제5장 피드백: 우수한 의사결정을 위한 필수 요소

- 피드백은 행위, 이벤트, 프로세스에 대한 평가 또는 교정 정보를 원본 또는 통제 중인 출발점으로 전송하는 것 피드백이 없다면 학습할 기회도 없다. 현실에 기반한 결정을 내리는 대신 추측만 가능하다.

선택과 행위의 결과를 알고 이해할 수 없다면, 발전하고 있는지도 알 수 없다.

피드백은 의사결정을 위한 근거의 원천을 확보하게 도와준다. 일단 이런 근거의 원천을 확보하면, 의사결정의 품질은 필연적으로 향상된다

5.1 피드백의 중요성을 보여주는 구체적인 사례

- 빗자루의 균형을 잡는 문제에 직면했다고 가정해 보자.

첫 번째 접근 방식은 폭포수 개발 모델과 유사하다. 동작은 하지만 실제로 동작할 가능성은 높지 않다고 상상할 수 있다. 이 모델은 완벽한 예측에 의존하고 있으며, 조금이라도 작은 변화가 생기거나 예측이 부정확하면
비질은 실패한다.

두 번째 접근 방식은 피드백에 기반한다. 이 방식은 설정이 더 빠르며, 피드백의 속도와 품질이 성공을 좌우한다. 피드백이 빠르고 효과적이면, 교정을 작게 할 수 있으며 비질이 안정적일 것이다

빗자루가 소프트웨어를 비교한 이유는 핵심 요지는 프로세스가 작동하는 방식에는 깊고 중요한 무언가가 있다는 점이다

첫 번째 경우는 계획적이고 예언적인 접근 방식이다. 이런 접근 방식은 모든 변수를 완벽하게 이해하고 이해나 계획을 변경할 만한 상황이 발생하지 않는 이상 잘 동작한다

대안적인 두 번째 접근 방식은 여전히 '빗자루의 균형을 잡을 것이다'라는 계획을 수반하지만, 계획은 균형을 달성하기 위한 메커니즘에 대해 아무 말도 하지 않는다.

두 번째 접근 방식이 임시방편적 이고 즉흥적으로 보일지는 몰라도 실제로는 더 효과적이고 더 안정적인 결과를 얻을 수 있다

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.

5.2 코딩 피드백

- 저자의 코딩 피드백 : 일반적으로 코드를 작성하기 위해 테스트 주도 접근 방식을 활용한다

테스트를 작성하고, 테스트가 실패하는지 보기 위해 실행한다. 실패의 특성은 직접 만든 테스트가 올바른지 이해하는 과정에 도움을 주는 피드백을 제공한다

나(저자)는 일련의 작은 단계로 코드를 변경한다

도움을 받기 위해 IDE에 탑재된 리팩토링 도구를 많이 사용하지만, 코드가 동작하는지 여부와 설계가 발전함에 따라 모양새가 내 마음에 드는지 단계별로 피드백을 받는다. 내가 코드를 변경할 때마다 현재 작업 중인 테스트를 다시 실행함과 동시에 피드백을 통해 변경 이후에도 코드가 계속 동작한다는 사실을 매우 빠르게 확인할 수 있다

작업을 일련의 작은 단계로 구조화하면 진행 상황을 반영하고 설계를 더 나은 결과물로 조정하기 위한 더 많은 기회를 제공한다.

5.3 통합 과정 피드백

- 지속적인 통합은 여전히 널리 오해를 불러일으키고 제대로 실행되지 못하는 개념이다. 소프트웨어 개발에 지적으로 엄격한 접근 방식을 확립하려면,아이디어의 장단점을 냉정하게 평가하기 위한 공학적인 접근 방식이 중요하다. 더 바람직해서라기보다는 기분을 더 좋게 만든다는 이유만으로 아이디어가 채택되는 경우가 많다

이에 대한 좋은 예가 지속적인 통합과 기능 브랜치 실무자 사이의 논쟁이다.

지속적인 통합은 시스템에 변화를 주는 다른 모든 변경사항과 함께 가능한 한 자주 최대한 '지속적으로' 시스템에 일어나는 변경을 평가하는 것이다

*CI는 모든 개발자의 작업 사본을 하루에 여러 차례 공유된 간선 main 브랜치 에 통합하는 관례다. 가 정의지만 CI는 하루에 적어도 한 번 평가로 타협 한다고 한다

"완료"라고 간주될 때까지 코드의 일부를 작업해서 전체로 병합한다. 이 시점에서 예상치 못한 모든 문제가 식별되어 병합 작업이 복잡해지고, 완료까지 예측하기 힘든 오랜 시간이 걸리는 상황이 벌어지는데 문제를 해결하기 위해 두 가지 접근 방식이 채택되었다. 하나는 CI였으며, 다른 하나는 병합 도구의 품질을 개선하는 것이 었다

지속적인 통합과 이의 큰형인 지속적인 배포는 작은 단계로 변경하고 모든 작은 단계 이후에 사용할
준비를 갖추도록 요구한다. 이는 몇 가지 중요한 방식으로 시스템 설계에 대한 우리의 사고 방식을 바꾼다 이런 접근 방식은 소프트웨어를 항상 릴리스 가능하게 만들며, 우리가 품질과

작업 적용 가능성에 대해 더 자주 더 세분화된 피드백을 얻을 수 있을 뿐만 아니라, 이런 접근 방식을 유지하는 방식으로 작업을 설계하게끔 권장한다.

