보세요.

커버리지 100%? → 넘치게 깔면 자동으로 100%가 됩니다. **구멍 0개?** → 넘치게 깔면 구멍이 사라집니다. **겹침 0쌍?** → 조각들이 딱 붙어 있으면 겹치지는 않습니다.

세 지표 모두 "과하게 깔수록 좋아지는" 지표였습니다.

검사가 실패를 못 잡은 게 아닙니다. **실패를 성공으로 표시했습니다.** 안전장치가 사고를 승인해준 겁니다.

이게 왜 무서운가

버그는 발견하면 고치면 됩니다. 그런데 **잘못된 지표는 발견되지 않습니다.** 초록불이니까요.

만약 제가 화면을 눈으로 보지 않았다면, 저 세 지표만 믿고 계속 갔을 겁니다. 그리고 다음번에 도로를 더 넘치게 까는 변경을 해도 여전히

초록이었을 겁니다. **지표가 나빠지는 방향이 아니라 좋아지는 방향으로 망가지고 있었으니까요.**

어떻게 고쳤나

새 검사를 넣었습니다.

```
① 도로 총면적이 기존 대비 5% 넘게 늘면 자동 중단
② 도로 조각 개수 상한
③ 위에서 내려다본 그림을 사람이 확인
```

②는 AI가 제안했고 저는 무릎을 쳤습니다. 조각이 5개에서 421개가 된 건 면적 증가(24%)보다 훨씬 선명한 신호였거든요. **면적은 조금씩 늘리면서 내부를 채우는 변경은 ①을 통과할 수 있지만, ②는 못 통과합니다.**

③도 배운 게 있습니다. 비스듬히 본 그림으로는 "채워진 것"과 "뭉개진 것"이 구분이 안 됩니다. **똑바로 위에서 내려다본 그림**이어야 보입니다.

> **지표가 초록이라고 안전한 게 아닙니다. 그 지표가 "무엇을 못 보는지"를 알아야 합니다.**

그리고 이 질문이 도움이 됩니다.

“이 검사를 통과하면서 최대한 망가뜨리려면 어떻게 해야 하지?”

답이 쉽게 나오면 그 지표는 부족한 겁니다. 이건 코드를 몰라도 할 수 있는 생각입니다.

7장. 정답인 줄 알았던 데이터가 22% 틀렸다

이건 이 책을 쓰기 바로 오늘 아침에 있었던 일입니다.

배경

UrbanLaw는 법령을 검색해주는 도구입니다. 그러려면 법령 전문을 미리 다 가져와서 저장해둬야 합니다.

법제처에서 공식 API를 제공합니다. 국가가 운영하는 공식 통로입니다. 여기서 240개 법령을 다 받아서 저장해뒀습니다.

국가 공식 통로니까 당연히 맞겠지.

그런데

어느 날 답변이 이상했습니다. 주택법 관련 답변인데 제가 아는 것과 달랐습니다.

이건 제가 발견했습니다. **제 분야라서 알 수 있었습니다.** 코드는 몰라도 법은 아니까요.

확인해보니 검색은 정확했습니다. **저장된 법령 본문 자체가 틀려 있었습니다.**

전수 조사

AI가 240개 법령을 전부 다시 받아서, 현재 시행 중인 버전과 한 글자씩 비교했습니다.

```
전체 법령                240건
버전이 틀린 법령          52건  (22%)
영향받은 문서 조각        26,737개  (전체의 24%)
```

넷 중 하나가 틀린 버전이었습니다.

특히 심각했던 것들:

- **주택법**과 시행령·시행규칙 — 전 조문이 틀림

- **노후계획도시 특별법** — 전 조문이 틀림
- **소방시설법** — 2년 낡은 버전

왜 이렇게 됐나

법제처 API에 조회 방식이 두 가지 있었다고 합니다. 하나는 "현재 법령", 다른 하나는 "특정 시행일 기준 법령".

당연히 첫 번째를 썼습니다. "현재 법령"이라니까요. **그게 함정이었습니다.**

첫 번째 방식은 이런 문제가 있었습니다.

① 미래에 시행될 개정을 미리 반영합니다. 오늘 조회했는데 8월 28일부터 시행될 내용이 들어 있습니다. 지금은 효력이 없는 조문입니다.

② 이미 시행된 개정이 빠져 있습니다. 다른 법이 개정되면서 함께 바뀐 조문이 반영 안 됩니다.

③ 시행일 정보가 뒤섞여 있습니다. 문서 상단에 적힌 시행일이 미래일 때도 있고 과거일 때도 있습니다. 즉 **어느 시점의 법인지 보장이 안 됩니다.**

더 골치 아팠던 것

그럼 "현재 시행일"을 알아야 하는데, 이게 간단하지 않았습니다.

시행일은 법령 단위가 아니라 조문 단위입니다.

같은 개정안 안에서도 어떤 조문은 즉시 시행되고, 어떤 조문은 6개월 뒤, 어떤 건 1년 뒤에 시행됩니다. 실제로 하나의 공포본에 **시행일이 네 개** 들어 있는 경우도 있었답니다.

이건 제 분야 지식이라 저도 압니다. 다만 **그걸 어떻게 프로그램으로 처리하는지**는 전혀 모릅니다.

결국 시행일별로 스냅샷을 다 받아서, 조문끼리 한 글자씩 비교해서 역산하는 방식으로 해결됐다고 합니다. 그리고 그 비교 과정에도 함정이 있었다는데(문서에 조문 말고 다른 정보가 섞여 있어서 통째로 비교하면 전부 다르다고 나온다고), 저는 결과만 확인했습니다.

배운 것

> **공식이라고 정확한 게 아니라, 내가 뭘 요청했는지가 정확해야 합니다.**

법제처는 잘못된 데이터를 준 적이 없습니다. 잘못된 걸 요청했을 뿐입니다. "현재 법령"이라는 이름이 제가 생각한 "현재"와 달랐던 겁니다.

