

문제 해결 관점에서의 AI 에이전트 활용법

이선협 @kciter

발표자 소개

- 이선협(@kciter)
- 12년차 개발자
- 마플코퍼레이션, 플랫폼 엔지니어 리드
- <https://github.com/kciter>
- <https://kciter.so>



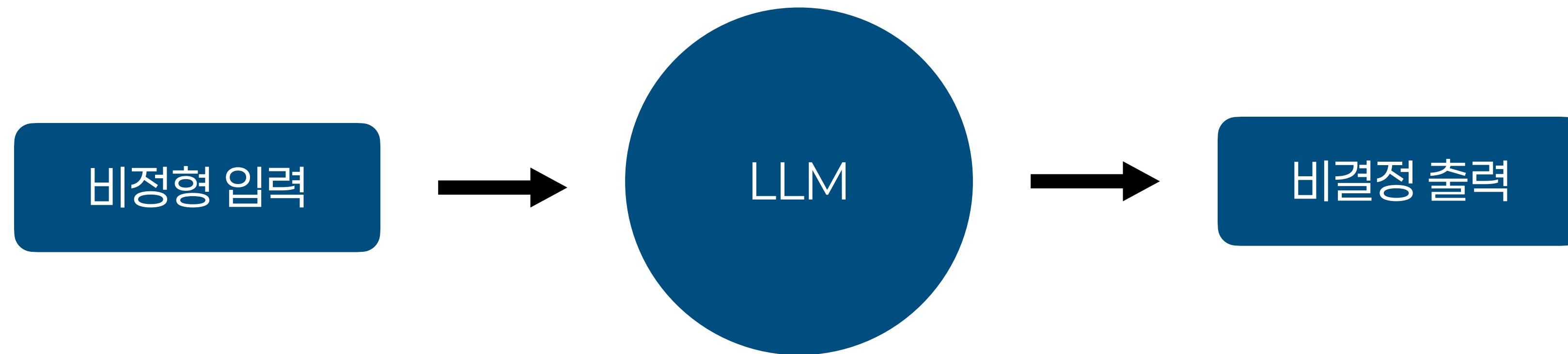
새로운 개발 패러다임

LLM의 등장



LLM의 등장





SL 1부터 100까지 홀수만 더하는 Python 코드 만들어줘

1부터 100까지의 홀수를 더하는 Python 코드를 작성해드리겠습니다. 간단하고 명확한 방법으로 구현하겠습니다.

>Create a Python script to sum odd numbers from 1 to 100

odd_numbers_sum.py

이 코드는 1부터 100까지의 홀수 합계를 계산하는 두 가지 방법을 보여줍니다:

1. **for** 반복문을 사용하는 방법:

- `range(1, 101, 2)`를 통해 1부터 시작해 2씩 증가하며 홀수만 선택
- 각 홀수를 `sum_odd_numbers`에 더함

2. 수학적 공식을 사용하는 방법:

- 1부터 100까지의 홀수 개수는 50개
- $50 * 50 = 2500$ 으로 합계 계산 (더 효율적)

[Python 홀수 합계 스크립트 보기](#)

이 스크립트를 실행하면 1부터 100까지의 홀수 합계인 2500이 출력됩니다. 두 방법 모두 정확한 결과를 제공하지만, 수학적 공식을 사용하는 두 번째 방법이 더 빠르고 효율적입니다.

```
1 # 1부터 100까지의 홀수 합계 계산
2
3 # 방법 1: for 반복문 사용
4 sum_odd_numbers = 0
5 for num in range(1, 101, 2): # 1부터 시작해 2씩 증가하며 홀수만 선택
6     sum_odd_numbers += num
7
8 print("1부터 100까지의 홀수 합계 (for 반복문):", sum_odd_numbers)
9
10 # 방법 2: 수학적 공식 사용 (더 효율적)
11 n = 50 # 1부터 100까지의 홀수 개수
12 sum_odd_math = n * n # (1 + 3 + 5 + ... + 99)의 합
13
14 print("1부터 100까지의 홀수 합계 (수학적 공식):", sum_odd_math)
15
```

코딩 AI 에이전트와 바이브 코딩의 등장

바이브 코딩

- 자연어 → 코드 생성
 - "말"로 프로그래밍하는 새로운 방법
- 자동 완성을 넘어 전체 기능 구현까지 가능
- "드디어 코드 작성에서 해방될 수 있을까?"
 - 개발자의 본질(이라고 생각하는) 비즈니스에 집중할 수 있을까?

