2부 소프트웨어 프로세스 개선을 위한 구체적인 실천 방안

제4장 개선을 위한 반복

- 개선을 위한 반복은 “일련의 연산을 반복해 성공적으로 원하는 결과를 얻기 위한 절차”로 정의된다.

우리는 무작위로 반복해서 여전히 목표를 달성할 수 있다. 더 멀어지는 단계를 버리고, 더 가까워지는 단계를 선호할 수 있다. 이는 진화의 본질적인 동작 방식이다. 이는 또한 현대적인 머신러닝의 핵심이기도 하다

*애자일 혁명

반복적으로 작업하는 방식은 훨씬 더 효과적인 전략이다 반복은 불필요하며 모든 단계에 대한 상세한 계획이 소프트웨어 개발 초기 단계의 목표라고 가정하며 시간을 보냈다. 반복은 모든 탐구 학습의 핵심이며, 모든 실제 지식 습득의 기본이다.

4.1 반복적인 작업의 복잡미묘한 장점

- 반복적으로 작업하는 과정에서 얻는 다양한 다른 장점은 언뜻 보기에 명백하지 않을 가능성이 높다. 더 반복적으로 작업하게 끔 작업 관례를 바꾸기 시작할 때, 자동으로 초점을 좁히고 더 작은 배치 단위로 생각하고 모듈성과 관심사 분리를 더 중요하게 생각하게 된다는 아이디어가 가장 중요할 것이다. 이런 아이디어는 작업 품질을 강화하는 선순환의 일부로 끝난다

스크럼과 익스트림 프로그래밍 양쪽의 공통 아이디어 중 하나는 ‘완료를 위해작은 작업 단위로 일해야 마땅하다

애자일 사고 과정은 완료된 기능을 측정할 수는 있기에, 완료되었을 때 확인할 수 있는 더 작은 기능을 대상으로 작업하자

하지만 ‘완료’할 때까지 얼마나 오래 걸릴지 알고 싶을 때 문제가 복잡해진다.

그렇다면 이런 컨텍스트에서 ‘완료’가 실제로 의미하는 바는 무엇일까

각 변경사항은 릴리스가 가능하기 때문에 완료되므로 ‘완료’를 합리적으로 측정하는 유일한 척도는 사용자에게 가치를 제공하는 것이다 ‘가치’를 구성하는 변경 집합을 추측하는 과정에서 문제가 생기는 이유는 시작
할 때 필요한 모든 기능을 알고 ‘완전함’이라는 아이디어로 향해가는 진행 상황을 파악할 수 있다는 가정에 의존하고 있기 때문이다.

반복적인 작업의 더 미묘한 장점은 우리에게 선택권이 있다는 사실이다. 소규모 배치 기반 접근 방식은 우리가 작업할 기능의 크기와 복잡도를 줄이도록 현업에 종사하는 우리를 격려해 왔으며, 이는 정말로 중요한 단계로 볼 수 있다

애자일 계획은 한 번의 스프린트나 반복 내에 기능을 완료할 수 있는 충분히 작은 조각으로 업무를 분해하는 역량에 상당 부분 의존했다

작고 명확하고 서비스에 바로 투입이 가능한 단계로 진행하는 반복적인 작업은 우리에 게 훌륭한 피드백을 제공한다!

4.2 방어적인 설계 전략으로서의 반복

- 반복적으로 작업하면 설계에 방어적인 접근 방식을 취하게 된다

폭포수와 애자일 사고의 차이점은 사실상 경제학의 문제라고 설명했다. 폭포수 사고방식은 시간이 지날수록 변화에 들어가는 비용이 더 많다는 가정에 기초한다.

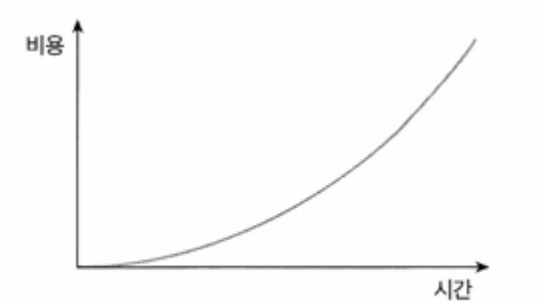

그림 4.1 전통적인 변경 비용

이 그래프의 문제는 프로젝트 생명 주기 초반에 가장 중요한 결정을 내리는 것임을 의미하지만,

여기서 프로젝트 생명 주기 초반에는 프로젝트에 대해 아는 바가 가장 적다는 어려움이고 프로젝트 생명 주기 초반에 정보를 캐내기 위해 아무리 열심히 일해도 잘못된 추측에 의거해 중요한 결정을 내릴 수밖에 없다.

소프트웨어 개발은 우리가 작업을 시작하기에 앞서 아무리 열심히 분석해도 결코 '모든 작업 단위가 완벽하게 이해된'채로 시작되지 않는다

소프트웨어 개발은 탐구와 발견의 과정이어서 놀라움, 오해, 실수가 당연하므로, 불가피하게 발생하는 실수로부터 우리를 보호하는 방법을 배우는 데 집중할 필요가 있다

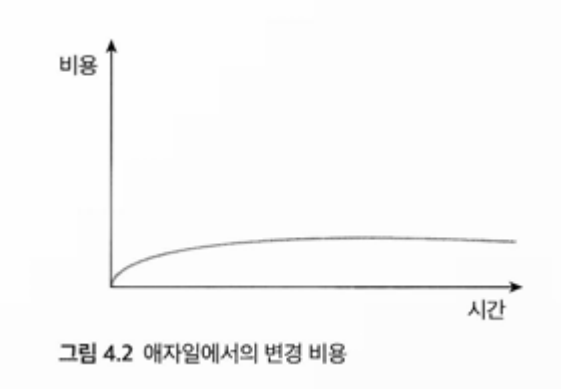

그림 4.2 애자일에서의 변경 비용

그렇다면 우리 생각을 바꾸고, 새로운 아이디어를 발견하고, 오류를 발견하고, 오류를 수정할 수 있다면 어떻게 될까? 변화 비용 곡선이 평평하다면 어떻게 될까 제품에 대한 이해, 제품을 구성하는 코드, 제품에 대한 사용자 경험을 지속적으로 개선하게 만드는 접근 방법을 채택할 수 있다.