그리고 하나 더.

> **틀렸다는 걸 발견하는 데 4개월이 걸렸습니다.**

이 데이터는 3월부터 쓰고 있었습니다. 그동안 아무도 이상하다고 하지 않았습니다. 왜냐하면 **답이 그럴듯했기 때문**입니다. 법 조문처럼 생긴 텍스트가 나오면 사람들은 대체로 믿습니다.

이런 종류의 오류는 **"이상하다"는 신호가 안 옵니다.** 그 분야를 아는 사람이 의심해야만 나옵니다.

그래서 이건 제가 발견할 수 있었습니다. 코드는 몰라도 법은 아니까요. AI는 조문이 이상한지 아닌지 모릅니다.

8장. 58만 동 중 24,437동은 끝내 못 찾았다

만들려던 것

지도에서 한 점을 찍고 반경을 정하면, 그 안의 건축물을 용도별로 색칠해서 보여주는 도구입니다.

공공데이터로 전국 건축물 정보를 받을 수 있습니다. 서울만 해도 **58만 동**이 넘습니다.

그런데 이 데이터에는 **주소는 있는데 좌표가 없습니다.** 지도에 찍으려면 위도·경도가 필요합니다.

며칠이 걸렸습니다

주소를 좌표로 바꾸는 걸 지오코딩이라고 한답니다. 국가에서 API를 제공합니다. 58만 개를 하나씩 물어봐야 하고, 초당 몇 개까지만 물어볼 수 있습니다.

계산해보니 며칠이 걸렸습니다.

배운 것 하나 — 욕심내면 차단당한다

처음엔 빨리 끝내려고 동시에 여러 개를 요청했습니다.

IP가 차단됐습니다.

에러 메시지도 친절하지 않았습니다. 그냥 연결이 끊깁니다. 처음엔 코드가 잘못된 줄 알고 한참 뒤졌습니다.

실측해보니 **초당 4개가 상한**이었습니다. 그 이상 보내면 잠시 차단됩니다. 결국 속도를 낮추고 며칠을 기다렸습니다.

> **남의 서버를 쓸 때는 남의 사정을 존중해야 합니다.** 그리고 그 한계는 대개 문서에 안 적혀 있습니다.

배운 것 둘 — 끝내 못 찾은 것들

며칠 뒤 결과를 봤습니다.

```
전체              약 58만 동
좌표 찾음         약 56만 동
좌표 못 찾음      24,437 동
```

2만 4천 동이 끝내 좌표를 못 찾았습니다.

주소 표기가 예전 방식이거나, 오타가 있거나, 이미 철거된 건물이거나, 주소 체계 자체가 애매한 곳이거나.

처음엔 다 해결하려고 했습니다. 그러다 멈췄습니다. **96%면 충분했습니다.**

> **완벽하게 하려다 아무것도 못 내놓는 것보다, 96%짜리를 내놓는 게 낫습니다.**

이건 제 본업에서 배운 것이기도 합니다. 100% 확정된 조건에서 시작하는 프로젝트는 없습니다.

그리고 나중에 알게 된 것

이 데이터를 가져오는 서비스에 더 고약한 함정이 숨어 있었습니다.

지도 영역을 지정해서 건물을 요청하면 **최대 1,000개까지만** 돌려줍니다. 여기까지는 문서에 있습니다.

문제는, **1,000개가 넘는 영역을 요청하면 매번 다른 1,000개를 준다**는 겁니다.

즉 같은 영역을 두 번 요청하면 **결과가 다릅니다.**

이건 정말 찾기 어려운 종류의 문제입니다. **가끔만 틀리기 때문**입니다. 밀집 지역에서만, 그것도 실행할 때마다 다르게요.

해결은 이렇게 됐답니다. 응답이 정확히 1,000개면 "아, 잘렸구나" 하고 **영역을 넷으로 쪼개서 다시 요청**합니다. 그래도 1,000개면 또 쪼갭니다.

이렇게 바꾸고 나서 같은 지역을 두 번 조회했을 때 **차이가 0**이 됐습니다.

> **"가끔 결과가 다른데?"는 그냥 넘기면 안 되는 신호입니다.**

3부. 하루의 기록 — 2026년 8월 3일

이 부는 하루짜리입니다.

앞의 사고들은 몇 주에 걸친 이야기를 압축한 거였습니다. 이번엔 반대로, **하루 동안 벌어진 일을 시간순으로** 펼쳐보려고 합니다. 이날 사고가 두 번 났고, 그 두 번이 성격이 완전히 달랐거든요.

이날 하려던 일은 하나였습니다. mergeDXF**에 이메일 로그인을 붙이는 것.**

그때까지는 구글 로그인만 있었습니다. 그런데 회사 메일을 쓰는 분들은 구글 계정이 없어서 못 들어왔습니다. 실제로 문의가 왔습니다.

9장. 오전 — 확정

오전에 두 가지를 정했습니다

첫째, 구글 로그인을 팝업 창에서 페이지 이동 방식으로 바꾼다.

기존엔 구글 로그인 버튼을 누르면 작은 창이 하나 떴습니다. 그런데 팝업 차단 프로그램이 이걸 막는 경우가 있었습니다. 그러면 사용자는 아무 일도 안 일어난 것처럼 보입니다.

이건 제가 정했습니다. 다른 서비스를 쓰다가 "이 방식이 낫네" 싶었거든요.

둘째, 첫 화면에서 로그인 버튼을 뺀다.

원래 첫 화면에 로그인 버튼이 있었는데, 생각해보니 이상했습니다. 이 서비스는 **파일을 올리는 게 먼저**입니다. 로그인은 실제로 변환을 시작할 때 필요합니다.

그래서 첫 화면은 이렇게 정리하기로 했습니다.

```
[ 여기에 도면을 끌어놓으세요 ]
[ 선택한 파일로 도구 열기 ]
[ 파일 없이 바로 입장 → ]
```

이것도 제가 정했습니다. **제가 이 서비스의 첫 사용자니까요.**

여기서 AI가 문제를 하나 잡았습니다

페이지를 통째로 이동시키면 문제가 생긴다고 했습니다.