AI 에이전트 환상

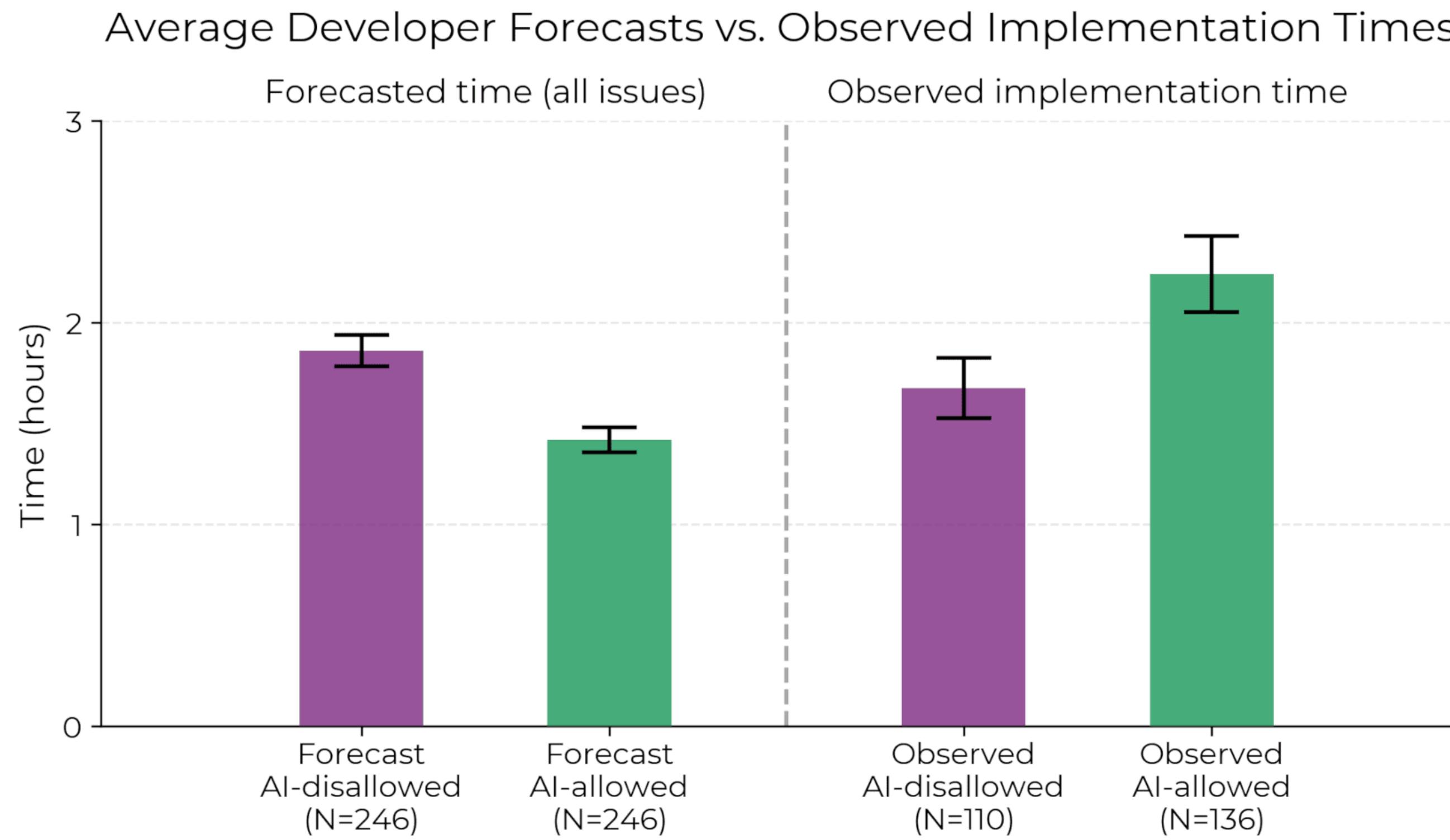
- 결국 마법이 아니라는 것을 깨달음
- 수많은 문제들
 - 보안 문제
 - 성능이 좋지 않은 코드
 - 컨벤션을 고려하지 않은 코드
 - 환각
- 품질 기준을 만족하지 못하는 경우가 많아짐

AI 에이전트 환상

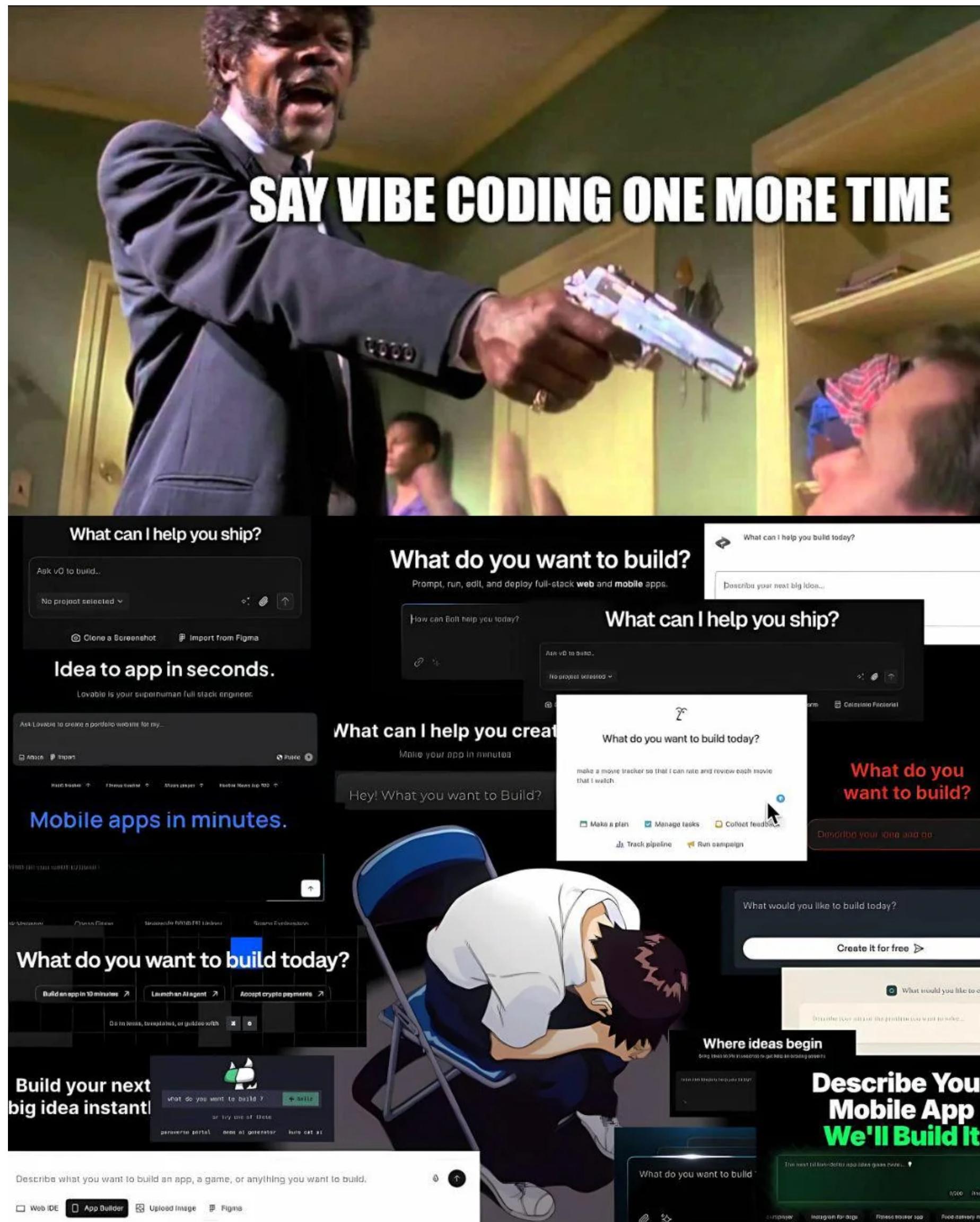
- 해결할 수 없을까?
 - 장문의 프롬프트
 - 컨텍스트 파일 수정 지옥
 - 의사코드화
- 매우 구체적으로 지시해야 한다면 이미 문제를 해결한 것이나 마찬가지

