4장 개선을 위한 반복

개선을 위한 반복

근복적으로 반복은 학습을 이끄는 절차다.

반복은 우리에게 학습하고, 반응하고, 학습한 내용을 조정하게 만든다.

반복의 진정한 위력은 우리가 정말로 우리 목표에 어떻게 접근할지 모를때조차도 반복하게 만든다는 데 있다.

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

애자일 혁명 : 조직과 일하는 방식 전체가 근본적으로 바뀐흐름

폭포수 스타일 사고

- 역사적 배경:

  1960년대 여러 팀들이 소프트웨어 개발을 위해 사용한 접근 방식.

- 특징:
  - 선형적이고 단계적인 진행 (요구사항 → 설계 → 구현 → 테스트 → 배포).
  - 각 단계가 끝나야 다음 단계로 넘어감.
  - 피드백 루프가 제한적이며, 변경에 유연하지 않음.
- 목표:

  프로덕션 중심. 처음부터 완벽한 계획 수립과 실행을 중시.

- 문제점:
  - 요구사항이 변할 경우 대응이 어려움.
  - 개발이 끝나기 전까지 결과물을 보기 어렵다.
  - 오류나 잘못된 방향을 나중에야 알게 됨.

애자일 사고

- 기본 가정:

  "*일은 불가피하게 잘못될 수 있다.*" → 변화와 실패를 전제로 시작.

- 핵심 개념:
  - 짧은 주기(Iteration)로 반복적인 개발.
  - 지속적인 피드백을 통해 점검 및 조정.
  - 고객과의 협업, 적응성, 가치 전달의 속도 중시.
- 실천 방법:
  - 스크럼, XP(eXtreme Programming) 등 다양한 프레임워크 활용.
  - 팀 간 협력과 자율성 강조.

어떤 유형의 소프트웨어 개발에 있어서든 애자일에 앞서 나온 프로덕션과 예측 중심의 폭포수
접근 방식보다는 애자일 방식이 훨씬 더 적합하다

따라서 반복은 모든 탐구 학습의 핵심이며, 모든 실제 지식 습득의 기본이다.

반복적인 작업의 복잡 미묘한 장점

반복적으로 작업하게끔 작업관계를 바꾸기 시작할 때, 자동으로 초점을 좁히고 더 작은 배치 단위로 생각하고 모듈성과 관심사 분리를 더 중요하게 생각하게 된다.

지속적인 배포에서는 모든 작은 변경사항이 하루에도 여러차례 일어날 수 있다.

이런 컨텍스트에서 “완료”가 실제로 의미하는 바는 무엇일까?

→ 완료를 합리적으로 측정하는 유일한 척도는 가치를 제공하는 것

즉 고객과 사용자의 좋은 피드백을 바탕으로 우리가 만들고 조정한 제품을 반복해서 수정하는 방법으로 더 가치가 높은 결과물을 제공

방어적인 설계 전략으로서의 반복

반복적으로 작업하면 설계에 방어적인 접근 방식을 취하게 된다.

