1부 소프트웨어 엔지니어링이란 무엇인가

제1장 소프트웨어 공학의 정의와 역사

공학이란 과학의 실용적인 응용 분야

소프트웨어 공학 정의의 재구성

다시, '소프트웨어 공학'(전진, 앞으로)

소프트웨어 공학의 탄생

패러다임의 전환

1.1 공학이란 과학의 실용적인 응용분야

- 소프트웨어 개발은 발견과 탐구의 과정이다. 따라서 성공하려면 소프트웨어 엔지니어 들은 학습의 전문가가 될 필요가 있다. 학습을 위한 인류 최고의 접근 방식은 과학이므로, 과학의 기술과 전략을 채택해 이를 우리 문제에 적용할 필요가 있다. 이 책은 소프트웨어에서 문제를 해결하기 위한 현실적이고 실용적인 접근 방법에 기초하고 있으며 과학적인 기초 원리의 비공식적인 채택, 다시 말해 공학(엔지니어링)에 기반한다

1.2 소프트웨어 공학 정의의 재구성

- 소프트웨어 공학(software engineering)은 소프트웨어의 현실적인 문제를 풀기 위한 효율적이고 경제적인 해법을 찾아 나서는 경험적이고 과학적인 접근 방식의 응용이다

소프트웨어 개발에 대한 공학적인 접근 방식의 채택은 두 가지

1)소프트웨어 개발은 항상 발견과 배움의 연습이며,

2)우리의 목표가 '효율적'이고 '경제적'이라면 우리의 학습 능력은 지속 가능해야만 한다

이는 우리가 새로운 지식을 배우고 이를 적용하는 방식으로 반드시 시스템의 복잡도를 관리해야만 한다는

사실을 의미한다. 따라서 우리는 학습의 전문가이자 복잡성을 관리하는 전문가가 되어 야만 한다

학습의 전문가가 되기 위해서는 구체적으로 다음과 같은 철학이 필요하다

반복 / 피드백 / 점진주의 / 실험 / 경험주의 는 복잡계의 창조를 위한 진화적인 접근 방식이다

학습에 집중적으로 신경을 써야 할 뿐만 아니라, 해답은 물론이고 종종 방향이 불확실할 때 조차도 전진하게 만드는 방식으로 일을 진행할 필요도 있다 이를 위해 복잡성을 관리하는 전문가가 될 필요가' 있다. 복잡성을 관리하는 전문가가 되기 위해서는 다음과 같은 개념이 필요하다.

모듈성 / 응집성 / 관심사 분리 / 추상화 / 느슨한 결합

소프트웨어 개발을 주도하기 위한 도구로서 이런 10가지 아이디어를 활용하는 방법을 기술한다. 그러고 나서 소프트웨어 개발을 위한 효율적인 전략을 추진하기 위해 실용적인 도구로서 사용할 수 있는 일련의 아이디어를 기술할 텐데, 여기에는 다음이 포함된다.
• 테스트 가능성• 배포 가능성• 속도• 변수 통제• 지속적인 배포

이런 아이디어를 적용할 때 고품질 소프트웨어를 만들고, 더 빠르게 많은 작업 결과물을 생산하며, 이런 원칙을 채택한 팀 내 사람들은 작업을 훨씬 더 즐기고 스트레스가 훨씬 덜하며 일과 삶의 균형을 훨씬 더 제대로
찾는다

1.3 다시 '소프트웨어 공학'

- 이 책 제목을 정하느라 고생했다. 뭐라고 부를지 몰랐기 때문이기보다는 우리 업계가 소프트웨어라는 컨텍스트에서 '공학'이 의미하는 바를 평가절하 하는 식으로 재정의해 왔기 때문에

소프트웨어 에서 '공학'은 사람들이 지나치 게 관료주의 적 이거나 절차적이라고 치부하는 '코드' 또는 유사한 무언가에 대한 동의어로 취급된다.