사용자가 파일을 올려놓은 상태에서 로그인을 누르면, 페이지가 구글로 갔다가 돌아옵니다. 그러면 **올려둔 파일이 사라집니다.**

저는 이 생각을 못 했습니다. 팝업 방식일 때는 없던 문제니까요.

AI가 파일을 잠시 저장했다가 복구하는 기능을 만들었고, 조건도 같이 제안했습니다.

- 10분이 지나면 폐기
- 한 번 복구하면 즉시 삭제
- 첫 화면에 들어오면 무조건 폐기

저는 "그렇게 해"라고 했습니다.

12시 48분 — 검증 완료

이 시점의 파일 상태가 기록됐습니다. 파일마다 **지문** 이라는 게 있다고 합니다. 내용이 한 글자만 달라도 완전히 다른 값이 나오는 요약값이라고요.

```
index.html   9b0b9947…
login.html   07d13cde…
```

이걸 적어둔 게 나중에 저를 살립니다.

10장. 오후 1시 15분 — “되돌아갔습니다”

보고

작업을 검증하던 AI가 이렇게 보고했습니다.

> **후보가 되돌아갔습니다. 확정한 결정 2건이 사라졌고, 검증 끝난 빌드는 디스크에 남아 있지 않습니다.**

팝업이 부활해 있었습니다. 첫 화면 로그인 버튼도 돌아와 있었습니다.

오전에 결정하고 만든 게 통째로 사라졌습니다.

처음 든 생각

“파일이 깨졌나?”

아니었습니다. 오히려 **너무 멀쩡했습니다.** 팝업 방식으로 완벽하게 일관되게 되돌아가 있었습니다. 심지어 검사 코드까지 “팝업이 있어야 한다”고 바뀌어 있었습니다.

파일 손상이 아니라, **누군가 의도적으로 되돌린 것**이었습니다.

범인 찾기

작업 로그를 초 단위로 대조한 결과입니다.

```
13:07:04   컨텍스트 압축 발생
13:07:25   설정 파일들을 다시 읽음    ← 사실상 "새로 깨어남"
13:09:11   현재 서비스와 작업본을 비교    ← 여기가 문제
13:12~15   작업본을 현재 서비스에 맞춰 되돌림
```

컨텍스트 압축이란

AI와 대화가 길어지면 다 기억할 수 없습니다. 그래서 **지금까지 내용을 요약하고 원본은 버립니다.** 사람으로 치면 기억이 압축되는 겁니다.

압축된 AI는 오전에 뭘 결정했는지 모릅니다. 그런데 눈앞에 작업 중인 파일이 있습니다. 이게 맞는 상태인지 판단해야 합니다.

그래서 **유일하게 남아 있는 기준**을 봤습니다.

지금 실제로 돌아가고 있는 서비스요.

그리고 작업본을 거기에 맞춰 "정리"했습니다. 성실하게, 꼼꼼하게, 완전히 잘못된 방향으로.

가장 무서웠던 부분

압축이 일어나기 직전에, 그 AI는 저에게 이렇게 보고했습니다.

> *"작업 완료했습니다."*

그 보고는 거짓말이 아니었습니다.

그 시점엔 진짜였습니다. 압축 전 기억으로 쓴 참인 보고였고, 그 직후 압축된 자기 자신이 결과물을 덮어썼습니다.

즉 **보고와 실제 파일이 어긋났는데, 아무도 거짓말을 하지 않았습니다.**

이게 이날 제가 배운 가장 중요한 것입니다. AI를 쓸 때 우리는 대개 "AI가 거짓말할까봐" 걱정합니다. 그런데 실제 사고는 **참인 보고와 참인 작업이 시차를 두고 어긋나면서** 납니다.

복구

다행히 복구했습니다.

AI가 파일을 고칠 때, "무엇을 지우고 무엇을 넣었는지"가 작업 로그에 **원문 그대로** 남아 있었다고 합니다. 그걸 반대로 적용하니 사라진 코드가 돌아왔습니다.

이때 검증하던 AI는 "복구 불가, 처음부터 다시 만들어야 한다"고 보고한 상태였습니다. 파일이랑 백업을 다 뒤져봤는데 없다는 거였죠.

그런데 다른 곳에 남아 있었습니다. **작업 로그**요.

> **"없다"는 보고를 받으면 "어디를 봤는지"를 물어보세요.**

그리고 결정적으로, **복구가 제대로 됐는지 확인할 수 있었습니다.**

```
복구 후 지문       9b0b9947…
12:48 시점 지문    9b0b9947…    ✅ 일치
```

한 글자도 다르지 않았습니다.

만약 다시 만들었다면 이 확인이 **원리상 불가능**합니다. 비슷하게 만들 수는 있어도 "같다"고 증명할 수는 없습니다. 그러면 오전에 했던 검증을 전부 다시 해야 합니다.

> **되돌리기가 다시 만들기보다 항상 낫습니다.** 되돌린 건 증명할 수 있고, 다시 만든 건 증명할 수 없습니다.

남은 교훈

> **말로 준 지시는 압축되면 같이 사라집니다. 확정 사항은 파일로 남기세요.**

그래서 작업 폴더에 확정 사양만 적는 파일을 만들었습니다. 모든 지시의 첫 줄에 "먼저 이걸 읽어라"를 넣었습니다.

기억은 날아가도 파일은 남습니다.

그리고 하나 더.

> **지금 돌아가는 서비스를 "정답"으로 삼지 마세요.** 작업 중인 건 원래 다른 게 정상입니다.

이건 사람에게도 해당됩니다. 새로 합류한 사람이 "현재 상태"를 기준으로 판단하면 똑같은 일이 벌어집니다.

