

제8장 실험주의 : 과학적 사고와 실천

- 문제 해결을 위해 실험적인 접근 방식의 선택은 매우 중요하다

소프트웨어 개발은 의식적으로 사용자가 무엇을 좋아할지 추측하는 기술 연습으로 진행된다.

8.1 물리학자 파인먼에게 배우는 실험주의

- “뭐가 되었든 권위를 존중하지 말고, 누가 말했는지 잊어버리고 그 대신 그가 무엇으로 시작해서 어디서 끝나는지 살펴보고 “그것이 합리적인가?”라고 자문해보자.” -물리학자 파인먼-

우리는 리처드 파인먼처럼 가장 중요하거나 카리스마가 있거나 유명한 사람의 말에 따라 의사결정을 내리는 것에서 벗어나 증거에 기반한 의사결정과 선택을 해야만 한다.

C#보다 클로저가 더 낫다는 주장을 하고 싶다면 누가 더 논쟁 과정에서 설득력이 있느냐에 따라 이런 결정을 내리는 대신, 완벽하지는 않더라도 어느 정도 근거에 기반해서 결정할 수 있다. 그 결과에 동의하지 않는다면 더 나은 실험을 통해 근거를 제시하자

실험주의자가 되는 것이 모든 결정의 근거를 엄격한 물리학에 두는 것을 의미하지는 않는다.

'실험주의자가 되는' 4가지 특징

- 피드백: 우리는 피드백을 진지하게 받아들여야 하며, 명확한 신호를 제공할 결과를 수집해 우리가 생각하고 있는 지점에 이 결과를 효율적으로 전달하는 방법을 이해할 필요가 있다. 루프를 닫을 필요가 있다.
- 가설: 평가하고자 하는 아이디어를 염두에 둘 필요가 있다. 우리는 무작위로 데이터를 수집하느라 방황하고 있지 않다. 데이터 수집 행위만으로는 충분하지 않다.
- 측정: 가설에서 테스트하려는 예측을 평가하는 방법에 대한 명확한 아이디어가 필요하다. 이 컨텍스트에서 '성공' 또는 '실패'는 무엇을 의미할까?
- 변수 통제: 실험이 우리에게 보내는 신호를 이해할 수 있도록 가능한 한 많은 변수를 제거할 필요가 있다.

8.2 피드백: 실험주의를 위한 원칙1

- 공학적인 관점에서 피드백의 효율성과 품질을 가속화할 때 얻을 수 있는 효과를 인식하는 것이 중요하다.

'질주 본능' 상자의 '전쟁 이야기'는 실험 기법을 작업에 적용하고 좋은 피드백을 위해 최적화하는 방식의 효율성을 보여준다.

8.3 가설: 실험주의 위한 원칙2

- 과학과 공학에 대해 이야기할 때 사람들은 종종 '추측을 배제한다'고 이야기 하지만 이는 잘못된 표현이다

한 가지 중요한 의미에서 과학은 추측에 기반을 두고 있으며, 문제 해결에 대한 과학적 접근 방식은 추측을 제도화하고 이를 '가설'이라고 부른다.

추측이나 가설은 출발점이다. 다른 덜 효과적인 접근 방식과 비교해 과학과 공학의 차이점은 출발점에서 멈추지 않는다는 사실이다

가설은 테스트할 필요가 있다. 테스트는 다양한 형태를 취할 수 있다. 현실 (서비스나 제품)을 관찰할 수도 있다.

가설을 검증하기 위한 일련의 실험을 진행하기 위해 우리의 생각과 작업을 구조화하는 것은 작업의 질을 향상하는 중요한 요소다.

8.4 측정: 실험주의 위한 원칙3

- 측정을 진지하게 받아들일 필요가 있다. 수집한 데이터가 무엇을 의미하는지 생각하고 비판적인 태도를 견지해야 한다

컨텍스트가 무엇을 의미하든 시스템 측정에 더 많은 주의를 기울일 것을 실험주의자에게 요구한다.

8.5 변수 통제: 실험주의 위한 원칙4

- 피드백을 수집하고 유용한 측정을 하기 위해서는 현실적으로 최대한 변수를 통제 할 필요가 있다 "변수를 제어해 릴리스를 안정적으로 만들자"

버전 관리는 상용 환경에 릴리스하는 변경사항을 훨씬 더 정확하게 제어하도록 만든다.