METR 2025: AI 사용 시 19% 느려짐

- 개발자들은 20% 빨라졌다고 믿었으나, 실제로는 반대 결과



<https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/>



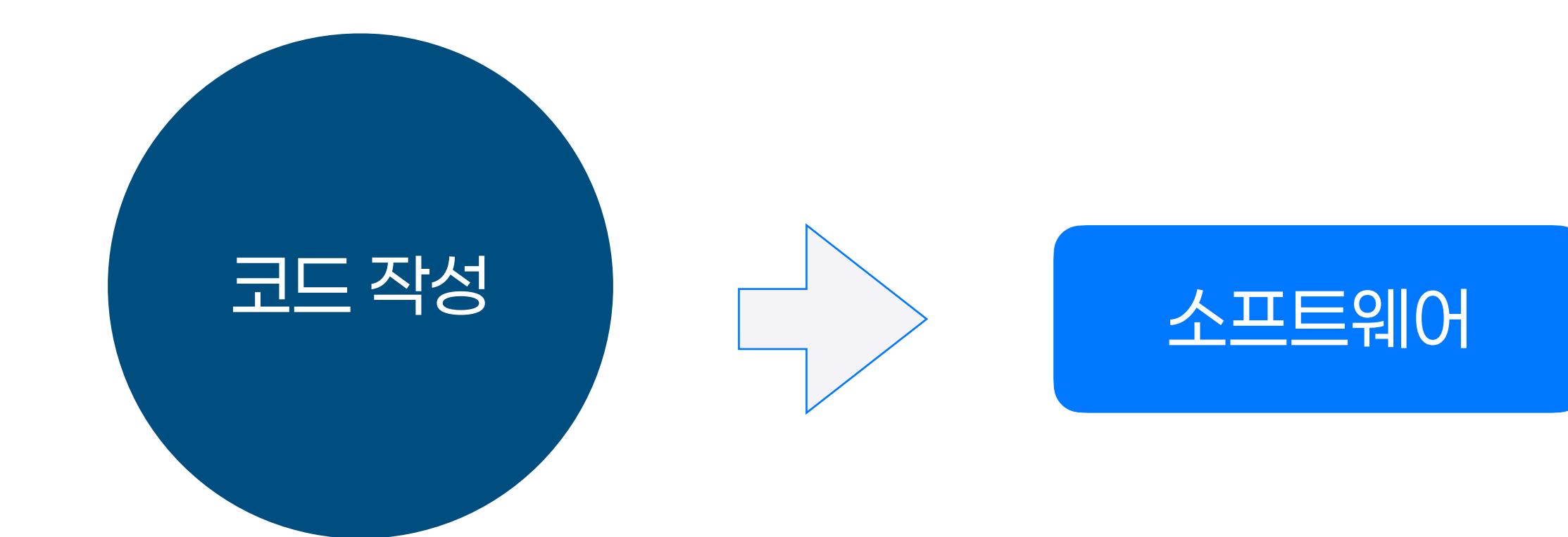
무엇이 문제일까?

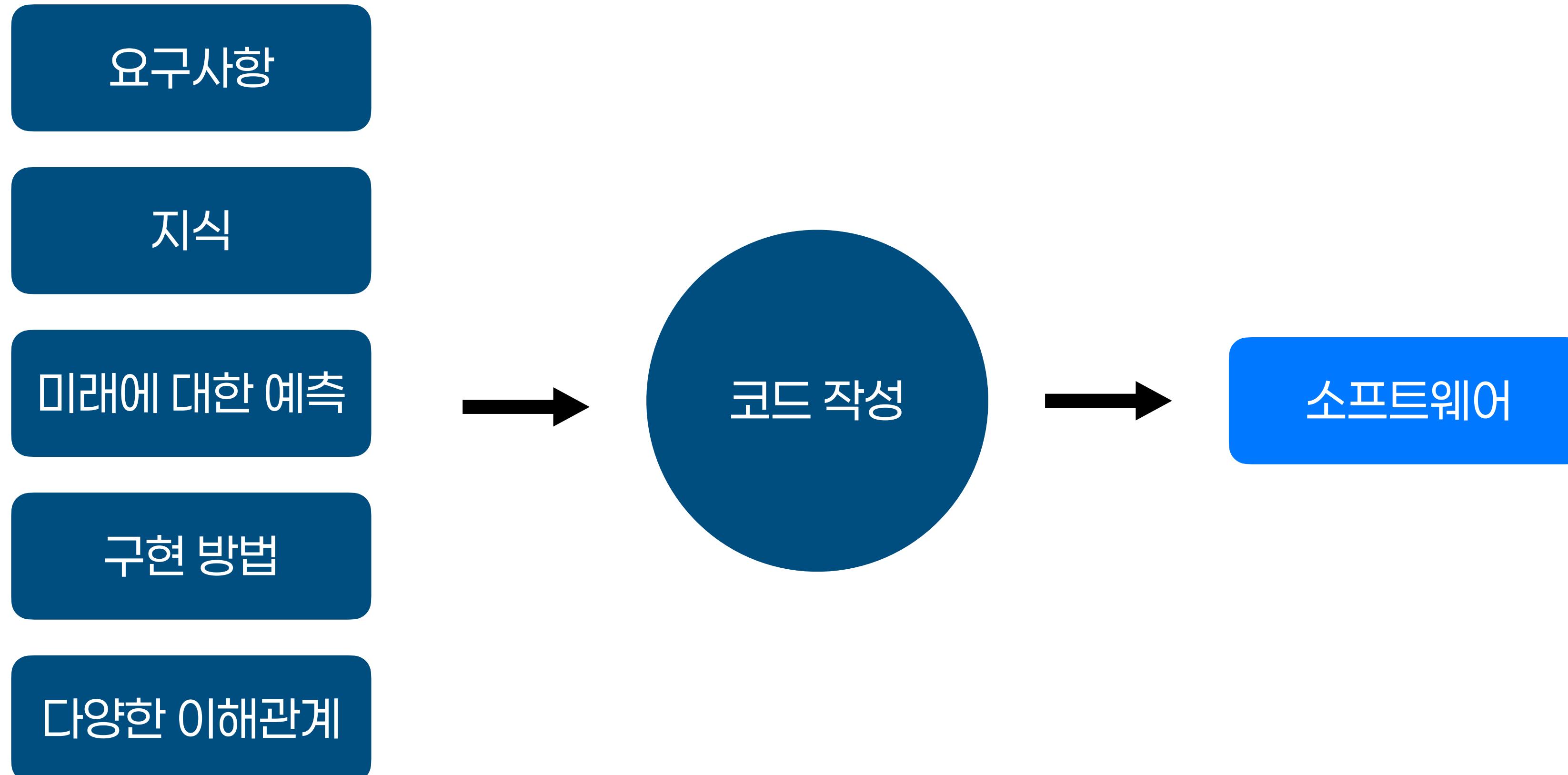
개발자의 역할은?

