W04
테스트와 환경 구성

작성중인 코드의 동작을 확인하고 싶다면

AI에게 코드를 만들어달라고 했다. 코드가 나왔다. 이게 진짜 되는 건가?

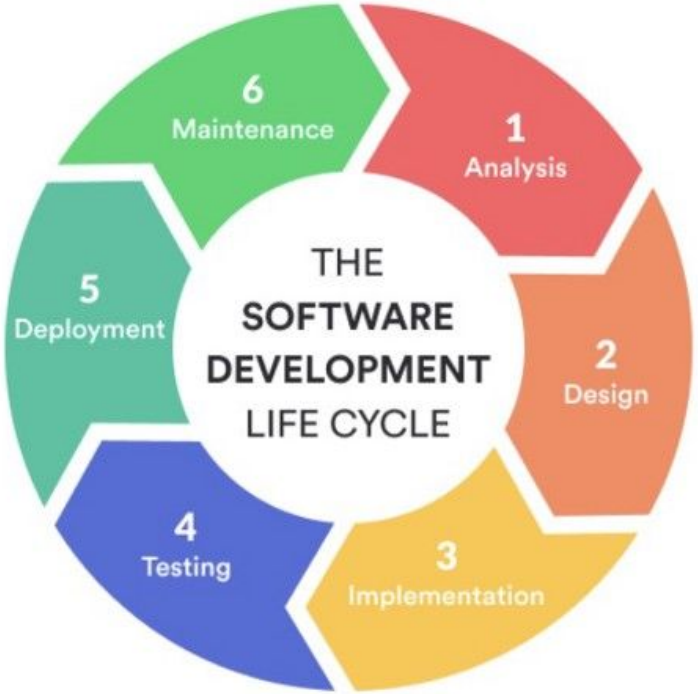
- 실행해보기 전까지는 모른다
- AI는 "될 것 같은" 코드를 만들어줄 뿐, 된다고 보장하지 않는다

확인 없이 다음으로 넘어가면?

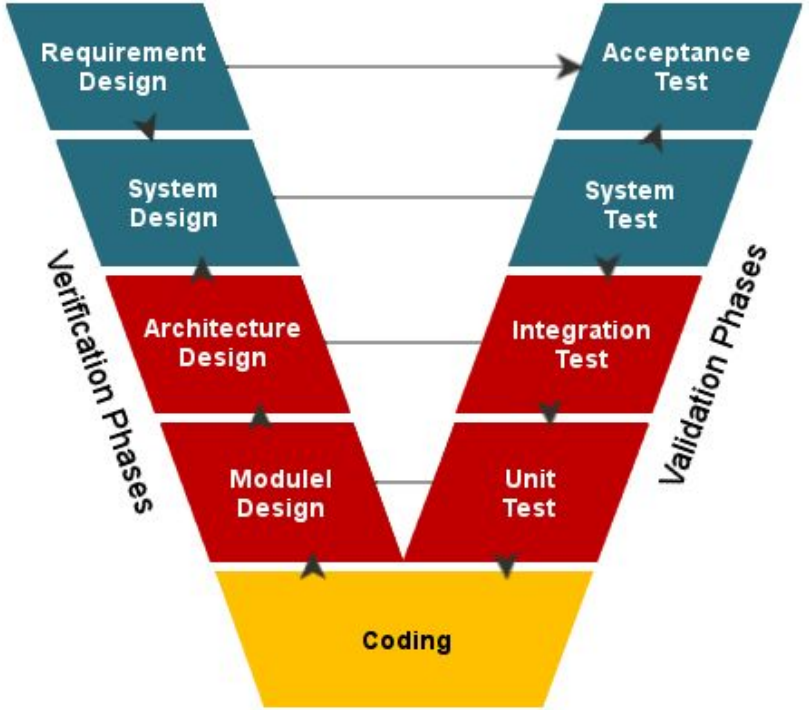
- 10개 만들고 나서 안 되는 걸 발견하면
- 10개 중 어디가 문제인지 찾아야 한다
- 1개 만들 때마다 확인했으면 바로 알았을 문제

그래서 만들 때마다 확인해야 한다 - 테스트

소프트웨어 개발 생애 주기



개발과 테스트의 이해 - V 모델



개발 환경 vs 실행 환경

개발 환경 (Development)

