6장 점진주의: 조금씩, 조금씩,앞으로

점진적으로 일하는 것은 가치를 점진적으로 구축하는 것이다.

간단 말해 시스템의 모듈화 또는 컴포넌트화를 이용하는 것이다.

점진적인 접근방식을 사용하면 업무를 세분화하고 단계별로 가치를 전달해 더 빨리 가치에 도달하고 더 작고 간단한 단계로 가치를 전달할 수 있다.

우주선 예시로 살펴보는 모듈성

아폴로 프로그램이 시작되었을때, 초기도약중 하나 “달 궤도 랑데부”

우주선을 일련의 모듈로 나눠 각각이 도전적인 임무의 특정 부분에 집중하는 것

- 서비스 모듈의 임무는 지구에서 달까지 모든 모듈을 운반하고 다시 돌아오는 것
- 사령선 모듈은 우주비행사들의 주요 거주지, 주요임무는 우주비행사들을 지구 궤도에서 지구 표면으로 귀한시키는 것
- 달 탐사 모듈은 하강 상승모듈
    - 하강모듈 : 우주비행사를 달 궤도에서 달표면으로 데려와줌
    - 상승모듈 : 우주비행사들을 달 궤도로 되돌려보내 사령선 모듈/서비스 모듈과 도킹한 후 지구도 돌아감

모듈성은 개별 컴포넌트를 문제의 한 부분에 집중하도록 구축하고 설계에서 타협할 필요가 적다.

다른 그룹은 거의 독립적으로 자신의 모듈에 집중, 단 서로 인터페이스하는 방식에 동의하기만 하면됨

소프트웨어의 관점 : 문제의 한부분을 해결하려는 목표로 문제를 여러 부분으로 나뉨

컴포넌트 기반 설계 접근 방식 (마이크로서비스나 서비스 지향 설계)

→ 각 컴포넌트는 테스트 쉬워짐, 배포 빨라짐, 다른컴포넌트와 독립적 배포 가능

단, 모듈식 접근 방식을 취한다면

시스템 모듈간의 경계 고려가 필요

효율 높은 조직 구성을 위한 비법

요즘 많은 조직들이 일하는 방식을 바꾸려는 **트랜스포메이션**을 시도하지만,

이 변화는 조직 전체에 새로운 방식을 퍼뜨리는 과정에서 **큰 어려움**을 겪는다.

1. 사람들에게 변화를 이해시키고 동기를 부여하는 일,
2. 기존 제도나 절차 같은 **조직적 장벽**이 큰 걸림돌이 됩니다.

그래서 많은 기업들이 업무를 **표준화**하려 하지만, 모든 일을 단계별로 규격화하기는 쉽지 않습니다.

특히 **창의적인 일을 하는 조직**일수록 사람의 **판단력과 창의성**이 필요하죠.
과도한 표준화나 자동화는 오히려 **유연성과 인간적인 대응력**을 잃게 만듭니다.

그렇다면 조직이 인간의 창의성을 활성화하려면,

업무 절차나 정책 안에서도 창의적으로 일할 수 있는 자유와 여유를 남겨둬야 한다는 뜻이에요.
특히 소규모 팀의 중요성이 강조됩니다.

연구와 경험에 따르면, 작은 팀이 큰 팀보다 훨씬 더 좋은 성과를 내는 경우가 많아요.
프레드 브룩스도 “유능한 사람 몇 명이 200명보다 낫다”고 말했죠.

애자일(Agile) 실무자 ≤ 8명 이하의 팀이 가장 효율적
why? 소규모 팀이 빠르게 의사결정하고, 스스로 실험하며, 점진적으로 발전할 수 있기 때문
→ 다른 부서나 상급자에게 일일이 허락받지 않아도 즉시 행동하고 수정할 수 있는 자율성이 큰 장점

조직의 변화를 이끌기 위한 가장 효과적인 방법은
이런 작고 자율적인 팀들이 독립적으로 변화할 수 있게 만드는 것

물론 완전히 제멋대로 움직여서는 안 되니, 전체적인 목표나 비전에 맞게 대략적인 방향만 맞춰주는 구조화된 자유가 필요합니다.

결국, 현대 조직이 추구해야 할 진정한 트랜스포메이션은

사람과 팀이 더 큰 자율성과 책임감을 가지고 창의적으로 일할 수 있게 하는 것입니다.

