



*Time goes now*



# 린 소프트웨어 개발

(Lean Software Development)

2015.06

SE Lab

김영기 책임



*If I sleep now I will have a dream, but if I study now I will make my dream com true ...*

## ❖ 린 소프트웨어 개발

- 2003SUS 메리 & 톰 포펜딕 부부 제안
- 도요타의 제조 프로세스(TSP)를 소프트웨어 개발에 적용한 방법론
- 목표
  - ✓ 시스템의 낭비를 줄이고, 고객에게 더 높은 가치를 만든다
- 특징
  - ✓ 낭비의 제거 = 지속적인 개선
    - Minimizing time
    - Minimizing capital invested
    - Minimizing cost
  - ✓ 낭비의 제거 → 수행 속도를 높임
  - ✓ 낭비의 제거는 효과적으로 품질을 개선

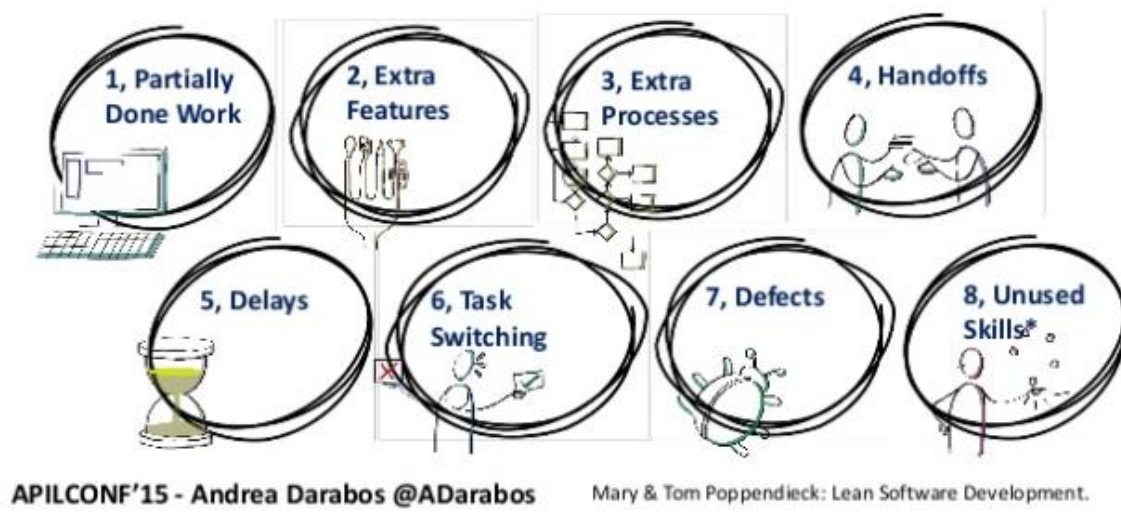


## ❖ 낭비

- Muda – 문자 그대로 “낭비”를 의미하지만 비 부가가치 활동을 의미
- 일반적으로 소프트웨어 개발에 있어서...



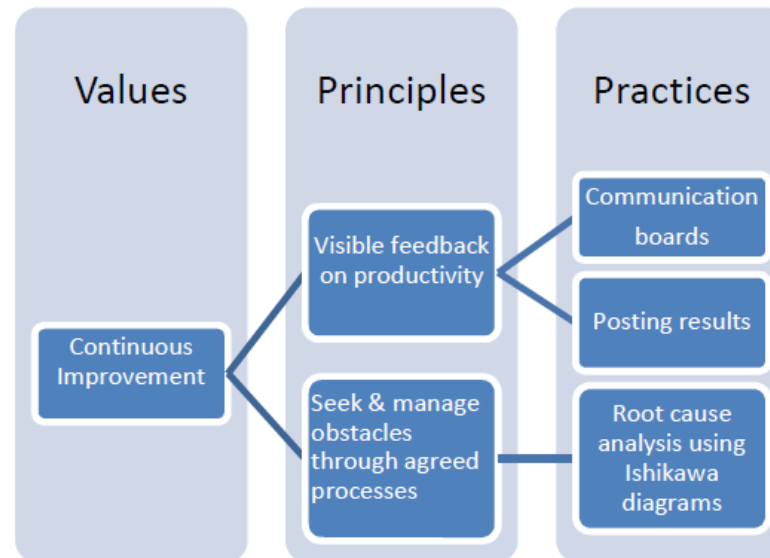
- 린에서 정의하는 8가지 소프트웨어 낭비들 (from Poppendieck's)



