5장. 피드백: 우수한 의사결정을 위한 필수요소

피드백

- 행위, 이벤트, 프로세스에 대한 평가 또는 교정 정보를 원본 또는 통제 중인 출발점으로 전송하는 것
- 생각보다 많은 사람과 조직이 피드백에 관심이 없다.
- 피드백은 의사결정을 위한 근거의 원천을 확보하게 도와준다.

피드백의 중요성을 보여주는 구체적 사례

빗자루의 균형을 잡는 문제에 직면했다고 가정해 보자.

빗자루의 구조를 주의 깊게 분석하고, 무게중심을 파악하고, 손잡이의 구조를 면밀히 살펴보고, 빗자루가 완벽하게 균형을 잡을 지점을 정밀하게 계산하기로 결정할 수 있다.

- 폭포수 개발 모델과 유사하다. 이 모델은 완벽한 예측에 의존하고 있으며, 조금이라도 작은 변화가 생기면 비질은 실패한다.

빗자루를 손에 쥐고 기울어지는 방향에 따라 손을 움직인다.

- 피드백에 기반한다. 이 방식은 설정이 더 빠르며, 피드백의 속도와 품질이 성공을 좌우한다.
- 이러한 접근방식은 너무나 성공적이어서 우주 로켓이 엔진 추력의 '균형'을 잡는 방법이 되었다.

피드백은 변화하는 환경에서 운영하는 모든 시스템의 필수 구성 요소다.

소프트웨어 개발은 끊임없는 학습의 과정이며, 개발이 진행되는 환경은 항상 변화하고
있다. 따라서 피드백은 효율적인 소프트웨어 개발 프로세스의 필수 요소다.

코딩 피드백

- TDD에 대한 소개를 함, 피드백 주기는 매우 짧아야 마땅합니다.
- 빠른 피드백 주기가 유용한 이유는 작업 중인 내용에 대한 관련성을 즉시 빠르게 파악할 수 있기 때문입니다.

통합 과정 피드백

- 코드 커밋 시 지속적 통합 시스템을 가동해 다른 모든 사람의 컨텍스트에서 내가 변경한 사안을 평가할 것입니다.
- CI 와 FB(feature Branch) 에 대한 내용이 나오며 CI 와 FB 간 코드 병합의 문제는 항상 발생할 수 있음을 말합니다.
- 또한 CI 를 통한 잦은 피드백을 받으려면 그만큼 자주 커밋해줘야 합니다.
- 이런 접근방식은 소프트웨어를 항상 릴리즈 가능하게 만들고 우리가 품질과 작업 적용 가능성에 대해 더 자주 세분화된 피드백을 얻을 수 있게 됩니다.

설계 피드백

- TDD 얘기를 하며 TDD 에서 테스트가 가능한 코드는 결국 아래의 사항을 따르게 된다고 합니다.
  - 모듈식
  - 관심사 분리
  - 응집성 높음 (하나의 모듈은 한 가지 일만 한다, 여러 일을 할 수록 응집성은 낮다)
  - 추상화 사용

    - 적절한 결합

아키텍처 피드백

피드백 주도 접근 방식을 적용하는 것은 구축하는 시스템의 광범위한 소프트웨어 **아키텍처에 미묘한 영향**을 미칩니다.

- 지속적인 배포(Continuous Deployment)는 개발 과정에서 고성능의 **피드백 주도 접근 방식**이며, 이의 초석 중 하나는 항상 프로덕션 환경에 투입할 수 있는 릴리스 준비가 된 소프트웨어를 제작해야 한다는 아이디어입니다.
- 이를 달성하려면 **높은 빈도와 품질의 피드백**이 필요하며, 시스템의*테스트 가능성(testability)과 **배포 가능성(deployability)** 같은 아키텍처적 특성을 진지하게 고려해야 합니다.
- 피드백에 걸리는 시간을 제한하는 것은 시스템을 배포하고 실행하는 데 걸리는 시간과 가장 느린 테스트 케이스를 실행하는 데 걸리는 시간에 달려있습니다.
- 더 쉽게 **테스트 가능하고** 더 쉽게 **배포 가능한 설계 방안**을 선택함으로써 더 짧은 시간 동안 더 효과적으로 피드백을 수집할 수 있습니다.
- 시스템을 더 작고 독립적인 모듈(예: **마이크로서비스**)로 분해하든, 아니면 효율성을 위해 하나의 긴밀하게 결합된 코드 기반(예: 모놀리스)을 개발하든, **지속적인 배포를 채택**하면 더 **모듈화되고, 더 추상화되며, 더 느슨하게 결합된 설계**를 촉진하게 됩니다.
- 이는 개발 접근 방식에서 피드백의 가치를 인정하고 우선순위를 높일 때 더 합리적이고 효과적인 아키텍처 의사결정을 촉진하게 됨을 의미합니다.

