고급 객체지향 개발론

진화적 설계

daumkakao
최윤상

당신은
개발할 때 어떤 절차로 하나요?


Red Bull
Red Bull
Red Bull
Red Bull

1. 준비를 한다


자유시간
자유시간
COOKIE
자유시간
SNICKERS
Ghana
Ghana


한라산물
순한소주
한라산


RAISON
Winston
SALEM
ZEST
vogue
vogue
ONE

2. 닥치고 코딩!

계획된 설계 /사전 설계
Planned Design / Upfront Design

분석 - 설계 - 구현 - 테스트 - 운영

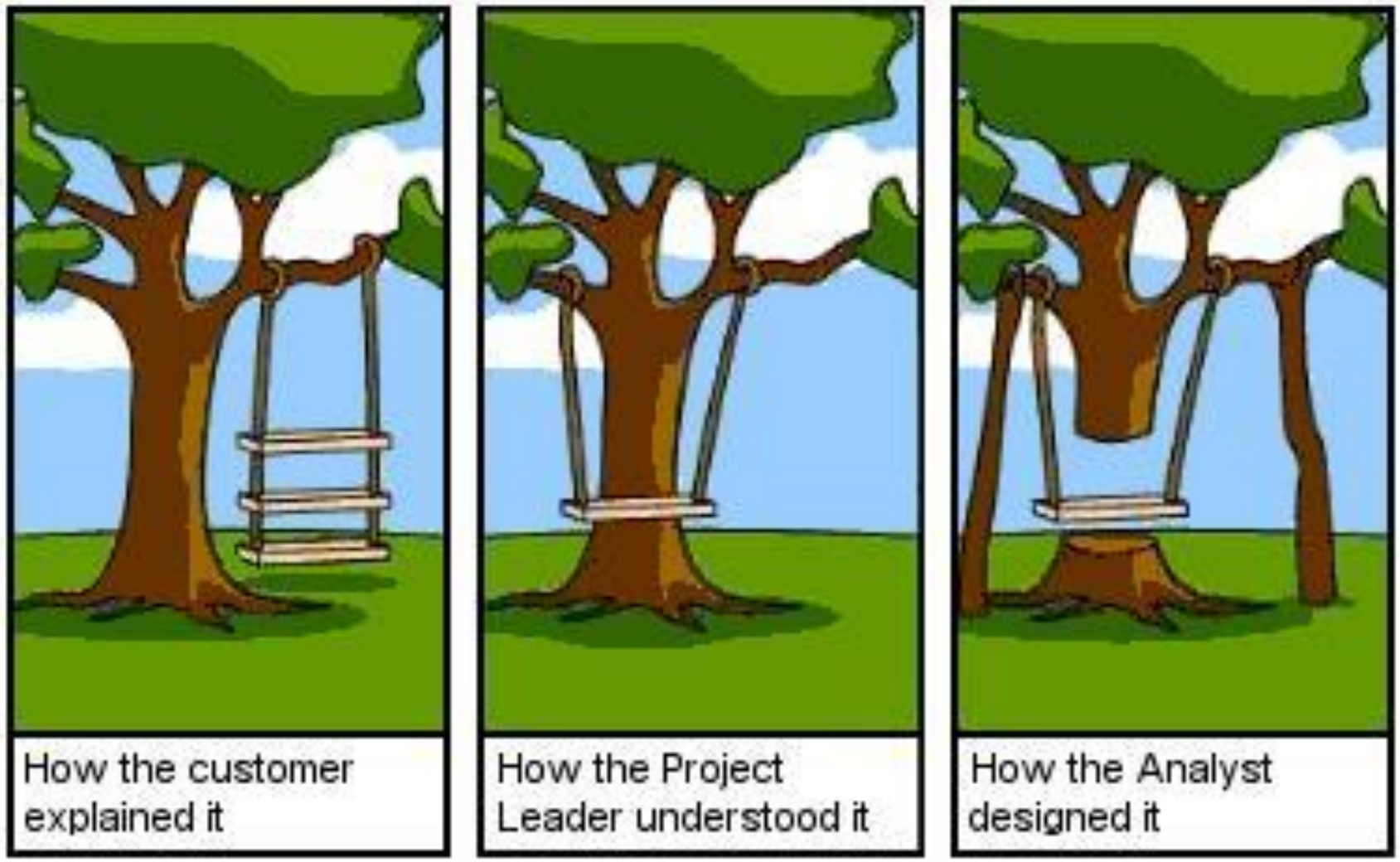
진화적 설계
Evolutionary Design

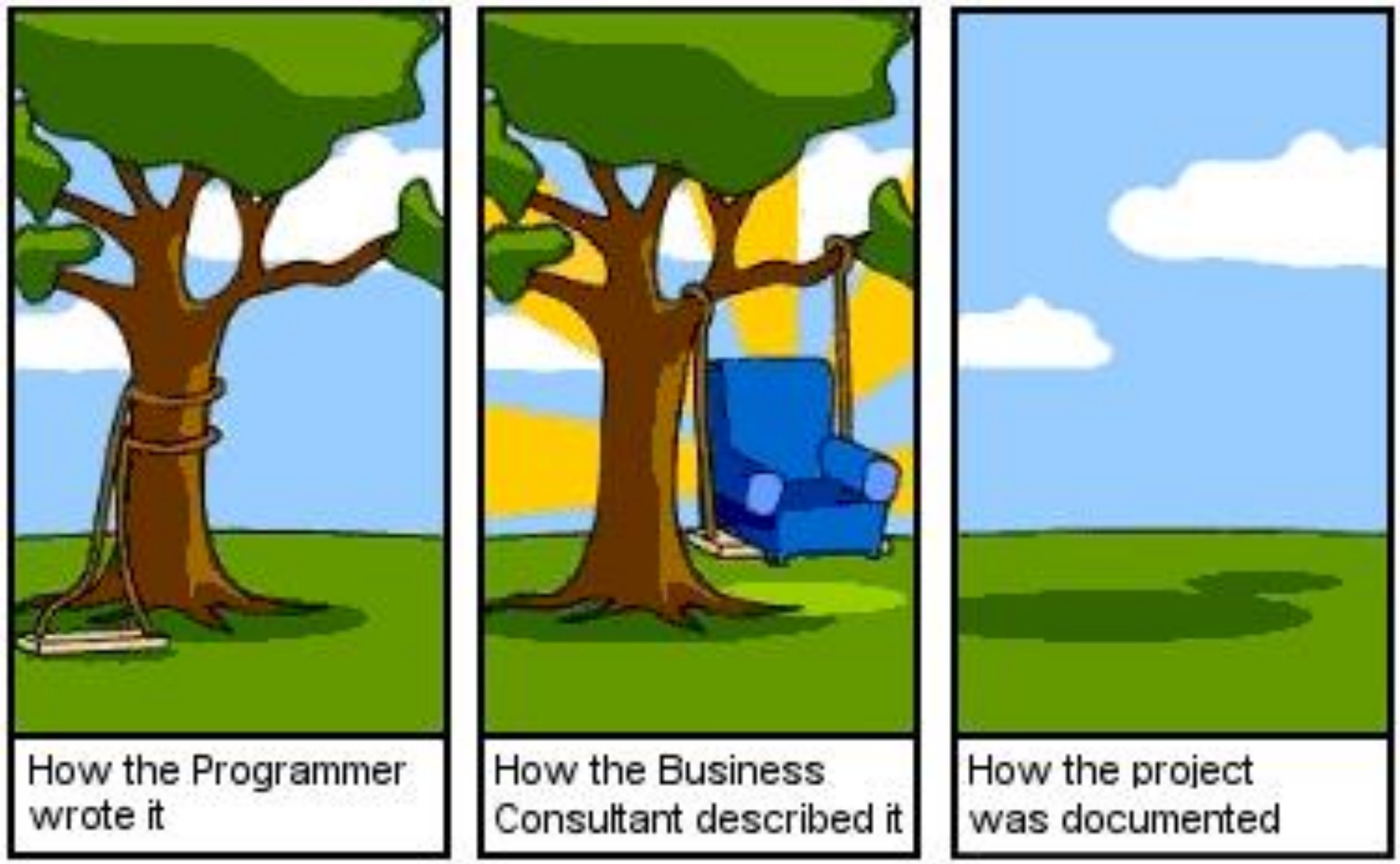
딱 필요한 만큼만!

