Time goes now

익스트림 프로그래밍
(eXtreme Programming)

Software Engineering Lab
김영기 책임
resious@gmail.com

XP란 무엇인가?

익스트림 프로그래밍 ?

❖ 익스트림 프로그래밍 (http://www.extremeprogramming.org)

- 기본 개념은 여러 가지 효과가 좋은 실천방법(XP Rules)을 극한으로 실천하는 것
- 전통적인 소프트웨어 개발 방법론과는 달리 문서를 강조하지 않고 변경을 장려
- **모호하고 빠르게 변하는 요구사항을 가지고 일을 하는 중소규모 팀을 위한 경량화 된 방법론 (Kent Beck)**

❖ XP 특징

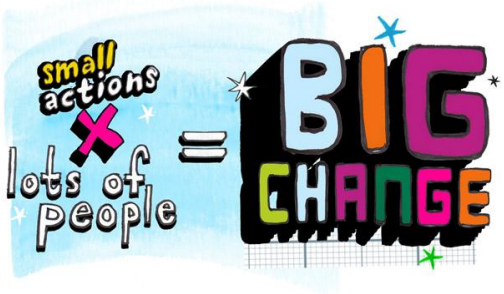
- 5명에서 10명 정도로 구성된 프로그래머가 고객과 함께 한 장소에서 일한다.
- 개발은 점진적으로 이루어지고, 각 단계마다 산출물의 기능을 덧붙여 나간다.
- 요구사항 정의 시에는 사용자가 새로운 기능을 스토리 형태로 구체적으로 작성한다.
- 프로그래머는 엄격한 코딩 규칙을 준수하고, 짝(Pair)을 이루어 일하며, 단위 테스트를 실시한다.
- 요구사항, 아키텍처, 디자인은 프로젝트 중간에 언제든지 발생한다.



깨어 있고, 적응하며, 변화하라

❖ **소프트웨어의 모든 것은 변한다.**

- 요구사항, 설계, 비즈니스, 기술, 팀, 팀원 등 모든 것은 변한다.
- 변화가 문제가 아니라, **변화를 극복하지 못하는 것이 문제다.**
- 작은 변화를 빈번하게 하는 것은 목표를 향한 변화에 적응을 쉽게 한다.



❖ **가치를 표현하기 위한 행동들은 장소, 시간, 팀에 따라 다르다.**

- 팀의 실천사항 중 일부는 팀을 목표로 이끌고, 일부는 목표로 부터 멀어지게 한다.
- 각각의 실천사항은 효율성, 의사소통, 자신감, 생산성을 개선하는 실험이다.

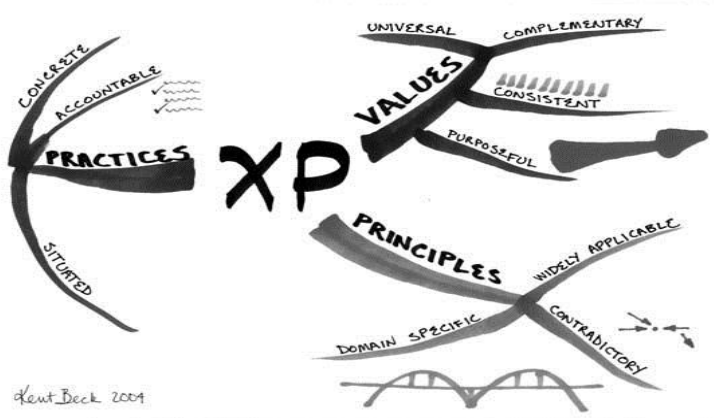
XP inside (1/2)

❖ 익스트림 프로그래밍(XP)은 …

- 소프트웨어 개발 기법이다
  - ✓ 프로그래밍 기법
  - ✓ 의사소통 기법
  - ✓ 팀워크(Team work)의 적용

상상 그 이상의 것을
성취하는 것에 집중하게
해주는 방법론이 XP이다.

- 다음의 사항을 포함한다.
  - ✓ 의사소통, 피드백, 단순성, 용기, 존중과 같은 **가치**들에 바탕을 둔 소프트웨어 개발 철학
  - ✓ 소프트웨어 개발을 개선하는데 쓸모가 있다고 증명된 **실천방법**들의 집합
  - ✓ 상호 보완적인 원칙들 : 가치를 실천방법으로 옮기는 지적인 기법들의 집합
  - ✓ 가치를 공유하고, 실천방법들을 공유하는 공동체
- 다음의 특징을 가진다.
  - ✓ 가볍다
  - ✓ S/W 개발의 제약 조건을 다루는 것에 기반한다.
  - ✓ 팀의 규모와 상관없이 적용 가능하다.
  - ✓ 모호하거나 빠르게 변화하는 상황에 적응이 가능



XP inside (2/2)

❖ 익스트림 프로그래밍(XP)이 다른 방법론과 다른 점은 …

- 짧은 개발 주기
  - ✓ 개발 초기부터 구체적이고 지속적인 반응을 얻게 해준다.
- 점진적 계획 접근
  - ✓ 전반적인 계획은 빨리 만들어 시작한 후, 프로젝트 동안 계획이 진화한다.
  - ✓ 기능 구현 일정을 유연하게 세워 변경되는 요구에 대응할 수 있는 능력
- 자동화된 테스트
  - ✓ 개발의 진정 상황을 관찰하고, 시스템이 진화할 수 있도록 한다.
  - ✓ 개발 초기부터 결함을 잡을 수 있도록 자동화된 테스터를 작성한다.
- 구두 전달, 테스트, 소스 코드에 의존
  - ✓ 시스템 구조와 시스템의 의도를 전달
  - ✓ 열심히 참여하는 개인들 사이의 긴밀한 협력에 의존
- 실천방법에 의존
  - ✓ 팀 구성원의 단기적 본능과 프로젝트의 장기적 이해관계 모두에 적용이 가능

XP의 Risk 관리

❖ XP는 잠재된 위험들에 대처하는 개발 규율

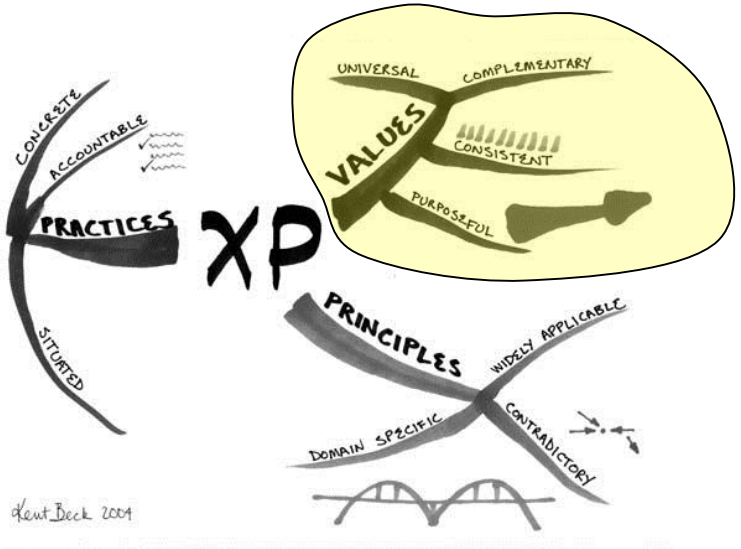
위험요소	XP에서의 대응 방법	효과
일정밀림	▪ 릴리즈 주기가 짧아야 한다.	▪ 개발팀이 주기 내에 문제를 해결할 수 있다.
프로젝트 취소	▪ 비즈니스에 가장 중요하고 작은 릴리즈를 우선 개발	▪ 배치 전 잘못될 가능성을 적어진다. ▪ 소프트웨어의 가치를 최고로 유지
시스템 이상	▪ 포괄적인 자동화 테스트 Suite을 만들고 유지한다. ▪ 시스템의 변화 시 테스트가 수행된다.	▪ 언제나 시스템을 배치 가능한 상태로 유지 ▪ 문제가 축적되는 것을 허용하지 않는다.
결함 비율	▪ 함수 단위로 테스트를 작성 – 프로그래머 ▪ 기능 단위로 테스트를 작성 - 고객	▪ 개발 품질과 제품 품질의 수준 유지
비즈니스에 대한 오해	▪ 비즈니스 파트의 인원도 정규 구성원	▪ 프로젝트 명세가 계속해서 정련된다. ▪ 고객의 요청이 개발에 빠르게 반영된다.
비즈니스의 변화	▪ 릴리즈 주기를 짧게 한다.	▪ 하나의 릴리즈에 처리할 양이 줄어든다. ▪ 새로운 기능의 요청이 자유롭다.
이름뿐인 기능	▪ 우선 순위가 높은 작업만 다룬다.	▪ 불필요한 작업이 줄어든다.
직원 교체	▪ 추정이 정확해지도록 프로그래머에게 피드백을 준다. ▪ 프로그래머의 추정을 존중해준다. ▪ 인간적인 접촉을 장려한다.	▪ 일에 대한 만족감과 책임감 증가 ▪ 팀간 의사소통의 증가

그래서 XP란 무엇인가 ?

