바이브 코딩을 이용한 소프트웨어 엔지니어링 레벨 업

발표자 소개

정도현

- ROBOCO.IO 수석 컨설턴트
- 1995년 12월 유니텔 사업부에서 프론트엔드 개발자로 시작
- AWS 시니어 개발자 및 트레이너로 8년 반 동안 재직
- 일본에서 14년동안 IT 컨설턴트 및 SI 개발자로 재직

바이브 코딩이란?

- **AI 지원 개발** – 자연어 지시로 코드 구현하는 새로운 개발 스타일 (Karpathy, 2025)
- **LLM과 페어 프로그래밍** – 개발자-AI 대화형 소프트웨어 개발
- **"느낌대로 코딩"** – 디테일한 구현은 AI가 담당하고 자연어로 완성된 모습을 정의 하는데 집중

왜 바이브 코딩인가? - 압도적인 생산성

- **고속 개발 사이클** – 아이디어 → 프로토타입 즉시 구현, MVP 개발 가속화
- **문제 해결 집중** – 구현보다 요구사항 정의와 솔루션 설계에 집중
- **빠른 반복 실험** – 저비용 아이디어 검증, 짧은 피드백 주기

왜 바이브 코딩의 생산성은 높은가?

- **소인원 대규모 개발이 가능해짐** - 기하급수적으로 증가하는 커뮤니케이션 비용을 억제 할 수 있음
- **코드베이스 인식 능력의 진화** - LLM으로 코드를 직접 탐색해서 읽은 다음 단계적으로 구현 계획을 세움
- **효과적인 지식관리** - 뛰어난 프로젝트관리, 문서화능력 (단, 현 단계에서는 사람이 오케스트레이션 할 수 있어야 함)

바이브 코딩 현황과 영향

- **급속한 확산** – Fortune 500 기업의 87%가 최소 하나 이상의 바이브 코딩 플랫폼을 사용 중
- **성과 지표** – 가트너 보고서에 따르면 개발자 생산성이 37% 증가하고, 신규 기능 출시 시간이 29% 단축되며, 코드 품질은 42% 향상되고, 개발자 만족도는 53% 상승됨
- **도구와 프로세스의 급속한 발전** - 모델과 도구, 프로세스 모두 업청나게 빠른 속도로 진화하고 있음

로보코에서 제안하는 바이브코딩 프로세스

`PRD` > `작업 분할` > `TDD구현` > `배포`

PRD 작성

> "AI가 여러분에게 무엇을 해 줄 것인가 묻지 말고, 여러분이 AI를 위해 무엇을 할 수 있는가를 물어 보십시오"

작업 분할

- PRD를 바탕으로 구현해야 할 작업들을 분할
- 작업은 가능한 한 작게 쪼개고, 독립적으로 구현/검증 가능해야 함
- 작업은 구현 뿐만 아니라, 문서화, 테스트 자동화, 리뷰, IaC, CI/CD, Ops 고도화 작업도 포함
- Github Issues, JIRA등을 사용한 작업 관리도 CLI도구를 사용해서 자동화 가능

TDD 구현

- PRD에 기반한 테스 케이스 작성 및 리뷰
- 작게 구현하고 자주 테스트
- 모듈이나 기능, 서비스 완성 시점에 통합테스트 및 E2E 테스트 작성

배포

- CI/CD 파이프라인을 이용한 자동 배포
- 배포 후 모니터링 및 운영 자동화
- 배포 후 피드백을 PRD에 반영

바이브 코딩과 소프트웨어 엔지니어링 레벨업

- 문서화, 테스트 자동화, 리뷰, IaC, CI/CD, Ops 고도화가 지원되지 않으면 바이브 코딩은 대량의 쓰레기 코드를 빠르게 생성해냄
- 바이브 코딩 도구들은 문서화, 테스트 자동화, 리뷰, IaC, CI/CD, Ops 고도화에 뛰어난 능력을 보임

> 바이브 코딩과 소프트웨어 엔지니어링 레벨업은 상호 보완적임!

바이브 코딩로 이룰 수 있는 엔지니어링 레벨 업

바이브 코딩으로 가능한 작업들

> 모두가 필요성을 인정하지만 잘 하지 않는 것들

1. 문서화
2. 테스트 자동화
3. IaC 코드 생성
4. CI/CD 파이프라인
5. Ops 자동화
6. 보안 자동화

문서화

바이브 코딩으로 개발하는 경우

- `PRD` > `Task 분할` > `TDD구현` > `배포`의 과정을 빠르게 빠르게 반복
- 이 과정에서 PRD가 항상 최신 설계를 반영함
- 테스크 완료 시점에 문서 최신화가 항상 진행 되어야 함

