TinyExperiment

거대한 문제 앞에서 길을 잃은 프로그래머를 위한 안내서

우리는 거대한 시스템과 싸운다

복잡해서 막막한 코드

처음부터 어디를 봐야 할지 감조차 오지 않는 레거시 코드베이스.

건드리면 고장 나는 시스템

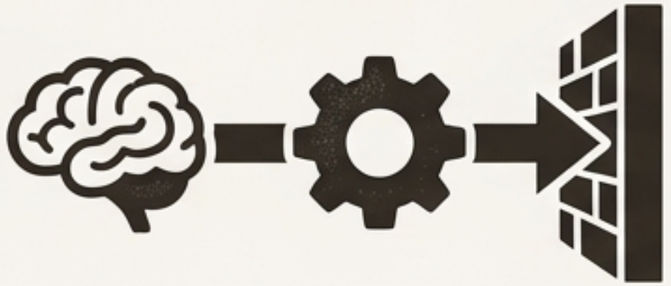
작은 수정 하나가 예상치 못한 연쇄 실패를 일으키는 취약한 아키텍처.

새로운 기능 추가의 두려움

기존 시스템을 망가뜨릴까 봐 새로운 시도를 주저하게 되는 상황.

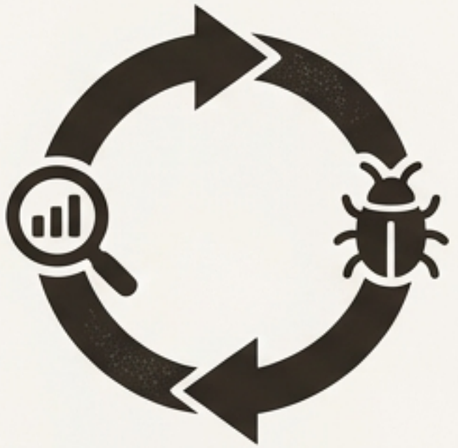
진짜 적은 시스템이 아니라, 우리의 '성급한' 습관이다

"전체를 한 번에" 해결하려는 전통적인 접근 방식



① 기획 → 작업 → 완성:
머릿속으로 전체를 상상하고, 한 번에 완벽하게 구현하려는 시도.

② 실행 → 기도 → 결과:
"아마 이렇게 하면 될 것이다"라는 추측에 기반해 코드를 실행하고, 잘 동작하기를 바라는 마음.



③ 검증 → 디버그 → 재시도:
실제 상황에서 문제가 터진 후에야 원인을 찾고 수정하는 과정을 반복.

이 습관이 우리를 가두는 4가지 감옥

늦은 피드백 (Late Feedback)

전체를 다 만들고 나서야 무엇이 잘못되었는지 알게 되어 엄청난 재작업이 발생.

큰 실패 비용 (High Cost of Failure)

잘못된 접근 방식이 큰 시스템 장애로 이어지고, 처음부터 다시 시작해야 하는 위험.

창의력 고갈 (Blocked Creativity)

