# 01-3 소프트웨어 개발방법론

#000_총류 #004_컴퓨터과학 #정보처리기사_필기

# 소프트웨어 개발방법론

> *Software Development life cycle models types - thinksys.com*

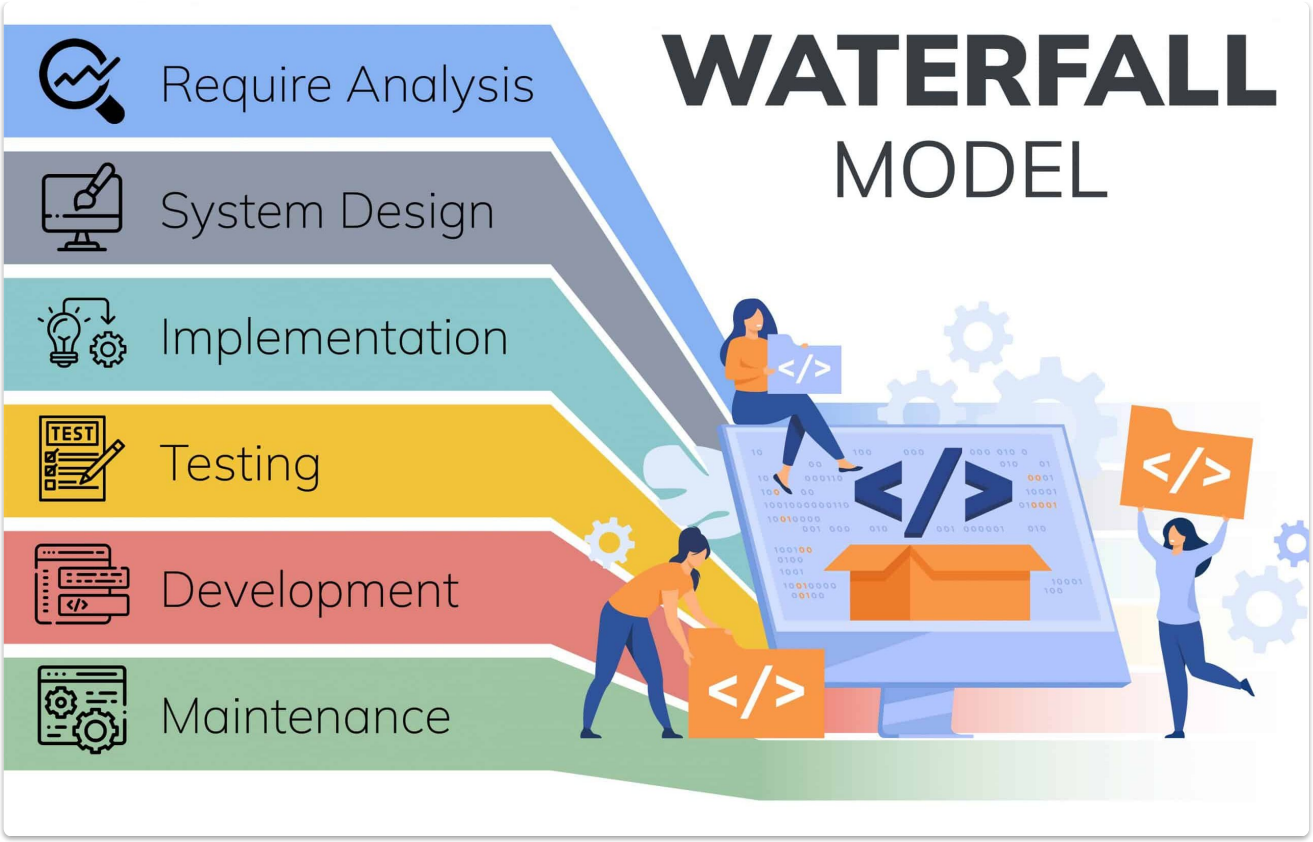
## 소프트웨어 설계 방법론

> *소프트웨어의 생명주기 (Software Life Cycle)은 소프트웨어 제품의 개념 형성에서 시작하여 운용/유지보수에 이르기까지 변화의 모든 과정이다.*

- 생명주기 과정
  1. 타당성 검토 : 개발하는게 맞을까?
  2. 개발 계획
  3. 요구사항 분석
  4. 설계
  5. 구현
  6. 테스트
  7. 운용
  8. 유지보수