## ❖ 가치(Value)

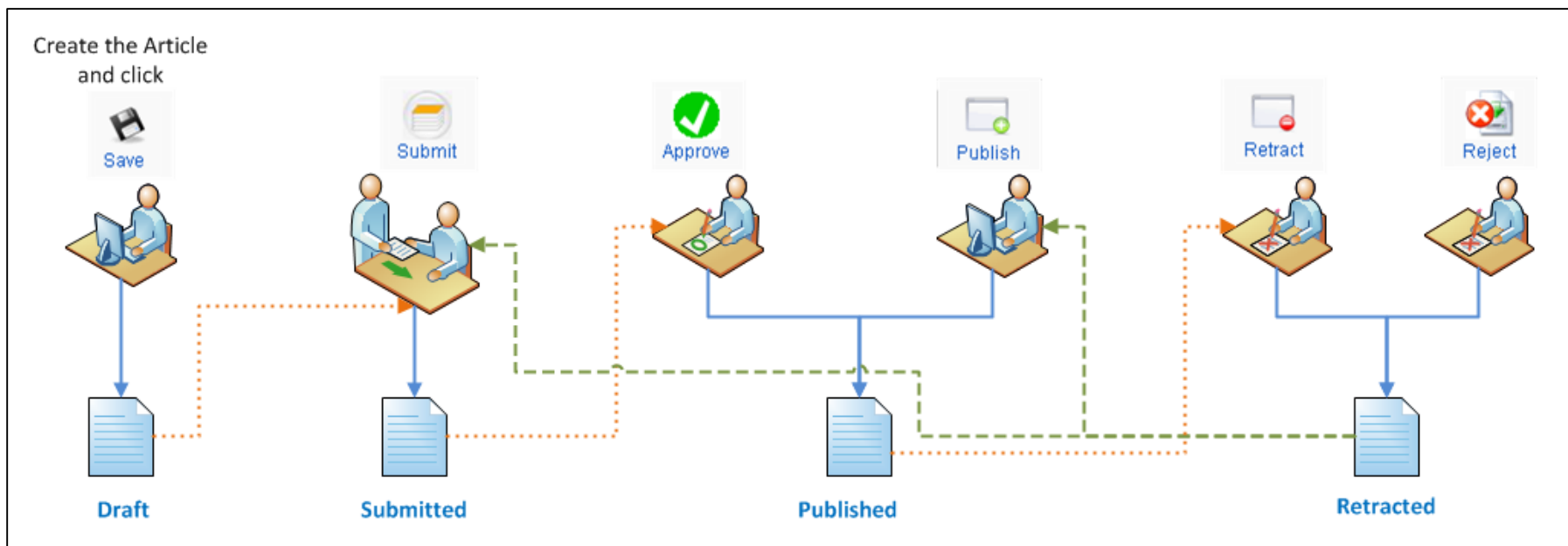
- Muda의 반대 개념
  - ✓ Muda – 문자 그대로 “낭비”를 의미하지만 비부가가치 활동을 의미
  - ✓ Mura – “공평”을 의미하며 “흐름의 가변성”으로 해석됨
  - ✓ Muri – “과중한 부담” 또는 “불합리”를 의미
- Muda가 없는 프로세스를 통해 고객만족을 위한 제품이나 서비스를 제공
- 가치의 규명 : 고객 관점에서 가치를 정의하고, 기업 관점에서 가치를 창출

가치를 제대로  
정의하기는 어렵다.  
그 이유는 ?