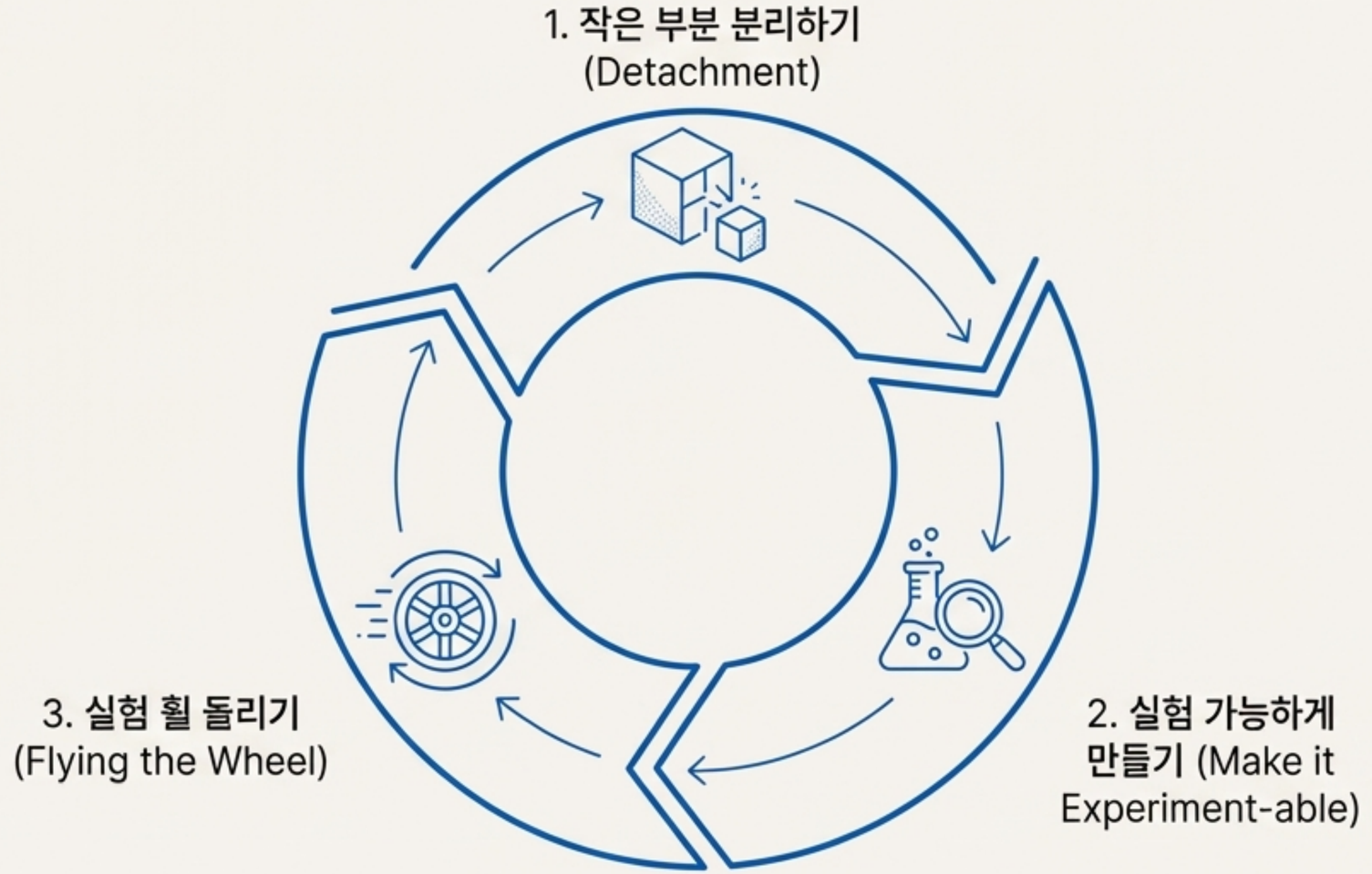
거대한 문제에 압도되어 새로운 해결책을 모색할 에너지를 잃고 현상 유지에만 급급해짐.

학습의 부재 (Absence of Learning)

왜 직동하고 왜 실패하는지에 대한 근본적인 메커니즘을 이해할 기회를 놓침.

TinyExperiment: 작게 분리하고, 빠르게 실험하고, 계속 배우는 기술

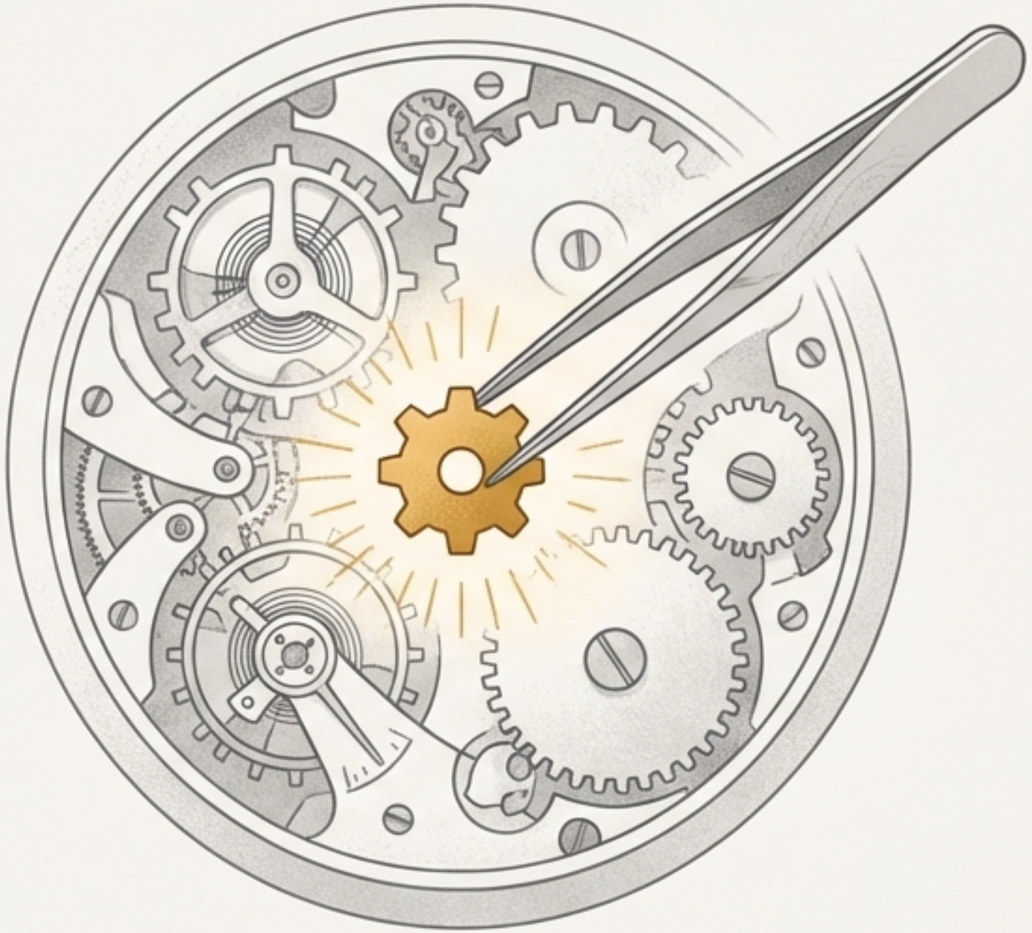
이 방법론이 단순한 기술이 아니라, 거대한 문제를 다루는 새로운 관점임을 강조하는 간결한 설명.