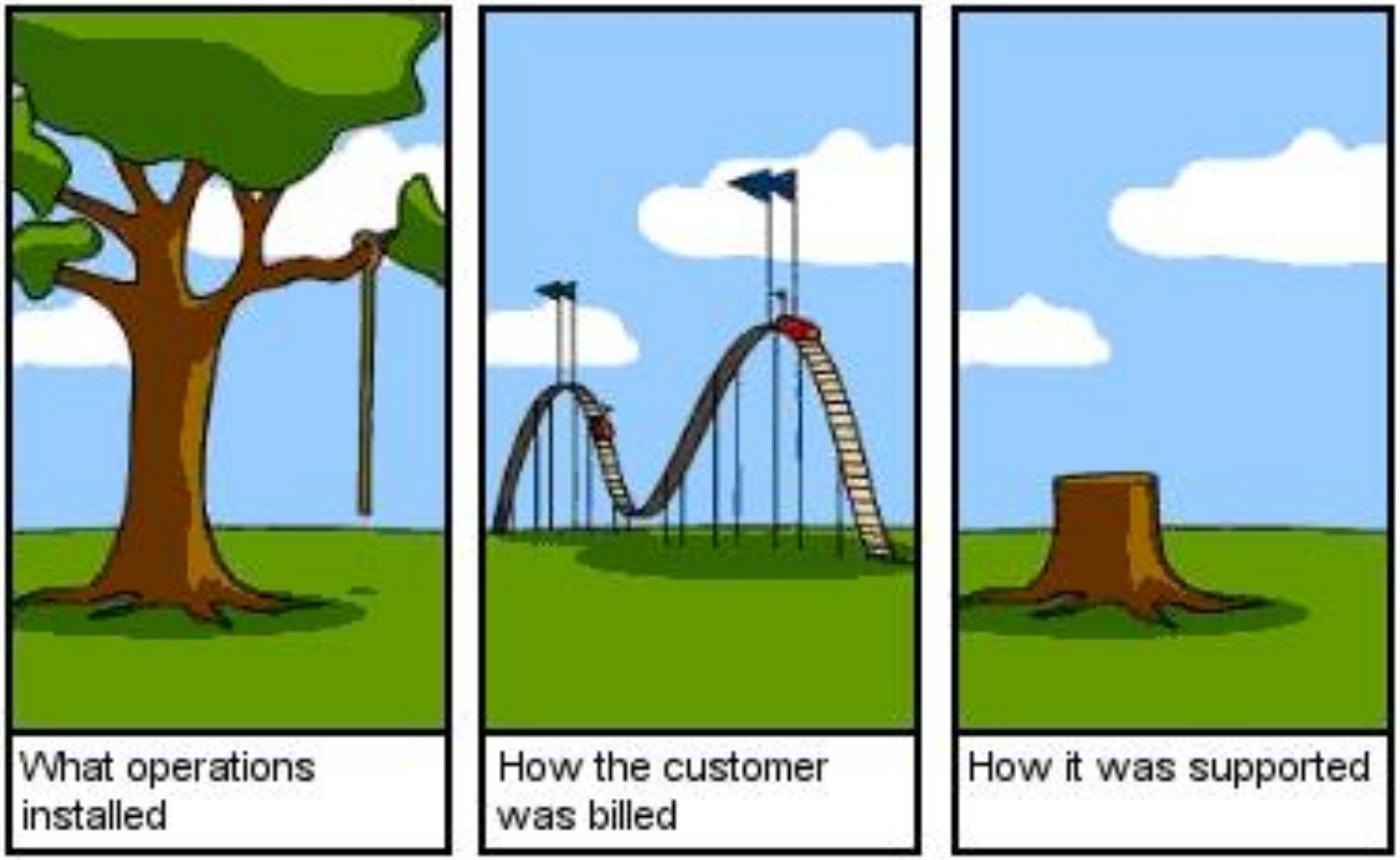
분석 - ((구현/테스트) - 설계) - 운영