## ❖ 작업 흐름(workflow)

- 비즈니스 프로세스를 수행하기 위해 일어나는 일련의 업무 흐름



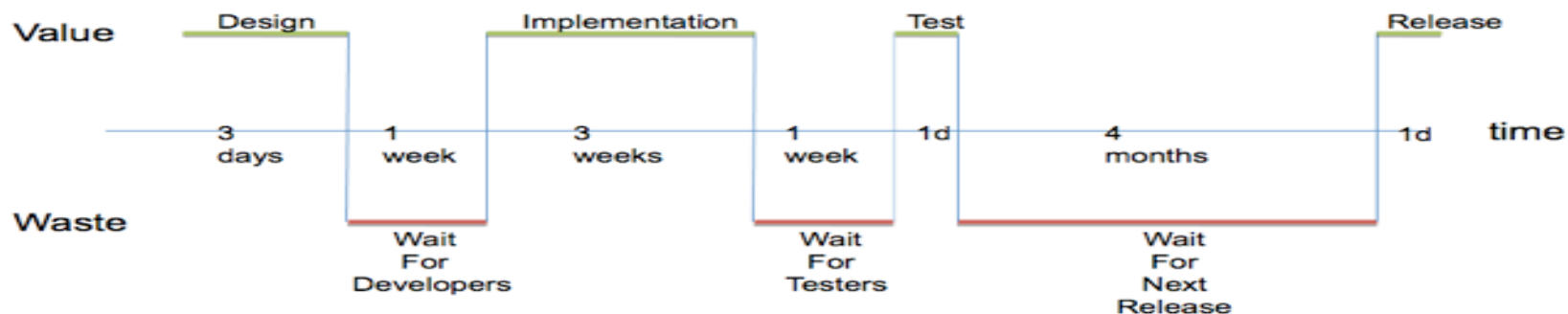
## ❖ 작업 흐름의 최적화

- 효율성 증가 = 리드 타임 감소
- 품질 향상 = 에러/낭비 감소
- 생산성 증가 = 비용 감소



## ❖ 가치 흐름 매핑(Value Stream Mapping)

1. 분석하고자 하는 제품이나 서비스를 정의한다.
2. 현재 프로세스의 스텝(Step), 대기열(queue), 지연(delay)과 정보흐름(information flow)을 식별하는 가치 흐름 맵을 생성한다.
3. 지연(delay), 낭비(waste), 제약사항(constraints)의 발견을 위해 현재의 맵을 검토한다.
4. 지연, 낭비, 제약사항을 제거하거나 줄임으로 최적화된 미래 상태(future state)의 새로운 가치 흐름 맵을 생성한다.
5. 미래 상태를 만들 수 있는 로드맵을 만든다.
6. 후에 계속해서 조정하고 최적화하기 위해 프로세스를 다시 논의 하기 위한 계획을 세운다.

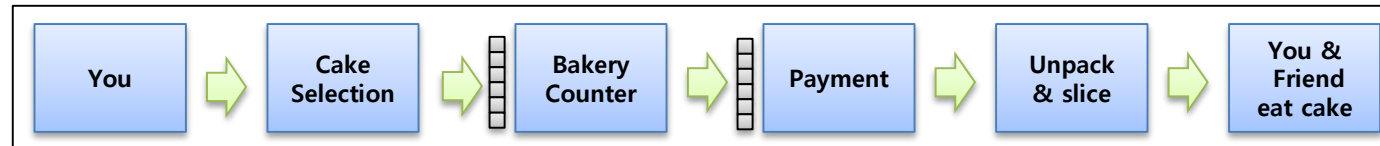


## ❖ 상황) 시험에 합격한 친구를 축하하기 위해 함께 축하 케이크를 먹기로 했다.

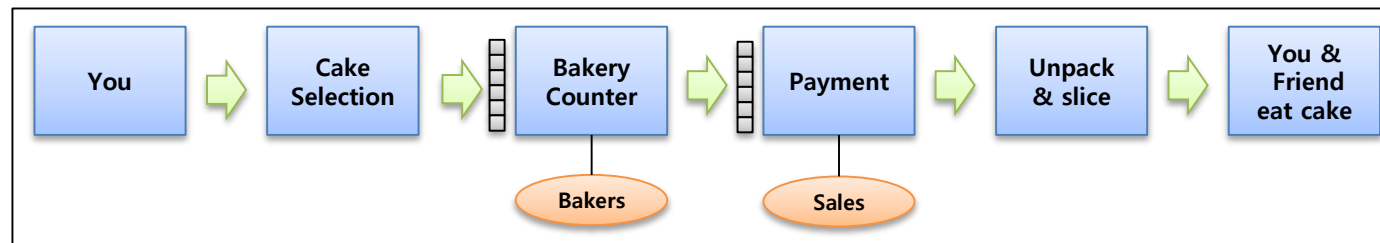
- 프로세스의 시작점과 끝점을 정의한다. (누가 시작하고, 최종 결과를 누가 얻는가?)



