1장. 소프트웨어 공학의 정의와 역사

공학이란 과학의 실용적인 응용 분야

소프트웨어 개발은 발견과 탐구의 과정이다.

소프트웨어 개발로 성공하기 위해서는 학습의 전문가가 될 필요가 있다.

학습을 위한 인류 최고의 접근 방식은 과학이므로, 과학의 기술과 전략을 채택해 이를 우리 문제에 적용할 필요가 있다.

[위키피디아에 기술된 과학적인 방법론]

- 특징 파악 : 현재 상태를 관찰 한다.
- 가설 수립 : 해당 관찰을 설명할 수 있는 이론인 설명을 만든다.
- 예측 : 가설에 따라 예측한다.
- 실험: 예측을 테스트 한다.

이런 식으로 생각을 정리하고 많은 작고 일상적인 실험을 바탕으로 진전을 이루기 시작할 때, 우리는 성급하게 부적절한 결론을 내리는 위험을 제한하고 훌륭하게 임무를 완수하기 시작 한다.

소프트웨어 공학 정의의 재구성

소프트웨어 공학은 소프트웨어의 현실적인 문제를 풀기 위한 효율적이고 경제적인 해법을 찾아나서는 경험적이고 과학적인 접근 방식의 응용이다.

소프트웨어 개발의 공학적 접근 방식이 중요한 이유

- 소프트웨어 개발은 항상 발견과 배움의 연습
- 우리의 목표가 '효율적'이고 '경제적' 이라면 우리의 학습 능력은 지속 가능해야만 함

우리는 학습의 전문가이자 복잡성을 관리하는 전문가가 되어야만 한다.

[학습의 전문가가 되기 위한 5가지 철학]

- 반복
- 피드백
- 점진주의
- 실험
- 경험주의

이런식으로 작업 시 우리가 안전하게 진행할 수 있는 방법에 제한이 가해지며 우리는 모든 소프트웨어 프로젝트의 핵심에 있는 탐사라는 여정을 촉진하는 방식으로 일할 수 있어야 한다.

학습뿐만 아니라 방향이 불확실할 때조차도 전진하게 만드는 방식으로 일을 진행할 필요도 있다.

이를 위해 복잡성을 관리하는 전문가가 될 필요가 있다.

[복잡성을 관리하는 전문가가 되기 위한 개념]

- 모듈성
- 응집성

- 관심사 분리
- 추상화
- 느슨한 결합

이 책의 목표는 이런 개념을 구조화해서 이런 개념이 제공하는 잠재력을 최대한 발휘하게끔 돕는 소프트웨어 시스템을 개발하기 위해 일관성 있는 접근법으로 통합하는 것이다.

다시. '소프트웨어 공학'

우리의 '소프트웨어 공학' 관례는 더 나은 소프트웨어를 더 빨리 구축하도록 허락하지 않으며 진짜 공학도 아니므로, 우린 이런 상황을 바꿀 필요가 있다.

성공에 대한 어떤 보장도 할 수는 없지만, 위와 같은 정신적 도구를 채택하여 원칙을 구조화하고 업무에 적용하면 성공의 기회가 확실히 늘어날 것이다.

전진, 앞으로

우리는 효과가 있는 방법과 효과가 없는 방법을 학습해 왔고, 계속해서 학습하고 있다.

우리는 전진하며 소프트웨어를 만들어나가기 위해선 활동에 지침을 제공하는 합의된 원칙과 규율이 필요하다.

- 단, 이런 사고방식을 잘못 적용 시 매우 엄격하고 지나치게 지배적인 '권위자의 결정'과 같은 스타일의 사고방식으로 변할 수 있다.

우리는 오래되었지만 제대로 확립된 나쁜 아이디어에 어떻게 이의를 제기하고 반박하며, 새롭고 잠재적으로 훌륭하고 시도되지 않은 아이디어를 평가할 수 있을까?

- 아주 확실한 예제가 있다. 독단적 신조에 이의를 제기하고 반박하며, 출처가 무엇이든 유행이나 상투적인 나쁜 생각과 훌륭한 생각을 구분하기 위해 우리에게 지적 자유를 허락하는 접근법이다?!
    - 이런 좋은 예제를 '과학' 이라고 부른다.
    - 또한 실질적 문제를 풀기 위한 이런 유형의 사고방식을 적용할 때 '공학' 이라고 부른다.

소프트웨어 공학의 탄생

1960년대에 **소프트웨어 공학**이라는 개념이 탄생했다.

MIT 계기장비 연구소의 소프트웨어 공학 부문 책임자가 된 마거릿 해밀턴이 처음 사용했다고 한다.

같은 시기에 NATO 는 독일의 가르미슈-파르텐키르헨에서 소프트웨어 공학이라는 용어를 정의하기 위한 회의를 소집했는데, 이것이 최초의 **소프트웨어 공학 컨퍼런스**였다.

... 이하 예제 생략

패러다임의 전환

패러다임의 전환이라는 아이디어는 우리가 어떤 변화를 만들 때 그 과정의 일부로서 이제는 더 이상 올바르지 않다고 알게된 몇 가지 아이디어를 버리게 될 것이라는 사상을 암묵적으로 포함한다.

정리

공학은 더 엄격한 관료주의를 부르는 규제가 아니라, 고품질 소프트웨어를 더 지속 가능하고 안정적으로 만들기 위한 우리의 역량을 향상하는 토대다.