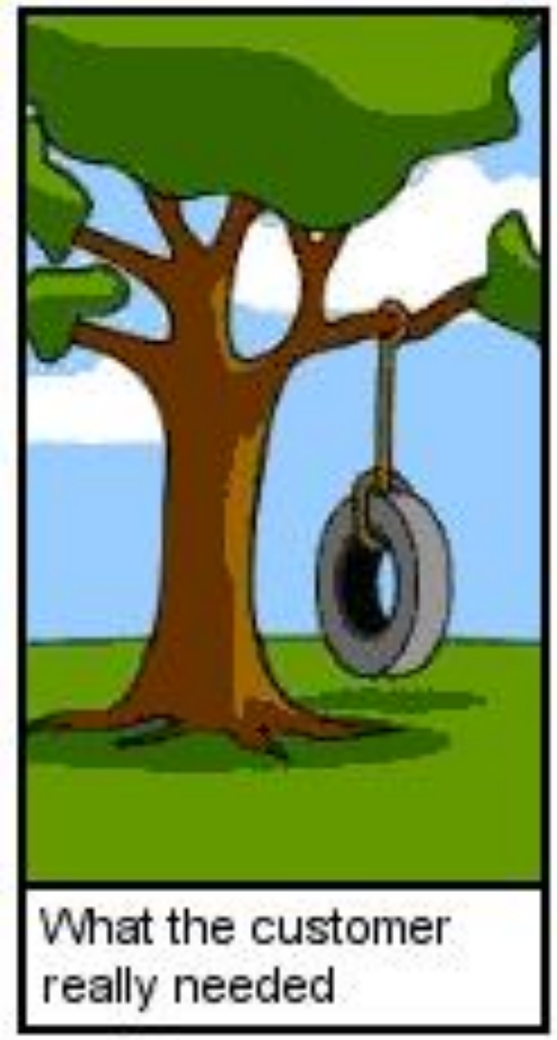
진화적 설계?
Why?