- 기본 흐름에 초점을 둔 프로세스 상의 고수준의 단계들, 재고, 그리고 큐를 정의한다.

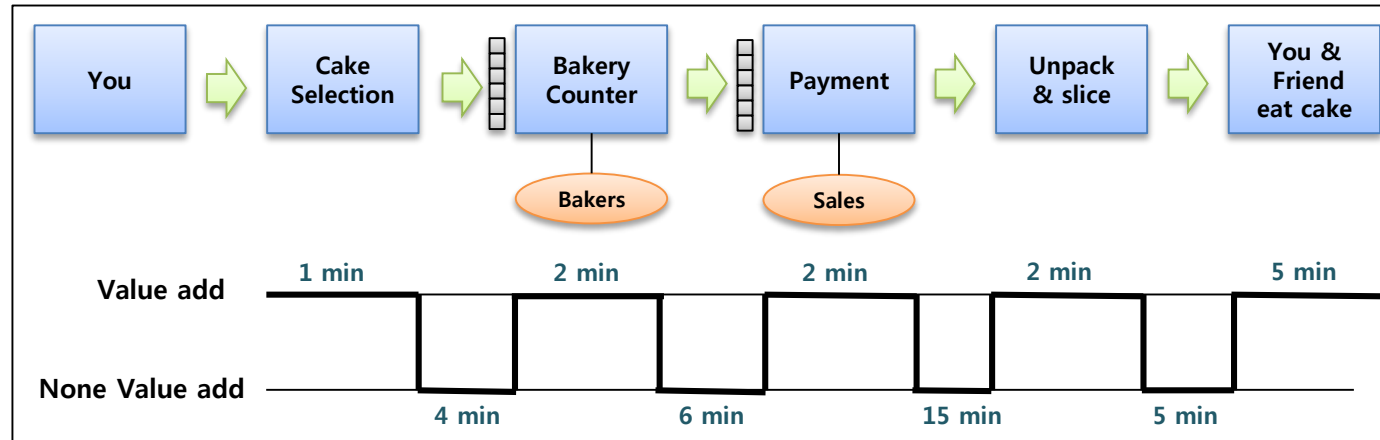


- 모든 지원 그룹과 다른 흐름을 정의한다.



## ❖ 예제 계속

- 가치를 추가하는 활동과 가치를 추가하지 않는 활동을 측정한다.

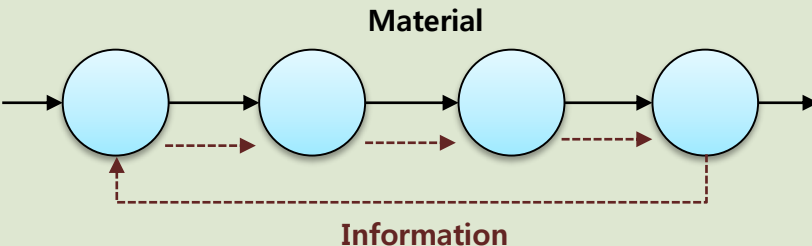
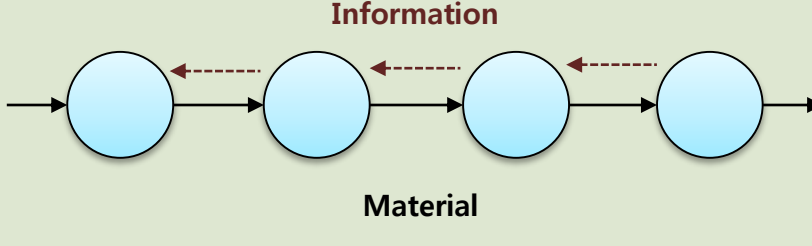
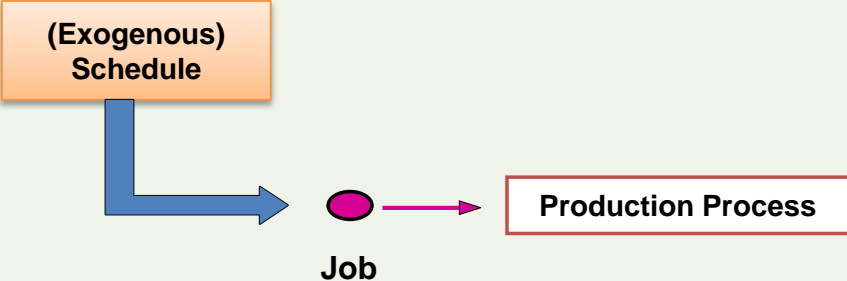
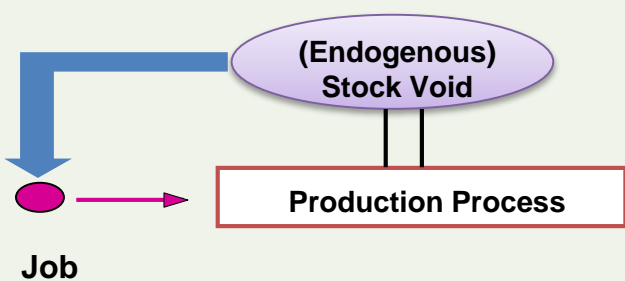