그렇다면, 변화 비용 곡선을 평평하게 만들기 위해 무엇이 필요할까?

우리는 경험을 압축할 필요가 있다. 우리는 반복적으로 작업할 필요가 있다. 우리 아이디어가 정말로 동작하는지를 확인하기 위해 고객과 사용자의 손에 쥐어줄 수준 만큼만 충분한 분석, 설계, 코딩, 테스트, 릴리스를 수행할 필요가 있다.

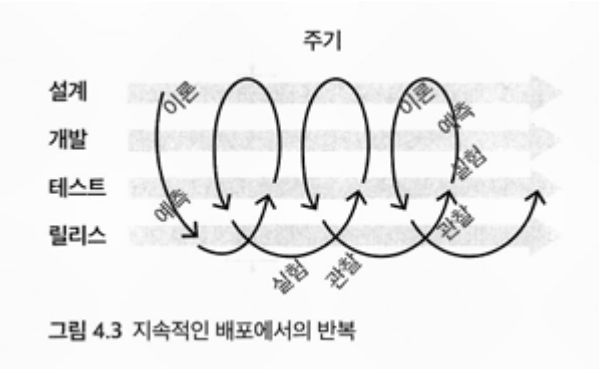

그림 4.3 지속적인 배포에서의 반복

4.3 계획이라는 유혹

- 폭포수 사고방식이 최선의 방법이라고 생각했지만, 업계는 수십 년에 걸쳐 폭포수 접근 방식이 동작하도록 노력해 왔지만, 성공하지 못했다.

‘시작하기에 앞서 주의 깊게 생각하라’와 ‘무엇을 할지 신중하게 계획하고 부지런히 실행하라’는 문구를 보면 폭포수 접근 방법이 아주 합리적으로 보인다. 잘 정의된 프로세스가 있다면, 이렇게 정의된 프로세스 제어 접근 방식은 매우 효과적이다

하지만 이제는 심지어 물리적인 제품 제조 분야에서 조차 이런 상황이 바뀌고 있다. 제조가 점점 더 유연해지고, 몇몇 제조 공장은 방향을 전환할 수 있게 되면서, 심지어 제조 공정에서 조차 이런 경직된 프로세스는 도전 받고 있으며 뒤집히고 있다

소프트웨어에서 애자일 혁명이 당시 규범을 뒤집어 엎었지만, 심지어 오늘날에도 많은 아마도 대다수 조직의 마음속 깊은 곳에는 계획 주도/폭포수 주도 개발이 남아 있다

조직이 다음과 같은 일을 할 수만 있다면 정말 좋을 것이다.
• 사용자 요구 사항을 정확하게 파악함
• 이런 요구 사항이 충족되었을 경우 조직에 주는 가치를 정확하게 평가함
• 이런 요구 사항을 충족하기 위해 얼마나 많은 비용이 들어갈지 정확하게 추정함
• 이익이 비용보다 더 큰지 합리적으로 의사 결정함
• 정확하게 계획을 수립함
• 편차 없이 계획을 수행함
• 마지막에 비용을 계산함

하지만 현실 세계에서 일어나는 소프트웨어 개발은 단순히 이런 식으로 작동하지 않는다.

4.4 반복 작업의 실제

- 그렇다면 이런 식으로 작업하기 위해 우리는 무엇을 할 수 있을까?

첫째, 작은 단위로 작업한다 각 변경의 범위를 줄이고 더 작은 단계로 변경할 필요가 있다 이렇게 하면 기법, 아이디어, 기술을 더 자주 시도해 볼 수 있다 작은 단위로 작업하는 방식은 또한 우리가 가정을 유지할 필요가 있는 시간 범위를 제한함을 의미한다

마지막으로, 작은 단계를 유지한다면 심지어 작은 단계가 상황의 변화나 우리 쪽의 오해 때문에 무효화되더라도 업무 손실이 적다

애자일 방식은 작고 고정된 시간 내에 완료되고 서비스 준비가 이뤄진 코드를 만든다는 아이디어를 장려한다

완전히 다른 규모에서 보면, 지속적인 통합과 테스트 주도 개발 이라는 관례를 본질적으로 반복적인 프로세스로 생각할 수 있다

지속적인 통합에서는 변경사항을 자주 커밋해 하루에도 여러 차례 수행한다

테스트 주도 개발은 종종 실패, 성공, 리팩토링으로 기술된다

- 실패 : 테스트를 작성하고 돌리고 실패하는지 확인한다.
- 성공 : 테스트를 작성하고 돌리고 성공하는지 확인하기에 충분한 코드만 작성한다.

- 리팩토링 : 코드와 테스트를 변경해서 더 깔끔하고 표현력 있고 우아하고 일반적으로 만든다. 작은 변경 직후 테스트를 돌려서 통과하는지 확인한다.

이는 매우 세분화되고 반복적인 접근 방식으로, 코드 작성의 근본 기술에 대해 상당히 반복적인 접근 방식을 권장한다

정리 :