과장된 SW Application

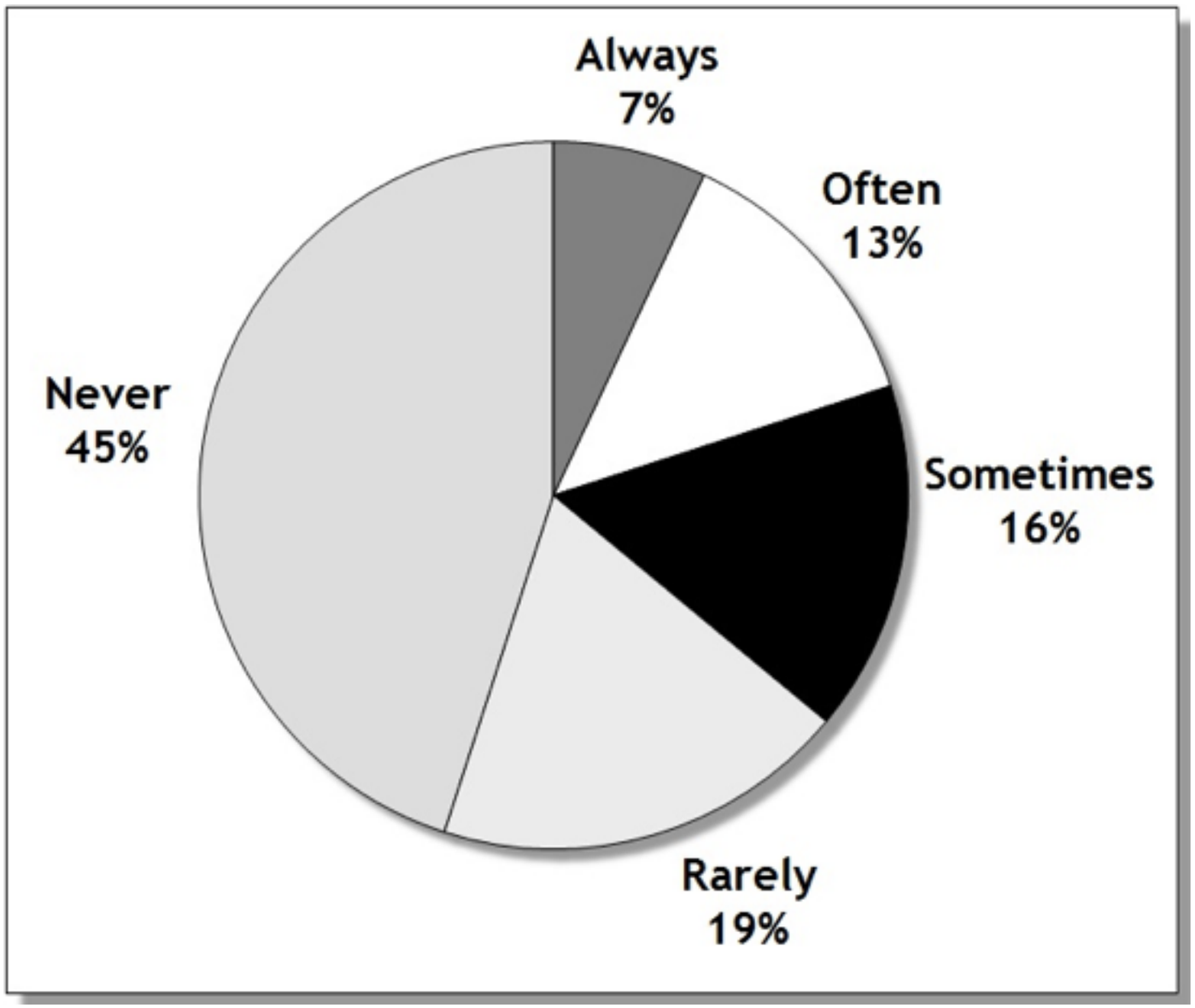

How the customer explained it
How the Project Leader understood it
How the Analyst designed it


How the Programmer wrote it
How the Business Consultant described it
How the project was documented


What operations installed
How the customer was billed
How it was supported


What the customer
really needed

구현된 기능 사용율



문제의 근원

“요구사항이 계속 변한다”

문제의 근원

요구사항 정리 -> 소프트웨어 설계

개발 싸이클 vs. 요구사항 수정 비용

기간 준수 vs. 요구사항 확정

문제의 근원

“왜 요구사항이 계속 변할까?”

사용해보기 전에는 원하는 기능이 뭔지 모름

실제 SW를 사용해 볼 수 있는 후반에만 피드백이 가능

잘못된 메타포 차용

다른 공학 분야(**건축**)에서 메타포 차용

건축가가 설계도면 완성
하도급자에게 전달하여 건물을 짓도록 함

잘못된 메타포 차용

건축 : 설계 -> 구축

SW : Coding -> Build

건축 : 구축에 많은 비용

SW : Coding에 많은 비용

잘못된 메타포 차용

설계과 **구현**을 완전히 분리

구현 전 모든 설계를 확정하는 방식

Big Design Up-Front (BDUF)

구현과정에서의 피드백을 통한 설계 개선 여지가 없음

SW 공학과의 차이점(목적)

