팀 단위 바이브 코딩 워크플로우 적용 결과 보고

IT본부 프론트엔드개발팀 김현우

26. 01. 08

1. **핵심 결론** 효과가 있었는가?
2. **배경** 왜 AI 코딩 워크플로우를 검토했는가?
3. **라이브 시연** 실제로 어떻게 동작하는가?
4. **파일럿 결과** 무엇을 확인했고, 한계는?
5. **핵심 인사이트** 전체 프로세스에 미치는 영향은?
6. **향후 개선 전략**

정량적, 정성적 개선 효과는 분명하지만 주요 한계도 인식해야 함

파일럿 기간	참여 규모	참고
2025.12.15 ~ 12.22 (6일)	5명 / 10개 티켓 완료	· 개발 시 필요한 레퍼런스를 실제 환경과 최대한 유사하게 구성 (Figma, Swagger) · 파일럿 결과: 클라이언트, 백엔드 Swagger, 한샘 Swagger MCP

주요 성과	주요 한계
80% ~ 90% 개발 시간 단축 (API 연동 시간 1/3, 구현 사전 조사(고민 시간), 스타일링 ...) (워크플로우 + Figma MCP + Hanssem Swagger MCP)	· 계획 리뷰, PR 리뷰 등의 간소화(파일럿) · 기획(확정된 기획) / 디자인(확정된 Figma 디자인) / API (확정된 OpenAPI 스펙의 Swagger) · 전체 프로세스에서 개발의 병목을 줄였을 때 효과는 대략 10% ~ 20% 불과
80%↑ 구현율(1 shot) (수기 개입 횟수 5회 ~ 15회)	· Greenfield만 검증 (레거시 미검증)

파일럿 결과 링크
파일럿 스웨거 링크
파일럿 피그마 링크
한샘 Swagger MCP

왜 "팀 기반" 워크플로우인가

[개인] AI 도구 활용 활발

Claude Code, Cursor, Copilot, Codex 등 다양한 도구 활용 중

Skills, MCP, Commands, Prompting 등 기법 활용

→ 개인 차원에서는 생산성 향상 확인

[팀] 협업 시 문제 발생

1. 도구 파편화 각자 선호하는 도구가 다름
2. 프롬프트 편차 같은 의도, 다른 표현 → 다른 결과물
3. 리뷰 난이도 증가 AI 생성 코드의 맥락 파악 어려움

→ 팀 단위 표준화 필요

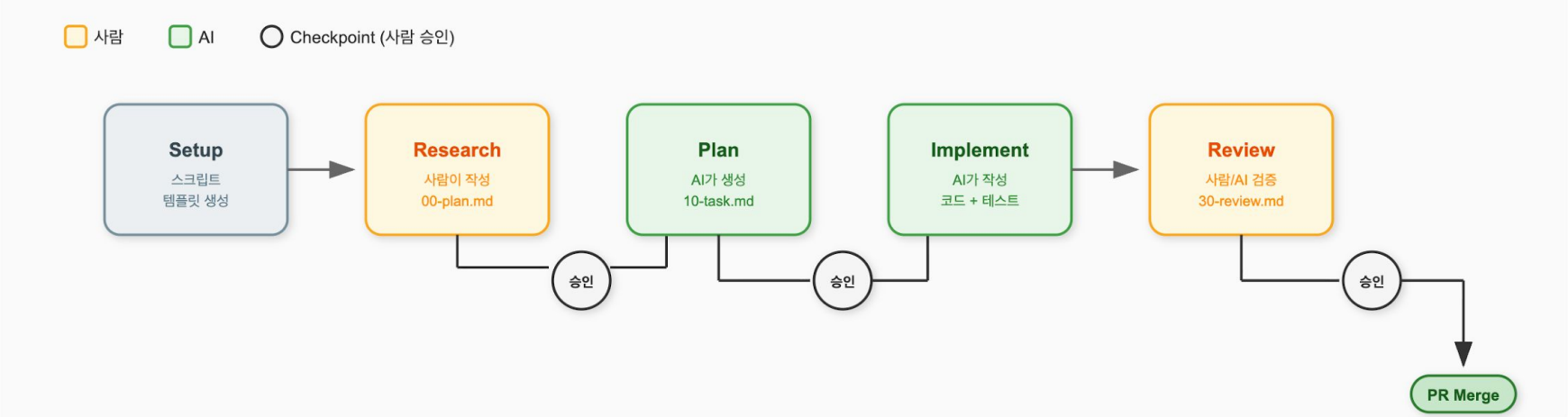
접근 방식: 표준 워크플로우

표준화된 워크플로우
공통 규칙 + 공통 템플릿 + 공통 명령어

↓

개발자 1 ≈ 개발자 2 ≈ ... 개발자 n
동일한 Input/Output 품질

핵심: 도구가 달라도 워크플로우는 동일
비결정적 응답을 팀 레벨에서 일관되게 유지하며 점진적, 지속적 개선



https://firsthubai.com/how-manus-ai-works/

Manus AI's Workflow Explained (Step-by-Step)

Let's visualize the **real journey** of how Manus AI works when you assign a task.