* 폭포수사고방식 비용모델

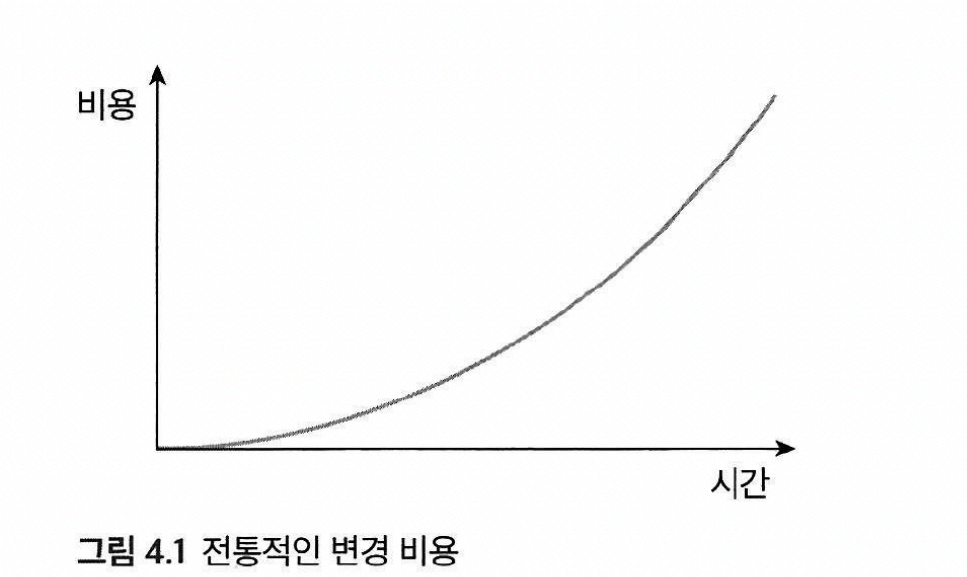


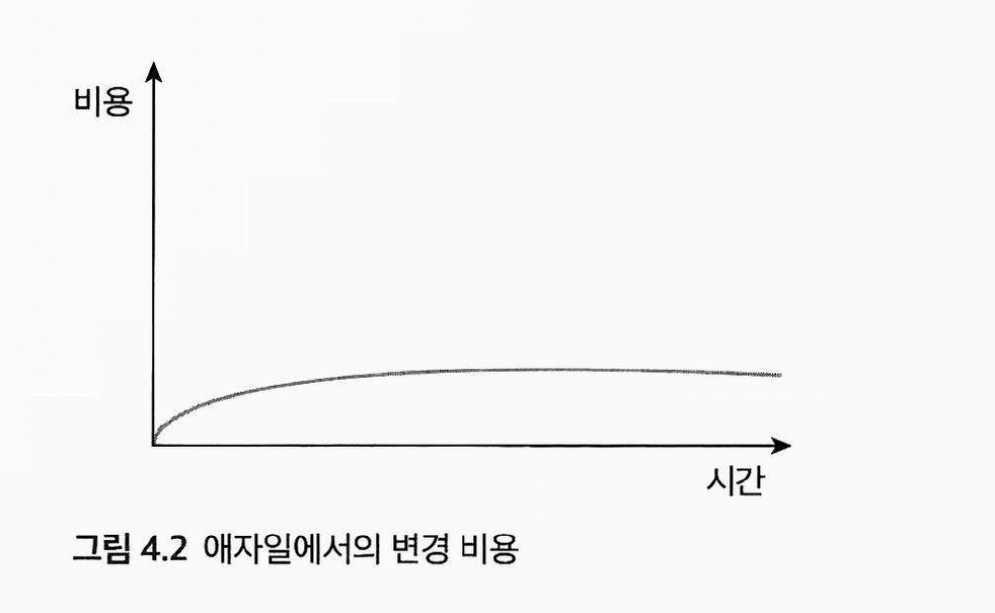
그림 4.1 전통적인 변경 비용

폭포수사고방식의 비용모델

합리적인 해법 : 프로젝트 생명 주기 초반에 가장 중요한 결정을 내리는것

- 생명주기 초반에는 프로젝트를 잘모름
- 아무리 열심히 일해도 잘못된 추측에 의거해 결정을 내림
- 불가피하게 발생하는 실수로부터 보호하는 방법이 필요

*** 애자일사고방식 비용모델**



애자일사고방식 비용모델

- 새로운아이디어를 발견, 오류발견, 오류수정 → 제품이해, 코드와 사용자 경험에 개선
- 반복적으로 분석, 설계, 코딩, 테스트, 릴리즈를 통해 변화 비용을 줄임

계획이라는 유혹

폭포수 사고방식은 '시작하기에 앞서 주의 깊게 생각하라, 무엇을 할지 신중하게 계획하고 부지런히 실행하라'
→ 합리적으로 보일수있음

하지만 실체가 있는 물리적인 제품을 만들 때, 프로덕션 공학의 문제점과 확장 문제는 종종 설계 문제보다 더 중요
제조가 점점 더 유연해지고, 경직된 프로세스는 도전 받으며 뒤집히고있음

→ 이런 패러다임이 근본적으로 잘못되었다는 사실을 인식하기 위해서는 어려운 지적인 도약이 필요

1. 우리는 사용자가 원하는바가 무엇인지 추측하는데 서툼

가장 효과적인 방법은 반복이다.
상당수가 잘못될것이라는 사실을 받아들이며, 최대한 빠르고 저렴하고 효율적으로 시도
방어적으로, 주의깊고 작은단위로 추측의 범위 또는 폭발 반경을 제한하고 이로부터 학습해야한다.