1단계: 거인에게서 돌멩이 하나를 꺼내라 (분리하기)

"전체 시스템이 아니라, 극히 작고 특별한 특정 기능만 분리한다."

- **함수 단위 분리**: 300줄짜리 함수에서 10줄의 핵심 로직만 따로 떼어내기.
- **데이터 단위 분리**: 전체 DB 대신 테스트용 레코드 3-4개만 준비하기.
- **처리과정 단위 분리**: 100단계의 프로세스 중 단 하나의 결과를 내는 1단계만 먼저 실행하기.
- **의존성 단위 분리**: 외부 시스템 없이도 독립적으로 작동하도록 만들기.



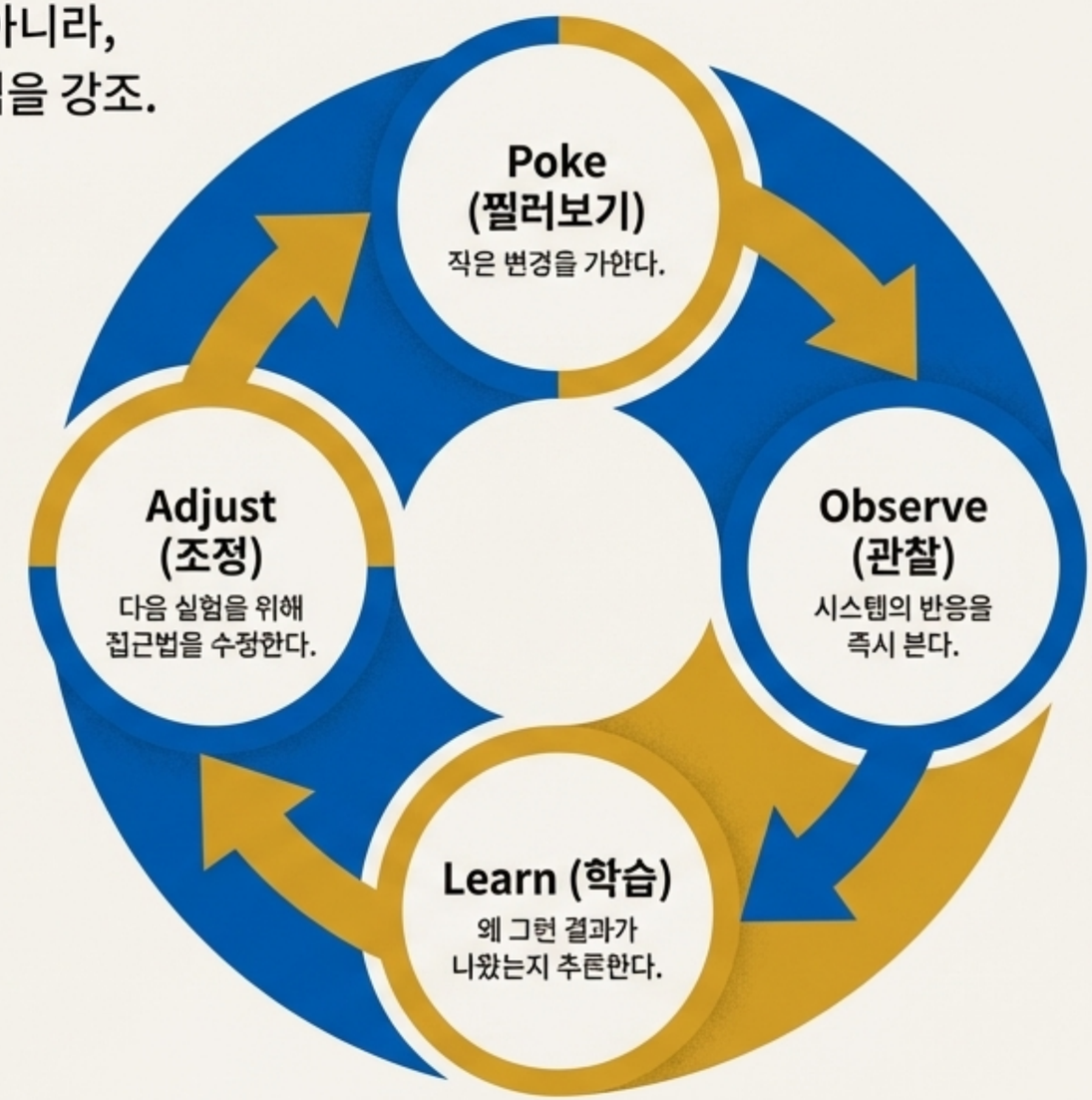
2단계: 나만의 작은 실험실을 구축하라 (실험 가능하게 만들기)