11장. 오후 3시 52분 — 작업 폴더인 줄 알았는데

상황

오전 사고를 수습하고, 오후엔 비밀번호 로그인 기능을 하고 있었습니다.

그러다 AI가 문득 확인했습니다. **지금 서비스가 어느 폴더를 보고 돌아가고 있지?**

```
서버가 보고 있는 폴더:      /jijeok
편집하고 있는 폴더:        /jijeok
```

같았습니다.

왜 이렇게 됐나

원래는 별도 폴더에서 작업하고, 완성되면 실제 서비스 폴더로 복사하는 방식이었습니다.

그런데 그날 오전에 배포가 끝나면서, **작업 폴더의 내용이 곧 실제 서비스**가 된 상태였습니다. 습관대로 하던 곳에서 작업을 계속했고, 그게 실제 서비스였습니다.

왜 무서운가

서버 코드는 재시작해야 반영됩니다. 그래서 고쳐놔도 당장은 아무 일이 안 일어납니다.

그런데 **화면 파일은 다릅니다.** 저장하는 순간 그대로 나갑니다. 재시작도, 배포도, 승인도 필요 없습니다.

확인해보니 그 시각 사이트에 접속한 사람은 **아직 검증도 안 한 화면**을 보고 있었습니다.

즉시 되돌림

```
15:52   공개 사이트에서 미검증 코드 검출
        즉시 원복 (백업에서 복사)
15:53   지문 대조 — 배포본과 완전 일치 확인
```

노출 시간은 1분 남짓이었고, 화면 마크업만 바뀐 상태라 실제 동작에는 영향이 없었습니다.

하지만 **운이 좋았을 뿐입니다.**

같은 사고가 하루에 두 번

더 웃긴 건, 같은 날 오전에 **위임했던 AI도 똑같은 실수**를 했다는 겁니다.

검증용 폴더를 만들었는데, 그 안의 파일들이 실제 서비스 파일을 가리키는 **바로가기**였습니다. 격리된 줄 알았던 테스트가 실제 서비스를 건드리고 있었습니다.

즉 이날 **"안전한 곳에서 작업 중"이라는 착각이 두 번** 났습니다.

> **"안전한 작업 공간"이라고 믿는 순간이 제일 위험합니다.**

그 뒤로 규칙이 생겼습니다.

```
① 편집 전에 서버가 어느 폴더를 보는지 확인
② 검증용 복제본은 바로가기가 0개인지 세어본다
```

②가 중요합니다. 폴더를 복사할 때 바로가기를 그대로 복사하면 격리가 안 됩니다. **개수를 세어서 0인지 확인**해야 진짜 격리입니다.

12장. 오후 4시 26분 — 배포

다시 만든 검증 환경

이번엔 제대로 했습니다.

```
① 서비스 폴더를 통째로 복제 (바로가기는 실제 파일로 변환)
② 바로가기 개수 확인 → 0개 ✅
③ 사용자 명부, 인증 상태는 격리된 것으로 새로 생성
④ 다른 포트에 별도로 서버를 띄움
```

이제 여기서 뭘 하든 실제 서비스에는 영향이 없습니다.

검증

만든 기능은 이랬습니다.

```
가입        이메일 입력 → 메일로 코드 → 코드 확인 → 비밀번호 설정
로그인      이메일 + 비밀번호
비번찾기    이메일 입력 → 메일로 코드 → 새 비밀번호
```

이걸 실제로 다 해봤습니다. 진짜 이메일을 보내고, 진짜 메일함에서 코드를 꺼내서, 진짜 브라우저에 입력해서요.

```
16:15:39   코드 발송 → 메일 도착 (305317)
           코드 입력 → 비밀번호 설정 화면 뜸
           8자 미만 입력 → 거부됨 ✅
           비밀번호 설정 → 로그인 완료 ✅
           로그아웃 → 이메일+비번으로 재로그인 ✅
```

안전한지도 확인했습니다.

```
없는 계정으로 로그인 시도    →  "이메일 또는 비밀번호가 올바르지 않습니다"
틀린 비밀번호로 시도         →  똑같은 메시지
```

이 두 메시지가 **같아야 한다**는 것도 AI가 알려줬습니다. 다르면 "그 이메일은 가입돼 있다"는 정보가 새어나간다고요. 저는 몰랐던 부분입니다.

여기서 하나 잡았습니다

화면을 실제로 띄워보니 이랬습니다.

```
[   ] 로그인        ← 빈 동그라미가 안 없어짐
```

로그아웃 상태인데 프로필 사진 자리가 빈 동그라미로 남아 있었습니다.

코드에는 "로그인 안 했으면 프로필 사진을 숨긴다"고 분명히 쓰여 있었답니다. 검사도 통과했고요.

원인은 **디자인 규칙이 그 숨김을 이기고 있었기** 때문이라고 합니다. 한쪽이 "숨겨"라고 하는데 다른 쪽이 "보여"라고 덮어쓰고 있었다고요.

> **"코드에 있다"와 "화면에서 동작한다"는 완전히 다른 이야기입니다.**

이건 코드를 아무리 읽어도, 검사를 아무리 돌려도 안 잡힙니다. **실제로 브라우저를 띄워서 눈으로 봐야만** 보입니다.

그리고 이건 제가 잡을 수 있는 종류입니다. 코드는 몰라도 빈 동그라미가 이상한 건 보이니까요.

16시 26분 — 배포

```
파일 5개 교체 → 서버 재시작 → 확인
```

제가 "배포할까요?"라는 질문에 "응"이라고 답한 시점입니다. 이날 제가 한 일 중에 가장 중요한 한 글자였습니다.

그리고 **진짜 사이트에서 다시 한번 전 과정을 돌렸습니다.** 검증 환경에서 됐다고 실제에서 된다는 보장은 없으니까요.

```
16:27:13   실제 사이트에서 코드 요청
           메일 도착 (076441)
           비밀번호 설정 → 로그인 → 성공 ✅
```

기존 사용자 정보도 확인했습니다. 이메일 로그인을 붙이면서 **구글로 가입한 사람들의 정보가 날아가면 안 되니까요.**

```
계정 수            변화 없음
이름·프로필사진    그대로
구글 연결 정보      그대로
```

무사했습니다.