5.4 설계 피드백

- 간단하고 효과적인 테스트를 만드는 능력과 설계의 효과는 우리가 '좋은' 코드 에서 중요하다고 고려하는 품질의 속성과 관련이 있다

다음 목록은 코드 품질에 대한 전형적인 특징으로 개발자들 사이에서 중요하다고 생각하는 목록이다

- 모듈성
- 관심사 분리
- 높은 응집성
- 정보 은닉（추상화）
- 적절한 결합도

이런 속성을 기반으로 '품질'을 어떻게 코드에 담을 수 있을까? TDD가 없다면, 이는 개발자의 경험과 헌신과 기술에 전적으로 달려 있다

TDD의 정의에 따르면 테스트를 먼저 작성한다 테스트를 먼저 작성한다면 삶을 더 쉽게 만드는 방법으로 이를 시도할 것이다

테스트가 가능한 코드는 어떤 모습일까?
- 모듈식이다
- 관심사 분리가잘되 어 있다
- 응집성이 높다
- 정보 은닉 (추상화)을 사용한다
- 적절히 결합되어 있다

TDD는 객관적으로 '더 높은 품질'의 코드를 작성하도록 압력을 가한다

TDD와 테스트 주도 접근 방식의 다른 측면은 우리가 만드는 코드 품질에 중요한 영향을 미친다.

5.5 아키텍처 피드백

지속적인 배포는 개발 과정에서 고성능이자 피드백 주도 접근 방식이다.

지속적인 배포의 초석 중 하나는 항상 프로덕션 환경에 투입할 수 있는 릴리스가 준비된 소프트웨어를 제작해야 마땅하다는 아이디어다. 이는 아주 높은 기준이며, 높은 빈도와 품질의 피드백을 요구한다.

이를 달성하려면, 표면에 드러나는 두 가지 측면은 우리가 구축하는 시스템의 아키텍처적인 특성일 수도 있다 우리는 시스템의 1)테스트 가능성과 2)배포 가능성을 진지하게 고려 할 필요가 있다

마이크로서비스 아키텍처 접근 방식은 팀이 독립적으로 서비스를 개발하고 테스트하고 배포하게 만든다

5.6 피드백은 빠를수록 좋다

일반적으로 가능한 한 초기에 확실한 피드백을 받으려고 노력하는 편이 효과적이다.

모든 테스트를 수행하면 작업의 품질과 적용 가능성에 대한 확신을 높일 수 있지만 그 대가로 결과를 얻기 위해 더 많은 시간이 걸린다. 따라서 먼저 (개발 환경의) 컴파일러 수준에서 결함을 식별한 다음 단위 테스트
에서 식별하고, 두 가지 형태의 검증이 성공한 다음에 다른 형태의 상위 수준 테스트에서 결함을 식별하는 방식을 선호한다.

이는 가장 빨리 실패하고, 최고의 품질을 얻고, 가장 효과적인 피드백을 얻을 수 있음을 의미한다

5.7 제품 설계 피드백

시스템의 품질에 대한 피드백을 받아들이는 관례의 영향력은 중요하지만, 소프트웨어 개발자들은 멋지게 설계 되고 쉽게 테스트할 수 있는 소프트웨어를 만들기 위해 월급을 받는 것이 아니라 회사나 조직을 위해 가치를 창출하기 위해 월급을 받는다. 이는 비즈니스에 집중하는 직원과 기술에 집중하는 직원 간의 관계에서 종
종 발생하는 갈등 중 하나이다.

이 문제는 '유용한 아이디어를 지속적으로 상용 환경에 배포'할 수 있도록 하는데 초점을 맞추면 해소될 수 있다.

아이디어의 소비자(사용자 또는 고객)로 부터 피드백을 받을 때 해결이 될 수 있다.

제품 아이디어 창출에 대한 피드백 루프를 닫고 상용 서비스에 가치를 제공하는 것이 바로 지속적인 배포의 진정한 가치다

5.8 조직과 문화 피드백

소프트웨어 개발의 측정 가능성은 성공을 어떻게 측정하고 개선을 어떻게 측정할 수 있을까? 우리가 만든 변화가 효과적인지 아닌지를 어떻게 알 수 있을까?

첫 번째는 에자일 개발 부문에서 한동안 확립되어 왔던 방식이다. 주관성을 완화하기 위해 합리적인 규율을 채택하려고 노력한다. 이런 접근법의 성공 여부는 필연적으로 참여하는 개인과 밀접한 관련이 있다. “프로세스와 도구보다 개인과의 상호작용” 이런 전략은 역사적으로 소프트웨어 개발에 대한 공식적이고 거창한 격식을 갖춘 접근 방식에서 벗어나는 데 중요한 역할을 했으며, 여전히 기본 원칙으로 중요하게 남아 있다.
에자일 개발 접근 방식은 팀과 ‘업무에 참여하는’ 사람들을 피드백 루프에 참여시킴으로써 사람들이 자신의 행동 결과를 관찰하고, 반성하며, 시간이 지남에 따라 상황을 개선하기 위해 선택을 구체화할 수 있도록 만들었다.

두 번째로, 피드백의 질을 높이기 위해 피드백에 대한 이런 주관적인 접근 방식에 더해 약간의 개선 목적으로 피드백의 성격을 구체화한다. 단계를 수행하면서 목표에 더 가까워졌는지 아니면 더 멀어졌는지를 확인하고 목표에 도달할 때까지 반복한다

정리:

제5장 피드백: 우수한 의사결정을 위한 필수 요소 (1)