❖ **"XP란 무엇인가"에 대한 답변은 ...**

- 오래되고 효과 없는 사회적 습관 대신 효과 있는 새로운 습관을 채택하는 것이다.
- 오늘 내가 기울인 모든 노력에 대해 자신이 인정해 주는 것이다.
- 내일은 좀 더 잘해보려고 애쓰는 것이다.
- 팀 전체가 공유하는 목표에 내가 얼마나 기여했는지 스스로 평가하는 것이다.
- 소프트웨어 개발을 하는 중에도 인간적인 요구 일부를 채우겠다고 요구하는 것이다.

소프트웨어 개발의 결과는 운에 의해 결정되는 것이 아니다.

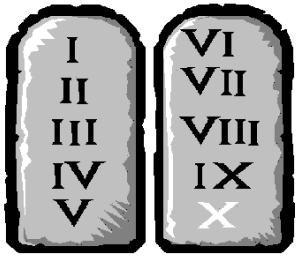


가치(Values), 원칙(Principles), 실천방법(Practices)

가치, 원칙, 실천방법

❖ 가치(Values)

- 가치는 좋아하고, 좋아하지 않는 것에 대한 근원
  - ✓ 가치가 없는 실천은 기계적인 활동이 되기 쉽다 ➔ 가치는 실천방법에 목적을 부여
- XP에서는 팀이 공유하는 중요한 5개의 핵심 가치를 정의
  - ✓ 의사소통(Communication), 단순성(Simplicity), 피드백(Feedback), 용기(Courage), 존중(Respect)

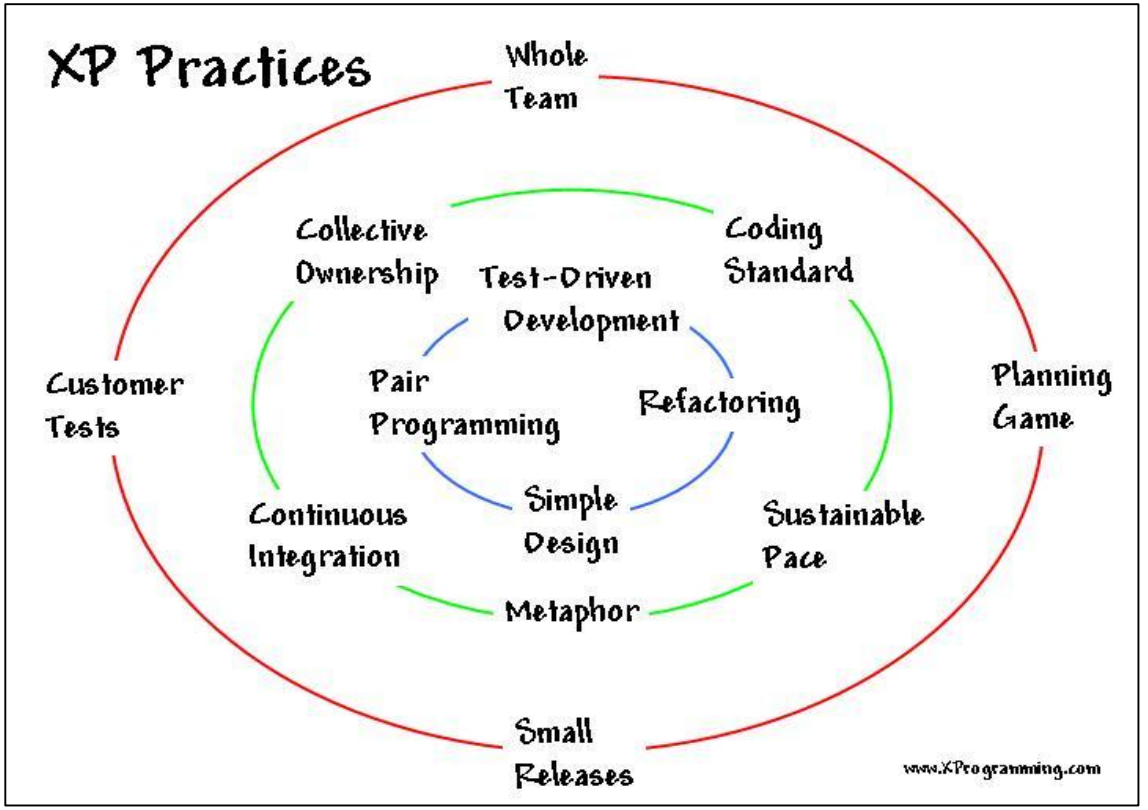


❖ 원칙(Principles)

- 특정 영역에서의 지침
- XP 방법을 규정짓는 14가지 기본원리

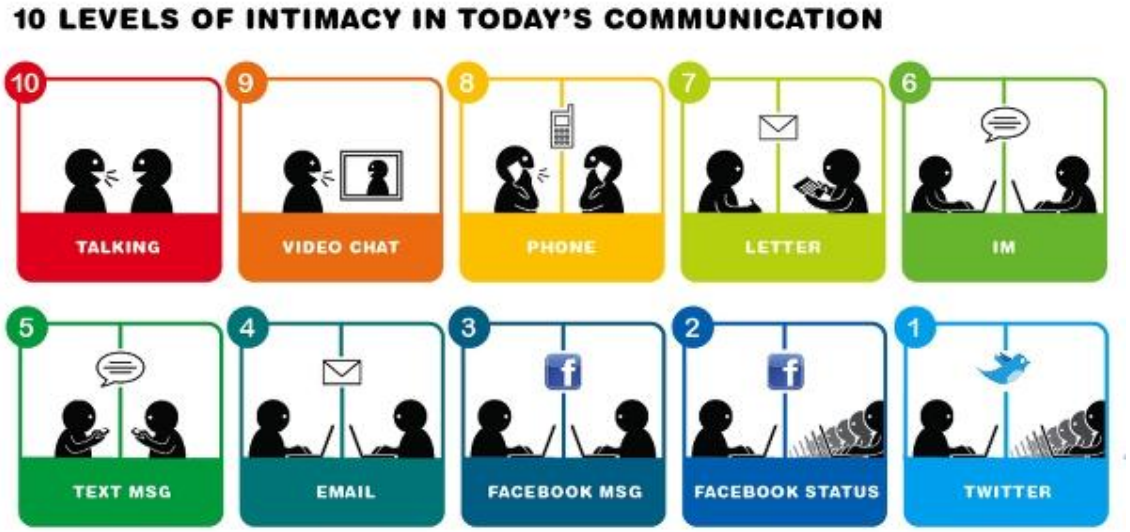
❖ 실천방법(Practices)

- XP가 실제로 동작하는 방법



❖ 의사소통(Communication)

- 의사 소통은 팀 개발에 있어서 가장 중요한 요소
  - ✓ 개발 과정의 문제는 이미 해결책을 알고 있는 경우가 많다.
  - ✓ 문제 해결 방법이 문제의 당사자에 전달되지 못하는 경우가 많다.
  - ✓ 문제를 찾은 다음에는 의사소통이 도움이 된다.

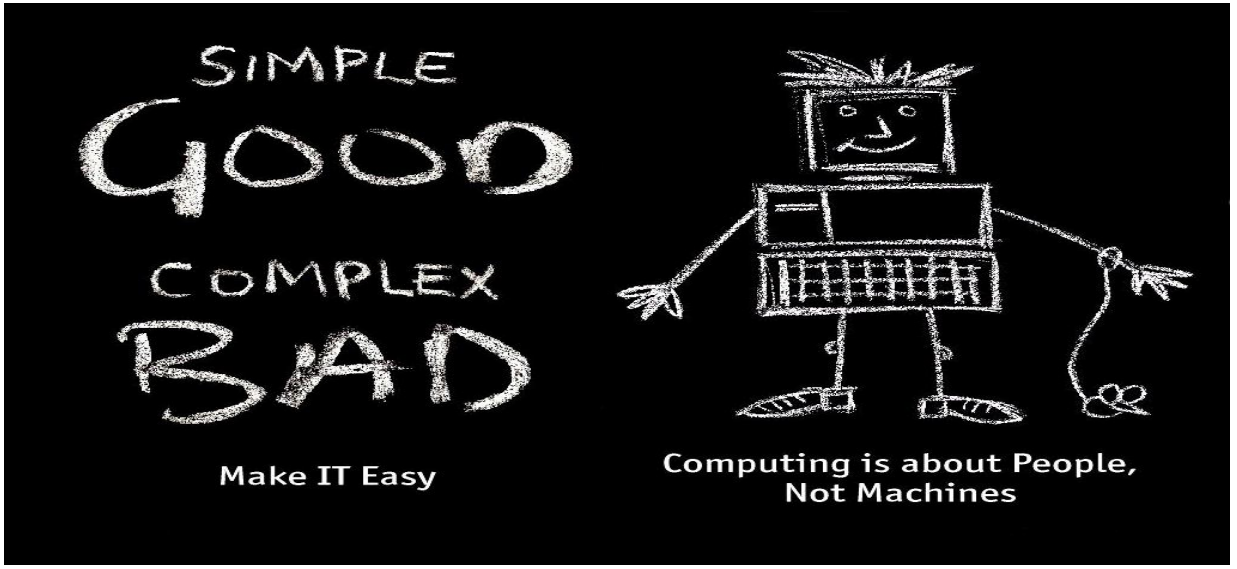


- 한 팀이라는 느낌을 만들고, 효과적으로 협동하려면 …
  - ➔ 팀원 간의, 팀과 고객 사이의 의사소통은 필수적이다.