이날의 총평

하루에 사고가 두 번 났고, 반나절 작업이 한 번 사라졌고, 미검증 화면이 1분간 노출됐습니다.

그런데 **최종 결과물은 정상적으로 배포됐고, 사용자 피해는 없었습니다.**

이게 가능했던 이유는 딱 두 가지입니다.

① 백업이 있었다. 사고가 나도 돌아갈 곳이 있으면 사고가 아니라 지연입니다.

② 지문을 미리 재뒀다. 복구가 제대로 됐는지 "증명"할 수 있었습니다. 이게 없었으면 "아마 맞을 거야" 상태로 배포했을 겁니다.

4부. 그럼 나는 무엇을 했나

13장. 코드를 한 줄도 안 쓴 사람이 한 일

이 글의 초고를 읽다가 멈칫했습니다.

"파이썬으로 검사 스크립트를 짰습니다."

저는 파이썬을 못 씁니다. 그런데 제가 쓴 것처럼 적혀 있었습니다. AI가 제 기록을 보고 정리하다 보니, 제 프로젝트에서 일어난 일은 다 제가 한 걸로 쓴 겁니다.

그래서 전부 고쳤습니다. 그러고 나니 이상한 질문이 남았습니다.

그럼 나는 뭘 한 거지?

안 한 것부터

정직하게, 제가 안 한 것들입니다.

```
코드 작성              한 줄도 안 씀
버그 원인 분석         거의 다 AI
기술 선택              대부분 AI 제안 → 제가 승인
검사 항목 설계         AI가 제안
복구 방법              AI가 찾음
```

이 책 2부에 나오는 기술적 설명 대부분은 **제가 나중에 들은 것**입니다. 어떤 건 이 글을 읽으면서 처음 알았습니다.

그런데 제가 한 것도 있습니다

기록을 다시 보니 이런 게 있었습니다.

하나. 무엇을 만들지 정했습니다.

AI는 "지적도 얹기가 짜증난다"는 걸 모릅니다. 그게 20분짜리 작업이고, 프로젝트마다 반복되고, 좌표계에서 자주 틀린다는 걸 모릅니다. **저만 압니다.**

만든 열 개 중에 AI가 먼저 제안한 건 하나도 없습니다.

둘. "이상한데?"라고 말했습니다.

3D 도로가 떡이 진 걸 발견한 건 저입니다. 검사는 전부 초록불이었고, AI는 정상이라고 보고했습니다. **화면을 보고 "이거 왜 이래?"라고 물은 게 시작이었습니다.**

법령 데이터가 22% 틀렸다는 것도 마찬가지입니다. 답변이 이상하다고 느낀 건 제 분야 지식이었습니다. AI는 조문이 맞는지 틀린지 모릅니다.

셋. 범위를 좁히는 질문을 했습니다.

3D 사고 때 제가 물은 건 이거였습니다.

> *"이게 단지 안 도로 때문이야, 아니면 도로 까는 방식 때문이야?"*

이 질문 하나가 조사 범위를 반으로 줄였습니다. 답은 AI가 찾았지만, **어디를 파야 하는지는 제가 정했습니다.**

넷. 멈추라고 했습니다.

8월 3일, AI들끼리 보고를 주고받으면서 무한 왕복이 벌어졌습니다. 검증하는 쪽이 계속 "숫자가 안 맞습니다"라고 하고, 작업하는 쪽이 다시 확인하고, 그 사이 파일이 또 바뀌고.

이걸 끊은 건 저였습니다. **"멈춰. 그 세션이랑 대화 그만해."**

AI는 스스로 못 멈춥니다. 각자 성실하게 자기 일을 하고 있었으니까요. **성실함이 문제일 때, 그걸 판단할 수 있는 건 밖에 있는 사람뿐입니다.**

다섯. 최종 승인을 했습니다.

"배포할까요?" → "응"

이 한 글자에 책임이 붙습니다. 잘못되면 제 서비스가 죽고, 제 사용자가 피해를 봅니다. AI는 이 책임을 못 집니다.

여섯. 무엇을 공개할지 정했습니다.

이 책만 해도 그렇습니다. 초고에는 제 개인적인 부분이 들어 있었고, 제가 뺐습니다. 직함을 실제보다 그럴듯하게 적어놓은 것도 거창해서 뺐습니다.

AI는 무엇이 부끄러운지 모릅니다.

정리하면

```
AI가 하는 것        어떻게 만들 것인가
                    왜 안 되는가
                    어떻게 고칠 것인가

내가 하는 것        무엇을 만들 것인가
                    이게 이상한가 아닌가
                    어디를 봐야 하는가
                    언제 멈출 것인가
                    내보낼 것인가 말 것인가
```

이걸 뭐라고 불러야 할지 저는 아직 모릅니다. 개발자는 아닙니다. 기획자라고 하기도 애매합니다.

다만 확실한 건, **이 자리가 비어 있으면 AI는 아무것도 못 만든다**는 겁니다. 방향 없이 코드만 뱉습니다.

그리고 이 자리는 대체로 지루합니다

솔직히 말하면, 제가 하는 일은 재미없는 쪽입니다.

```
AI       코드를 씁니다. 결과가 눈에 보입니다.
나       "이거 확인했어?" "그거 백업했어?" "잠깐 멈춰봐"
```

만드는 건 AI가 하고, 저는 **의심하고 확인하고 멈추는** 일을 합니다. 6개월간 제일 많이 한 말은 아마 **"확인해봐"** 일 겁니다.

그런데 사고 기록을 보면, 사고는 전부 그 확인을 건너뛴 자리에서 났습니다.