### 폭포수 모형 (Waterfall Model)의 개요

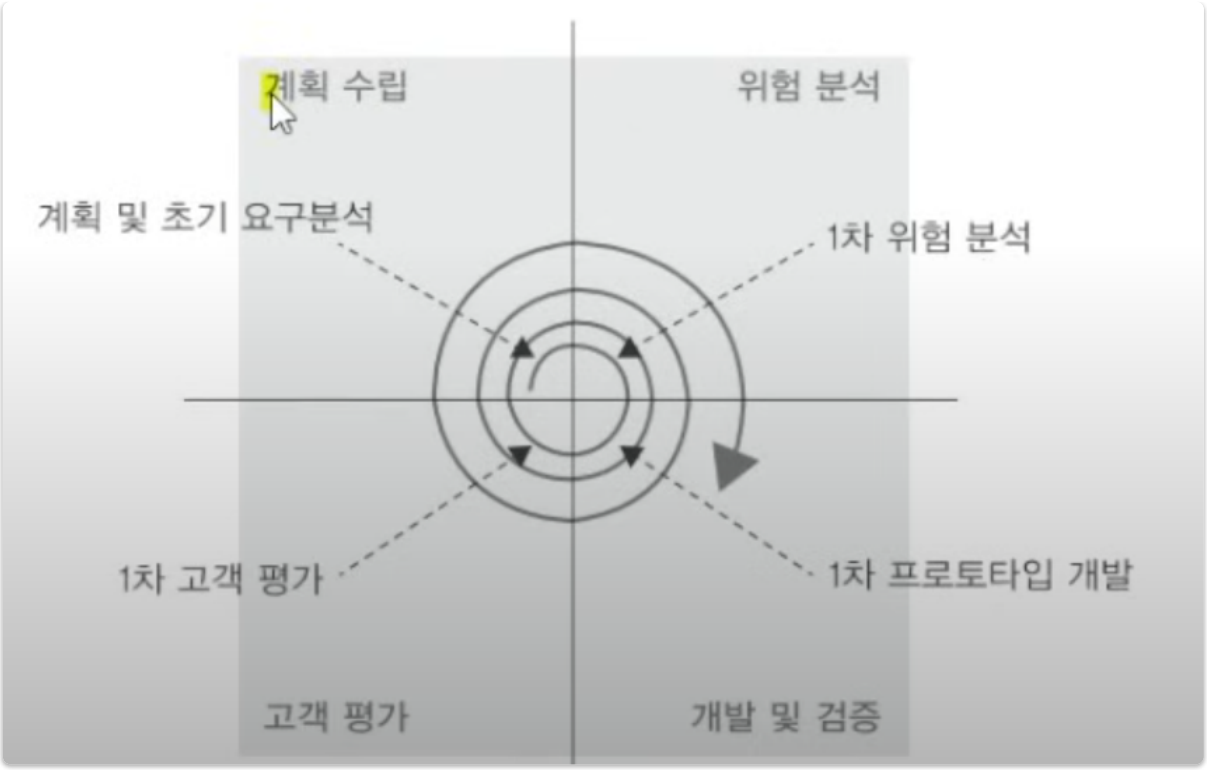
> *선형 순차적 모델이라고도 하며 Boehm이 제시한 고전적 생명주기 모형으로, 소프트웨어 개발 과정의 각 단계가 순차적으로 진행되는 모형이다. 후에 나올 모형들의 기본이다.*




- **장점 :**
  새로운 환경에 잘 적응하며, 명확한 로드맵과 업격한 구조를 가진 계획을 적용한다. 작은 프로젝트, 짧은 시간 및 간단한 요구사항에 적합하다.
- **단점 :**
  대규모 프로젝트에는 이상적인 모델이 아니다. 초기 요청이 명확하지 않으면 효율적이지 않을 수 있으며, 제품 생성 중에 오류를 수정하기 어렵다. 이로 인해 개발이 완료된 후 버그를 수정하는 것이 어렵고 비용이 많이 들게 된다.