가치(Values)

❖ 단순성(Simplicity)

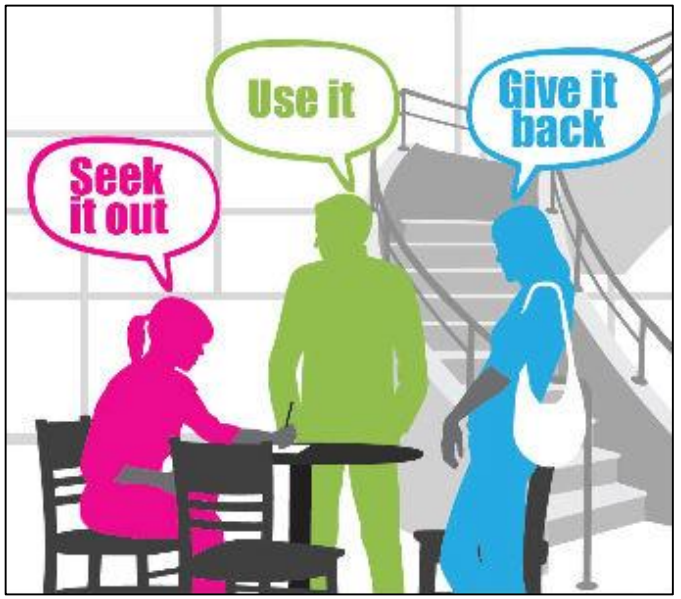
- 꼭 필요한 것만 한다.
  - ✓ 간단한 설계 부분은 코드로 바로 작성
  - ✓ 복잡한 설계 부분은 지속적으로 잘못된 부분을 리팩토링
- 불필요한 복잡성을 줄인다.



- 단순성의 의미는 상황에 따라 결정된다.
  - ✓ 단순성을 성취하면, 의사소통 해야 할 것도 줄일 수 있다.

❖ 피드백(Feedback)

- 소프트웨어 개발에서 변화는 불가피 ➔ 변화는 피드백을 필요
  - ✓ 한번에 완벽하게 해결되기보단, 점진적 개선에 만족해야 하는 경우



- 피드백을 이용하면 목표에 점점 다가갈 수 있다.
  - ✓ 빠른 피드백은 프로젝트 팀과 고객의 이해를 도우며 조율이 가능하게 함
  - ✓ 피드백의 단위는 상황에 따라 다르지만, 빠를수록 좋다.
  - ✓ 피드백은 의사소통의 핵심

❖ 용기(Courage)

- 두려움에 직면했을 때 가장 효과적인 행동
  - ✓ 문제가 무엇인지 안다면, 그것에 대해 무슨 일이든 해보는 것
  - ✓ 단, 결과를 생각하지 않고 일을 하는 것은 효과적이지 않음

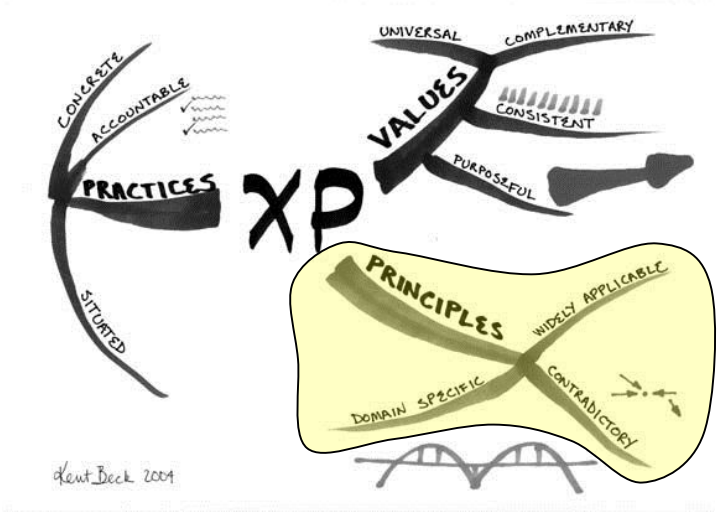
- 때때로 용기는 인내로 표현
  - ✓ 문제를 정확히 모르는 경우, 기다리는 것도 필요
- 용기는 다른 가치와 조화를 이룰 때 강력해진다.
  - ✓ 진실을 말할 수 있는 용기는 의사소통과 신뢰를 증대
  - ✓ 실패하는 해결책을 버리고 새로운 해결책을 찾는 용기는 단순함을 증대
  - ✓ 진짜 답변, 구체적인 답변을 추구하는 용기는 피드백을 낳는다.

가치(Values)

❖ 존중(Respect)

- XP는 팀원 간의 상호 존중을 강조한다.
  - ✓ XP는 기본적으로 팀 기반의 활동이다.
    - 팀원간 존중이 있어야 공동작업, 의사소통, 피드백이 가능
  - ✓ 팀원 간의 상하관계나 중요도의 차이는 없다고 가정

- S/W 개발에서 생산성과 인간성을 동시에 개선하려면 …
  - ✓ 팀에 속한 모든 개인의 기여를 존중해야 한다.
    - 모든 사람은 인간으로서 동등한 가치를 지닌다.
    - 다른 사람보다 본질적으로 더 가치 있는 사람은 없다.



가치(Values), 원칙(Principles), 실천방법(Practices)

원칙(Principles)

❖ **원칙을 이용하면, 실천방법을 더 잘 이해할 수 있다.**

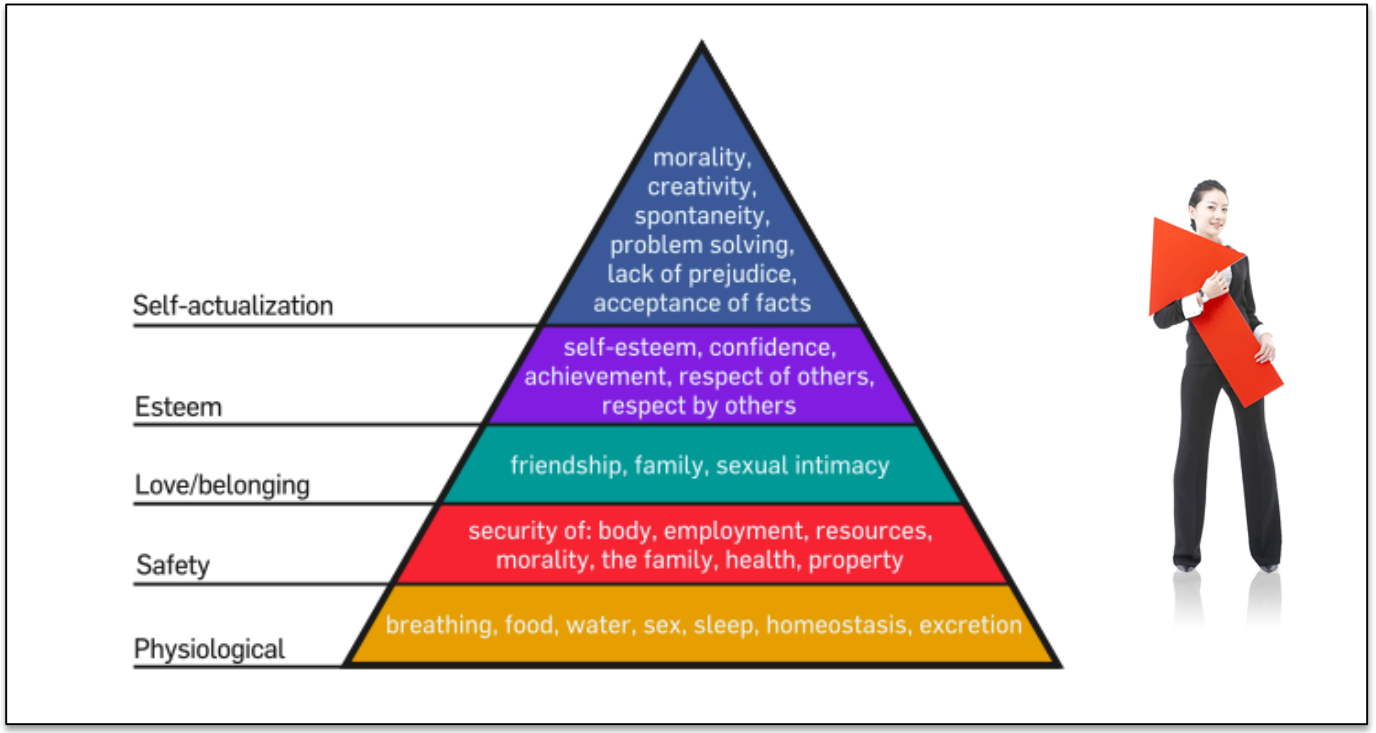
- 상황과 환경에 맞추어진 실천 방법들의 목록은 존재하지 않는다.
- 목적에 맞는 실천방법이 없다면, 새로운 실천방법을 고안하라

❖ **원칙은 실천 방법이 달성하기 위한 목적을 알려준다.**

- 원칙을 이해하면, 목표와 조화를 이루면서도, 기존의 실천방법과 조화롭게 작동하는 새로운 실천방법을 만들 수 있다.

