4장 개선을 위한 반복

반복의 개념과 중요성

반복의 정의란:

반복은 일련의 연산을 반복해서 성공적으로 원하는 결과를 얻기 위한 절차입니다.

반복의 특징은:

학습을 이끄는 절차입니다. 반복을 통해 우리는 학습하고 반응하며, 학습한 내용을 조정하게 만듭니다.

반복과 피드백을 수집하는 밀접히 관련된 활동이 있다면, 계속적으로 학습할 기회도 생깁니다.

반복을 수행함으로서 점진적으로 맞춤 목표에 접근하게 만들어 줍니다.

목표에 점점 더 가까워지는지 아니면 멀어지는지를 알려주는 수단이 있는 한, 우리는 무작위로 반복해서 여전히 목표를 달성할 수 있습니다.

이는 진화의 본질적인 동작 방식이자 현대적인 머신러닝(machine learning)의 핵심이기도 합니다.

** 일련의 연산을 반복해 성공적으로 원하는 결과를 얻기 위한 절차 ** 는 곧 반복 자체의 개념인데, 소프트웨어 개발에 반복 과정을 적용하여 효과를 얻기 시작하여 혁명으로 불리기 시작하였는데 이것을 “애자일 혁명” 이라고 부르기도 하였습니다.

애자일 사고방식과 반복적 작업

애자일의 과학적 접근

과학적 공유 방식

- 희의적인 관점에서 출발해 아이디어에 접근하고, 아이디어가 올바르다는 사실을 증명하기보다는 **틀렸다는 사실을 증명**하기 위한 길을 모색하는 것이 훨씬 더 과학적인 사고방식

예측 가능성과 탐구

- 두 가지 사고 학파는 프로젝트 조직과 팀 관례에 근본적으로 다르고 호환이 불가능한 접근 방식을 조성

애자일 사고방식의 실천방법

- 안전하게 실패
- 실수를 쉽게 관찰
- 변경하여 다시시도
- 이상적으로는 다음에 더 잘할 수 있게 만듦

구체적으로 시도할 방법으로는 스크럼(Scrum), 익스트림 프로그래밍(Extreme Programming), TDD(Test-Driven Development) 등이 있습니다.

반복적으로 작업하는 방법의 특징은 다음과 같습니다.

- 더 계획적이고 순차적인 접근 방식과 비교해 **덜 가치 근본적인 차이**점이 존재
- 하지만 **훨씬 더 효과적인 전략**

소프트웨어 개발 역사 중 상당 기간 동안, 반복은 불필요하며 모든 단계에 대한 상세한 계획이 소프트웨어 개발 초기 단계의 목표라고 가정하며 시간을 보냅니다.

반복적인 작업의 복잡미묘한 장점

소프트웨어 공학을 발견과 학습 과정으로 접근한다면, **반복이 핵심이 되어야** 합니다.

반복적인 작업을 할 때의 효과는 **자동으로 초점을 좁히고** 더 작은 배치 단위로 생각하고 모듈성과 관심사 분리를 더 중요하게 생각하게 된다는 아이디어가 여기서 가장 중요할 것입니다.

애자일에서의 완료 개념.

애자일 사고 과정은 '소프트웨어 개발에서 진행 상황을 측정하기 어렵지만,
완료된 기능을 측정할 수는 있기에, 완료되었을 때 확인할 수 있는 더 작은 기능을 대상으로 작업하자' 였습니다.

하지만 '완료'를 때까지 얼마나 오래 걸릴지 알고 싶을 때 **문제가 복잡**해집니다

왜냐면, 이유는 다음과 같다.

- 지속적인 배포에서는 모든 작은 변경사항이 하루에도 여러 차례 일어날 수 있습니다

- 안전하고 신뢰성 높게 소프트웨어를 어느 시점에서든 프로덕션 환경에 릴리스할 수 있는 수준으로 마무리되어야 마땅합니다

그러면, 이런 컨텍스트에서 '완료'가 실제로 의미하는 바는 무엇일까?