## 나선형 모형 (Spiral Model) **

*Boehm이 제시하였으며, 반복적인 작업을 수행하는 점증적 생명주기 모형이다.*




| 개발 단계 | 설명 |
|---|---|
| 계획 수립 | 기능 제약 등의 세부적 계획 단계이다 |
| 위험 분석 | 위험 요소 분석 및 해결 방안 설정 단계이다. |
| 개발과 검증 | 기능 개발 및 검증 단계이다. |
| 고객 평가 | 결과물 평가 및 추후 단계 진행 여부를 결정하는 단계이다. |

- 장점 :
  - 점증적 모형, 집중적 모형이라고도 하며 유지보수 과정이 필요 없다.
  - 소프트웨어 개발 중 발생할 수 있는 위험을 관리하고 최소화하는 것이 목적이다.
  - 나선을 따라서 돌아가면서 각 개발 순서를 반복하여 수행하는 점진적 방식으로 누락된 요구사항을 추가할 수 있다.

## 하향식과 상향식 설계 **

### 하향식 설계

> *소프트웨어 설계 시 제일 상위에 있는 Main User Function에서 시작하여 기능을 하위 기능들로 나눠가면서 설계하는 방식이다.*

### 상향식 설계

> *가장 기본적인 컴포넌트를 먼저 설계한 다음 이것을 사용하는 상위 수준의 컴포넌트를 설계하는 방식이다.*

## 프로토타입 모형(Prototype Model)

> *실제 개발될 시스템의 견본(Prototype)을 미리 만들어 최종 결과물을 예측하는 모형이다.*

- 개발이 완료되고 나서 사용을 하면 문제점을 알 수 있는 폭포수 모형의 단점을 보완하기 위한 모형으로 요구사항을 충실 반영할 수 있다.

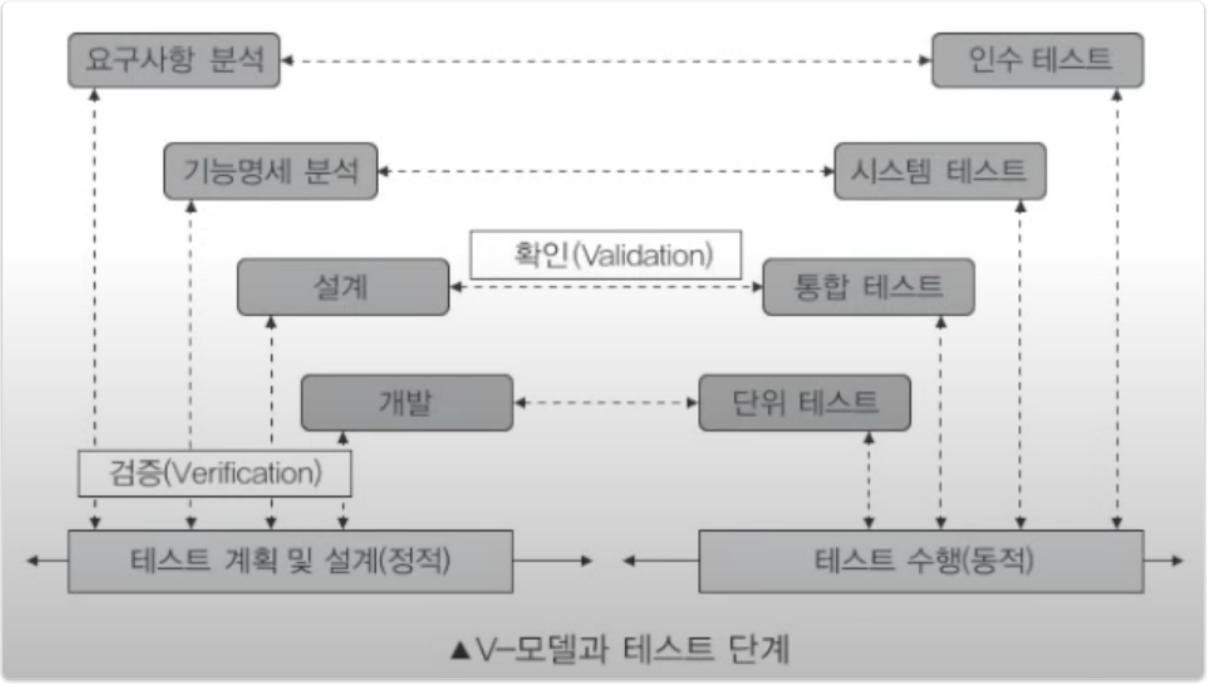
## HIPO (Hierarchy Input Process Output) **