1. Task Assignment & Planning

- The user defines **the task goal**
- Manus AI **analyzes the scope**
- Generates **todo.md** plan

2. Research & Resource Collection

- Accesses the **Datasource APIs**
- Conducts **web searches**
- Gathers & verifies info from **multiple sources.**

3. Execution & Automation

- **Writes code**, executes scripts.
- **Browses websites**, extracts data.
- **Automates workflows** step-by-step
- **Builds apps**, deploys websites

4. Communication & Interaction

- Sends **progress notifications** (message_notify_user)
- Asks for **essential approvals** (message_ask_user)
- Suggests **browser takeover** (for sensitive tasks)

5. Task Completion & Result Delivery

- Finalizes the **todo.md checklist**
- Provides **outputs** (reports, apps, dashboards)
- Enters **standby mode** for the next task

Google Antigravity

CLAUDE CODE

CURSOR

표준화된 워크플로우가 해결하는 3가지 문제

❌ 워크플로우 없이 AI 사용 시

1. Prompt Drift

"이 함수 리팩토링해줘" vs "클린 코드 원칙에 맞게 개선해줘"
→ 같은 의도, 다른 결과물 → 코드베이스 일관성 붕괴

2. Input/Output 예측 불가

AI 비결정성 + 개인별 프롬프트 차이
→ 산출물 품질 편차, 재현 불가능

3. 리뷰 부담 가중

AI가 대량 코드 생성 → 맥락 파악 어려움
→ "이 코드가 왜 이렇게 됐지?" 추적 불가

✅ 워크플로우 적용 후 개선 효과

1. 일관된 산출물

공통 Rules + Templates → 누가 실행해도 유사 결과
숙련도 30% 인원도 90% 유사 품질 달성

2. 예측 가능한 프로세스

plan.md로 구현 전 산출물 예상 가능
Plan 단계에서 품질 이슈 사전 발견 → 재작업 방지

3. 리뷰 효율화

plan → task → result 체인으로 의사결정 추적
task.md에 리뷰 포인트 집약 → 시작점 명확

개발 시간 80% 단축, 구현 유사도 80~95% 달성

생산성 지표 (정성적, 정량적)

항목	기존	워크플로우 적용	개선
Plan 작성	-	10분	-
Task 작업	1일	2시간	**80%↓**
구현유사도	-	80~95%	-
수동 개입	-	5~15회/태스크	개선 필요
리뷰	-	-	실제 시나리오 테스트 필요

개별 측정 사례

- 윤종호: 계획+구현 **4~5일 → 15분** (단순 페이지)
- 이지영: API 연동 **1/3로 단축** (Swagger MCP 활용)

MCP Tool 연동 효과

MCP Tool	효과
Figma MCP	디자인 재현율 **90% 이상**
Swagger MCP	API 연동 시간 **1/3로 단축**
Mermaid 활용	정확도 추가 개선

MCP Tool은 워크플로우 내에서 참조 자료(Figma, Swagger)를
AI가 직접 조회하도록 연결 → 수동 복붙 제거

품질 지표

✓ UI/UX: Figma 디자인 90% 이상 재현

✓ API 연동: Swagger 기반 자동 타이핑, hooks 연동 이슈 없음

✓ 유닛 테스트: 케이스 고려 잘됨, Mocking 포함

> **"구현 계획을 거의 구현 수준으로 볼 수 있다"**

단계별 긍정 피드백

Plan 단계

- 구현 전 산출물 예상 가능 → 사전 피드백 가능
- 접근성, 성능최적화 등 놓치기 쉬운 부분 자동 챙김

Implementation 단계

- 1-shot 결과 나쁘지 않음 (UI/UX, 로직)
- Figma 디자인 거의 정확하게 재현
- Swagger → 타이핑 → API → hooks 연동 이슈 없음

Review 단계

- task.md에 리뷰 포인트 집약 → 변경 사항 파악 용이
- 리뷰 경중 파악, 우수 사례 제시, 보안 인사이트 제공

참여자 코멘트

박대윤

"task.md에 미리 구현을 작성해서 보여줘서 피드백 가능해짐"

이지영

"Swagger MCP 사용 시 작업 시간 **1/3로 단축**"
(자연어로 참조할 API 작성 → MCP 조회 → 스키마 조회 후 타이핑 및 API 연동까지 자동으로 완성)

윤종호

"계획+구현 소요 시간 **4~5일 → 15분**" (단순 페이지)

주요 의견

"기술 스택에 대한 숙련도가 높지 않아도 시작할 수 있음."