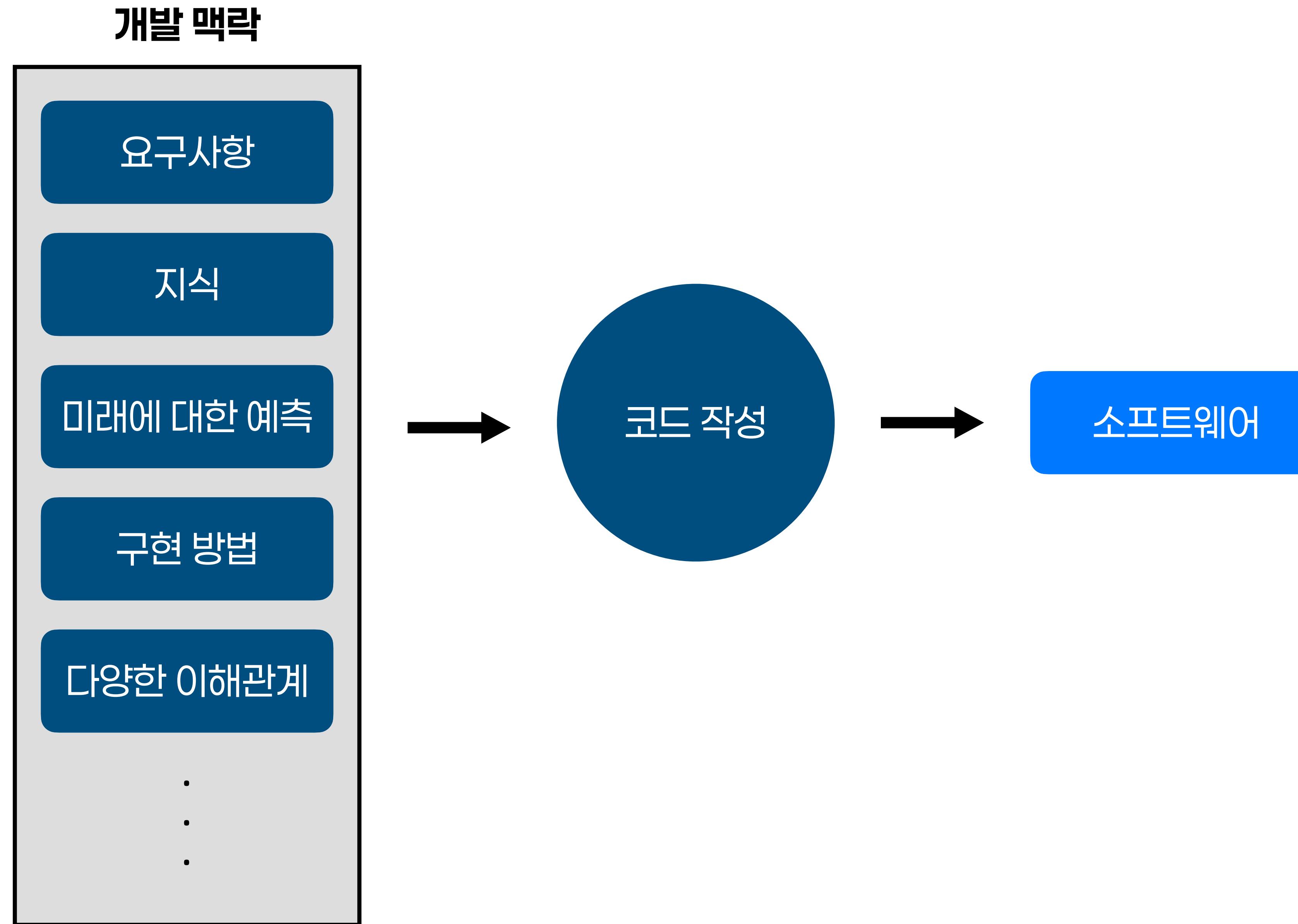
맥락을 고려하여 코드를 통해 소프트웨어를 만드는 것



코드 작성







소프트웨어 개발은 매우 수공업적이다..!