- 내 컴퓨터에서 코드를 작성하고 테스트하는 환경
- 에러가 나면 상세한 메시지가 보인다
- 테스트용 가짜 데이터를 사용한다
- 디버깅 도구가 켜져 있다

실행 환경 (Production)

- 실제 사용자가 서비스를 이용하는 환경
- 에러가 나면 "문제가 발생했습니다"만 보인다
- 실제 사용자 데이터가 흐른다
- 보안을 위해 디버깅 도구가 꺼져 있다

개발할때는 되는데 배포하면 안되는 이유

개발 환경과 배포 환경은 다르다

- 개발 환경: 코드를 작성하고 실행하기에 좋은 환경
- 배포 환경: 실제로 완성된 제품을 사용하는 환경

왜 구분해야 하는가?

- 개발 환경에서 잘 되던 코드가 실행 환경에서 안 될 수 있다
- 환경이 다르면 같은 코드도 다르게 동작한다
- AI도 이 차이를 모르면 "내 컴퓨터에선 됐는데요?"라고 대답한다

테스트의 목적에 따라 환경이 구성되어야 한다

- 유닛 테스트, 연동 테스트: 개발 환경에서도 가능
- 시스템 테스트, 인수 테스트: 배포 환경과 유사하게 구성해야

유닛 테스트

유닛 테스트(Unit Test): 하나의 기능이 혼자서 제대로 동작하는지 검증

- 시나리오 단위에서 진행 가능
- 하나의 시나리오 = 하나의 유닛 테스트
- Given-When-Then이 곧 테스트 케이스가 된다

예시: Scenario 1-1을 유닛 테스트로 변환

- Given: 등록된 사용자 존재
- When: 올바른 이메일/비밀번호로 로그인 요청
- Then: JWT 토큰 반환, 200 OK

→ "올바른 정보로 로그인하면 토큰이 반환되는가?"만 검증

연동 테스트

연동 테스트(Integration Test): 여러 기능이 함께 제대로 동작하는지 검증

- 워크 패키지 안의 시나리오들이 연결되어 동작하는지 확인
- 선행 의존성이 있는 시나리오들 사이의 연결 검증

예시: Work Package "사용자 로그인"의 연동 테스트

- Scenario 1-1 (로그인 성공) → Scenario 1-3 (토큰으로 API 호출)
- → "로그인해서 받은 토큰으로 다른 기능을 쓸 수 있는가?"
- 각각 테스트 통과해도 1-1에서 받은 토큰이 1-2에서 진짜 쓸 수 있는지는 별개 문제

워크 패키지 안에서 시나리오들이 이어지는지 확인

- 선행 의존성이 있는 시나리오들 사이의 연결 검증
- AI가 시나리오별로 따로 만들면 이 연결을 놓치기 쉽다

시스템 테스트

시스템 테스트(System Test): 전체 시스템이 요구사항대로 동작하는지 검증

- 마일스톤의 목표가 달성되었는지 확인
- 배포 환경과 유사한 환경에서 테스트해야 의미 있다

예시: 마일스톤 1 "사용자 인증 시스템 완성"의 시스템 테스트

- 회원가입 → 이메일 인증 → 로그인 → 토큰 발급 → API 호출
- 전체 흐름이 끊김 없이 동작하는가?

왜 시스템 테스트를 하는가?

- 마일스톤: "여기까지 되면 의미 있는 결과물"
- 시스템 테스트는 그 결과물이 진짜 되는지 확인

인수 테스트

인수 테스트(Acceptance Test): 사용자 관점에서 서비스가 목적을 달성하는지 검증

- PRD의 사용자 스토리가 실현되었는지 확인
- 기술적으로 동작하는 것과 사용자가 쓸 수 있는 것은 다르다

예시: 사용자 스토리 기반 인수 테스트

- "강사로서, 나는 학생의 GitHub URL을 입력해 코드 리뷰를 받고 싶다, 그래서 반복적인 리뷰 작업을 줄일 수 있다."
- 실제로 URL 입력 → 리뷰 생성 → 피드백 확인이 가능한가?