❖ 인간성(Humanity)

- 소프트웨어는 사람이 개발하고, 사람을 위해 쓰인다.
  - ✓ 사람에게 초점을 맞추고, 자신에게 알맞은 프로세스를 만들어 문제를 해결
- 좋은 개발자가 되기 위해 필요한 것들
  - ✓ 기본적인 안전, 성취감, 소속감, 성장, 친밀감
- 개인의 욕구와 팀의 욕구간의 균형을 맞추는 것이 필요



원칙(Principles)

❖ 경제성(Economics)

- 기술적 성공에만 집착하지 않고, 비즈니스 목표와 필요를 충족
- 경제성은 다음의 두 가지 측면에 영향을 받음
  - ✓ 돈의 시간적 가치
    - 돈은 일찍 벌어들이고, 나중에 지불하는 것이 더 가치가 있다.
  - ✓ 시스템과 팀의 기술적 가치
    - 미래의 선택 사항을 늘릴 수 있다.

❖ 상호이익(Mutual Benefit)

- 모든 활동은 그 활동에 관련된 모든 사람에게 이익이 되어야 함
  - ✓ 한쪽만 이익이 되는 해결 방법은 결과적으로 손해가 나는 경우가 많다.
  - ✓ 지금 내게 이익이 되고, 나중에도 이익이 되고, 고객에게 이익이 되게 …
- 상호이익이 되기 위해서는 …
  - ✓ 해결책이 야기하는 문제 < 해결책이 처리하는 문제의 수

원칙(Principles)

❖ 자기 유사성(Self Similarity)

- 어떤 해결책의 구조를 다른 맥락에 적용한다.
  - ✓ 규모의 차이는 문제가 되지 않는다.
- 자기 유사성은 언제가 효과가 있는 것은 아니다.
  - ✓ 한 맥락에서 효과가 있다고 모든 상황에 효과적이지는 않다.

❖ 개선(Improvement)

- "완벽하다"는 없고, "완벽해지도록 노력한다"
  - ✓ XP는 개선을 통해 탁월한 소프트웨어 개발에 도달하려는 것
  - ✓ 현실과 이상의 간극을 좁히기 위해 날마다 노력하는 것은 가능하다
  - ✓ 새로운 기술적 효용성을 이용해 효과적인 구조로 가져가는 것
- 어떤 것을 시작하면 좋을지 찾아보고, 일단 시작 후 개선하라

원칙(Principles)

❖ 다양성(Diversity)

- 개발에 있어서 문제와 함정을 발견하려면 …
  - ✓ 팀 안에 다양한 종류의 기술, 사고방식, 관점들이 있어야 한다.
  - ✓ 문제를 해결하기 위해 협동할 것과 서로의 의견을 존중해야 한다.
- 다양성이 있으면 갈등도 있기 마련이다.
  - ✓ 갈등이 없는 팀은 없다. 중요한 것은 갈등을 생산적으로 풀 수 있느냐다

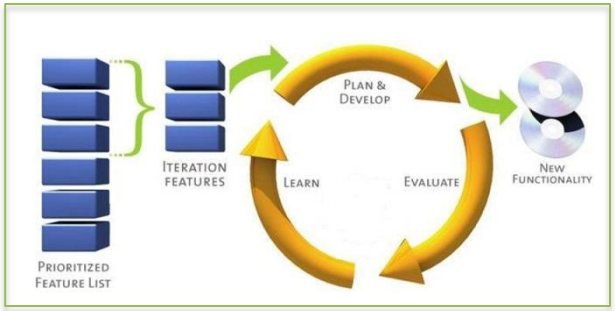
❖ 반성(Reflection)

- 좋은 팀은 일만하지 않고, 어떻게, 왜 일하는지를 고민한다.
  - ✓ 자신들의 성공과 실패를 분석하고, 실수를 통해 배운다.
  - ✓ 반성이 지나친 경우, 실제 개발에 문제가 생기지 않아야 한다.
  - ✓ 배움이란 행동이 반성을 거친 것이다.
  - ✓ 지성이 조율된 감정은 통찰력의 원천이다.

원칙(Principles)

❖ 흐름(Flow)

- 개발의 모든 단계를 동시에 작업함
  - ✓ 가치 있는 S/W를 끊임없이 제공하는 것
  - ✓ 개선을 위해 가치를 조금씩, 점진적으로, 계속해서, 자주 배치한다.
  - ✓ 흐름에서 떠나게 될 때 마다 반드시 제자리로 돌아오겠다고 결심해야 한다.



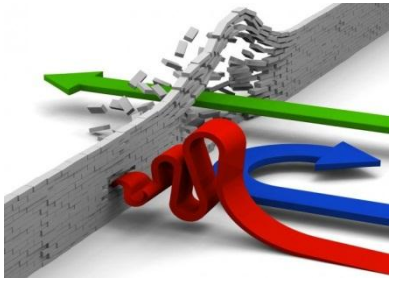
❖ 기회(Opportunity)

- 문제를 기회를 보도록 생각을 전환하라
  - ✓ 뛰어난 실력을 갖추려면, 문제를 단지 생존의 문제가 아닌, 배움과 개선의 기회로 전환할 줄 알아야 한다.
  - ✓ 문제를 기회로 전환할 기회는 개발 과정 곳곳에서 생긴다.
    - 전환은 강점을 늘리고, 약점을 최소화 할 수 있는 기회이다.

원칙(Principles)

❖ 중복(Redundancy)

- 핵심적이면서도 해결하기 어려운 문제에는
  - ✓ 해결 방법을 여러 개 마련해 놓아야 한다.
- 중복은 자원을 소모한다.
  - ✓ 그러나 중복이 위해 드는 비용보다 재앙을 회피하는 이득이 더 큰 경우
  - ✓ 정당한 목적이 있는 중복은 제거하지 않도록 해야 한다.



❖ 실패(Failure)

- 실패를 통하여 교훈을 얻는다.
  - ✓ 실패가 지식을 늘려주는 한, 그것은 허비가 아니다.
- 무엇을 해야 할지 모르는 경우
  - ✓ 실패를 감수하는 것이 성공으로 가는 가장 짧고, 확실한 길이다.

원칙(Principles)

❖ 품질(Quality)

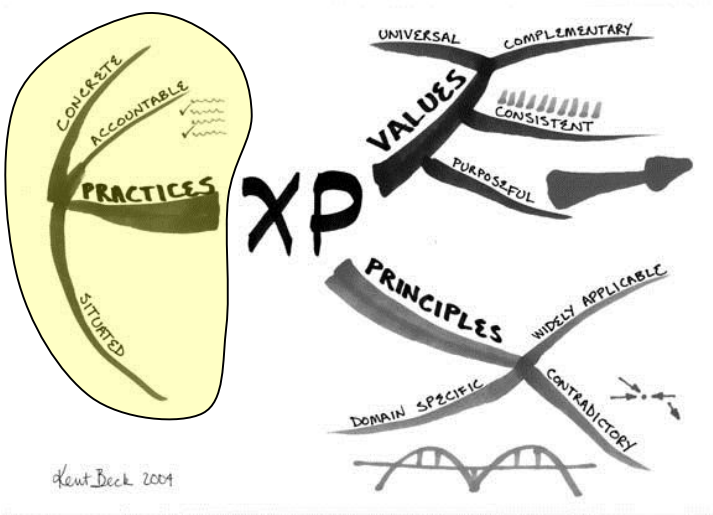
- 품질을 희생하는 것은 프로젝트의 관리 수단으로 비효과적이다.
  - ✓ 확실한 방법을 모른다면, 가장 최선책이고 생각하는 방법으로 한다.
  - ✓ 시간 안에 할 수 있는 방법으로 최대한 진행 후 , 마무리를 생각한다.
- 품질이 주는 이익에는 한계가 없다.
  - ✓ 품질을 높이기 위한 이해의 한계만이 존재한다.
  - ✓ 프로젝트를 계획하고, 기록하고, 방향을 잡기 위한 수단은 범위이다.