개발자는 매일 여러 맥락 사이에서 균형을 찾아 최선의 판단을 내려야함

암묵지의 문제

- 개발자는 수많은 맥락을 머릿속에 담고 있지만 지식의 대부분은 암묵지로 남음
 - 규칙
 - 도메인 지식
 - 시스템 이해
 - 프로세스
- 왜 이런 일이 생길까?
 - 시간의 문제
 - 귀찮음
 - 전달하기 어려움
 - 비용 절감

개발자의 본질은 코드를 통해 소프트웨어를 만드는 것이 맞음

하지만 숨겨진 비용(맥락 파악)이 코드 작성보다 큰 경우가 훨씬 많음

결국 개발자는 코드 작성보다 맥락 처리에 더 많은 시간과 에너지를 소모함

진짜 문제는 '맥락 처리'

맥락 문제를 해결할 수 있다면 코드도 더 빠르게 생산할 수 있다

AI 에이전트와 엔지니어링

그렇지만 안쓸 이유가 없다

- AI 에이전트가 '제대로' 구현해줬을 때 생산성이 매우매우매우 높은 것은 틀림없는 사실
- 사람이 5분 안에 1000줄을 짜는 건 거의 불가능하다
- 거의 코드 생산의 자동화 공장 (주문 제작 가능한)

AI도 맥락의 지배를 받는다

- 맥락 없이는 AI도 무력함
- 자연어 입력의 편의 ≠ 맥락 자동 이해
- 비정형 입력이더라도 여전히 입력 자체는 필요하며, 입력 품질에 결과가 좌우됨
- 결국 LLM은 맥락과 지시에 따라 무언가를 출력하는 도구일 뿐

우리는 AI 에이전트를 '사람'이 아닌 '도구'로 봐야한다

절대 알아서 무언가를 해주지 않는다

컨텍스트 엔지니어링

- "Context engineering is the #1 job of engineers building AI agents" - Cognition (Devin)
- 프롬프트 엔지니어링 → 어떻게 지시할까?
- 컨텍스트 엔지니어링 → 어떤 맥락을 줄까?
- 컨텍스트가 중요하다는 것을 깨닫고 관련한 연구가 활발히 진행 중

그러나 문제가 있음

- 우리도 모든 맥락을 알지 못함
- 알지 못하는 맥락을 AI에 전달하면 AI 출력 품질의 저하로 이어짐
- 코드 생성 속도는 매우 빠르나 실제 생산성은 내려가게됨
 - 리뷰, 잘못된 코드 수정, 환각 대응 등

거꾸로 생각해보자

AI 에이전트를 맥락 해결에 이용할 수 있다면?

AI 에이전트를 엔지니어링의 영역으로

- 맥락을 AI 에이전트로 다룰 수는 없을까?
 - 자동화 파악 / 생성
- 계획과 검증
 - 구현을 시작하기 앞서 충분한 정보가 모였는가?
 - 구현 후 맥락에 따라 검증되었는가?