> *입력, 처리, 출력으로 구성되는 시스템 분석 및 설계와 시스템 문서화용 기법이다.*
> *계층적 입력 처리 출력*

- 일반적으로 가시적 도표(Visual Table of Contents), 총체적 다이어그램(Overview Diagram), 세부적 다이어그램(Detail Diagram)으로 구성된다.
- 구조도(가시적 도표, Visual Table of Contents), 개요, 도표(Index Diagram), 상세 도표(Detail Diagram)로 구성된다.
- 가시적 도표는 전체적인 기능과 흐름을 보여주는 구조이다.
- 기능과 자료의 의존 관계를 동시에 표현할 수 있다.
- 보기 쉽고 이해하기 쉬우며 유지보수가 쉽다.
- 하향식 소프트웨어 개발을 위한 문서화 도구이다.

## V-Model **

> *폭포수 모형에 시스템 검증과 테스트 작업을 강조한 모델이다. 중앙 기준으로 왼쪽은 HIPO라고 봐도 무방하나 테스트 또한 단계별로 하는 것에 의의를 둔다. 정적 테스트는 코드를 분석하는 것이며 동적 테스트는 실제 실행시 문제가 생기는지 확인하는 것이다.*



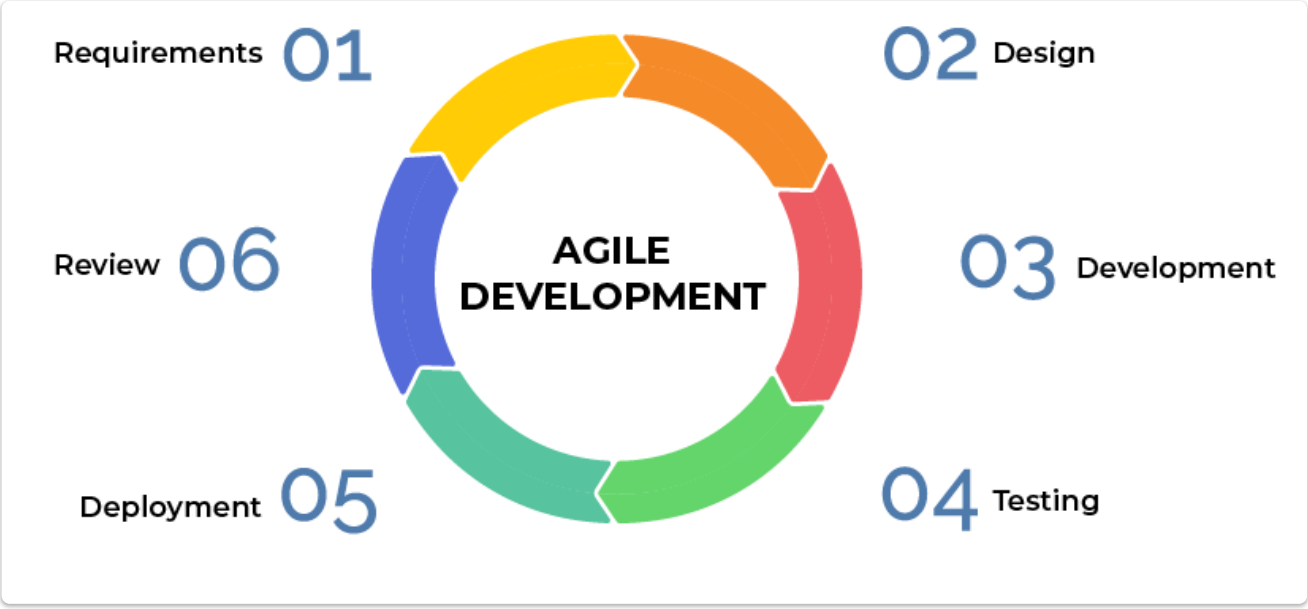
▲V-모델과 테스트 단계

- 개발 단계의 작업을 확인하기 위해 테스트 작업을 수행한다.

- 생명 주기 초반부터 테스트 작업을 지원한다
- 코드뿐만 아니라 요구사항과 설계 결과도 테스트할 수 있어야한다.
- 폭포수 모형보다 반복과 재처리과정이 명확하다
- 테스트 작업을 단계별로 구분하므로 책임이 명확해진다.