❖ 잔걸음(Baby Step)

- 중요한 변화를 한번에 몰아서 시도하는 것은 위험하다.
  - ✓ 단계를 작게 나누는 것이 실패해서 다시 시작하는 것보다 효율적이다.
  - ✓ 단, 위의 언급이 정체나 느린 변화를 정당화하지 않는다.
- 변화의 요구 대상은 사람이다.
  - ✓ 사람들이 변하는 속도에는 한계가 있다.

❖ 수용된 책임감(Accepted Responsibility)

- 어떤 일을 하겠다고 한 사람이, 그 일의 평가도 내리는 것
  - ✓ 스토리 구현 책임자 → 스토리의 설계, 구현, 테스트까지 책임
- 책임이 있다면, 권한도 따라온다.
  - ✓ 단, 채임과 권한이 잘못 연결하면, 팀의 의사소통이 왜곡된다.
  - ✓ 잘못된 책임과 권한의 연결은 감정적인 부담의 비용도 유발한다.



가치(Values), 원칙(Principles), 실천방법(Practices)

실천 방법(Practices)

❖ 실천 방법

- XP이 매일 실천하는 것이지만, 절대적인 것은 아니다.
- 실천 방법 자체만으로는 부족하다.
  - ✓ 가치를 통해 목적을 가지지 않은 실천 방법은 기계적인 규칙일 뿐이다.
  - ✓ 동기와 목표를 제공해야 한다.
- 실천 방법은 상황에 따라 달라져야 한다.
  - ✓ 상황에 맞추어 적절한 실천 방법을 선택해야 한다.
  - ✓ 문제 영역이 달라지면, 새로운 원칙이 필요할 수 있다.
  - ✓ 그러나, 가치는 새로운 상황에 따라 변할 필요가 없다.
  - ✓ 어떤 실천 방법을 적용할 것인지는 팀의 선택에 달려있다.
- 실천 방법은 함께 사용할 때 효과가 증폭된다.
  - ✓ 새로운 실천방법을 최대한 빨리 받아 들일수록 팀에는 이득이 된다.

❖ 함께 앉으라(Sit Together)

- 팀 전체가 들어가는 충분할 정도로 크고 열린 공간에서 작업하라
  - ✓ 팀이 준비되기 전 공간을 나누는 파티션은 생산성을 저해한다.
  - ✓ 기술적인 해결책만으로 충분하지 않는 경우가 많다.
- 함께 같은 공간에서 모든 감각을 이용하는 의사소통이 중요하다.
  - ✓ 필요한 경우 함께 앉기를 천천히 진행시켜도 된다.
  - ✓ 팀원이 물리적 공간의 가까움이 의사소통을 향상시킨다는 사실을 이해한다면 기꺼이 공간을 개발할 것이다.

❖ 하나의 팀(Whole Team)

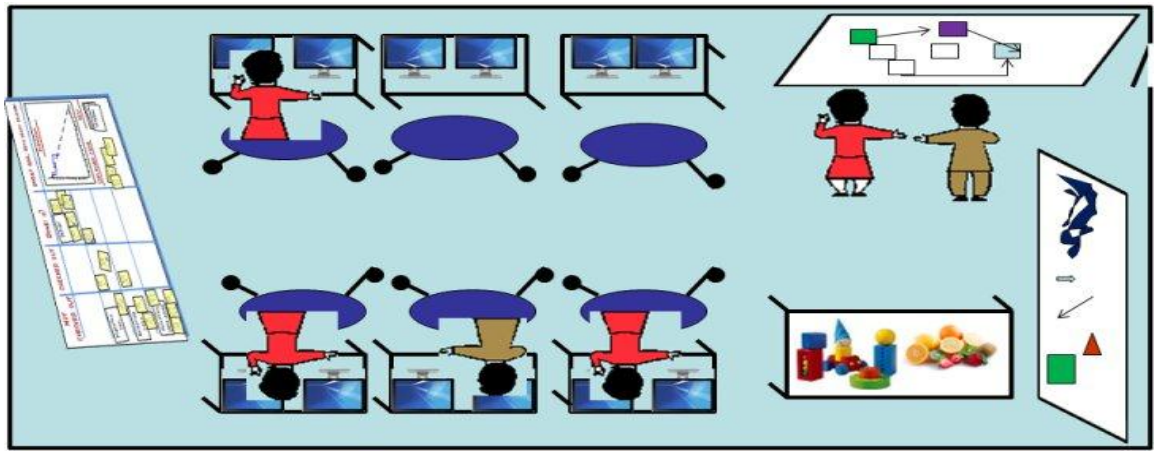
- 필요한 기술과 시야를 지닌 사람들을 모두 팀에 포함시켜라.
- 사람들은 "팀"에 속한다는 느낌이 필요하다.

우리는 소속되어 있다.
우리는 이 안에 함께 있다.
우리는 서로의 작업, 성장을 돕는다.

- 팀을 구성하는 요소는 동적으로 변한다.
  - ✓ 팀에 필요한 사람을 데려오고, 필요 없는 사람은 다른 곳으로 간다.
  - ✓ 이상적인 팀의 규모 ➔ 팀의 지속성이 끊어지는 두 지점 12와 150.
    - 규모가 너무 크다면 적절한 규모로 나누어야 한다.
- 팀에 한 사람의 일부분만을 소속시키는 것은
  - ✓ 팀에 속한다는 정체성을 파괴한다.
  - ✓ 작업의 전환에 따른 생산성을 떨어뜨리는 요인이다.

❖ 정보를 제공하는 작업공간(Informative Workspace)

- 작업 공간을 작업에 대한 것들로 채워라
  - ✓ 누구든지 팀이 사용하는 공간에 들어와서 15초안에 프로젝트가 어떻게 진행되는지 대략적인 감을 잡을 수 있어야 한다.
  - ✓ 더 자세히 관찰할 경우, 지금 있는 문제와 앞으로 생길지 모르는 문제에 대한 정보를 얻을 수 있어야 한다.
- How to ?
  - ✓ 벽에 스토리 카드를 붙이는 방법을 활용
  - ✓ 크고 잘 보이는 차트의 활용한다
  - ✓ 차트가 필요 없다면, 지금 일어나는 주용한 정보를 위해 사용한다.

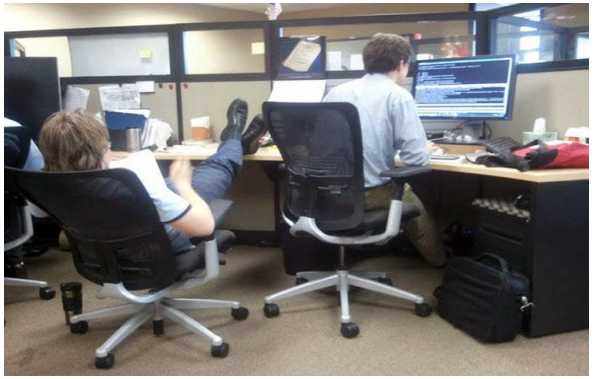


❖ 열정적인 작업(Energized Work)

- 생산적으로 일할 수 있는 정도의 시간만 일하라.
  - ✓ 근무 시간은 점진적으로 개선할 수 있다.
  - ✓ 40시간 / 1주
- 그리고 일의 활력을 유지할 수 있는 정도의 시간만 일하라.
  - ✓ 하루를 혹사하고, 다음 이틀을 망치는 것은 팀과 자신에게 손해다.
  - ✓ 같은 양의 시간을 더욱 잘 활용하기 위해 노력하라.
  - ✓ 피곤할 때는 스스로 가치를 떨어뜨리고 있다는 것을 깨닫기 힘들다.
- 소프트웨어 개발은 통찰력의 싸움이다.
  - ✓ 통찰력은 준비되고, 잘 쉬고, 긴장이 풀린 마음에서 생겨난다.