**밑바닥 창의성 (Bottom-up Creativity)* - 실제 상황에서서 떨어지 떨어진 최소한의 환경을 만들어 작은 기능이 혼자서도 돌아가게 하는 능력.

```
function testDataParsing() {
  console.log("날짜 파싱:", parseDate("2024-01-15") ==
= "2024/01/15" ? "PASS" : "FAIL");
  console.log("빈 데이터:", handleEmpty("") === null ?
"PASS" : "FAIL");
  console.log("숫자 변환:", toNumber("123") === 123 ?
"PASS" : "FAIL");
}
```

3단계: 찔러보고, 관찰하고, 배우고, 조정하라 (실험 휠 돌리기)

이 사이클은 정답을 찾는 과정이 아니라,
이해를 깊게 만드는 학습의 과정임을 강조.



이것이 최고의 학습 환경인 이유

삼각측량 피드백 (Triangulation Feedback)

나의 변경(입력)과 시스템의 반응(출력) 사이의
직접적인 인과관계를 명확하게 파악하여
"왜?"라는 질문에 즉시 답을 얻음.

안전한 실패 (Safe Failure)

실패 비용이 거의 없어(몇 초의 시간)
심리적 안정감 속에서 과감한 시도가
가능해지고, 실패를 통해 학습함.

창조의 경험 (Experience of Creation)

정해진 답을 따르는 것이 아니라, 직접 실험하며
해결책을 '발견'하는 기쁨과 깊은 이해를 얻음.

실행 사례 #1: 15분 만에 복잡한 정규표현식 해독하기

Before (전통적 접근)

```
/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])
[A-Za-z\d@$!%*?&]{8,}$/
```

After (TinyExperiment 접근)

```
function testRegexPart(pattern, testString) {
  const regex = new RegExp(pattern);
  console.log(`${testString}` ->
${regex.test(testString) ? 'MATCH' : 'NO MATCH'}`);
}

// 예시 실행
testRegexPart('(?=.*[a-z])', 'Hello123!'); // 소문자
존재 확인
```

결과*: **15분 만에 복잡한 정규표현식을 자신 있게 분석하고 수정할 수 있게 됨.