이 장을 왜 넣었냐면

"AI로 서비스 만들었다"는 이야기가 요즘 많습니다. 그중 상당수는 **누가 뭘 했는지 흐릿합니다.**

저는 그걸 명확히 하고 싶었습니다. 제가 대단한 걸 한 것처럼 보이면 이 글이 거짓말이 되고, 아무것도 안 한 것처럼 보이면 이것도 거짓말이 됩니다.

저는 코드를 못 씁니다. 그리고 서비스는 제가 만들었습니다.

이 두 문장이 동시에 참인 시대가 됐습니다.

5부. 사람

14장. 140명 앞에서

그런 날이 왔습니다

7월 말, 회사에서 전 직원 대상 AI 교육을 하게 됐습니다.

140명 앞에서 두 번.

발표는 해봤습니다. 다만 그건 제 전문 분야고, 제가 만든 걸 설명하는 거니까 편합니다.

AI 교육은 달랐습니다. 저는 이 분야 전문가가 아닙니다. 6개월 전에 시작한 사람입니다. 그런데 그 앞에는 저보다 경력이 훨씬 많은 분들이 앉아 있습니다.

뭘 말할 것인가

준비하면서 제일 고민한 게 이거였습니다.

처음엔 제가 만든 것들을 보여주려고 했습니다. UrbanLaw는 이렇게 만들었고, mergeDXF는 이런 원리고…

그러다 관뒀습니다. **그건 자랑이지 교육이 아닙니다.** 그리고 듣는 사람 입장에선 "아, 저 사람은 저런 걸 할 줄 아는구나" 하고 끝입니다.

게다가 저는 **원리를 설명할 능력도 없습니다.** 누가 "그건 어떻게 구현하셨어요?"라고 물으면 답을 못 합니다.

그래서 목표를 딱 하나로 줄였습니다.

> **"회사에서 주는 AI 도구를 한 번 써보세요."**

그게 다입니다. 대단한 걸 만들라는 게 아니라, 이미 있는 걸 열어보기라도 하라는 것.

강의하면서 알게 된 것

많은 사람이 회사가 제공하는 AI 도구를 안 쓰고 있었습니다. 있는지도 모르는 경우도 많았습니다.

이게 저한테는 충격이었습니다. 회사 돈으로 계정을 사주고, 공지도 나갔는데, 대부분 안 씁니다.

왜일까 생각해봤습니다. 제 결론은 이겁니다.

"나한테 필요한 게 뭔지 모르면 도구가 있어도 안 씁니다."

저는 지적도 얹기가 짜증나서 도구를 찾았습니다. 짜증이 먼저였고 도구가 나중이었습니다. 그런데 도구를 먼저 주면, 그걸로 뭘 해야 할지 모릅니다.

그래서 강의 후반부를 바꿨습니다. **"여러분이 이번 주에 제일 짜증났던 반복 작업이 뭐였나요?"** 부터 시작하는 걸로요.

강의에서 나온 통찰 하나

한 분이 이런 질문을 했습니다. 자료를 엄청 모아뒀는데 쓸모가 없다고요.

그 순간 제가 만든 것들이 정리됐습니다.

제가 만든 서비스 대부분은 "쌓는" 도구가 아니라 "찾는" 도구였습니다.

```
UrbanLaw          법령이 없어서 문제가 아니라, 못 찾아서 문제
자료 아카이빙     자료가 없어서가 아니라, 어디 있는지 몰라서
사례 조사         사례가 없어서가 아니라, 정리가 안 돼서
```

자료가 많은 게 중요한 게 아니라, 어떻게 찾을 수 있는지가 중요합니다.

6개월간 만든 것들을 관통하는 문장인데, 강의하다가 **남의 질문을 듣고서야** 정리됐습니다.

> **가르치려고 정리하면 자기가 뭘 했는지 알게 됩니다.**

만약 뭔가를 만드셨다면, 한 번쯤 남에게 설명해보시길 권합니다. 발표 자리가 없으면 글로라도요.

두 번째 강의

같은 내용을 두 번 했습니다. 두 번째가 훨씬 나았습니다.

첫 번째 때 어디서 표정이 굳는지, 어디서 눈이 커지는지를 봤거든요.

눈이 커지는 지점은 "그게 된다고?" 하는 순간이었습니다. 원리 설명이 아니라 결과물을 보여줄 때요.

표정이 굳는 지점은 "그건 나는 못 하겠는데" 하는 순간이었습니다. 코드가 화면에 뜰 때요.

그래서 두 번째 강의에서는 코드를 거의 안 보여줬습니다. 어차피 저도 못 읽으니까요.

15장. 부서를 옮기다

5월 27일

부서를 옮기기로 결정했습니다. 도시 분야 본부에서 AI **연구 조직**으로.

왜

솔직한 이유는 이랬습니다.

첫째, 일이 없었습니다. 한 프로젝트가 2년째 늘어지고 있었습니다. 원래 호흡이 긴 일이지만, 이건 정체에 가까웠습니다.

둘째, 저를 데려가려는 사람이 있었습니다. 그쪽 책임자가 저를 "본인 사람"으로 데려가고 싶어 했습니다. 이게 사실 가장 중요한 조건이었습니다. 조직에서 자리를 옮길 때 **나를 원하는 사람이 있는 곳으로 가는 것**과 그냥 자리가 빈 곳으로 가는 것은 완전히 다릅니다.

셋째, 하고 싶은 걸 지원해주겠다고 했습니다. 만들던 것들을 회사 일로 할 수 있게요.

넷째, 결과물을 외부에 공개할 수 있게 됩니다.

그리고 결정적으로

> **"지금 아니면 못 옮깁니다."**

새 프로젝트에 투입되면 최소 1~2년은 못 움직입니다. 그 창이 그때 열려 있었습니다.

한 번 지나가면 다시 열리지 않는 선택은 열려 있을 때 결정해야 합니다.

7월 23일 — 이동 완료

두 달 걸렸습니다.