- 릴리스가 가능하기 때문에 완료되므로 **'완료'를 합리적으로 측정**하는 유일한 척도는 사용자에게 가치를 제공하는 것입니다
- 이는 매우 주관적인 기준입니다
- 하지만 우리 사용자에게 '가치'를 제공하기 위해 변경사항이 얼마나 필요한지를 어떻게 예측할까?
- 대다수 조직은 여러 개를 조합해 '가치'를 제공하는 기능 목록을 추측하는 방법을 사용하지만, 소프트웨어의 생명 주기 중 어느 시점이든 릴리스할 수 있다면 이는 **다소 모호한 개념**입니다.

반복의 실용적 예시는 다음과 같습니다.

애자일 계획은 한 번의 스프린트나 반복 내에 기능을 완료할 수 있는 충분히 작은 조각으로 업무를 분해하는 역량에 **상당 부분 의존**했습니다:

- 초기에 이는 진행 상황을 측정하는 수단으로 홍보되었지만
- 정기적으로 우리 작업의 품질과 적절성에 대해 명확한 피드백을 제공한다는 **훨씬 더 심오한 영향을 미쳤습니다**
- 이는 우리의 학습 속도를 높여줍니다

방어적인 설계전략으로서의 반복

프로그램 개발 작업방법에는 크게 2가지가 있습니다.

- 전통적인 변경:폭포수 방식
- 애자일 방법

반복적 접근의 이점

댄 노스(Dan North)의 통찰

애자일 사고의 기초에 대한 흥미로운 견해를 내 친구인 댄 노스가 제시했습니다:

- **폭포수와 애자일 사고의 차이점**: 사실상 경제학의 문제
- **폭포수 사고방식**: 시간이 지날수록 변화에 들어가는 비용이 더 많음

그림 4.1: 전통적인 변경 비용 모델

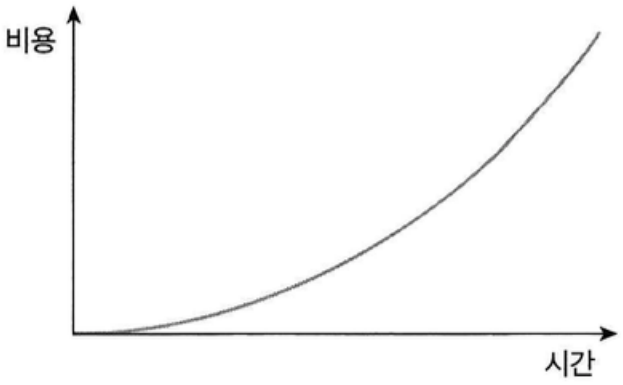

그림 4.1 전통적인 변경 비용

시간이 지날수록 변경 비용이 기하급수적으로 증가

전통적 방법의 문제점

이 세계관에는 **문제가 있습니다**:

올바른 해법을 찾기 어려움

- 유일하게 합리적인 해법은 **프로젝트 생명 주기 초반에 가장 중요한 결정을 내리는 것**
- 여기서 프로젝트 생명 주기 초반에는 **프로젝트에 대해 아는 바가 가장 적다는 어려움**이 존재
- 따라서 프로젝트 생명 주기 초반에 정보를 캐내기 위해 아무리 열심히 일해도 우리는 **잘못된 추측에 의거해 중요한 결정을 내릴 수밖에 없음**

소프트웨어 개발의 현실

소프트웨어 개발은:

- 우리가 작업을 시작하기에 앞서 **아무리 열심히 분석해도** 결코 **'모든 작업 단위가 완벽하게 이해된' 채로 시작되지 않음**
- 우리는 결코 **'잘 정의된 업무 집합'으로 시작하지 않으므로**, 아무리 부지런히 계획을 세우더라도, **정의된 프로세스 모델 또는 폭포수 접근 방식은 첫 번째 장애물에 부딪힘**
- 소프트웨어 개발을 이렇게 부적절한 틀에 맞추기란 불가능

반복적 접근의 필요성

소프트웨어 개발은 **탐구와 발견의 과정**이므로:

- 놀라움, 오해, 실수가 당연하므로
- **불가피하게 발생하는 실수로부터 우리를 보호하는 방법을 배우는 데 집중할 필요가 있음**

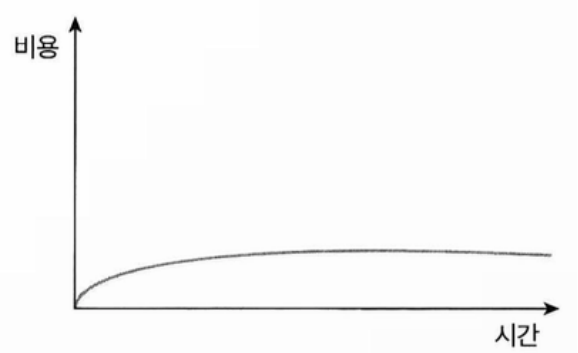

그림 4.2 애자일에서의 변경 비용

초반에는 비용이 증가하다가 일정 수준에서 평평해짐

애자일의 핵심 차이

매번 거의 동일한 비용으로:

- 우리 생각을 바꾸고
- 새로운 아이디어를 발견하고
- 오류를 발견하고
- 오류를 수정할 수 있다면 변화 비용 곡선이 평평해진다.

변화 비용 곡선이 평평하다면 어떻게 될까? 그러면:

- **새로운 사실을 발견하고 그 발견으로부터 이익을 얻는 자유**를 얻을 것
- 이렇게 되면 제품에 대한 이해, 제품을 구성하는 코드, 제품에 대한 사용자 경험을 **지속적으로 개선하게 만드는 접근 방법**을 채택할 수 있음

변화 비용 곡선을 평평하게 만들기

필수 요건

- 무엇이 필요한가? **우연가를 만들지 않고서** 분석이나 설계에 많은 시간을 할애할 수 없음
- 이유: 무엇이 정말 동작하는지 충분하게 **시간을 들여 배우지 못함을 의미**하기 때문

해결책: 경험을 압축

- **경험을 압축할 필요가 있음**
- 우리는 **반복적으로 작업할 필요가 있음**
- 우리 아이디어가 정말로 동작하는지를 확인하기 위해 **고객과 사용자의 손에 귀여를 수준 만큼만 충분한 분석, 설계, 코딩, 테스트, 릴리스를 수행할 필요가 있음**
- 우리는 이를 **되돌아보고, 학습을 바탕으로 이를 활용할 수 있게 다음 단계에 반영**해야 함

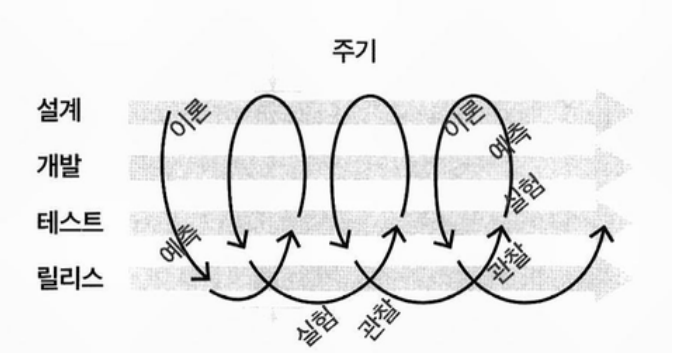

그림 4.3 지속적인 배포에서의 반복

반복 작업의 실제

효과적인 반복작업 방법.

첫째, 작은 단위로 작업한다.

- 각 변경의 범위를 줄이고 → 위험감소
- 더 작은 단계로 변경할 필요가 있음 → 가독성 좋음.
- 일반적으로 변경사항은 작을수록 더 좋음
- 이렇게 하면 **기법, 아이디어, 기술을 더 자주 시도**해 볼 수 있음 → 요건이 변경되었을 때 유연성있게 변경가능.

반복적 작업의 조건

반복적인 작업이 가능하려면

우리가 구축하는 시스템의 설계:

- 우리의 작업을 구조화하는 방식
- 작업하기 위한 조직을 구조화하는 방식에 **영향을 미침**

반복이 가능한 시스템의 특징

반복이라는 개념은:

- 이 책과 여기서 내가 제시하는 소프트웨어 공학 모델 이면을 깊이 파고듦
- 모든 아이디어는 서로 깊이 연결되어 있으며
- 종종 **반복이 끝나고 피드백이 시작되는 지점을 파악하기가 까다로울 수도 있음**

공짜 점심은 없다는것.

- 반복적 작업의 이점을 얻으려면
- 시스템, 작업 방식, 조직 전체를 **반복을 촉진하는 방향으로 재설계**해야 함
- 하지만 그 노력은 **더 나은 소프트웨어와 학습, 발견**이라는 가치 있는 결과를 가져옴!