서비스 개요에서 정의한 "해결하려는 문제"가 진짜 해결되었는지 최종 확인

테스트 환경을 배포 환경과 동일하게 맞출 수 있을까

문제는 배포 환경을 내 컴퓨터에서 똑같이 만들기 어렵다

- 실제 서버는 돈이 든다
- 실제 사용자 데이터를 테스트에 쓸 수 없다
- 외부 서비스는 제한이 있다

이 과정에서의 현실적 접근

- 유닛 테스트, 연동 테스트: 시나리오, 워크 패키지 만들 때마다 한다
- 시스템 테스트: 가능한 범위에서 마일스톤 완료 전에 한다
- 인수 테스트: 배포 후 직접 써보면서 확인한다

구성하기 어려운 환경(1) - 로컬 환경으로 제한

웹 환경 배포

- 사용자는 브라우저(Chrome, Safari)로 접속한다
- 서버는 AWS, Vercel 같은 클라우드에서 돌아간다
- 내 컴퓨터가 아니라 인터넷 어딘가에 있는 컴퓨터에서 실행된다

모바일 배포

- 사용자는 앱스토어에서 앱을 다운받는다
- iPhone과 Android는 동작 방식이 다르다
- 내 컴퓨터에서는 실제 스마트폰 환경을 완벽히 재현할 수 없다

어플리케이션(설치 프로그램) 배포

- 사용자가 .exe나 .dmg 파일을 다운받아 설치한다
- Windows와 Mac은 동작 방식이 다르다
- 내가 개발중인 서비스를 실제로 설치해보지 않으면 확인이 어렵다

구성하기 어려운 환경(1) - 로컬 환경으로 제한

내가 만드는 서비스는 어디서 실행되나?

- 내 IDE에서 설치된 환경을 완벽히 테스트할 수 없다
- 내 컴퓨터에서 실제 서버 환경(메모리, 네트워크)을 똑같이 만들 수 없다
- 내 컴퓨터에서 아이폰/안드로이드 환경을 완벽히 재현할 수 없다

이 과정에서의 현실적 접근

- 로컬에서 할 수 있는 테스트를 최대한 철저히 한다
- 로컬에서 안 되는 건 배포 후 직접 확인한다
- 환경 차이로 생길 수 있는 문제를 미리 인식한다

구현하기 어려운 환경(2) - 서버 구성

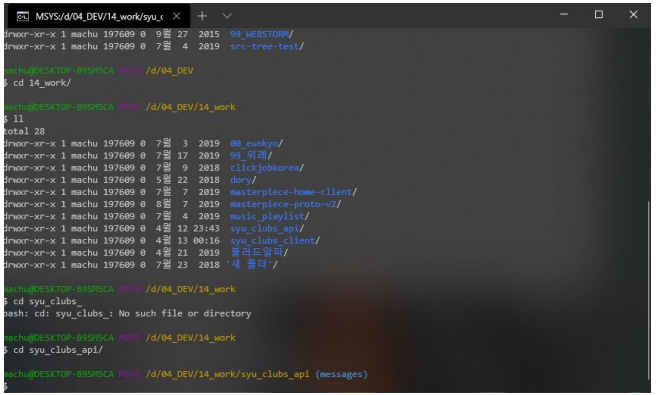
AI가 만든 코드 중에 로컬에서 테스트하기 어려운 것들

- 데이터베이스: 실제 서버 DB와 내 컴퓨터 DB는 설정이 다를 수 있다
- 외부 API: GitHub API, OpenAI API는 호출할 때마다 제한이 있고 비용이 발생한다
- 인증/보안: 실제 환경에서는 HTTPS가 필수인데 로컬에서는 보통 HTTP로 개발한다

어떻게 대응하나?

- 진짜 대신 가짜(Mock)를 써서 테스트한다
- "GAPI를 호출하면 이런 응답이 온다"고 가정하고 테스트
- AI에게 "Mock 데이터로 테스트 코드 작성해줘"라고 요청할 수 있다

테스트를 위한 지식(1) - 터미널



터미널이 뭔가?

- 컴퓨터에게 글자로 명령을 내리는 창
- 마우스로 클릭하는 대신 텍스트를 입력해서 컴퓨터를 조작한다
- Cursor IDE 하단에 있는 검은 화면이 터미널이다

왜 터미널을 써야 하나?