- 효율성을 계산하고, 낭비, 병목지점, 개선 작업을 정의한다.

✓ Total Cycle Time = Value Add Time + Non Value Add Time = 42 min

✓ Process Cycle Efficiency =  $\frac{\text{Total Value Add Time}}{\text{Total Cycle TIME}} = \frac{12 \text{ min}}{42 \text{ min}} = 29\%$



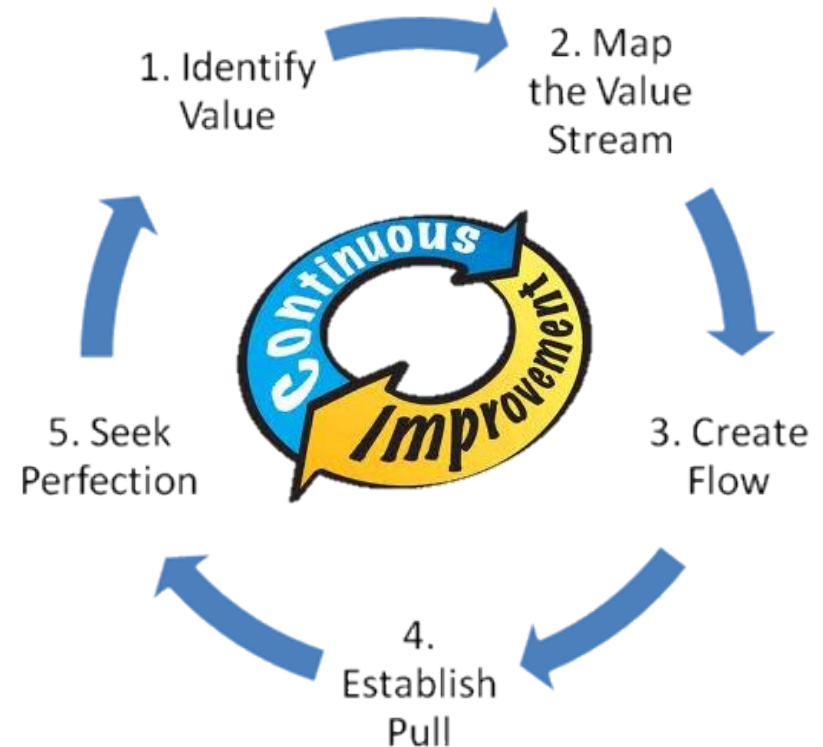
Push Systems	Pull Systems
 <p><i>Push systems do not limit WIP in the system.</i></p>	 <p><i>Pull systems deliberately establish a limit on WIP.</i></p>
	
<ul style="list-style-type: none"> <li>▪ Schedule work releases based on demand</li> <li>▪ Plan, Work arrival trigger production</li> <li>▪ Inherently due-date driven</li> <li>▪ Control release rate, observe WIP level</li> <li>▪ 미래의 예측된 결과를 기반으로 Action을 결정</li> </ul>	<ul style="list-style-type: none"> <li>▪ Authorize work releases based on demand</li> <li>▪ Event, demand arrival triggers production</li> <li>▪ Inherently rate driven</li> <li>▪ Control WIP level, observe throughput</li> <li>▪ 과거의 정보를 끌어들이며 현재의 상황에 적용</li> </ul>

## ❖ 린(Lean) 활동을 통한 개선

- 여전히 시간, 공간, 노력 및 실수를 줄여서 "낭비"를 제거할 수 있는 여지가 있다.