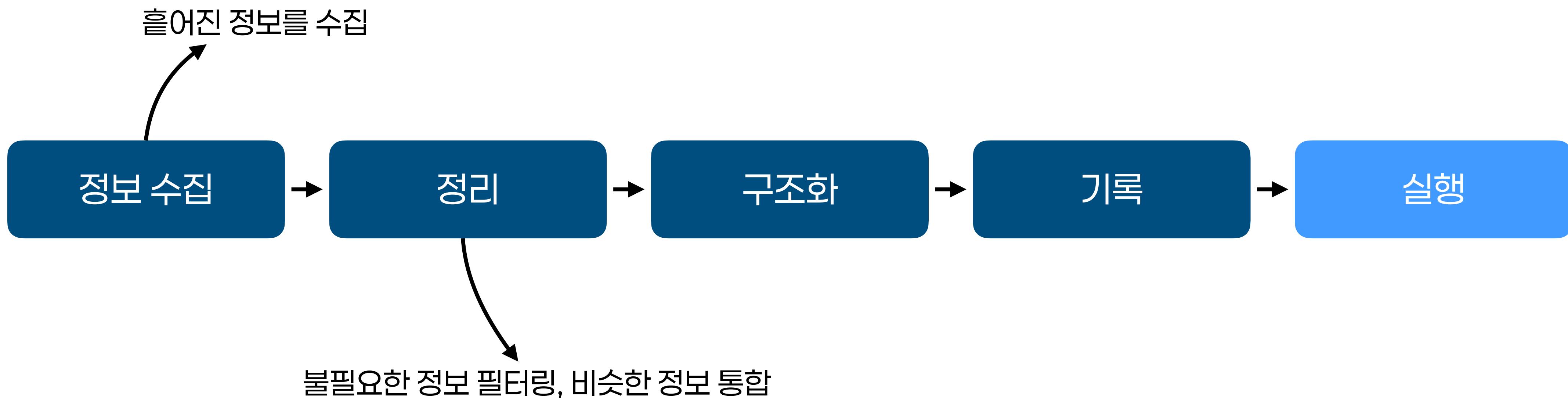
계획을 세우기 위한 절차



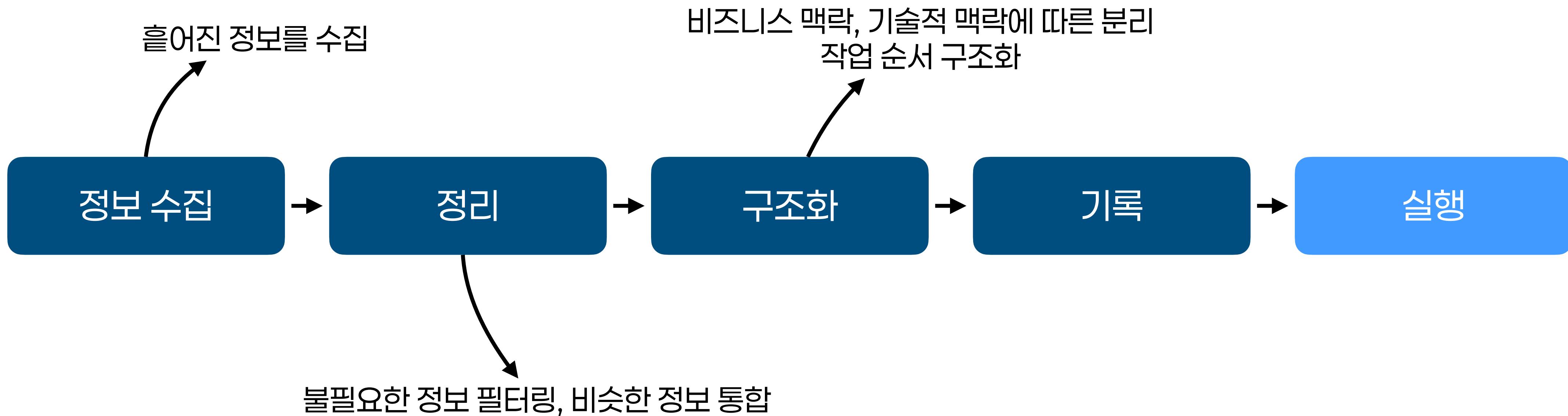
계획을 세우기 위한 절차



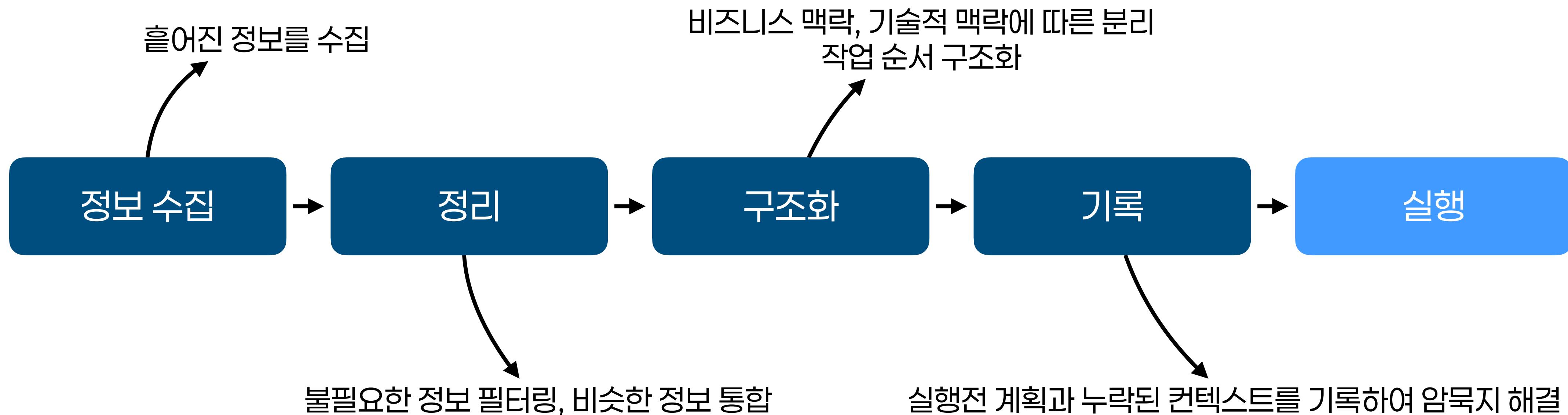
계획을 세우기 위한 절차



계획을 세우기 위한 절차



계획을 세우기 위한 절차



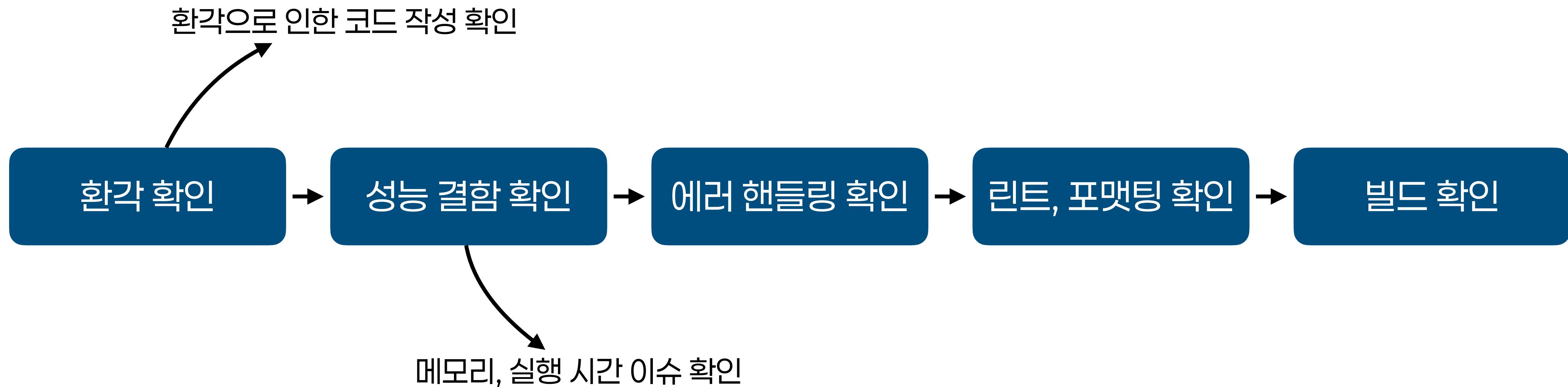
작업에 대한 검증 절차



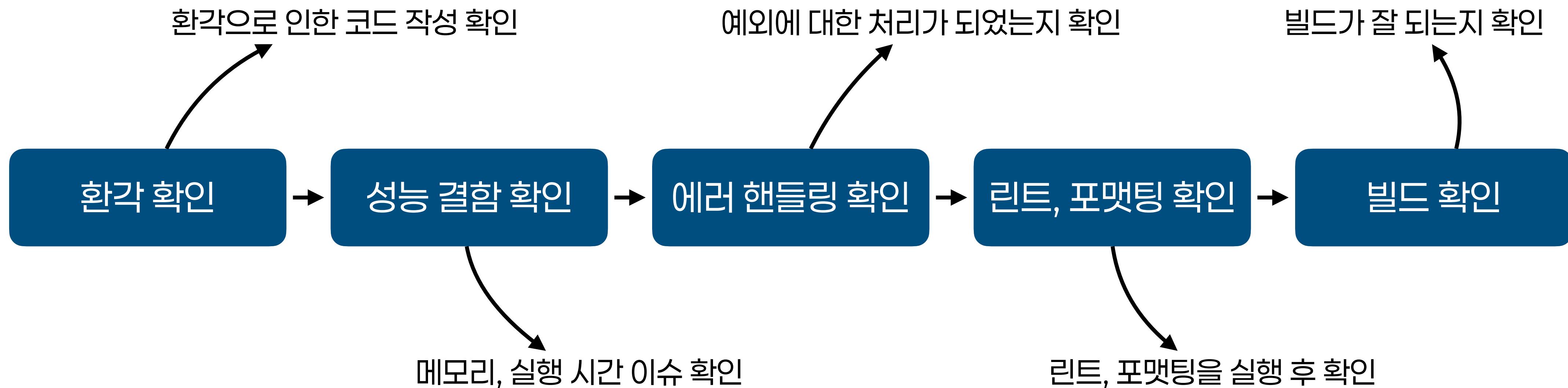
작업에 대한 검증 절차



작업에 대한 검증 절차



작업에 대한 검증 절차



컨텍스트 엔지니어링

- 재사용 가능한 컨텍스트 단위
 - Agentic Primitives - 재사용 가능한 맥락 블록
- 컨텍스트는 AI 에이전트의 코드나 마찬가지



이런 작업을 AI 에이전트가 할 수 있도록 만들자!

Claude Code와 함께하기

증강 코딩

- AI 에이전트는 모든 것을 대신하는 것이 아니라 옆에서 도와주는 역할
- 휴먼 인 루프
 - 판단과 결정
 - 리뷰
 - AI 에이전트가 제대로 못하는 것은 직접하기
- 어떤 판단?
 - 암묵지 찾기
 - 컨텍스트 보충
 - 기술적 판단을 위한 학습



Claude Code로 맥락을 관리하는 방법

- **CLAUDE.md (AGENTS.md)**
 - Claude Code가 알아야하는 공통 맥락을 정리
- **Skills**
 - 재사용 가능한 지식 맥락
 - 암묵지를 명시화
 - Git으로 버전 관리 가능
 - Agentic Primitives
- **Hooks**
 - 규칙에 기반하여 자동 실행
 - 특정 이벤트(도구 사용, 세션 시작/종료 등)에 자동으로 적용

Skills

- 출시된지 얼마안된 따끈따끈한 기능 (발표일 기준 보름 정도..)
- 개발팀 내부의 지식을 마크다운으로 명시적 작성 가능
 - 암묵적인 규칙, 주의할 점을 마크다운으로 정리
- <https://github.com/anthropics/skills>
 - 공식 저장소

About This Repository

This repository contains example skills that demonstrate what's possible with Claude's skills system. These examples range from creative applications (art, music, design) to technical tasks (testing web apps, MCP server generation) to enterprise workflows (communications, branding, etc.).

Each skill is self-contained in its own directory with a `SKILL.md` file containing the instructions and metadata that Claude uses. Browse through these examples to get inspiration for your own skills or to understand different patterns and approaches.