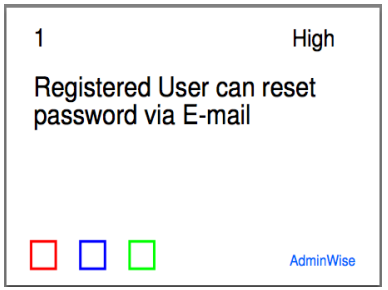
❖ 짝 프로그래밍(Pair Programming)

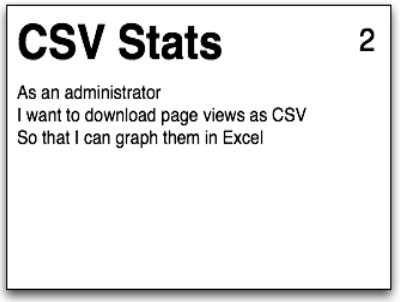
- 모든 프로그램을 두 사람이 컴퓨터 한 대에 앉아 작성하라.
  - ✓ 두 사람이 편안하게 나란히 앉을 수 있도록 컴퓨터를 배치
  - ✓ 마우스와 키보드를 서로 주고 받을 수 있도록 한다.
- 짝 프로그래밍을 한다고 해도, 혼자 생각할 시간은 필요하다.
  - ✓ 필요하면, 혼자서 고민하고 작업하는 것도 가능하다.
  - ✓ 단, 고민이 끝난 경우, 다시 동료와 함께 작업한다.
- 짝을 자주 바꾸도록 한다.
  - ✓ 더 많은 사람과 공유하고, 프로젝트 전체에 도움이 되도록 한다.
  - ✓ 개인적인 차이를 존중하는 것이 중요하다.



❖ 스토리(Story)

- 스토리와 요구사항은 다르다.
  - ✓ 요구사항은 변화를 포용하지 못하게 한다. (강제성 때문에...)
- 고객이 볼 수 있는 기능 단위로 계획을 짜라.
  - ✓ 스토리를 작성 후 구현에 필요한 개발 노력을 추정하라.
    - 스토리 실천방식과 요구사항의 차이는 일찍 추정하기이다.
  - ✓ 스토리를 인덱스 카드에 적어 자주 다니는 벽에 붙여놓는다.
- 스토리에는 짧은 글이나 그림 설명 외에, 이름도 필요하다.
  - ✓ 새로운 정보가 추가되면 상황을 이해하는데 도움이 된다.





❖ 주 단위 주기(Weekly Cycle)

- 한 번에 일주일 분량의 일을 계획하라.
  - ✓ 주초에 주간 회의를 통하여 계획한다.
  - ✓ 계획에 들어가는 시간을 조금씩 줄이도록 노력해야 한다.
  - ✓ 주초에 자동화 Test case를 작성하면, 스토리의 구현이 빨라진다.
- 주간 회의에서는 …
  - ✓ 진행 사항의 확인 (지난 주 실제 진행 사항 검토 포함)
  - ✓ 금주의 구현 스토리를 고객이 고르도록 한다.
  - ✓ 스토리를 여러 태스크(Task)로 나누고, 팀원에 할당
  - ✓ 각각의 태스크에 대한 추정을 한다.

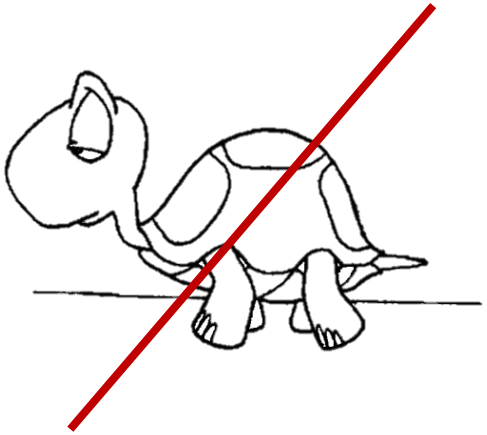
주 단위의 맥박 치는 프로젝트 진행은,
팀과 개인 차원의 실험을 편리하고, 빈번하게
예측 가능하게 할 수 있는 환경을 제공한다.

❖ 사분기 주기(Quarterly Cycle)

- 한 번에 한 분기 분량의 일을 계획하라.
  - ✓ 분기마다 한 번씩, 팀과 프로젝트, 진행 정도 목표 등을 놓고 검토한다.
- 분기별 계획에서는
  - ✓ 병목, 특히 팀의 힘이 미치지 못하는 외부에서 생기는 병목을 찾아본다.
  - ✓ 수선(Repair) 작업을 시작한다.
  - ✓ 이블 분기의 주제(Theme)을 계획하고, 한 부기 분량의 스토리를 고른다.
  - ✓ 프로젝트가 조직에서 차지하는 위치에 초점을 맞춘다.

❖ 느슨함(Slack) - 여유

- 계획에는 일정이 뒤쳐질 경우 포기할 수 있는 덜 중요한 태스크를 포함하라.
  - ✓ 계획된 것을 지키는 것도 중요하다.
  - ✓ 약속을 지키는 것은 긴장을 완화하고 신뢰를 회복시켜준다.
  - ✓ 예정보다 더 많은 스토리를 제공하는 것은 가능하다.
    그러나 과잉 약속은 낭비를 부른다.
  - ✓ 팀원에게 고르게 업무를 배당하는 것에 중점을 두어야 한다.



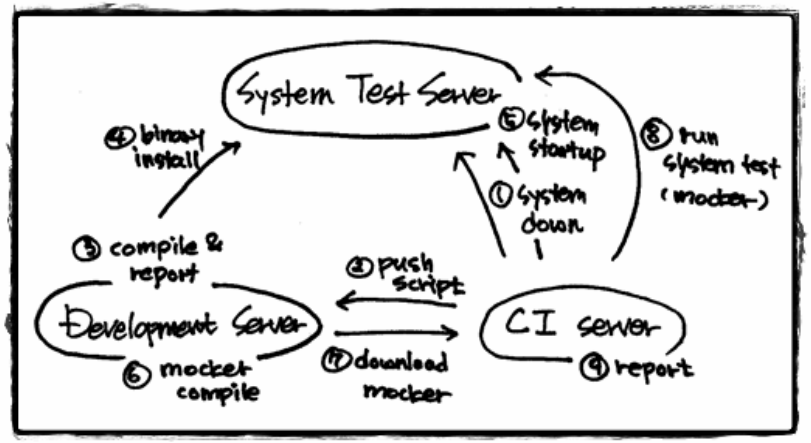
❖ 10분 빌드(Ten-minute Build)

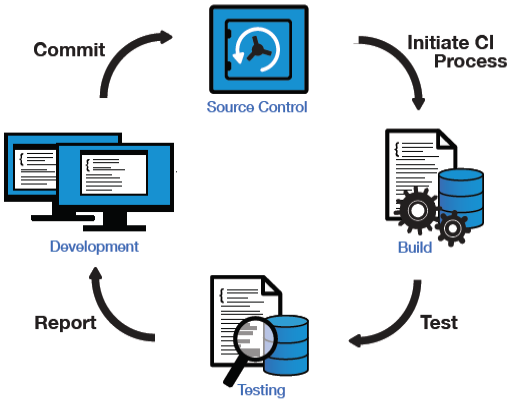
- 10분만에 자동으로 시스템을 빌드하고, 테스트를 돌려라
  - ✓ 10분보다 빌드가 오래 걸린다면, 실행 횟수가 매우 줄어든다.
  - ✓ 10분보다 짧다면, 피드백과 휴식을 위한 시간이 없다.
- 자동화 된 빌드는 수작업이 필요한 빌드보다 가치가 있다.
  - ✓ 빌드가 자동화 되지 않았다면, 자동화를 위해 노력하라
  - ✓ 자동화와 테스트 수행을 연동하라
  - ✓ 일부만 자동화 된 빌드라도, 빌드를 하지 않는 것 보다 낮다.
  - ✓ 자동화 된 빌드는 긴박할 때, 스트레스 해소자가 되어준다.

❖ 지속적인 통합(Continuous Integration)

- 팀 구성원들이 작업한 것을 자주 통합하는 S/W 개발 실천방법
- 지속적으로 퀄리티 컨트롤을 적용하는 프로세스를 실행
  - ✓ 소프트웨어의 질적 향상
  - ✓ 소프트웨어 배포 시간의 단축
- 동기적인 방식 > 비동기적 방식
  - ✓ 동기적인 방식은 짧은 피드백 주기를 만들 수 있다.
- 통합과 빌드의 결과는 완제품이어야 한다.