소프트웨어 개발의 많은 기술적인 변수를 통제함으로써 지속적인 배포는 이전 보다 훨씬 더 자신 있게 진전을 이루도록 만든다. 따라서 소프트웨어 개발 팀은 이 책의 핵심인 학습을 위한 최적화 기술을 실제로 활용할 수 있게 된다

소프트웨어가 항상 릴리스 가능한 상태가 되도록 작업 한다는 지속적인 배포의 핵심 아이디어는 작업 품질에 대한 피드백을 극대화하고 작은 단계로 작업하도록 강력히 권장하는 아이디어다

8.6 TDD에서 배우는 자동화 테스트

- 코드의 예상 동작에 대해 작은 예측을 하고 이를 통해 소프트웨어의 기능을 점진적으로 향상할 수 있는 일련의 반복적인 실험을 중심으로 개발을 구조화하는 방식, 테스트에 의한 소프트웨어 개발, 즉 테스트 주도 개발 test-driven development(TDD)이다. TDD는 테스트를 시스템 동작에 대한 실행 가능한 명세로 사용하는 효과적인 전략이다.

이런 TDD 접근 방식은 다양한 세부 수준에서 운영할 수 있다. 먼저 행위 주도 개발 behavior-driven development(BDD)이라고도 하는 승인 테스트 주도 개발 acceptance test driven development(ATDD) 기법을 사용해 사용자 중심의 명세를 만들면서 시작할 수 있다

8.7 테스트는 새로운 지식을 끌어내는 원천

- 소프트웨어는 다른 공학 분야와 달리 '우주'를 완전히 스스로 만들고 통제할 수 있다.

윤리적·실용적 제약 없이 수백만 번의 반복 실험이 가능하며, 이는 최신 머신러닝이 작동하는 방식과 같았다.

테스트 환경(IaC 등)을 통해 동일한 우주를 지속적으로 재현하고 변수를 통제할 수 있다.

테스트 전체는 특정 우주 안에서 시스템이 어떻게 동작해야 하는지에 대한 **지식 체계**이다.

새로운 기능(아이디어)을 추가하려면 먼저 테스트를 만들어 '이 우주에서 어떤 사실을 기대하는지'를 정의해야 한다.

테스트가 실패한다면 그 아이디어는 기존 지식 체계와 충돌하는 것으로 수정해야 한다.

소프트웨어는 물리학처럼 완벽하지 않더라도 얼마든지 '과학'처럼 접근할 수 있다.

잘 설계된 테스트 체계는 몇 분 만에 시스템 전체의 일관성과 타당성을 검증할 수 있게 해준다.

100% 이상적인 상태가 아니어도, 80%만 충족해도 엄청난 가치가 있다.

이는 소프트웨어를 개인의 감(기교)에만 의존하지 않고 **공학적인 프로세스**로 발전시키는 길이다.

8.8 품질을 높이는 TDD 적용 사례 하나

- 소프트웨어 개발 과정에서 글쓴이가 주기적으로 수행하는 일반적인 실험 유형 중 하나를 설명한다

"TDD를 연습할 때 나는 테스트를 통해 의도한 코드 변경을 시작한다. 여기서 목표는 실패하는 테스트를 만드

는 것이다. 테스트를 실행해 테스트가 실제로 무언가를 테스트하는 과정에서 실패하는 사실을 확인하고 싶다. 그래서 나는 테스트 작성부터 시작한다. 내가 원하는 방식을 따르는 테스트를 확보하고 나면 테스트가 실패할 것으로 예상되는 '정확한 오류 메시지'를 예측한다."

- 나는 문제에 대해 생각하고 특성을 파악했다: "내가 원하는 시스템의 동작 방식을 결정하고 이를 테스트 케이스로 캡처했다."
- 나는 가설을 세웠다: "테스트가 실패할 것으로 예상한다!"
- 나는 예측을 했다: "실패하면 다음 오류 메시지와 함께 실패할 것이다."
- 내가 만든 실험을 수행했다: "나는 테스트를 실행했다."

작업의 품질에 상당히 긍정적인 영향을 미쳤다

정리:

'실험'의 결과를 신뢰할 수 있기를 바란다. 우리가 구축하는 시스템의 기술적인 컨텍스트에서, 효과적인 자동화된 테스트나 IaC 같은 지속적인 배포 기술을 통해 실험적으로 작업하고 가능한 한 변수를 통제하면 실험을 더 안정적이고 반복 가능하게 만들 수 있다. 하지만 더 깊게 들어가면, 이런 방식은 또한 소프트웨어를 훨씬 더 결정론적으로 만들어 더 높은 품질, 더 예측 가능하고 신뢰할 수 있는 사용성을 제공한다