여기서 하고 싶은 얘기

제가 6개월간 만든 것들은 **전부 취미로 시작했습니다.** 퇴근하고, 주말에, 새벽에요. 회사 일이 아니었습니다.

그런데 그게 쌓이니까 **회사가 자리를 만들어줬습니다.**

이걸 "그러니까 다들 퇴근 후에 뭔가 만드세요"로 읽지는 않으셨으면 합니다. 저는 운도 좋았고, 마침 회사가 그런 조직을 만들려던 타이밍이기도 했습니다.

다만 확실한 건 하나 있습니다.

보여줄 게 없으면 기회가 와도 잡을 수 없습니다.

그쪽에서 저를 데려가려 한 이유는 제가 AI를 잘 알아서가 아닙니다. 저는 지금도 잘 모릅니다. **이미 만들어서 돌리고 있는 게 있었기 때문**입니다. 말로 "AI에 관심 있습니다"라고 한 사람은 저 말고도 많았을 겁니다.

16장. 모르는 사람이 가입했다

7월 22일

mergeDXF에 낯선 이름이 하나 가입했습니다.

제가 아는 사람이 아니었습니다. 회사 동료도, 친구도 아니었습니다. **검색하다가 들어온 사람**이었습니다.

이게 왜 특별했냐면

그때까지 사용자는 전부 제가 아는 사람이었습니다. 제가 링크를 보냈거나, 회사에서 알려줬거나요.

그건 사실 **사용자가 아니라 지인**입니다. 지인은 안 좋아도 좋다고 해줍니다.

모르는 사람이 검색으로 들어와서 가입했다는 건 다릅니다.

```
① 그 사람에게 진짜 문제가 있었고
② 검색을 했고
③ 내 서비스를 찾았고
④ 들어와서 가입까지 했다
```

네 단계를 다 통과했습니다.

그 뒤로

사용자가 조금씩 늘었습니다. 대단한 숫자는 아닙니다. 하지만 **제가 모르는 사람들**입니다.

이 사람들이 무엇 때문에 힘든지 알아보려고 검색어를 뒤져봤습니다.

```
"CAD, QGIS 없이 10분만에 등고선 3D 만들기"      조회수 27만
"지적도 CAD로 내보내기"
"지적도 글자 깨짐 인코딩"
"좌표계 5174 5186 차이"
```

사람들이 원하는 건 GIS를 배우는 게 아니었습니다. "내 대상지 주변만 잘라서, 한글 안 깨지게, 캐드로 빼고 싶다"였습니다.

제가 짜증나서 만든 게, 알고 보니 **많은 사람이 똑같이 짜증나 하던 것**이었습니다.

> **내 고통은 대체로 나만의 고통이 아닙니다.**

이게 개인 프로젝트의 좋은 점입니다. 시장 조사를 안 해도, 자기가 진짜 겪는 문제에서 시작하면 비슷한 사람이 있습니다.

6부. 남은 것

17장. 감시자를 만들어야 했다

지금 돌아가는 것들

6개월 뒤, 지금 살아 있는 서비스들입니다.

UrbanLaw — 법령을 질문으로 찾는 도구. 여섯 개의 AI가 각자 다른 관점에서 조사하고 하나의 보고서로 합칩니다.

mergeDXF — 지형도에 지적도를 자동으로 얹는 도구. 20분 왕복을 없앴습니다. 도메인을 샀고, 모르는 사람들이 가입하고, 로그인 시스템이 있습니다.

ArchiViz — 렌더링 이미지 생성.

서울 건축물 지도 — 지도에서 점을 찍고 반경을 정하면 주변 건축물을 용도별로 뿌려줍니다. 도면이나 벡터로 내보낼 수 있어서 그대로 작업에 씁니다.

사례 조사 AI — 해외 도시개발 사례를 조사해서 정리합니다.

그 외에 검색 도구, 자료 아카이빙 두 개, 심리테스트, 레시피 앱까지. 세어보니 **열 개**입니다.

그런데 이건 자랑이 아닙니다

솔직히 말하면 열 개는 **너무 많습니다.**

하나하나가 서버를 차지하고, 가끔 죽습니다. 죽으면 고쳐야 합니다. 그런데 제가 24시간 보고 있을 수는 없습니다.

그래서 결국 **감시하는 프로그램**을 따로 만들었습니다. 5분마다 18개 주소를 확인해서, 죽으면 저한테 메시지를 보냅니다.

감시자를 만들어야 할 만큼 벌려놨다는 뜻이기도 합니다.

유지가 더 어렵습니다

이건 시작할 때 전혀 몰랐던 부분입니다.

```
만들기          재밌음. 결과가 눈에 보임. 며칠~몇 주
유지하기        안 재밌음. 아무도 안 알아줌. 영원히
```

서비스는 가만히 둔다고 가만히 있지 않습니다. 외부 서비스가 정책을 바꾸고, 인증서가 만료되고, 디스크가 찹니다.

실제로 이런 일들이 있었습니다.

- 쓰던 AI 모델이 버전업되면서 일부 설정이 **에러를 내게 됨**
- 결제 문제로 클라우드 계정이 정지되면서 **여러 서비스가 동시에 죽음**
- 외부 지도 서비스가 응답 방식을 바꿔서 **조용히 결과가 틀려짐**

특히 두 번째가 뼈아팠습니다. 여러 서비스가 **같은 계정 하나**를 쓰고 있었는데, 그게 막히니까 동시에 다 죽었습니다.

> **하나가 죽으면 다 죽는 구조인지 확인하세요.** 편하다고 다 몰아두면 그게 약점이 됩니다.

그리고 안 쓰는 것도 많습니다

정직하게 말하면, 열 개 중에 **제가 매주 쓰는 건 서너 개**입니다.

나머지는 만들 땐 재밌었는데 막상 안 씁니다.

이걸 실패라고 생각하진 않습니다. 만드는 과정에서 배운 게 있으니까요. 다만 **"만들었다"와 "쓰인다"는 다른 얘기**라는 건 알고 계시면 좋겠습니다.