Better,
Faster,
Cheaper

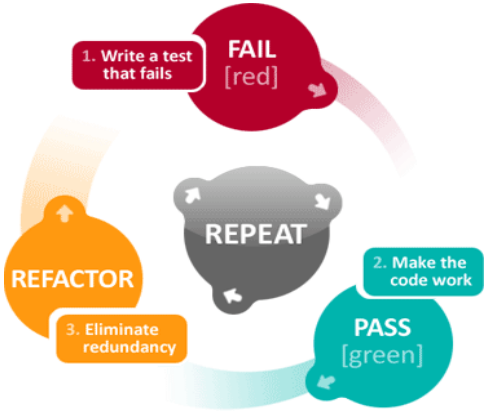


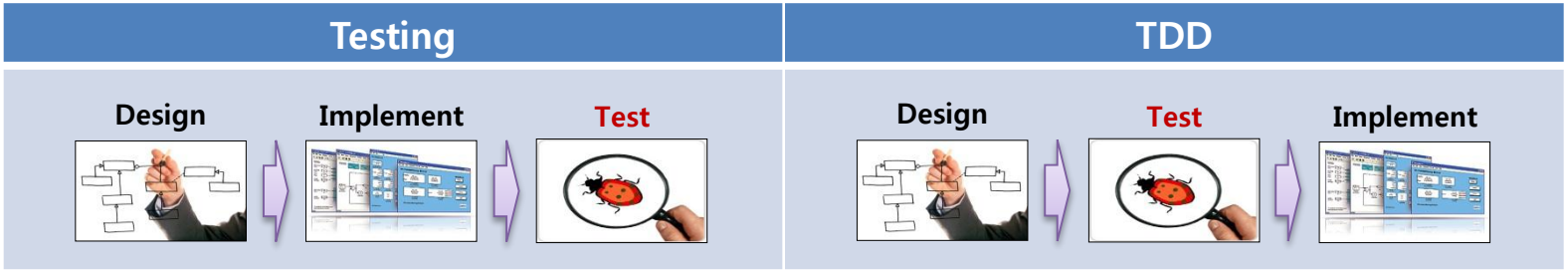


실천방법(Practices)

❖ 테스트 주도 개발(Test Driven Development)

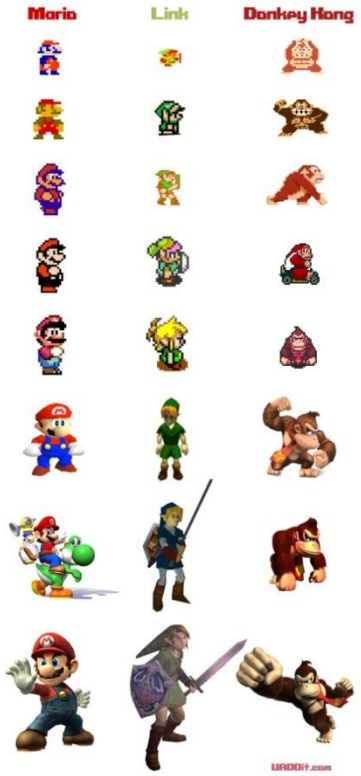
- 프로그램을 작성하기 전에 테스트를 먼저 하라
  - ✓ 테스트 코드를 만든 후 실제 프로그램 코드 작성
  - ✓ **"작성 종료 조건을 미리 정하고, 코딩을 시작한다."**
- 동작하는 깨끗한 코드를 작성하는 것이 목표
  - ✓ 짧은 개발 사이클을 반복한다.
  - ✓ 명세에 기반하여 결과를 확인한다.
- Testing 과 TDD는 다르다.





❖ 점진적 설계(Incremental Design)

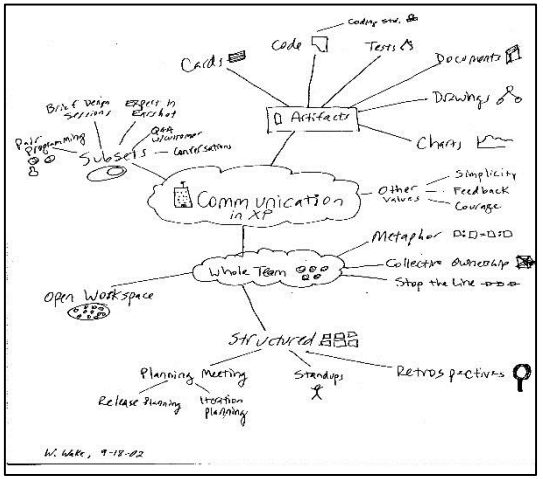
- 시스템의 설계에 매일 투자하라.
  - ✓ 어떤 설계가 가능한 한도에서 최선의 설계인지 이해하라
  - ✓ 현재의 설계가 최선의 설계와 일치하도록 점진적이지만, 지속적으로 노력
- 기존의 개발 방법과 다른 접근
  - ✓ 기존 방식은 설계를 마친 후 개발
    - 변경에 따른 비용 문제 때문
  - ✓ XP는 마지막 책임 지점까지 설계를 미룬다.
- 설계 투자를 최소화하는 것이 아니다.
  - ✓ 시스템의 필요에 비례하도록 설계 투자를 유지하라.
  - ✓ 시스템 설계 시 중복을 제거하라.
  - ✓ 경험을 얻기 전 설계를 하면 문제가 된다.
    - 설계 사용 시점과 가까울 때 작업하면 효율적
  - ✓ 리팩토링을 활용한다.

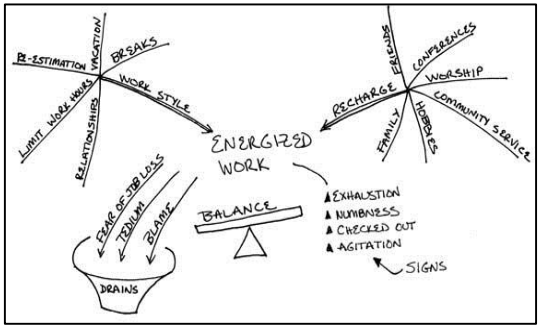


XP 적용하기

❖ Start XP

- 모든 사람에게 다 적합한 시작 지점은 없다.
  - ✓ 팀의 목표와 일치하고, 가치를 표현하는 실천 방법들을 추가하면서 적용한다.
    - 팀이 어떤 일을 하는지, 무엇을 이루고 싶은지를 살펴보고, 실천 방법을 선택한다.
    - 한 번에 하나씩 바꾸는 방식으로 시작하는 것이 좋다.
    - 꼭 느린 속도로 변해야 하는 것은 아니다.
- 변화의 필요함을 팀 모두가 공감해야 한다.
  - ✓ **변화는 깨달음과 함께 시작된다.**
    - 변화의 깨달음은 감정, 본능, 사실, 외부자의 피드백에서 나온다.
    - 수치화와 측정은 깨달음은 낳기도 한다.
  - ✓ **변화가 필요함은 인지한다면, 변화를 시작할 수 있다.**
    - 변화의 시작점은 기본 실천 방안을 활용한다.
    - 실천 방법에 대한 확신을 주기 못한다면 문제가 발생한다.
    - 실천 방법을 팀에 강요하면, 신뢰가 파괴되고, 원망만 ...
  - ✓ **먼저 실천 방법들의 지도를 활용하자**