# 애자일 (Agile) 개발 방법론

> *날렵한, 재빠른 이란 사전적 의미가 있다. 특정 방법론이 아닌 소프트웨어를 빠르고 낭비 없이 제작하기 위해 고객과의 협업에 초점을 두고 소프트웨어 개발 중 설계 변경에 신속히 대응하여 요구사항을 수용할 수 있다.*




## 특징

- 절차와 도구보다 개인과 소통을 중요시하고 고객과의 피드백을 중요하게 생각한다.
- 소프트웨어가 잘 실행되는데 가치를 두며, 소프트웨어 배포 시차를 최소화할 수 있다.
- 짧은 릴리즈와 반복, 점증적 설계, 문서 최소화
- 사용자 참여, 비공식적인 커뮤니케이션 변화

## 종류

- **익스트림프로그래밍** (XP, eXtremeProgramming) :
- **스크럼** (SCRUM) :
- **린** (Lean) :
- DSDM (Dynamic System Development Method) :
  동적 시스템 개발 방법론으로
- FDD (Feature Driven Development) :
  기능 중심 개발
- Crystal

- ASD (Adaptive Software Development)
  적응형 소프트웨어 개발 방법론
- DAD (Disciplined Agile Delivery)
  학습 애자일 배포

## Agile 선언문

> *고객과의 의사소통을 통해 고객이 원하는 것을 만들어주자.*

1. 프로세스나 도구보다 개인과의 소통이 더 중요하다.
2. 완벽한 문서보다 실행되는 소프트웨어가 더 중요하다
3. 계약 협상보다 고객과의 협업이 더 중요하다
4. 계획을 따르는 것보다 변경에 대한 응답이 중요하다.

# XP (eXtremeProgramming) **

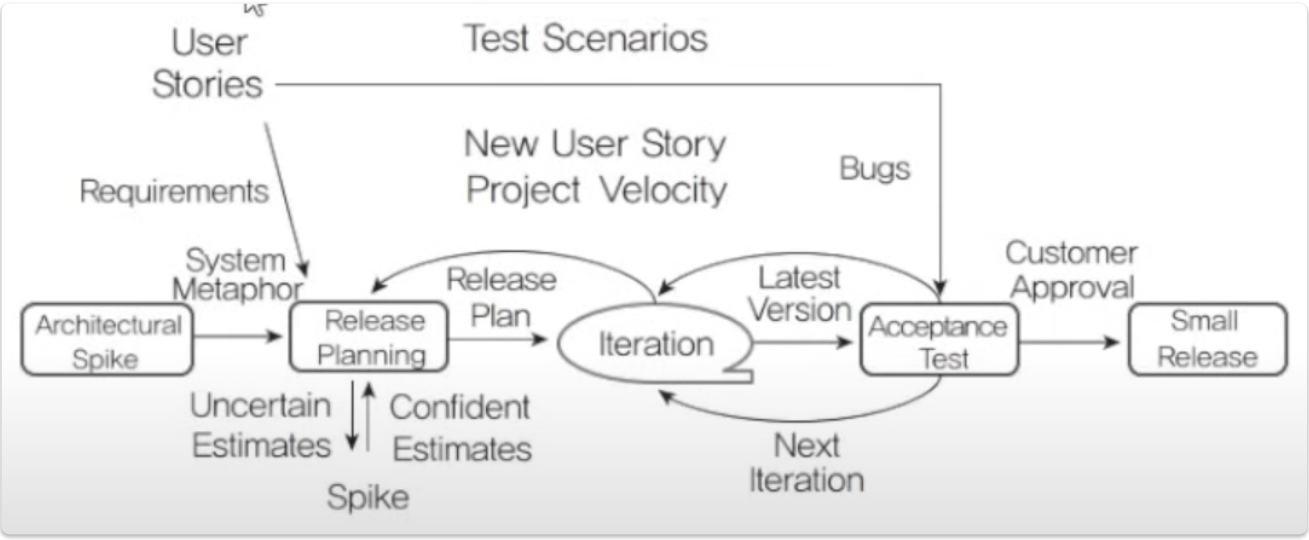
> *1999년 Kent Beck이 제안하였으며, 개발 단계 중 요구사항이 시시각각 변동이 심한 경우 적합한 방법론이다. 요구에 맞는 양질의 소프트웨어를 신속하게 제공하는 것을 목표로 한다*

## 핵심가치

- **소통 :**
  개발자, 관리자, 고객 간의 원활한 소통을 지향한다
- **단순성 :**
  부가적 기능 또는 미사용 구조와 알고리즘은 배제한다.
- **Feedback :**
  소프트웨어 개발에서 변화는 불가피하다. 이러한 변화는 지속적 테스트와 통합, 반복적 결함 수정 등 빠르게 피드백한다.
- **용기 :**
  고객 요구사항 변화에 능동적으로 대응한다.
- **존중 :**
  개발 팀원 간의 상호 존중을 기본으로 한다.