"유닛 테스트를 알아서 케이스 고려해서 잘 만들어줌. Mocking 등 실제 개발자가 하면 오래 걸리는 작업들을 단시간에 해줌."

파일럿 전제 조건과 미검증 영역을 인식해야 함

⚠️ 전제 조건 (모두 충족된 상태)

1. 확정된 기획

요구사항 변경 없이 진행

2. 확정된 디자인

Figma 디자인 확정 상태에서 시작

3. 확정된 API 명세

OpenAPI 스펙의 Swagger 완비

4. Greenfield 환경

레거시 코드 없는 신규 개발

→ 실제 프로젝트에서는 기획 변경, 디자인 수정, API 미확정 등 대기 시간 발생. 이 대기 시간은 AI로 줄일 수 없음

⚠️ 파일럿에서 간소화된 영역

계획 리뷰 / PR 리뷰

실제 운영 환경 수준의 리뷰 프로세스 미적용

⚠️ 전체 프로세스 관점

개발 병목만 해소 시 전체 프로세스 단축 효과: 10% ~ 20%

기획(2주) → 디자인(2주) → BE(2주) → FE(2주) → QA(2주) 가정 시
개발 80% 단축해도 전체 일정은 20%만 개선

❌ 미검증 영역

Brownfield (레거시 코드베이스)

기존 로직 파악, 사이드 이펙트 예측 등 추가 복잡도 존재
→ 별도 검증 필요

> **"코딩은 안하지만 볼게 너무 많다"**

단계별 부정 피드백

Plan 단계

- 볼 항목이 많음
- 계획을 꼼꼼히 체크해야 함
- 동일 파일명, 중복 파일, type 충돌 가능성

Implementation 단계

- GNB/BNB 등 연계 작업 명시 안 하면 누락
- 디테일한 스타일링(간격 등)은 수동 개입 필요
- 긴 페이지에서 텍스트/이미지 잘못 매핑

Review 단계

- 리뷰 결과가 문서화되지 않음 (30-review.md 부재)
- task.md와 실제 코드 연결이 쉽지 않음
- 태스크 클수록 변경 사항 부담 증가

참여자 코멘트

윤종호

"너무 쉽고 맥락 모르는 채로 코드 만들기 때문에 **리뷰하기 싫어질 수 있음**"

→ 리뷰 회피 리스크 존재

주요 의견

"코딩은 안하지만 볼게 너무 많다..."

"처음 PR을 열면 리뷰 때 뭘 봐야할지 조금 막막함."

"사람 개입없이 완벽하게는 아직 갈 길이 멀다..."

도구/프로세스 마찰

- Figma 그룹화 안 된 디자인 URL은 깨짐
- MCP 프롬프트 미세 차이로 다른 Tool 호출
- plan 생성 시 입력 항목 과다

구현 고민 + 리서칭 + 구현 시간 → 계획 + 계획 리뷰 + 코드 리뷰 시간으로 이동

시간 배분의 변화

기존

- "이 로직을 어떻게 구현하지?" 고민, 서칭, 문서 탐색
- 코드 작성 사전 조사 및 코딩 시간에 대부분의 시간 소요
- 조사 및 구현 역량이 핵심 경쟁력

AI 워크플로우 적용 후

- 코드 작성 사전 조사 시간 대폭 단축
- 실제 코딩 시간 대폭 단축
- Plan 리뷰: "내가 계획하고 AI 보강한 이 계획이 맞는가?" 검증
- Code 리뷰: "이 코드가 의도대로 동작하는가?" 검증

→ 계획 하고, 계획을 리뷰하고 의사결정하는 역할로 전환

역할 변화 요약

영역	기존	AI 워크플로우
핵심 역량	조사 / 구현 능력	**계획 리뷰 / 코드 리뷰 능력**
시간 배분	조사 / 코딩 80%	리뷰 + 의사결정 80%
산출물	코드	**검증된 코드 + 문서화된 의사결정**