## ❖ 완전성에 이르는 방법

- 지속적인 개선(Kaizen)
- 급진적인 개선(Innovation)
- 실제로 두 가지 방법 모두 필요



## ❖ 가치(Value)의 규정

- 고객의 관점에서 가치를 창출하는 제품이나 서비스를 정확하게 규정
- 고객이 원하는 가격/장소/시간에 고객이 필요로 하는 제품/서비스 제공
- 가치는 동적임, 시간이 지남에 따라 변함

## ❖ 가치흐름(Value Stream)의 개선

- 가치흐름
  - ✓ 제품의 완성이나 서비스의 전달을 위한 모든 처리 단계와 과정
- 제품이나 서비스의 생산공정에 대해 가치흐름을 파악, 연구 및 개선
  - ✓ 가치를 부가하지 않는 처리 단계 제거

## ❖ 공정흐름(Flow)의 개선과 낭비 제거

- 공정 내의 흐름을 단순하고, 원활하며, 실수가 없게 함으로써 낭비를 피함

## ❖ 낭비 제거

- 제품이나 서비스의 가치에 기여하지 못하는 것은 모두 낭비!
  - ✓ 낭비는 가치는 추가하지 못하면서 오히려 비용만 발생



원칙	설명
1. 낭비를 제거하라 (Eliminate waste)	<ul style="list-style-type: none"> <li>▪ 고객에게 가치를 전달하지 않는 모든 것이 낭비이다.</li> <li>▪ 모든 낭비를 제거함으로 기간을 단축하라.</li> </ul>
2. 품질을 내재화하라 (Integrating Quality)	<ul style="list-style-type: none"> <li>▪ 결함을 예방하는 테스트를 통해 코드의 품질을 내재화하라.</li> </ul>
3. 지식을 창출하라 (Creating Knowledge)	<ul style="list-style-type: none"> <li>▪ 잦은 반복(Iteration)과 고객의 피드백을 통해 경험과 지식을 창출하라.</li> </ul>
4. 확정을 늦춰라 (Postpone Commitments)	<ul style="list-style-type: none"> <li>▪ 예측이 아닌 사실을 기반으로 결정을 할 수 있을 때까지 확정을 늦춰라.</li> </ul>
5. 빨리 인도하라 (Delivering Fast)	<ul style="list-style-type: none"> <li>▪ 고객의 확실한 요구사항을 파악하기 위해 제품을 가능한 빨리 인도하라.</li> <li>▪ 품질의 내재화가 필수 조건이다.</li> </ul>
6. 사람을 존중하라 (Respect People)	<ul style="list-style-type: none"> <li>▪ 사람을 대체 가능한 인력으로 다루지 말라.</li> <li>▪ 사람을 신뢰하고, 전문 기술을 가질 수 있도록 인재를 육성하라.</li> </ul>
7. 전체를 최적화하라 (Optimize the whole)	<ul style="list-style-type: none"> <li>▪ 부분 최적화가 전체를 최적화하지 않을 수 있다.</li> <li>▪ 문제의 근본 원인을 찾아 전체를 최적화하라.</li> </ul>

## ❖ 낭비

- 고객에게 가치를 주지 못하는 모든 것
- 낭비를 줄임 → 리드 타임을 단축

낭비	예제
미완성 작업(Partially done work)	배포되지 않은 코드
가외 기능(Extra Processes)	필요하지 않은 기능 추가
재학습(Relearning)	알던 것을 잊었다가 다시 생각해 내는 것
이관(Handoff)	암목지가 전수되지 못함, 한번 이관 시 지식이 누락
작업 전환(Task Switching)	작업 전환 시간 = 낭비
지연 (Delay)	다른 일을 하는 개발자를 기다림
결함(Defect)	결함은 추가적인 작업을 추가

## ❖ 낭비 제거 3 단계

### 1. 가치 인지

“고객이 무엇이 가치가 있다고 평가할 것인가?”