## XP Process **

> *Spike : 어려운 요구사항, 잠재적 솔루션을 고려하기 위해 작성하는 간단한 프로그램이다.*
> *User Stories : 사용자의 요구사항을 간단한 시나리오로 표현(UML에서의 Use Case와 같다)*

> *여기서 가장 중요한 것은 Iteration.*
>
> 
> 

| 용어 | 설명 |
|---|---|
| User Story | 일종의 요구사항으로 UML의 유즈케이스와 같은 목적으로 생성되나 형식이 없고 고객에 의해 작성된다는 것이 다르다 |
| Release Planning | 몇 개의 스토리가 적용되어 부분적으로 기능이 완료된 제품을 제공하는 것으로 부분/전체 개발 완료 시점에 대한 일정을 수립한다. |
| Iteration | 하나의 릴리즈를 세분화한 단위이며 1~3주 단위로 진행된다. 반복 진행 중 새로운 스토리가 추가 될 때 진행 중 반복이나 다음 반복에 추가될 수 있다. |
| Acceptance Test | 릴리즈 단위의 개발이 구현되었을 때 진행하는 테스트로 사용자 스토리에 작성된 요구사항을 확인하여 고객이 직접 테스트한다. 오류가 발견되면 다음 반복에 추가한다. 테스트 후 고객의 요구사항이 변경되거나 추가되면 중요도에 따라 우선순위가 변경될 수 있다. 완료 후 다음 반복을 진행한다. |
| Small Release | 릴리즈 단위를 기능별로 세분화하면 고객의 반응을 기능별로 확인할 수 있다.<br>최종 완제품일 때 고객에 의한 최종 테스트 진행 후 고객에 제공한다. |

## XP의 12가지 실천사항 **

| 구분 | 12 실천사항 | 설명 |
|---|---|---|
| Fine Scale FeedBack | Pair Programming | 두 사람이 짝이 되어 한 사람은 코딩을 다른 사람은 검사를 수행하는 방식이다. 코드에 대한 책임을 공유하고, 비형식적인 검토를 수행할 수 있다. 코드 개선을 위한 리팩토링을 장려하며, 생산성이 떨어지지 않는다 |
| | Planning Game | 게임처럼 선수와 규칙, 목표를 두고 기획에 임한다. |
| | Test Driven Development | 실재 코드를 작성하기 전에 단위 테스트부터 작성 및 수행하며, 이를 기반으로 코드를 작성한다. |
| | Whole Team | 개발 효율을 위해 고객을 프로젝트 팀원으로 상주시킨다. |
| Continuous Process | Continuous Integration | 상시 빌드 및 배포를 할 수 있는 상태로 유지한다. |

| 구분 | 12 실천사항 | 설명 |
| --- | --- | --- |
|  | Design Improvement | 기능 변경 없이 중복성/복잡성 제거, 커뮤니케이션 향상, 단순화, 유연성 등을 위한 재구성을 수행한다. |
|  | Small Releases | 짧은 주기로 잦은 릴리즈를 함으로써 고객이 변경사항을 볼 수 있게 한다. |
| Shared Understanding | Cording Standards | 소스 코드 작성 포맷과 규칙들을 표준화된 관례에 따라 작성한다. |
|  | Collective Code Ownership | 시스템에 있는 소스 코드는 팀의 모든 프로그래머가 누구든지 언제라도 수정할 수 있다. |
|  | Simple Design | 가능한 가장 간결한 디자인 상태를 유지한다 |
|  | System Metaphor | 최종적으로 개발되어야 할 시스템의 구조를 기술한다. |
| Programmer | Sustainable | 일주일에 40시간 이상 작업 금지 2주 연속 오버타임을 금지한다. |

## 효과적인 프로젝트 관리를 위한 3대 요소

- 사람(People) - 인적 자원
- 문제(Problem) - 문제 인식
- 프로세스(Process) - 작업 계획