레거시 코드의 경우

- README.md를 포함해서 기존 문서들을 손쉽게 최신화
- 현재 소스코드 뿐만 아니라, 커밋이력, 이슈 트래커, Wiki도 콘텍스트 소스로 활용 가능
- AI 뿐만 아니라 신입/외부 협업자 온보딩 가속화에도 기여 가능
- 생성된 문서에 대해서 신경써서 리뷰를 진행해야 함
    - 코드나 기존 문서가 잘못된 경우가 있을 수 있음

테스트 자동화

테스트 자동화 구현 순서

- 테스트 자동화도 하나의 신규 프로젝트로서 다룬다
- 문서 최신화(요구사항 명확화) 부터 시작
- 정적 코드 체커 > UT > IT >E2E의 순서로 진행

AI를 활용한 코드 가독성 향상 - 정적 코드 체크 결과 자동 수정

- 정적 코드 체커의 규칙들은 코드 가독성과 밀접한 관련이 있음
- AI를 활용해서 정적 코드 체커의 지적사항을 자동으로 수정 수정

Git hooks을 사용한 빠르고 빈번한 테스트

- Git hooks기능을 이용해서 git 이벤트에 자동으로 검증이 이루어 질 수 있도록 설정 한다
  - pre-commit: Lint, Unit Test
  - pre-push: Integration Test, E2E Test
- husky등을 사용하면 git hook을 편리하게 관리 가능

IaC 코드 생성

- 기존 인프라 → **IaC(Infrastructure as Code)** 변환
  - Terraform / AWS CDK 코드 자동화
  - 실행코드와 마찬가지로 PRD>테스크분할>구현>검증 의 단계로 진행
- 장점:
  - 인프라 변경 이력 관리
  - 재현 가능한 환경 제공
  - 멀티 리전/클라우드 이식성 확보

CI/CD 파이프라인

- Vibe Coding으로 **CI/CD Workflow 자동 생성**
  - GitHub Actions, GitLab CI, AWS CodePipeline 등을 활용하여 구현
  - 실행코드와 마찬가지로 PRD>테스크분할>구현>검증 의 단계로 진행
- 기능:
  - 빌드/배포 파이프라인 자동화
  - 보안 스캔 & 품질 검사 포함
- 효과:
  - 릴리즈 주기 단축
  - 배포 실패율 감소

Ops 자동화

- **운영 지표 기반 자동화**:
  - 알람 규칙/대응 시나리오 자동 생성
  - CloudWatch, Prometheus, ELK 연동
- 예시:
  - CPU 초과 시 자동 스케일 아웃
  - 장애 발생 시 Slack/Teams 알림 & 이슈 생성

운영 자동화 스크립트

> AI가 정말 잘 하는 영역

- 자주 반복되는 운영 작업 스크립트 자동화:
  - 로그 분석 / 백업 / 배치 작업
  - 보안 패치 적용 스크립트
- 기대효과:
  - 운영 리소스 최적화
  - 휴먼 에러 감소

보안 자동화

- **AI 기반 보안 자동화**:
  - AI를 활용해서 정기적으로 보안 규정 준수 여부를 포함 각종 보안 리뷰를 실행 할 수 있음
  - 규정 준수여부 실시간 감시 가능
  - 잘 알려진 규정 준수 프레임 워크(OWASP top 10, ISO27001)를 활용
- 기대효과:
  - 실질적 보안 강화
  - 문서화와 같은 귀찮은 작업을 대부분 AI에게 떠넘길 수 있음

소프트웨어 엔지니어링 레벨업 수행 팁

- 문서화가 제일 중요
  - 문서는 AI와 사람의 `Single Source Of Truth`
- 모든 콘텍스트들을 AI가 쉽게 접근 가능한 마크다운 형태로 만들어서 소스 리포지토리와 함께, 또는 별도 문서 리포지토리를 만들어서 관리
- 공통적으로 참고되는 내용은 문서 리포지토리를 만들고 서브 모듈로 포함시킴
- 모든 작업은 기본적인 바이브 코딩 프로세스를 동일하게 지켜나갈것
  - `PRD작성` > `테스크 분할` > `테스트가 포함된 구현` > `리뷰/배포`

결론: 바이브 코딩과 소프트웨어 엔지니어링

- AI를 믿지 말고
- 사람을 믿지 말고
- 믿을 수 있는 프로세스를 만들어라!

앞으로의 과제 - 리뷰를 어떻게 자동화 할 것인가?

- 바이브 코딩에 익숙해지면 결국 사람에 의한 수동 리뷰가 최종적인 보틀넥으로 등장할 가능성이 높음
- 리뷰/검증을 어디까지/어느정도 수준으로 자동화 할 것인가가 바이브 코딩 레벨을 나누는 중요한 척도로 등장함
- 리뷰 자동화도 AI에 의한 자동화와 마찬가지임. 원칙을 세우고 이를 지킬 수 있도록 프롬프트 전략을 수립.

경청해 주셔서 감사합니다.