Skills - 쉽게 시작하기

- 어떻게 시작할지 막막하다면 에이전트에게 시켜보자

> [REDACTED]에서 암묵적으로 사용되는 코딩 컨벤션을 정리해줄래?

• [REDACTED]의 코딩 컨벤션을 정리하기 위해 코드베이스를 분석하겠습니다.

• [REDACTED]의 암묵적 코딩 컨벤션을 정리했습니다. 주요 내용은 다음과 같습니다:

주요 암묵적 규칙 요약

네이밍 규칙

- 파일명: 캐밥 케이스
- 클래스명: PascalCase
- 함수명: camelCase (
- **Enum**: E 접두사 (ECa
- **Type**: T 접두사 (TCh
- **Exception**: {원인 }Ex

컨트롤러 패턴

- 하나의 컨트롤러
- 내부 DTO는 {
- Success DTO는 }

o 접미사 사용

DTO 패턴

- **InputDto**: 검증 데코레이터 + **@ApiProperty**로 문서화
- **OutputDto**: 생성자 없이 객체 리터럴로 생성
- **SuccessOutputDto**만 생성자 포함

> 이 규칙을 Claude Code skills로 정리할 수 있을까?

- 네, Claude Code skills로 정리할 수 있습니다! 먼저 Claude Code의 skills 기능에 대해 확인해보겠습니다.
- **Fetch**(https://docs.claude.com/en/docs/clause-code/clause_code_docs_map.md)
 - ↳ Received **26.1KB** (200 OK)
- **Fetch**(<https://docs.claude.com/en/docs/clause-code/skills.md>)
 - ↳ Received **14.9KB** (200 OK)
- 네, 완벽하게 가능합니다! [REDACTED]의 코딩 컨벤션을 Claude Code Skill로 만들어드리겠습니다.
- **Bash**(`mkdir -p .claude/skills/[REDACTED]-conventions`)
 - ↳ (No content)

∴ Thinking...

+ **SKILL.md** 파일 작성 중... (**esc** to interrupt · **ctrl+t** to hide todos · 11s · ↓ 1.4k tokens)

- ↳ ☐ .claude/skills 디렉토리 구조 생성
- ☐ **SKILL.md** 파일 작성
- ☐ 참고 문서 파일 작성

Skills - 주의할 점

- SKILL.md는 500줄 이하로 작성하는 것을 Anthropic이 공식적으로 권장
- 상세한 내용은 별도 파일로 분리하고 SKILL.md에 레퍼런스를 달아두기
- 중요한 것은 꾸준히 관리할 것
- 가끔 Claude가 필요할 때 자동으로 Skill을 발견하지 못하는 문제가 있음
 - Hook을 이용하여 Skill을 찾도록 지시 가능

Hooks

- Skills보다는 늦지만 나온지 몇 달 안된 기능
- 이름처럼 어떠한 이벤트 이후로 강제 실행할 수 있는 도구
 - 예를 들어, 파일 편집 후 lint나 formatter를 무조건 실행하게 할 수 있음
 - 불필요한 토큰 낭비를 줄여줌 (빌드 결과 확인 후 수정하는 등)