- 건축학
  - 설계 도면에 어그러짐 없이 건물을 건설하는 것
- 제조업
  - 설계에 부합하는 제품을 제공하는 것
- SW
  - 구체적인 피드백과 배움을 통해 설계를 성장시켜 나가는 분야
  - 프로젝트에 합류한지 일주일만에 설계한 모듈
  - vs.
  - 6개월 후의 재 설계한 모듈의 품질 차이

공학적 메타포로 오염된 개발 프로세스

구현을 통한 피드백 원천 차단

과도한 사전 설계에 순응하는 코드 생성 요구

어떠한 장치에도 불구하고 변경되는 요구사항

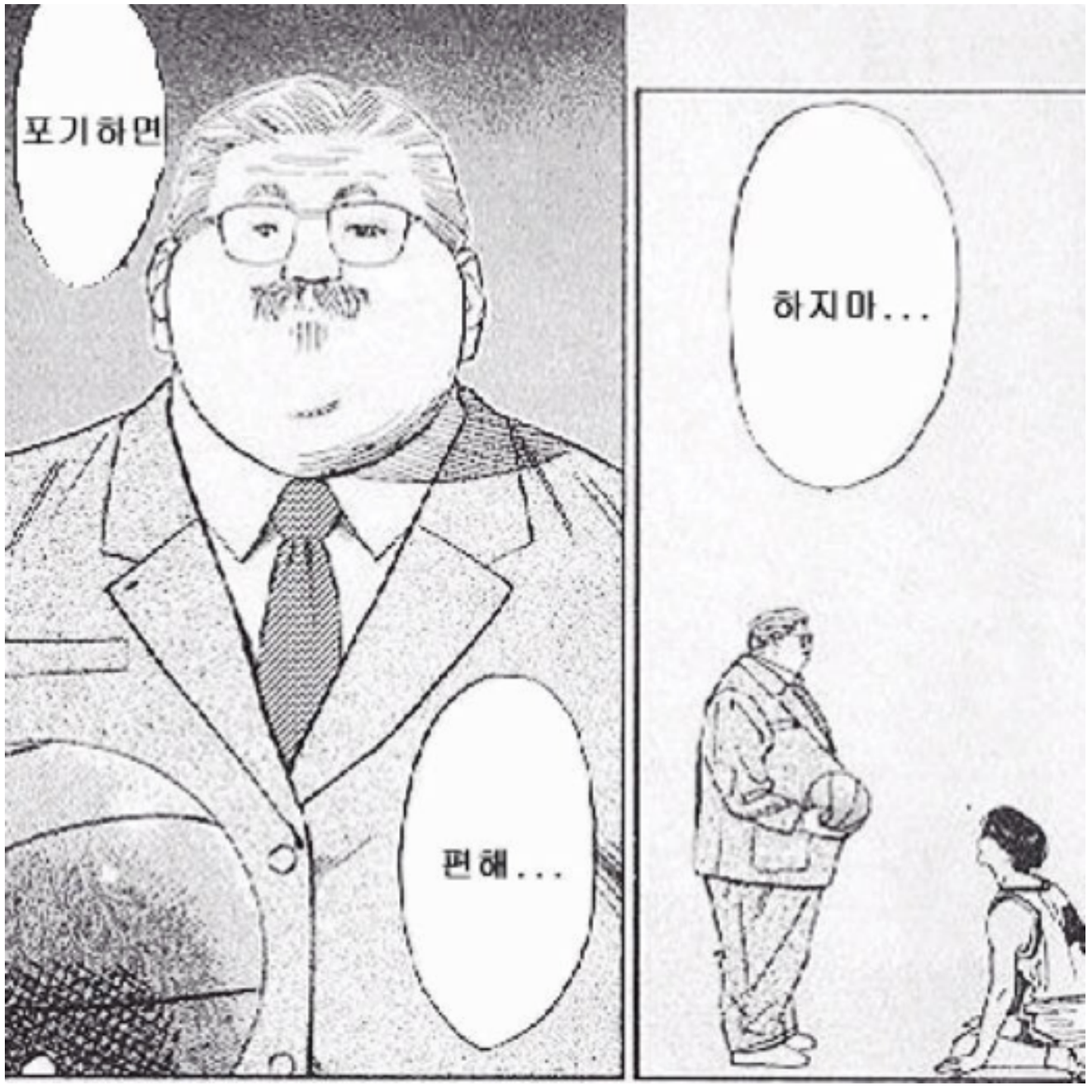
공학적 메타포로 오염된 개발 프로세스

요구사항 변경에 대한 설계자의 최후 해결책

- 변경 가능한 모든 항목들을 미리 설계
- 불필요한 유연성 / 복잡성
- 머피의 법칙

“Just in case code”

“요구사항이 변한다는 건 당연해.
그냥 받아들여”



“변경을 수용하라”

요구사항 변경은 오류가 아니다

사용자가 SW에 대해 더 잘 알게 되었다는 환영의 신호

개발할 SW에 대해 더 잘 알게 되었다는 긍정의 신호

개발 과정을 둔하게 하는 요인

충분한 정보가 없는 상태에서 불안정한 결정을 내림

내린 결정을 변경하지 않으려고 함

사전 설계 방식의 문제

명확하지 않음에도 요구사항 베이스 라인 확정

요구사항이 변경될 수 있는 상황에서 미리 설계 확정

요구사항 변경이 악이라는 가정에서 출발한 결과

