프론트 엔드 개발자가 알아야 하는 컴퓨터 공학 지식

소프트웨어 공학(SW Engineering)

소프트웨어 공학 | 프론드 엔드 개발자가 알아야 하는 CS 지식

강사 나동빈

프론트 엔드 개발자가 알아야 하는 컴퓨터 공학 지식

소프트웨어 공학(SW Engineering)

리팩토링(Refactoring)

- 외부적으로 드러나는 기능을 변경하지 않는다. (결과의 변경 없이 코드를 조정)
- 소스 코드의 가독성과 유지 보수의 용이성을 높이기 위해 내부 구조를 변경한다.

→ 기존 기능은 유지하되 **개발 과정에서 편리한 코드**가 되도록 재작성하는 것이다.

→ 새로운 기능을 추가하거나, 버그를 제거하기 위한 목적은 아니다.

리팩토링의 필요성

- 대규모 프로젝트를 진행하기 위해 **유지보수의 용이성**이 요구된다.

→ 소프트웨어의 구조, 구현, 설계를 개선(깔끔하고, 단순하고, 표현력이 풍부한 코드)하면서 기능을 보존한다.

- **소스 코드의 가독성**이 떨어지는 경우, 다른 사람(혹은 미래의 나)과의 협업이 어려울 수 있다.
- 리팩토링을 통해 코드의 중복을 제거하고, 수정이 편리해질 수 있다.

→ 코드의 품질이 좋아 수정하기 용이하다면, 궁극적으로는 **개발 속도가 개선**된다.

리팩토링을 해야 할 때

- **유사한 코드가 세 번 이상 반복**될 때, 리팩토링을 고려하자.

→ 이 경우에도 무조건적인 리팩토링이 아닌, 개발 일정 및 계획을 감안해야 한다.

- **코드 리뷰**를 통해 다른 사람에게 코드를 공유할 때, 리팩토링을 고려하자.

→ 코드 리뷰를 통해 코드의 질을 높일 수 있지만, 너무 많은 인원이 참여하지 않는 것을 권장한다.

- **새로운 기능을 추가**할 때 리팩토링을 고려하자.

→ 현재 코드를 유지한 상태로 새 기능을 추가하는 게 어렵다면, 리팩토링을 고려해 보자.

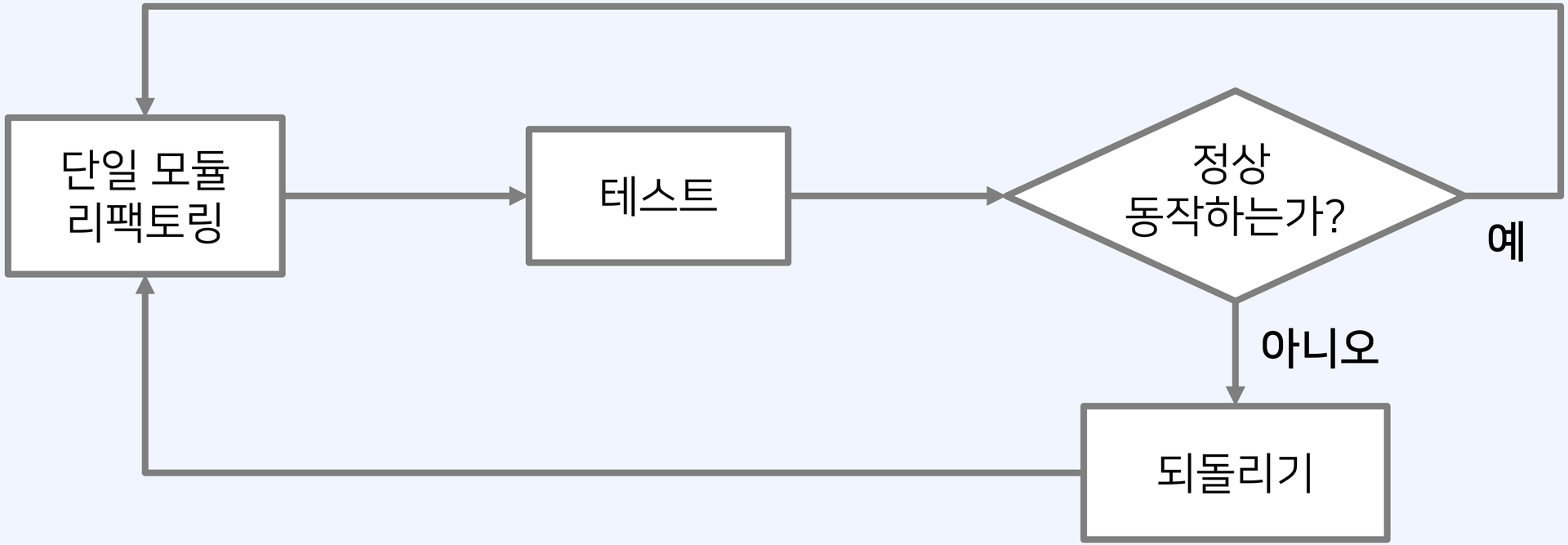
- **버그를 수정**해야 할 때 리팩토링을 고려하자.

리팩토링을 하지 말아야 할 때

- 현재의 코드가 정상 동작조차 하지 않는 상태라면, 리팩토링보다는 **재작성**이 필요하다.
- 개발 일정 마감이 다가 온 상황에서는 가급적 리팩토링을 피한다.

리팩토링의 과정

- **기능을 유지**하면서, 소스코드의 내부 구조를 변경한다.



리팩토링(Refactoring): Two Hats

- Two Hats 원칙을 지키면서 리팩토링을 진행하는 것이 권장된다.

1. **기능을 추가할 때**: 기존 코드를 수정하지 말고, 기능 및 테스트만 추가한다.
2. **리팩토링 할 때**: 기능을 추가하지 말고, 기존 동작을 유지한 상태로 내부 구조만 변경한다.

→ 추가 오류를 최소화하기 위하여, 리팩토링에만 집중할 수 있도록 한다.

성능 향상과 리팩토링

- 개발 과정에서는 소프트웨어 자체의 성능을 높이기 위한 고민을 끊임없이 해야 한다.
- 리팩토링을 하게 되면 단기적으로는 소프트웨어 속도가 더 느려질 수 있다.
- 하지만 주요 함수의 성능을 최적화하는 과정까지 고려하면, 리팩토링이 소프트웨어 튜닝을 더 쉽게 만들어 줄 수 있다.

→ 이러한 과정에서 프로파일러를 사용해 병목 현상 발생 부분이 어디인지 확인할 필요가 있다.

클린 코드와 리팩토링

- 코드 리팩토링은 **클린 코드**를 포함한 전반적인 유지 보수의 용이성을 목표로 한다.

→ 따라서, 클린 코드의 기능을 리팩토링이 포함한다.

- **클린 코드**는 기본적인 코드 작성 과정에서부터 잘 이루어지는 것이 좋다.