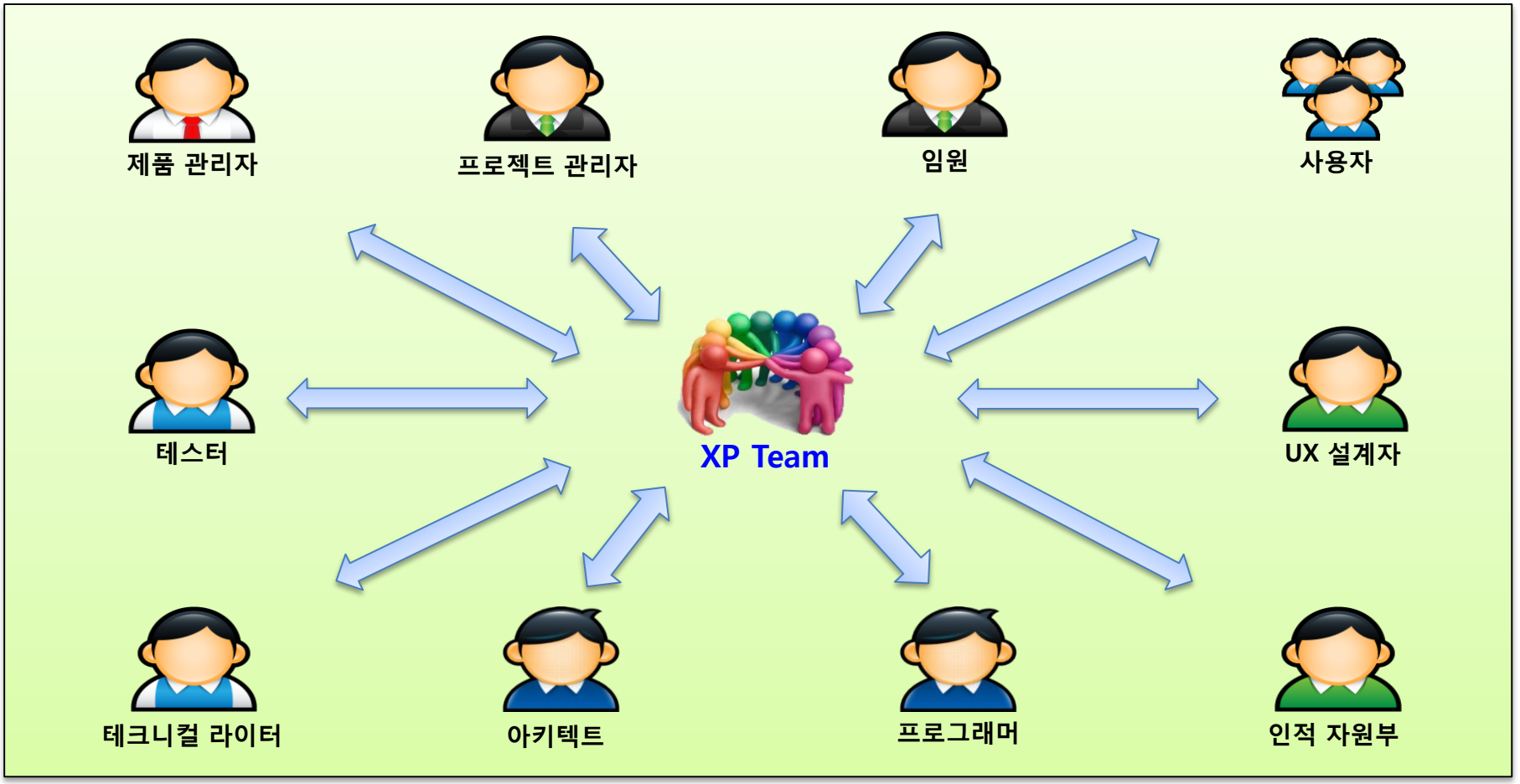




XP 팀

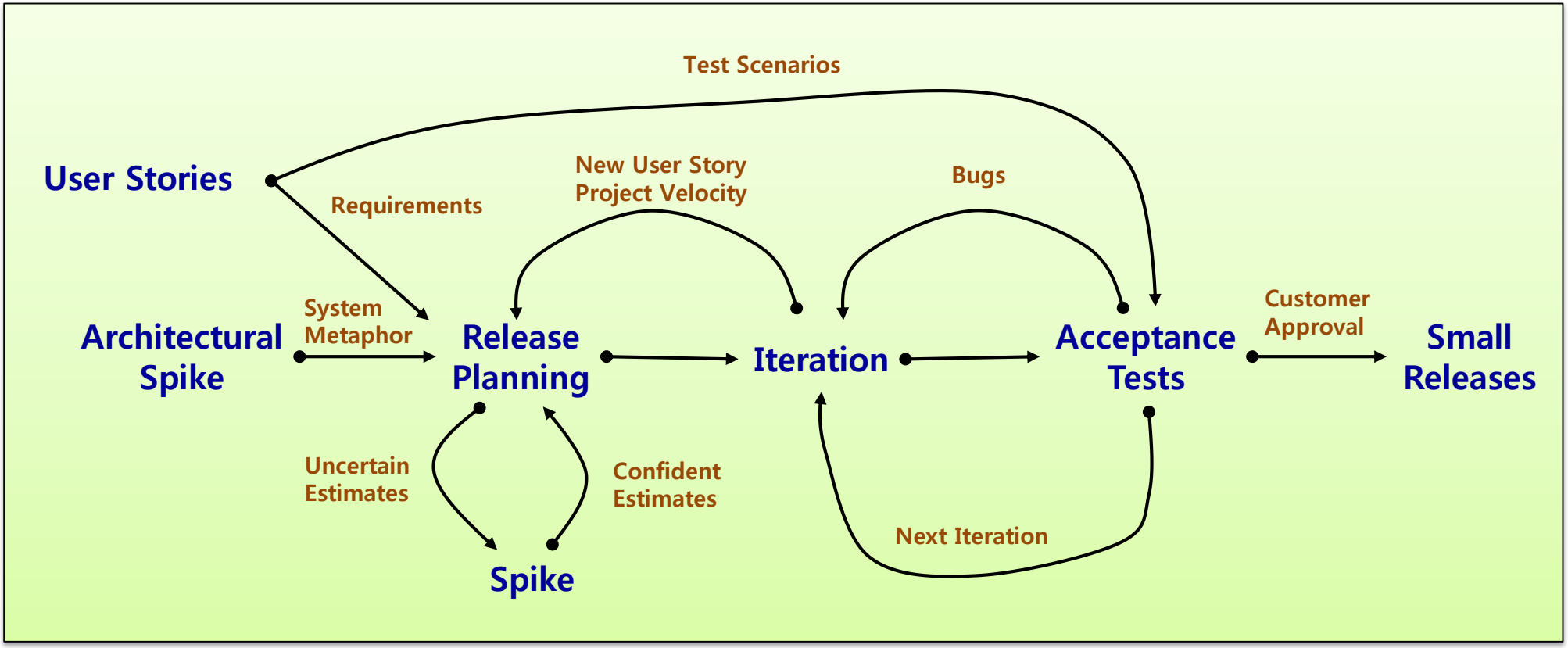
❖ **효율적인 SW 개발을 위해서는 많은 사람의 시각을 반영해야 한다.**

- 성숙한 XP 팀에서 역할들은 융통성 없이 고정된 것이 아니다.
  - ✓ 권위와 책임을 연결해야 한다.



❖ XP Process Model

- 출시 계획의 2가지 입력 : 사용자 스토리와 스파이크
- 테스트 시나리오의 작성 : 고객
- 프로세스 결과물 : 문제점에 대응하는 작은 출시물(들)



XP 제약사항

❖ XP가 효과적으로 적용되지 않는 경우

- 시간이 흐를수록 비용이 기하급수적으로 늘어나는 프로젝트 환경
- 안정성이 중요한 시스템을 개발하는 경우
  - ✓ 추적성에 대한 문서화가 필요
  - ✓ 외부 전문가에 의한 정형화 된 설계 검토가 필요
- 프로젝트에 정형화되거나, 문서화 된 요구사항, 분석 설계가 강요되는 경우
- 주당 40시간 근무가 어려운 프로젝트나 회사
  - ✓ 기존 방법론에 비해 프로그래머나 테스터의 업무 강도가 높아 초과 근무는 생산성은 낮춘다.
- 지속적인 통합과 테스팅이 불가능한 프로젝트
  - ✓ 시스템 테스트가 어렵거나 불가능한 경우 피드백이 어렵다.
  - ✓ 테스트가 어려운 경우, 코드가 완성됐다고 인정하지 않는다.
- 팀이 지리적으로 분리된 경우
  - ✓ XP는 함께 위치한 소규모 팀에서 가능하다.

[Appendix] 보조적인 실천방법들

진짜 참여 고객	▪ 통찰력 있는 고객은 분기별 계획과 일주일 단위의 계획에 참여할 수 있다. ▪ 개발 프로세스에 고객을 포함시키는 것은 신뢰를 조성하고 지속적인 개선을 북돋운다.
점진적 배치	▪ 당장 다룰 수 있는 작은 기능이나 제한된 데이터 집합을 찾고 배치한다.
팀 지속성	▪ 효율적인 팀은 계속 함께 하도록 하라 ▪ 팀을 하나로 유지하라고 해서 팀에 전혀 변화를 주지 말아야 한다는 의미는 아니다.
팀 크기 줄이기	▪ 팀의 능력이 신장되면, 작업량을 일정하게 유지하면서 점차 팀의 크기를 줄여라.
근본 원인 분석	▪ 개발 후 결함이 발견될 때마다, 결함과 그 원인을 모두 제거하라 (Five Whys)
코드 공유	▪ 팀 구성원 누구든지 언제라도 시스템의 어떤 부분이든 개선할 수 있다. ▪ 지속적인 통합은 집단 소유를 실시하기 전에 해야 하는 중요한 전제 조건이다.
코드와 테스트	▪ 코드와 테스트만 영구 산출물로 유지하고, 다른 문서들은 코드와 테스트에서 생성되도록 한다. ▪ 쓸모 없는 산출물들을 제거하는 것이 이러한 개선을 가능하게 한다.
단일 코드 기반	▪ 코드 흐름은 오직 하나뿐이어야 한다. 잠시 분기를 하더라도 빠른 시간 내 합쳐져야 한다. ▪ 코드 기반을 늘리는 대신, 단일 코드 기반으로 충분하지 못하게 만드는 근본적인 설계 문제를 해결하라.
매일 배치	▪ 매일 배치를 실천하기 위해서는 해야 할 것들이 많다. ▪ 팀 내부의 신뢰도와 고객과의 신뢰도가 높은 수준이어야 한다.
범위 협상 계약	▪ 소프트웨어 개발 계약 시 시간, 비용, 품질은 확정해도, 시스템의 정확한 범위는 계속 협상해 나가자고 요청하라. 범위 협상 계획은 소프트웨어 개발에 좋은 충고가 된다.
사용 별 지불 (Pay-Per-Use)	▪ 돈의 흐름을 소프트웨어 개발에 직접 연결하면, 개선을 이끌어갈 정확한 정보를 얻을 수 있다. ▪ 구독 모델(Subscription Model)의 활용


ASK
QUESTIONS