진화적 설계

인간의 본성

명확한 오늘에 충실하기 보다 불확실한 내일에 대비

제어할 수 없는 것을 제어할 수 있다고 착각

요구사항 변경, 과도한 사전 설계 발생

진화적 설계의 모토

내일이 아니라 오늘에 충실하자

오늘 주어진 정보가 충실하지 않다면 내일로 미뤄라

오늘은 오늘 주어진 정보만으로 가능한 작업을 하라

진화적 설계: 현재를 위한 설계

- 오늘 구현에 필요한 요구사항에 대해서만 고객과 커뮤니케이션 하라
- 내일 구현할 요구사항은 간단한 목록 정도로만 책상 한편에 치워둬라
- 언제 변경될 지, 언제 구현될 지 모르는 요구사항에 너무 많은 시간을 소비하지 마라
- 오늘의 요구사항을 만족시킬 정도만 설계하라
- 오늘 필요로 하지 않는 유연성은 추가하지 말라

진화적 설계: 피드백 통한 진행

- 피드백 주기가 짧을 수록 오늘을 위해 작업하고 있는지 여부를 더 확실하게 알 수 있다.
- 고객의 피드백 주기를 줄이는 방법
  - 더 짧은 주기로 SW를 배포
- 설계에 대한 피드백 주기 줄이기
  - 짧은 개발 주기 (iteration)로 나누고
  - 각 iteration에서 구현해야 하는 부분에 대해서만 설계하고 곧바로 구현
- 점진적(incremental) 개발로 요구사항 수용

~~설계 후 구현~~

구현 후 설계

구현 후 설계:
TDD (Test Driven Development)