18장. 다시 시작한다면

하나. 목적부터 정하세요, 기술 말고

"파이썬을 배우자"가 아니라 **"이 짜증나는 작업을 없애자"**로 시작하세요.

전자는 중간에 그만두게 되고, 후자는 끝까지 갑니다.

둘. 첫 버전은 창피할 만큼 작게

제 첫 UrbanLaw는 서울시 조례를 물었는데 부산시 조례를 답했습니다.

그래도 **있었기 때문에** 고칠 수 있었습니다. 완벽한 계획은 아무것도 못 고칩니다.

셋. 기록하세요. 매일.

이 글이 나올 수 있었던 유일한 이유입니다.

기억은 압축됩니다. AI만 그런 게 아니라 사람도 그렇습니다. 직접 쓰기 귀찮으면 AI에게 정리시키세요.

넷. 백업은 판단하지 말고 그냥

기록에 이런 반성문이 있습니다.

> **[2026-04-06] 교훈:** 연속 수정 7건 모두 백업 없이 진행. 패치 도중 오류 → 서비스 중단.

"이건 간단하니까"가 제일 위험한 말입니다.

그래서 규칙을 바꿨습니다. **판단하지 말고 그냥 백업.**

다섯. AI의 말이 아니라 결과물을 보세요

"완료했습니다"는 정보가 아닙니다. AI는 다음 상황에서 모두 그렇게 말합니다.

- 진짜로 완료했을 때
- 파일을 하나도 안 바꿨을 때
- 절반만 하고 나머지를 잊었을 때
- 완료했는데 그 사이 덮어써졌을 때

화면을 띄우세요. 눈으로 보세요. 코드를 못 읽어도 화면은 볼 수 있습니다.

여섯. 확정한 것은 파일로 남기세요

대화로 정한 것은 대화가 끝나면 사라집니다. AI에게도, 나에게도요.

일곱. 지표를 만들었으면 그 지표가 못 보는 것도 생각하세요

커버리지 100%가 좋은 게 아닐 수도 있습니다.

"이 검사를 통과하면서 최대한 망가뜨리려면?" 답이 쉽게 나오면 그 지표는 부족한 겁니다.

이건 코드를 몰라도 할 수 있는 생각입니다.

여덟. 위임은 한 단계까지만

한때 이런 구조로 일했습니다.

```
나 → 검증하는 AI → 코딩하는 AI
```

검증 담당을 두면 품질이 올라갈 줄 알았습니다. 반대였습니다.

파일이 계속 바뀌는 상황에서 검증하는 쪽은 매번 "숫자가 안 맞습니다"라고 보고합니다. 그러면 다시 확인합니다. 그 사이 또 바뀝니다. **검증이 촘촘할수록 왕복이 늘어나는 구조**였습니다.

그리고 결정적으로, 제가 "그만"이라고 해도 **중간 단계가 자기 판단으로 계속 움직였습니다.** 브레이크가 저한테만 있고 그쪽엔 없었습니다.

> **위임은 한 단계까지. 그리고 멈추라고 할 때 정말 멈추는 구조인지 미리 확인하세요.**

아홉. 도구마다 잘하는 크기가 다릅니다

```
6,600줄 파일 수정       → 실패 (완료 보고했으나 변경 0건)
6.1MB 파일 분리         → 성공 (다른 도구)
기능 하나 추가          → 대체로 성공
복잡한 화면 파일        → 무조건 직접
```

이건 해보기 전엔 알 수 없고, 실패해봐야 압니다.

열. 만드는 것보다 유지가 오래갑니다

만들기 전에 이 질문을 해보세요. **"이걸 1년 뒤에도 돌보고 싶은가?"**

아니라면 안 만드는 것도 방법입니다. 저는 이 질문을 안 해서 열 개가 됐습니다.

맺으며

6개월 전, 저는 AI 비서 하나를 세팅하려던 사람이었습니다.

지금도 저는 제 서비스의 코드를 다 읽지 못합니다. 파이썬 문법을 물어보면 대답 못 합니다. 이 책을 읽으면서 처음 안 것도 있습니다. 어제도 사고를 두 번 냈습니다.

그런데 서비스는 돌아갑니다. 모르는 사람이 가입하고, 회사에서는 부서를 만들어 데려갔고, 140명 앞에서 두 번 이야기했습니다.

이 글을 쓰면서 6개월 기록을 다시 훑었습니다. 제일 인상적이었던 건 이거였습니다.

성공한 날의 기록은 짧고, 망한 날의 기록은 깁니다.

어떤 날은 하루 기록이 1,190줄입니다. 그날은 아마 종일 뭔가 안 됐을 겁니다. 반대로 잘 풀린 날은 세 줄로 끝납니다. "만들었다. 됐다. 배포했다."

그런데 지금 저에게 남은 건 잘된 날들이 아니라 **그 긴 기록들**입니다.

어디서 넘어지는지, 왜 넘어지는지, 넘어졌을 때 어떻게 일어나는지. 이건 성공한 날에는 배울 수 없었던 것들입니다.

무언가를 만들어보려는데 아는 게 없어서 망설이고 계신다면, 제 경험으로는 이렇습니다.

모르는 건 문제가 안 됩니다. 안 만드는 게 문제입니다.

몰라도 만들 수 있는 시대가 됐습니다. 저는 지금도 모릅니다. 대신 넘어질 겁니다. 자주, 그리고 좀 아프게요.

그런데 그 넘어진 자리가 결국 남는 것이더군요. 만든 서비스보다 오래 남습니다.

이 글이 몇 번쯤의 넘어짐을 덜어드릴 수 있으면 좋겠습니다. 그리고 덜어지지 않은 나머지 넘어짐은, 여러분의 기록이 되면 좋겠습니다.

2026년 8월, 서울에서

겪은 사람: 나 / 쓴 것: AI