Hooks - 가능한 이벤트

- UserPromptSubmit: 사용자가 프롬프트를 제출할 때 (Claude가 처리하기 전)
- PreToolUse: 도구 호출 전 (실행을 차단할 수 있음)
- PostToolUse: 도구 실행 후
- Notification: 알림이 필요할 때
- Stop: Claude Code 세션 종료 시
- SubagentStop: 서브에이전트 작업 완료 시
- PreCompact: 컴팩션 작업 전
- SessionStart: 새 세션 시작 또는 기존 세션 재개 시

Hooks - Skill 탐색하게 만들기

- 사용자가 지시해도 Claude Code가 Skill을 탐색하지 않을 때가 있음
- CLAUDE.md에 명시해도 누락되는 경우 있음
- 조금 더 탐색할 확률을 늘리기 위하여 Hook 사용 가능

```
{
  "hooks": {
    "UserPromptSubmit": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "echo '💡 Consider running /skills to explore available skills for this task' && sleep 1"
          }
        ]
      }
    ]
  }
}
```

Hooks - Skill 탐색하게 만들기

- 프롬프트 내용을 읽어서 출력하는 것도 가능

```
#!/bin/bash
# 프롬프트 내용 읽기
read -r prompt

# 특정 키워드 감지하면 경고 출력
if echo "$prompt" | grep -iE "(document|spreadsheet|pdf|create|modify|edit)" > /dev/null; then
  echo "⚠️ This task might benefit from using a skill. Consider checking available skills." >&2
fi

echo "$prompt"
```

Hooks - 작업 후 자동화

- 파일 편집 후 Lint나 Formatter 실행 가능
- 불필요한 빌드 실패, 타입 에러 탐색에 대한 토큰을 아낄 수 있음

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "~/.claude/hooks/lint-and-format.sh"
          }
        ]
      }
    ]
  }
}
```

놓친 암묵지를 AI 에이전트가 말해주게 만들기

- CLAUDE.md를 이용하여 말해주게 만들 수 있음
- 별거 아닌 기능이지만 사용자가 놓친 것을 알 수 있게 해줌
- 하지만 당연히 누락될 수 있으므로 사용자가 주의하기

작업을 마쳤을 때

- * 논의한 내용과 작업 기록을 기반으로 기존 SKILL 문서에 누락된 것이 없는지 확인합니다.
- * 누락된 내용이 있다면 내용을 추가합니다.

SKILL 문서를 읽을 때

- * 문서의 내용과 현재 상황이 일치하는지 확인합니다.
- * 필요한 경우 문서를 업데이트 합니다.
- * 업데이트 하는 경우 마지막 업데이트 시간을 남깁니다.

작업 맥락 저장하기

- Claude Code는 작업 중 맥락이 유실될 수 있음
- 잊지 않도록 작업 맥락을 저장해두면 품질이 좋아짐
- 개발자가 작업 파일을 보고 맥락을 파악할 수 있음
- CLAUDE.md에 다음과 같이 지시할 수 있음

계획 모드 종료할 때

계획 모드를 종료하기 전에 지금까지 논의된 계획과 맥락을 정리해야 합니다.

다음과 같이 파일을 만듭니다.

```
mkdir -p .claude/tasks/[task-name]-plan.md
```

`plan.md`에는 다음 내용이 기록되어야 합니다.

- * 수락된 계획 내용
- * 논의된 맥락
- * 작업 체크리스트

작업을 진행할 때

- * `.claude/tasks` 디렉터리를 확인하여 작업 문서가 있는지 확인합니다.
- * 작업 문서를 읽은 후 작업을 진행합니다.
- * 작업이 마무리되면 체크리스트를 확인하여 완료 표시를 합니다.
- * 맥락이 변경되면 작업 문서를 업데이트 합니다.

MCP를 사용하지 않는 이유

- 비즈니스 맥락은 취합하기 매우 어려움
- Slack, Notion, GitHub 등 모두 탐색하기 위한 토큰이 매우 많이 소비됨
- Human in the Loop가 필요한 이유
- 가급적 발견한 비즈니스 맥락은 Skills로 기록

중요한 것은 암묵지를 모아두는 것!

- 여기서는 Claude Code의 Skill과 Hook을 이용했지만, 다른 AI 에이전트도 가능
- AI 에이전트가 코드를 생산하는 것보다 중요한 것은 맥락을 찾을 수 있게 만드는 것!

마치며

코딩 AI 에이전트는 의외로 범용

- 꼭 코딩이 아니더라도 글쓰기, 맥락 정리 등에 이용할 수 있음
- 구성하기에 따라 범용 에이전트로 충분히 활용 가능
- 발표자의 경우 코딩, 글쓰기, 맥락 정리, 학습에 모두 Claude Code 사용

AI 에이전트 어디까지 이용해야해?

- 기준은 있을 수 없음
- 소프트웨어는 일반화 할 수 없다
 - 금융 소프트웨어와 게임은 같은 소프트웨어지만 정말로 같다고 할 수 있을까?
 - 백엔드와 프론트엔드 개발이 같다고 할 수 있을까?
 - 결국 공을 이용하는 것은 같지만 농구와 축구가 같다고 말하는 꼴
- AI 에이전트도 당연히 만드는 소프트웨어에 따라 이용할 수 있는 정도가 다르다

중요한 것

- 엔지니어링 감각을 유지하는 것이 중요
 - 미래는 계속 바뀔 수 밖에 없음
 - 모델이 **정말정말정말** 너무 좋아지면 AI 에이전트는 어떻게 될까? (모름)
 - Spec Driven Development가 정말 가능할까? (모름)
 - 끊임없이 진짜 '문제'는 무엇인가에 집중할 필요가 있음
- AI 에이전트를 이용하는 것은 어렵지 않지만 하지만 팀에 맞는 방법은 각자 다를 수밖에 없음
 - 가설과 검증을 통해 우리 팀만의 AI 에이전트 이용법을 만들자

감사합니다

이선협 @kciter