- 테스트 실행은 터미널에서 한다
- AI가 "이 명령어를 실행하세요"라고 하면 터미널에 입력해야 한다
- 테스트 결과(성공/실패)가 터미널에 출력된다

테스트를 위한 지식(2) - 디버깅

- 디버깅: 테스트가 실패했을 때 원인을 찾고 고치는 과정

디버깅 루프

1. 테스트 실행 → 실패
2. 에러 메시지 복사
3. AI에게 붙여넣기 + "이 에러 해결해줘"
4. AI가 수정한 코드 적용
5. 다시 테스트 실행
6. 통과할 때까지 반복

테스트를 위한 지식(2) - 디버깅

AI에게 에러를 전달하는 좋은 방법

- "아래 테스트가 실패했어. 원인을 찾아서 수정해줘.
- [에러 메시지 전체 복사]

왜 에러 메시지가 중요한가?

- 에러 메시지는 AI에게 주는 가장 정확한 힌트
- "안 돼요"보다 "Expected 400, Received 200"이 100배 유용하다
- AI도 에러 메시지 없이는 추측할 수밖에 없다

AI가 만든 테스트의 한계

1. 엣지 케이스

- "비밀번호가 빈 문자열이면?" "이메일에 한글이 있으면?"
- AI는 보통 정상 케이스 위주로 테스트를 만든다
- 사용자는 예상 못한 방식으로 서비스를 쓴다

2. 실제 사용 패턴

- AI는 기술적으로 가능한 것을 테스트한다
- 사용자가 실제로 어떻게 쓰는지는 모른다
- 예: 로그인 10번 연속 실패 시 계정 잠금

3. 비즈니스 로직 오류

- 코드는 맞는데 요구사항을 잘못 이해했을 수 있다
- 테스트가 통과해도 "원래 원했던 것"이 아닐 수 있다

테스트가 통과해도 확인해야 할 것

시나리오 의도대로인가?

- 코드는 돌아가는데, 내가 원했던 게 맞는지 확인
- AI가 요구사항을 잘못 이해했을 수 있다

빠진 케이스가 없는가?

- 성공하는 경우만 테스트하고 실패하는 경우는 빠졌는지
- 시나리오에 정의 안 한 상황은 테스트도 없다

직접 써봤는가?

- 테스트 코드가 통과하는 것과 직접 써보는 건 다르다
- 최종 확인은 결국 사람이 해야 한다

질문 1. 테스트 실행

테스크코드 작성

- “XXXX 시나리오에 대해 유닛 테스트를 위한 코드를 작성해줘. 코드는 XXXXXX 문서를 모두 참고해서 작성해.”

테스트 실행

- “작성한 코드에 맞게 XXXX 시나리오의 목적을 명확히 달성했는가를 테스트해줘. 테스트 후 문제가 있는 경우 이를 해결하기 위한 작업을 진행해줘"

질문 2. 디버깅

“실행 결과 다음과 같은 오류가 발생했어. 원인을 파악해줘"

“이 원인을 근본적으로 해결할 수 있는 방안에 대해 탐색하고 실제로 문제를 해결해줘"

“같은 문제가 계속 발생하고 있어. 너가 파악하지 못한 문제가 뭔지 근본적으로 파악해주고 이를 해결하는 방법을 제안해줘"

오늘 해결해야 할 문제

1. 앞서 작성한 로드맵의 첫 번째 시나리오를 Cursor에게 전달하여 코드를 생성합니다.
2. 생성된 코드에 대한 유닛 테스트 코드를 AI에게 요청합니다. 시나리오의 실제 내용을 확인하고 생성하게 합니다.
3. 터미널에서 테스트를 실행하고, 결과를 확인합니다.
4. 테스트가 실패하면 에러 메시지를 AI에게 전달하여 디버깅합니다. 통과할 때까지 반복합니다.
5. 테스트가 통과한 코드를 GitHub에 커밋합니다.
6. (선택) 단계별로 실행이 완료된 경우 연동 테스트, 시스템 테스트까지 위 과정을 확장합니다.

Q&A