실행 사례 #2: 페이지 로딩 시간 8초에서 1.5초로 단축하기

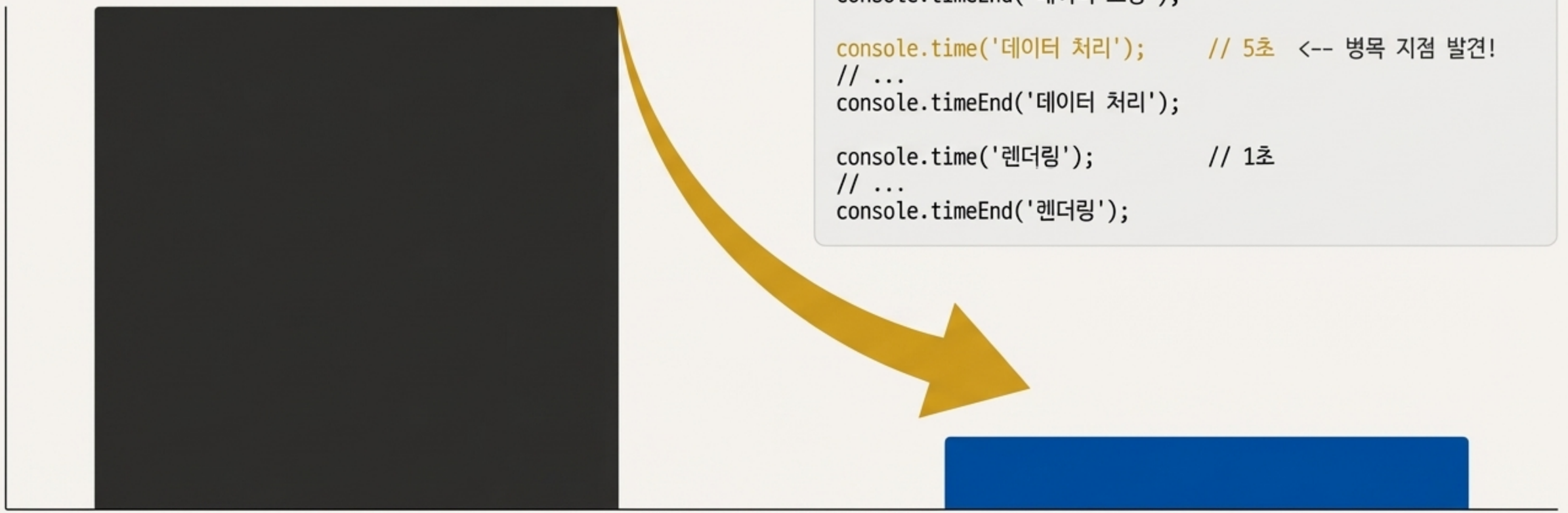
문제 상황 : 페이지 로딩에 8초가 걸리는 심각한 성능 저하.

TinyExperiment 접근

```
console.time('데이터 로딩');      // 2초
// ...
console.timeEnd('데이터 로딩');

console.time('데이터 처리');      // 5초  <-- 병목 지점 발견!
// ...
console.timeEnd('데이터 처리');

console.time('렌더링');           // 1초
// ...
console.timeEnd('렌더링');
```



BEFORE: 8s

AFTER: 1.5s

실행 사례 #3: 2시간 만에 새로운 API 핵심 파악하기

Before (전통적 접근)

After (TinyExperiment 접근)

```
curl -X GET "https://api.example.com/users/1" -H
"Authorization: Bearer token"
```

결과*: 2시간 만에 API의 핵심 동작을 이해하고, 실제 애플리케이션 코드 작성을 시작할 수 있게 됨.

흔히 빠지는 3가지 함정 피하기

① 전체를 이해하려는 욕구

"모든 것을 완벽히 이해하고 시작해야 해"라는 생각 때문에 영원히 시작하지 못하는 문제.

② 완벽한 환경을 기다리기

"제대로 된 테스트 인프라부터 갖춰야지"라며 환경 구축에만 시간을 쏟는 문제.

③ 한번에 너무 많이 바꾸기

"이것도 고치고 저것도 고치고..." 여러 변경을 동시에 적용하여 어떤 것이 효과가 있었는지 알 수 없게 되는 문제.

가장 작은 첫걸음

이 질문들이 당신의 새로운 무기가 된다

The Experiment Mindset

"이것 중에 **가장 작고 성공할만한 것**은 무엇인가?"

"**5분 안에 답을 확인**할 수 있을까?"

"**망쳐도 괜찮게 되돌릴 수** 있을까?"

"어떤 부분만 **먼저 시험**해볼 수 있을까?"

두려워하는 개발자에서 탐구하는 프로그래머로

창의력 (Creativity): 큰 문제 앞에서도 작은 시작점을 발견하는 능력.

학습 속도 (Learning Speed): 빠른 피드백을 통한 깊은 이해.

자신감 (Confidence): 실패에 대한 두려움 없이 과감하게 시도하는 태도.

문제 해결력 (Problem-Solving Power): 체계적인 작은 접근법의 체화.