즉 최소 비용으로 아이디어를 시험하는 방법을 찾아내어, 해당아이디어가 나쁘다면 빠르고 상대적으로 저비용으로 이를 확인해야한다.

2. 범위가 제한된 아이디어, 그렇지 않은 아이디어

구분	정의된 프로세스 제어 모델 (Defined Process Control Model)	경험적 프로세스 제어 모델 (Empirical Process Control Model)
핵심 개념	모든 과정을 **사전에 계획하고 정의**한 뒤, 정해진 절차대로 수행	**실험·관찰·피드백을 통해 점진적으로 학습하고 조정**
특징	- 입력이 일정하면 출력도 항상 동일- 프로세스가 완전히 이해되어야 함- 계획이 완벽해야 동일한 결과 확보 가능	- 불확실한 상태에서도 시작 가능- 반복적 시도와 피드백으로 점진적 개선- 변경과 학습을 전제로 함
적용 조건	문제와 해결 방법이 모두 **명확하고 예측 가능**할 때	문제와 해결책이 **불확실하거나 복잡**할 때
장점	- 일관성 높은 결과- 관리와 통제가 용이	- 변화에 유연- 실제 상황에 맞게 조정 가능
단점	- 불확실성에 취약- 계획 오류 시 전체 실패 가능	- 예측 불가능성 존재- 명확한 계획 부재로 초기 혼란 가능
예시	폭포수 모델(Waterfall)	애자일(Agile), 스크럼(Scrum)
Ken Schwaber의 관점	“작업 하나하나를 완벽하게 이해해야 하며, 입력이 일정하면 항상 동일한 결과가 나온다.”	“탐구와 피드백을 통해 점진적으로 학습하고 개선해 나가는 개방적인 과정이다.”

3. 정리

계획이라는 유혹은 잘못된것이다.

오히려 더 제한적이고 예감과 추측에 기반하며, 현실적으로 작고 단순하고 잘 이해되고 잘 정의된 시스템에서만 작동한다.

- 자동차 여행처럼 일상적인일에서도 선택한 경로가 최적일지
- 교통정보를 반복적으로 업데이트해 변화하는 여행 환경을 점검

계획과 실행에 대한 반복적인 접근 방식은 예측적이고 이론적이며 항상 부정확한 상황 대신에 실제 우리가 처한 상황에 대한 가장 최신의 그림을 유지하게 만든다.

반복 작업의 실체

이런식으로 작업하려면 무엇을 해야할까?

1. 작은 단위로 작업한다.
2. 지속적인 통합
3. 테스트주도개발을 도입한다.

테스트주도개발

실패 : 테스트를 작성하고 돌리고 실패하는지 확인한다.

성공: 테스트를 작성하고 돌리고 성공하는지 확인하기에 충분한 코드만 작성한다.

리팩토링: 코드와 테스트를 변경해서 더깔끔하고 표현력있고 우아하고 일반적으로 만든다.

예를들면

코딩과정에서 새로운 클래스, 변수, 함수, 매개변수를 도입해 테스트를 돌려 코드가 계속해서 동작하는지를 주기적으로 점검

정리

반복은 중요한 아이디어이며 학습, 발견, 더나은 소프트웨어와 소프트웨어 제품에대한 좀 더 통제된 접근 방식을 향해 나아가는 우리 능력의 토대이다.

반복적인 작업은 우리가 구축하는 시스템의 설계, 우리의 작업을 구조화하는방식, 작업하기 위한 조직을 구조화하는 방식에 영향을 미친다.