Test Driven Design

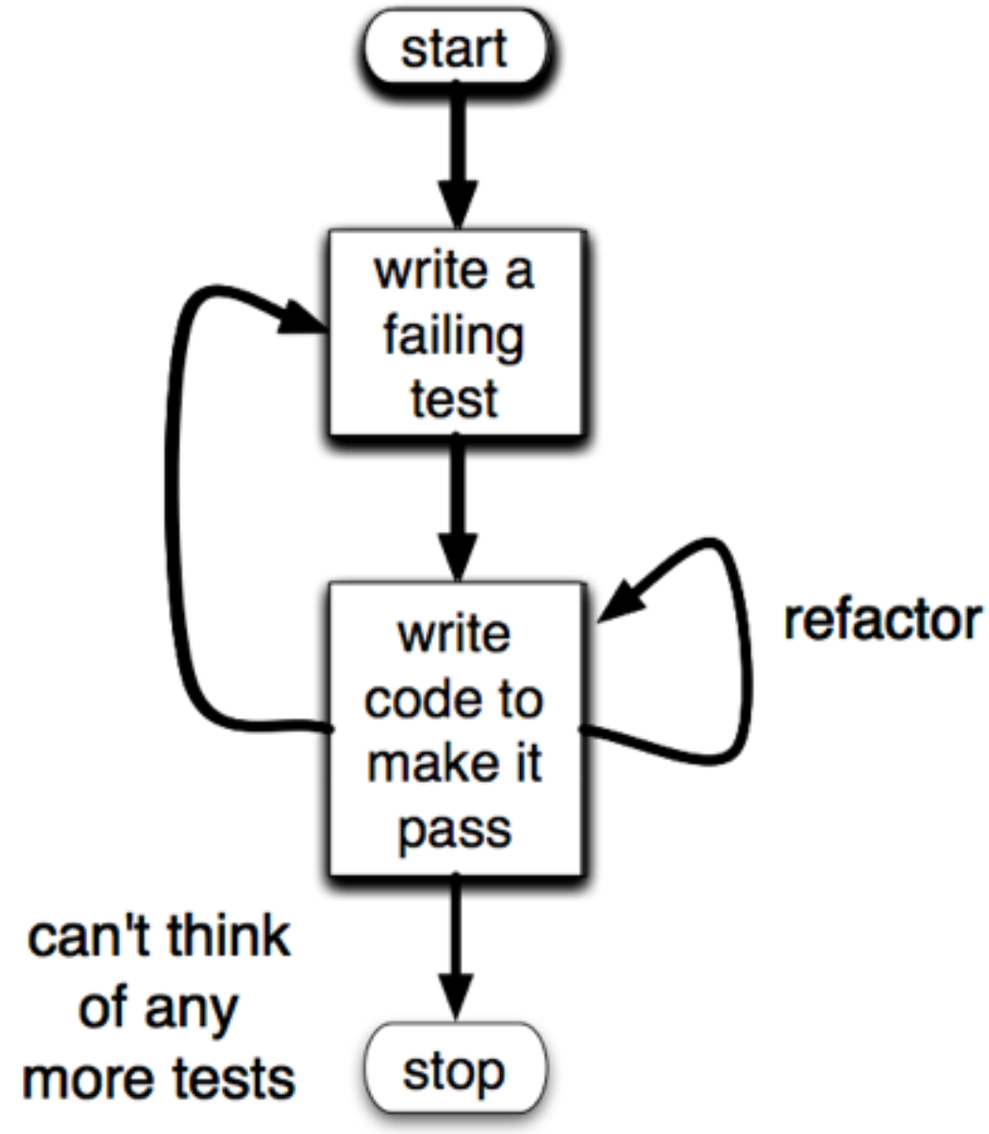
Test First Development

그리고....

Refactoring

Practice and Principle : TDD

- 테스트 작성
- 테스트 통과시키기
  - 더러워도 일단 돌아가게...
- 리팩토링



Practice and Principle : Refactoring

- 화려하지만 과도한 설계 vs. 투박하지만 적합한 설계
- 적합한 설계에 이를 수 있는 방법
  - 안정망 (TDD)
  - Clean Code
  - Boy Scout Rule

TDD & Refactoring

- 과도한 설계와 분석 마비 상태에서 허비하는 시간을 줄이고
- 다양한 기능을 포함하는 품질 높은 코드를 빠른 시간 안에 작성할 수 있게 됨

That’s all?

TDD와 Refactoring을 하면

“진화적 설계”가 되는 겁니까?

설계를 진화시키려면...

훌륭한 설계 원칙과 원리, 다양한 이론에 대한 학습

이를 지속적으로 적용시킬 수 있는 기술이 필요

The Agile Manifesto