우리의 '소프트웨어 공학' 관례는 더 나은 소프트웨어를 더 빨리 구축하도록 허락하지 않으며 진짜 공학도 아니므로, 우리는 이런 상황을 바꿔야 마땅하다! 이게 바로 이 책의 핵심을 이루는 근본적인 생각이고, 이 책의 목표는 훌륭한 소프트웨어 개발의 근간을 이루는 몇 가지 기본 원칙을 통합해 지적으로 일관된 모델을 기술하는 데 있다

전진 앞으로

- 소프트웨어 개발은 복잡하고 정교한 활동이다 소프트웨어는 어떤 면에서 인류가 시도하는 매우 복잡한 활동 중 하나다.

모든 사람이 모든 것에 거부권을 행사한다면, 업계 또는 팀으로서 우리는 어떻게 전진하고 소프트웨어를 만들어 낼 수 있을까? 우리는 우리 활동에 지침을 제공하는 합의된 원칙과 규율이 필요하다

이런 사고방식의 위험은 만일 잘못 적용한다면 매우 엄격하고 지나치게 지배적인 '권위자의 결정'과 같은 스타일의 사고방식으로 변한다는 것이다

우리는 이런 문제를 해결하는 방법은 독단적인 신조에 이의를 제기하고 반박하며, 출처가 무엇이든 유행이나 상투적인 나쁜 생각과 훌륭한 생각을 구분하기 위해 우리에게 지적 자유를 허락하는 접근법이다.

또한 나쁜 생각을 더 좋은 생각으로 대체하고 좋은 아이디어를 개선하도록 돕는다. 근본적으로 우리는 개선된 접근 방식, 전략, 프로세스, 기술, 해법을 성장 시키고 발전하게 만드는 몇몇 구조가 필요하다.

실질적인 문제를 풀기 위해 이런 유형의 사고방식을 적용할 때, 우리는 이를 '공학'이라고 부른다

1.4 소프트웨어 공학의 탄생

- 개념으로서의 소프트웨어 공학은 I960년대 말에 만들어졌다. 소프트웨어 공학이라는 용어는 나중에 MIT 계기장비 연구소의 소프트웨어 공학 부문 책임자가 된 마거릿 해밀턴이 처음으로 사용했다.

북대서양 조약 기구North Atlantic Treaty Organization（NATO)는 독일의 가르미슈-파르텐키르헨에서 소프트웨어 공학이라는 용어를 정의하기 위한 회의를 소집했는데, 이것이 바로 최초의 소프트웨어 공학 컨퍼런스였다

1.5 패러다임의 전환

-'패러다임의 전환'은 물리 학자인 토마스 쿤이 주장했다

때때로 우리는 근본적으로 무언가에 대한 우리의 관점을 바꾸며 이런 과정에서 무언가를 배우지만, 이는 또한 우리가 전에 배웠던 내용을 버려야만 한다는 사실을 의미한다.

18세기, 평판이 좋은 생물학자（그 당시에는 그렇게 불리지 않았다）들은 몇몇 동물이 자연적으로 발생한다고 믿었다. 다윈Darwin은 四세기 중반에 등장해서 자연 선택의 과정을 설명했고, 이는 자연 발생설이라는 아이디어를 완전히 뒤집었다

케플러,코페르니쿠스, 갈릴레오는 당시 지구가 우주의 중심이라는 일반적인 통념에 도전했다.

패러다임의 전환이라는 아이디어는 우리가 이런 변화를 만들 때 그 과정의 일부로서 이제는 더 이상 올바르지 않다고 알게 된 몇 가지 아이디어를 버리게 될 것이 라는 사상을 암묵적으로 포함한다

소프트웨어 개발을 과학적인 방법론과 과학적인 합리주의에 근본을 두는 진정한 공학 분야로 취급함으로써 파급되는 영향은 심오하다. 이는 디지털 트랜스 포메이션 엔진에 감명 깊게 기술된 영향과 효과뿐만 아니라, 이런 접근 방식이 대체하는 기존의 아이디어를 버리기 위한 본질적인 필요성을 다루기 때문에 심오하다. 이는 훨씬 더 효과적으로 학습하고 훨씬 더 효과적으로 나쁜 아이디어를 버리기 위한 접근 방법을 제시한다