필요한 변화

✓ 리뷰 역량 강화 (Plan 리뷰, Code 리뷰)
✓ AI 산출물 품질 판단 능력
✓ 요구사항 → 명확한 계획 전환 능력

개인 최적화가 아닌, 팀 워크플로우의 점진적 발전이 핵심

❌ 개인별 AI 활용

- 각자 프롬프트 최적화 → Prompt Drift 심화
- 개인 노하우 축적 → 팀 공유 안됨
- 도구별 설정 파편화 → 유지보수 비용 증가

→ 단기 생산성 ↑, 장기 기술부채 ↑

✅ 팀 워크플로우 점진적 발전

- 공통 Rules/Templates 지속 개선
- 파일럿 피드백 → 워크플로우 반영
- 성공/실패 사례 팀 전체 학습

→ 개선이 모든 팀원에게 즉시 적용
→ 비결정적 AI를 팀 레벨에서 일관되게 제어

전체 프로세스 단축이 목표

현재: 개발 단계만 80% 단축 → 전체 10~20% 개선

목표: 각 단계별 병목 해소 → 전체 리드타임 단축

기획 → 디자인 → **개발** → QA

각 단계의 대기 시간, 재작업 시간 최소화 필요

명세 품질이 AI 효과를 결정

AI가 올바르게 참조할 수 있는 명세 필요:

- **기획**: 명확한 요구사항 정의
- **디자인**: 구조화된 Figma (그룹화, 네이밍)
- **API**: 완전한 OpenAPI 스펙 (Swagger)

→ 명세가 불완전하면 AI 효과 급감
→ 상위 단계 품질이 개발 단계 생산성 좌우

워크플로우를 설정 파일로 관리, 어떤 도구에서든 동일하게 실행

핵심 아이디어

워크플로우 = 설정 파일

- 워크플로우 정의를 코드처럼 버전 관리
- 팀/프로젝트별로 커스터마이징 가능
- 개선 사항이 전체 팀에 즉시 반영

MCP 기반 상호운용성

- Cursor, Claude Code, JetBrains 등 어떤 도구든
- MCP 클라이언트 지원 시 동일 워크플로우 실행
- 도구가 달라도 산출물 품질 일관

계층 상속 구조

→ 상위 필수 규칙은 하위에서 제거 불가
→ 하위에서 확장/오버라이드 가능

워크플로우를 YAML로 정의하고, 팀/프로젝트별로 관리

워크플로우 정의 예시

```yaml
# .ai/workflows/feature.yaml
name: feature-development
triggers: [기능, feature, 컴포넌트]
steps:
  - id: research
    type: human
    output: 00-plan.md
    checkpoint: true
  - id: plan
    type: prompt
    grounding: [rules/base.md]
    output: 10-task.md
    checkpoint: true
  - id: implement
    type: prompt
    output: 20-result.md
  - id: review
    type: prompt
    output: 30-review.md
    checkpoint: true
```

사용 사례

```
사용자: "UserProfile 컴포넌트 만들어줘"
→ triggers: [컴포넌트] 매칭
→ feature-development 워크플로우 실행
[research] 사람이 00-plan.md 작성
  ↓ checkpoint 승인
[plan] AI가 분석 → 10-task.md 생성
  ↓ checkpoint 승인
[implement] AI가 구현 → 20-result.md
  ↓
[review] AI가 리뷰 → 30-review.md
  ↓ checkpoint 승인 → 완료
```

핵심 포인트

- ✓ **triggers:** 자연어 → 워크플로우 자동 매칭(MCP Tool)
- ✓ **checkpoint:** 사람 승인 후 다음 단계
- ✓ **grounding:** 팀 규칙 + 프롬프트 자동 주입

파일럿 한계를 MCP 기반 아키텍처로 해결

파일럿 방식	MCP 기반 워크플로우
.cursor/, .claude/ 등 도구별 개별 설정 필요	단일 워크플로우 설정 → MCP Client로 동작하는 모든 도구에서 동일 실행
워크플로우 개선 시 도구별 각각 반영	설정 파일 1회 수정 → 전체 팀에 즉시 적용
팀/프로젝트별 설정 공유 어려움	계층 상속 (Global → Team → Project)

핵심: 워크플로우를 코드처럼 관리하고, 어떤 도구에서든 동일하게 실행

개인 최적화가 아닌, 팀 워크플로우의 점진적 · 지속적 발전

결론

> 파일럿을 토대로 팀 기반 바이브 코딩의 효과는 확인했지만
>
> 잘 정제된 자료(기획/디자인/API)와 전체 프로세스 병목 해소가 전제되어야 합니다.
>
> 다음 단계로 MCP 기반 상호운용 워크플로우를 통한 팀 표준화가 필요할 것으로 생각됩니다.

감사합니다