즉, 크게 움직이기보다 작게, 분산적으로, 점진적으로 변화하는 접근이 핵심

점진주의를 적용하기 위한 실천 도구

점진적으로 변경하고 싶다면, 다른영역에 미치는 영향을 제한하면서 변경할수 있어야한다.

1. 리팩토링

코드 개선, 최소한 수정할 수 있는 작고 간단하게 변경하는 것이다.

이러한 작은 변경은 결과가 마음에 들지 않는다고 판단되면 쉽게 철회할 수 있다.

2. 버전제어, 위의 리팩토링과 결합

리팩토링 도구와 버전 관리 시스템을 활용하면, 개발자는 코드를 작고 안전하게, 점진적으로 개선

할 수 있다. 잘못된 변경은 쉽게 되돌릴 수 있어서,

항상 안정적인 상태를 유지하며 발전 할 수 있다.

3. 테스트

테스트를 최대한 단순하게 유지하고 시스템을 최대한 테스트 가능한 코드로 설계

이런 세 가지 기법의 조합은 점진적으로 변경할 수 있는 우리의 능력을 크게 향상한다.

변경의 부작용을 최소화하자

복잡한 시스템을 다룰 때는 큰 변화보다 작고 점진적인 발전이 안전하다.

시스템을 독립적인 컴포넌트 분리하기

시스템을 독립적인 컴포넌트로 분리하고, 포트와 어댑터 패턴을 이용하면 각 팀이 자율적으로 개선 하면서도 전체가 안정적으로 발전할 수 있다.

- 포트 어뎁터 패턴

  두 컴포넌트가 연결되는 지점에 어댑터(중간 번역기)를 두는 방식

  이 어댑터가 입력과 출력을 변환해 주기 때문에,

  한쪽의 코드를 바꿔도 다른 쪽에는 영향을 주지 않는다.

→ 코드 변경의 영향을 줄이고, 주의가 필요한 부분을 안전하게 관리 한다.

→ 다른기술이여도 다 가능, 즉 더많은 주의를 구현하기위한 전략이다.

피드백의 속도

예를 들어, 내가 다른 사람의 코드를 망가뜨렸다고 합시다.

그 사실을 즉시 알게 되면 문제 해결이 쉽고 영향이 적지만,

몇 달 후에야 알게 되면 이미 상용 환경에 반영되어 피해가 커질 수 있습니다. 즉, 문제를 빨리 발견하고 수정하는 것이 변경 관리에서 매우 중요하다.

정리하자면

시스템을 설계할 때 자율적으로 변경할 수 있는 환경을 만들고, 작은 변경을 빠르게 공유·검토하여 피드백과 문제를 효과적으로 관리하면 변경의 부작용을 최소화시킬수있을 것이다.

점진적인 설계

애자일과 '무한의 시작'

- 모든 답을 알지 못해도 일단 시작할 수 있는 점이 중요하다.
- 점진적으로 진전을 이루면서 배우는 것이 핵심 아이디어다.

정신적 전환과 자신감

- 미래에 어떤 문제가 생길지 모른다는 무지와 불확실성을 받아들이고, 이를 관리할 수 있는 자신감이 필요하다.
- 모르는 것을 인정하고 빠르게 배우는 태도가 공학적 사고로 나아가는 첫걸음이다.

점진적 설계와 복잡성 관리

- 코드를 작은 단위로 나누고 관심사를 분리하며, 점진적으로 개선 가능하도록 설계한다.
- 구조화된 코드는 미래 변경이나 예기치 않은 문제 발생 시 유연성을 제공한다.

피드백과 실수 관리

작은 변경을 자주 테스트하고 평가하여 피드백을 빠르게 받는다.

- 실수나 변경의 영향 범위를 제한하면서 안전하게 발전할 수 있다.

경험과 내부 경보 시스템

- 경험을 통해 코드가 너무 복잡하거나 결합도가 높으면 경고 신호를 인식하고 조정한다.
- 단순히 작고 단순한 코드가 아니라, **배우고 변경할 수 있는 코드**가 목표다.

결론

점진적, 반복적 작업은 더 높은 품질의 소프트웨어를 만드는 기본 원칙이다. 자유롭게 변경하고 학습할 수 있는 환경을 만드는 것이 공학적 우수성의 핵심이다.

정리

복작합 시스템을 구축할 대는 점진적인 작업이 기본이다.

시스템

진전을 이루면서 점진적으로 지식과 이해가 축적되는 작업의 결과물