4장. 개선을 위한 반복

4장: 개선을 위한 반복 (Iteration for Improvement)

1. 반복의 정의와 본질

- **반복**은 **"일련의 연산을 반복해 성공적으로 원하는 결과를 얻기 위한 절차"**로 정의됩니다.
- 반복은 근본적으로 **학습(learning)**을 이끄는 절차입니다. 이를 통해 학습하고, 반응하며, 학습한 내용을 조정하여 발전하고 개선할 수 있습니다.
- 반복의 진정한 위력은 **목표에 어떻게 접근할지 모를 때조차도** 점진적으로 목표에 접근하게 만든다는 데 있습니다.

2. 반복과 폭포수 모델의 비교

- **전통적인 폭포수 사고방식:** 폭포수 모델은 프로젝트 수명 주기 초반에 가장 중요한 결정을 내리는 것이 합리적이라고 가정합니다. 이는 시간이 지날수록 변화에 드는 비용이 급증한다는 가정에 기초합니다.
- **소프트웨어 개발의 현실:** 소프트웨어 개발은 **탐구와 발견의 과정**이므로, 일이 불가피하게 잘못될 것이라는 가정에서 출발하는 애자일 사고(반복적 접근 방식)가 더 적합합니다.
- **목표: 변화 비용 곡선 평탄화:** 반복적인 접근 방식의 목표는 변화 비용 곡선을 평평하게 만드는 것입니다. 이렇게 되면 새로운 아이디어를 발견하고 오류를 수정하는 비용이 거의 동일하게 유지되어, **제품과 코드를 지속적으로 개선할 자유**를 얻게 됩니다.
- 세계 최고의 소프트웨어 회사에서 아이디어의 3분의 2가 0이나 음의 가치를 창출한다는 업계 데이터에 근거하여, **가장 효과적인 접근 방식은 반복**입니다. 즉, 아이디어 중 일부가 잘못될 것임을 받아들이고 최대한 **빠르고 저렴하며 효율적으로 시도**해 보게끔 작업하는 것입니다.

3. 반복적인 작업의 실천 방안과 장점

- **작은 단위로 작업:** 반복적으로 작업하려면 **작은 단위**로 작업해야 합니다. 작은 단계는 상황 변화나 오해로 인해 무효화되더라도 업무 손실이 적습니다.
- **설계 품질 강화:** 더 반복적으로 작업하면 자동으로 초점이 좁아지고, 더 작은 배치 단위로 생각하며, **모듈성**과 **관심사 분리**를 더 중요하게 생각하게 됩니다.
- **빠른 피드백 제공:** 작고 명확하며 서비스에 바로 투입이 가능한 단계로 진행하는 반복적인 작업은 정기적으로 작업의 품질과 적절성에 대한 명확한 피드백을 제공하여 **학습 속도를 높여줍니다**.
- **점진적인 설계 (Incremental Design):** 복잡한 시스템은 어떤 천재적인 창조자의 머리에서 온전히 만들어지지 않으며, 점진적으로 진전을 이루면서 배우는 것이 이 책의 핵심 아이디어입니다.
- **구체적인 반복 기술:**

◦ **지속적인 통합(CI)**: 변경사항을 자주 커밋하여 하루에도 여러 차례 수행함으로써 작업이 다른 사람의 코드와 함께 동작하는지 이해할 기회를 제공합니다.

◦ **테스트 주도 개발(TDD)**: 실패(Red), 성공(Green), 리팩토링(Refactor)의 단계를 거치는 **매우 세분화되고 반복적인 접근 방식**을 권장합니다. TDD는 코드 작성의 근본 기술에 대해 상당한 반복적 접근 방식을 적용하는 것입니다.

결론적으로, 반복적인 작업은 **변화하는 상황을 위한 유일하게 효과적인 전략**이며, 학습, 발견, 더 나은 소프트웨어 제품에 대한 통제된 접근 방식을 향해 나아가는 능력의 토대가 됩니다.