

# 6장 점진주의 조금씩, 조금씩 앞으로

## 점진주의

“점진적인 설계는 더 나은 성능을 보장하기 위해 개선될 경우 컴포넌트를 자유롭게 대체할 수 이쁘니 모든 모듈식 설계 어플리케이션과 관련이 있다.”

초안을 기준으로 더 나은 방향으로 다듬고 개선한다.

## 모듈성

각 요소들을 큰 대분류 기능을 하는 것들을 모듈 단위로 나눠서 각 모듈의 기능에 집중할 수 있도록 하기위한 목적이다.

이러한 방식을 웹개발로 가져오면, “마이크로서비스” 나 “서비스 지향 설계” 같은 컴포넌트 기반 설계 접근 방식을 예로 들 수 있다.

모듈 방식으로 기능을 구성하게되면 모듈에 집중해서 개발을 할 수 있으며, 모듈식 접근방식을 취할 때 모듈간의 경계를 고려하며 개발에 임해야 한다.

## 효율높은 조직 구성을 위한 비법

대규모 단위의 팀으로 운영하기 보다 오늘날에는 8명 이하의 소규모 팀으로 구성한 뒤 자율성을 부여하여 각 팀이 변화할 수 있는 기회를 주도록 하는 방법의 접근 방식을 많이 도입한다. 즉, 분산적이고 점진적인 변화가 핵심이 된다.

모듈형 조직은 소프트웨어 개발을 위한 더 전통적이고 조직적인 구조보다 훨씬 더 유연하고 확장성이 뛰어나며 효율적이다.

## 점진주의를 적용하기 위한 실천도구

### 핵심 개념

- 학습과 복잡성 관리를 위한 5가지 원칙들은 서로 깊이 연결되어 있음

- 점진적으로 일하기 위해서는 구체적인 도구와 기술이 필요함

## 3가지 핵심 도구

### 1. 리팩토링 (Refactoring)

- 작고 안전한 단계로 코드를 개선하는 기술
- 한 곳의 변경이 다른 부분에 미치는 영향을 최소화
- 개발 환경의 자동화 도구 활용 (메서드 추출, 매개변수 도입 등)

### 2. 버전 제어 (Version Control)

- 작은 변경을 쉽게 되돌릴 수 있게 함
- 안전한 지점으로 언제든지 돌아갈 수 있어 반복적 작업 가능
- 점진적이고 안정적인 개발 지원

### 3. 자동화된 테스트 (Testing)

- 변경 시 확신을 갖고 앞으로 나아갈 수 있게 보호
- 테스트 주도 개발(TDD)로 모듈화된 설계 유도
- 실수의 영향 범위를 제한하고 안전한 변경 가능

이 세 가지 기술의 조합이 점진적 변경 능력을 크게 향상시킴

- 피드백과 실험을 통한 지속적 개선
- 모듈성과 관심사 분리로 안전한 변경 환경 구축

## 변경의 부작용을 최소화 하자

변경의 부작용을 최소화 하는 방법으로 포트와 어댑터 패턴 이라는 강력한 기법을 사용할 수 있다.

결합을 떨어뜨리고 싶은 시스템의 두 컴포넌트 사이의 인터페이스 지점인 "포트" 에서 입력과출력을 변환하는 분리된 코드조각인 "어댑터"를 정의. 이렇게하면 포트를 통해 상호작용하는 다른 컴포넌트에 변경을 강요하지 않고서도 어댑터 뒤의 코드를 변경할 수 있는 자유를 얻게된다.

>> 자바로 치면 interface로 메서드를 선언하고, 인터페이스를 구현부가 변경되어도 해당 메서드를 사용하는 사람은 변경할 필요가 없는 경우와 같다.

즉, 포트, 어댑터의 구조적 설계로 모든 모듈을 구성하고 지속적인 통합과 지속적인 배포를 통해 시스템을 작은단위로 변화시켜서 통합시키는 것이 중요.

- 지속적 통합: 시스템을 독립적 부분으로 분해, 변경내역을 자주 통합, 피드백을 통해 속도와 품질개선
- 지속적 배포: 문제를 빠르게 발견하고 신속히 수정, 코드가 이미 상용 환경에서 사용중이라 수정에 유의하여 배포할것.

### 포트와 어댑터 패턴으로 구성후

점진적 변경 + 빠른 피드백 + 독립적 컴포넌트 설계 = 안전하고 효율적인 시스템 개발 재시도

## 점진적 설계

복잡한 시스템은 한번에 나오지 않는다.

간단한 시스템을 시작으로 점진적 탐구를 통해 나온 성과이다.

점진적 설계를 통해 시스템이 커지게되면 "복잡성을 관리" 하게 되는 단계가 된다.

시스템이 커지면서 복잡성이 올라갈 수 있다. 이 때, 복잡성을 관리하는것도 중요하다. 복잡성이 올라가서 실수가 발생되었을 때, 실수의 범위를 제한 할수 있기도 하기에 복잡성을 관리하는 것이 중요하다는 것이다.

(실수 했을때 초래될 수 있는 피해의 폭발반경을 제한할 수 있다.)

코드의 결합도가 낮고, 모듈성을 높힘으로서 복잡성을 낮추는 것이 중요한데,

- 그 방법으로 코드가 10줄이상 으로 길어지거나,
- 매개변수가 4개를 초과하는 함수를 피한다.

이 두 가지를 지켜나가면 시간이 지남에 따라 기능이 명확해 졌을 때 점진적으로 성장 할 수 있는 코드가 된다.

## 정리

복잡한 시스템을 구축할 때는 점진적인 작업이 기본.

복잡한 시스템은 머릿속에서 한번에 '완전히 형성' 되지 않는다.

우리가 진실을 이루면서 점진적으로 지식과 이해가 축적되는 작업의 결과물 처럼, 학습을 촉진해 다양한 사례에 대해서 검증하고 검증된 것들을 구조화 함으로서 우리가 보지 못했던 새로운 것들을 발견하는 방향으로 나갈 수 있다.

이런 과정이 우리가 효과적으로 발전할 수 있는 핵심 이 될 수 있다.