피드백은 빠를수록 좋다

일반적으로 가능한 한 초기에 **확실한 피드백을 받으려고 노력하는 편이 효과적**입니다.

- 가장 빠르고 저렴하며 가치 있는 피드백 루프는 코딩할 때 타이핑하는 동안 개발 도구(IDE)를 사용해 코드에서 오류를 강조하는 것입니다. 타입 시스템 같은 기술로 정확도를 높여 오류 피드백을 이용할 수 있습니다.
- **테스트 주도 개발(TDD)** 접근 방식은 자동화된 단위 테스트를 생성하며, 이는 로컬 개발 환경에서 정기적으로 실행되어 **매우 빠르게 피드백**을 받을 수 있게 합니다(보통 몇 초 또는 밀리초 단위).
- 이처럼 짧고 빠른 피드백 주기는 작업 중인 내용에 대한 관련성을 즉시 파악할 수 있도록 하여 믿기 어려울 만큼 유용합니다.
- 개발 프로세스에서는 **가장 빨리 실패하는 것**을 선호하며, 컴파일러 수준에서 결함을 식별한 다음 단위 테스트에서 식별하고, 이후 상위 수준 테스트에서 결함을 식별하는 방식을 선호합니다.
- 이러한 조기 실패 선호('왼쪽으로 자리 옮김' 또는 '빨리 실패하라!')는 고품질 소프트웨어와 결과물을 위한 필수 요소로 간주됩니다.

조직과 문화 피드백

소프트웨어 개발의 성과 측정은 오랫동안 어려웠으며, 과거에는 성공과 현실적인 상관관계가 없는 지표(예: 코드 행 수, 테스트 커버리지)에 의존하는 경우가 많았습니다.

- **애자일 개발**은 팀과 조직이 피드백 루프에 참여하여 행동의 결과를 관찰하고, 반성하며, 시간이 지남에 따라 상황을 개선하기 위해 선택을 구체화하는 주관적인 피드백 중심 접근 방식입니다. 이는 애자일의 가장 근본적인 아이디어인 '점검하고 적응한다' 의 기반을 닦았습니다.
- 더 나은 접근 방식은 과학적 방법론을 경량으로 적용하는 것입니다. 즉, 현재 상태와 목표 상태를 명확히 하고, 올바른 방향으로 나아가기 위한 단계를 기술하며, 목표에 더 가까워졌는지 아닌지를 판단하는 방식을 결정하고 반복하는 것입니다.
- DORA(DevOps Research and Assessment) 연구와 같은 최근의 연구는 조직 및 문화적 변화를 평가하는 데 유용한 구체적이고 덜 주관적인 측정 기준을 제시합니다.
- 이 연구에서는 안정성(Stability)과 처리량(Throughput)이라는 두 가지 핵심 속성을 기반으로 소프트웨어 개발 팀의 효율성을 평가합니다.
- 이러한 측정 결과는 우리의 노력을 더 나은 결과로 이끄는 '적합성 함수(fitness function)'로서 활용될 수 있습니다. 조직은 프로세스, 기술, 조직, 문화에서 변화를 만들 때 안정성과 처리량 관점에서 점수를 추적함으로써 변화가 실제로 유익한지 확인할 수 있습니다

정리

- **피드백은 우리의 학습 능력에 필수적인 요소**입니다. 빠르고 효과적인 피드백이 없다면 추측에만 의존하게 됩니다.

- **피드백의 속도와 품질** 모두 중요하며, 너무 늦거나 잘못된 피드백은 그 가치를 잃습니다.
- **지속적인 배포**와 **지속적인 통합**은 개발 프로세스를 최적화하여 수집하는 **피드백의 품질과 속도를 극대화**한다는 아이디어에 기반을 두고 있습니다