- ✓ 고객의 피드백
- ✓ 고객 입장에서 생각(고객과 시장은 항상 변한다)

### 2. 낭비 인지

“고객에게 가치를 주지 못하는 것은 무엇인가?”

- ✓ 7가지 대표적인 낭비 사례
- ✓ 가치 흐름도 작성

### 3. 낭비 제거

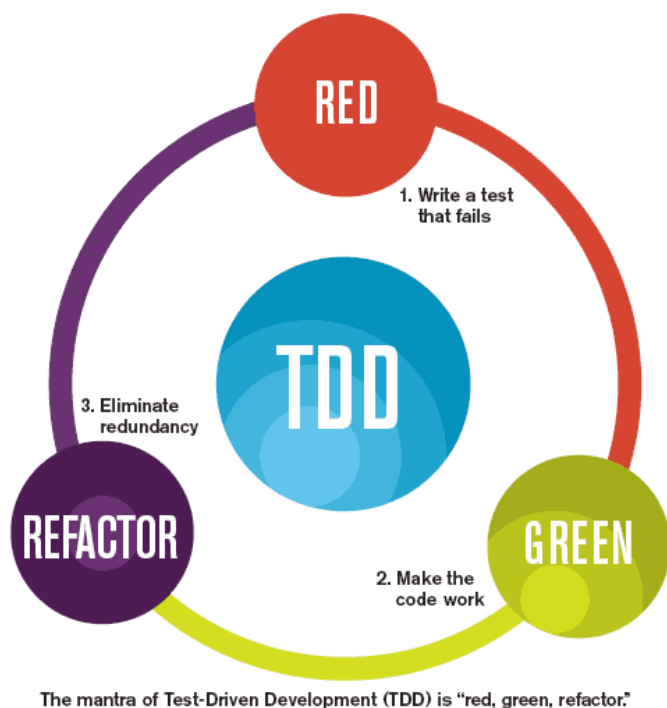
“어떻게 낭비를 제거할 것인가?”

- ✓ 낭비를 제거할 수 있는 실행력

종 류	사 례
코딩 (Waste in code development)	<ul style="list-style-type: none"> <li>▪ <b>미완성 작업</b> <ul style="list-style-type: none"> <li>✓ 일정을 지키기 못하고, 사용을 못하게 된 가능성이 있다.</li> <li>✓ 해결책: 코드 모듈화, 반복적인 개발</li> </ul> </li> <li>▪ <b>결함</b> <ul style="list-style-type: none"> <li>✓ 결함 수정 및 반복 테스트 필요</li> <li>✓ 해결책: 테스트 스위트의 갱신, 고객 피드백</li> </ul> </li> </ul>
프로젝트 관리 (Waste in project management)	<ul style="list-style-type: none"> <li>▪ <b>추가적인 프로세스</b> <ul style="list-style-type: none"> <li>✓ 낭비적이고, 불필요한 문서화</li> <li>✓ 해결책: 문서 리뷰</li> </ul> </li> <li>▪ <b>코드 이관</b> <ul style="list-style-type: none"> <li>✓ 이관 시 지식의 손실</li> <li>✓ 해결책: 이관 횟수를 줄인다 → 프로세스 간소화</li> </ul> </li> <li>▪ <b>여분의 기능</b> <ul style="list-style-type: none"> <li>✓ 고객이 필요로 하지 않는 기능</li> <li>✓ 해결책: 고객과의 지속적인 상호작용을 통한 피드백</li> </ul> </li> </ul>
개발자 (Waste in workforce potential)	<ul style="list-style-type: none"> <li>▪ <b>작업 전환</b> <ul style="list-style-type: none"> <li>✓ 다중 작업의 비효율성</li> <li>✓ 해결책: 출시를 위한 각각의 작업에 집중</li> </ul> </li> <li>▪ <b>인프라 활용이나 정보를 얻기 위한 대기</b> <ul style="list-style-type: none"> <li>✓ 개발자는 대기하기에는 값비싼 인력</li> <li>✓ 개발자 스스로 갖은 정보에 기반하여 판단하고 행동</li> </ul> </li> </ul>

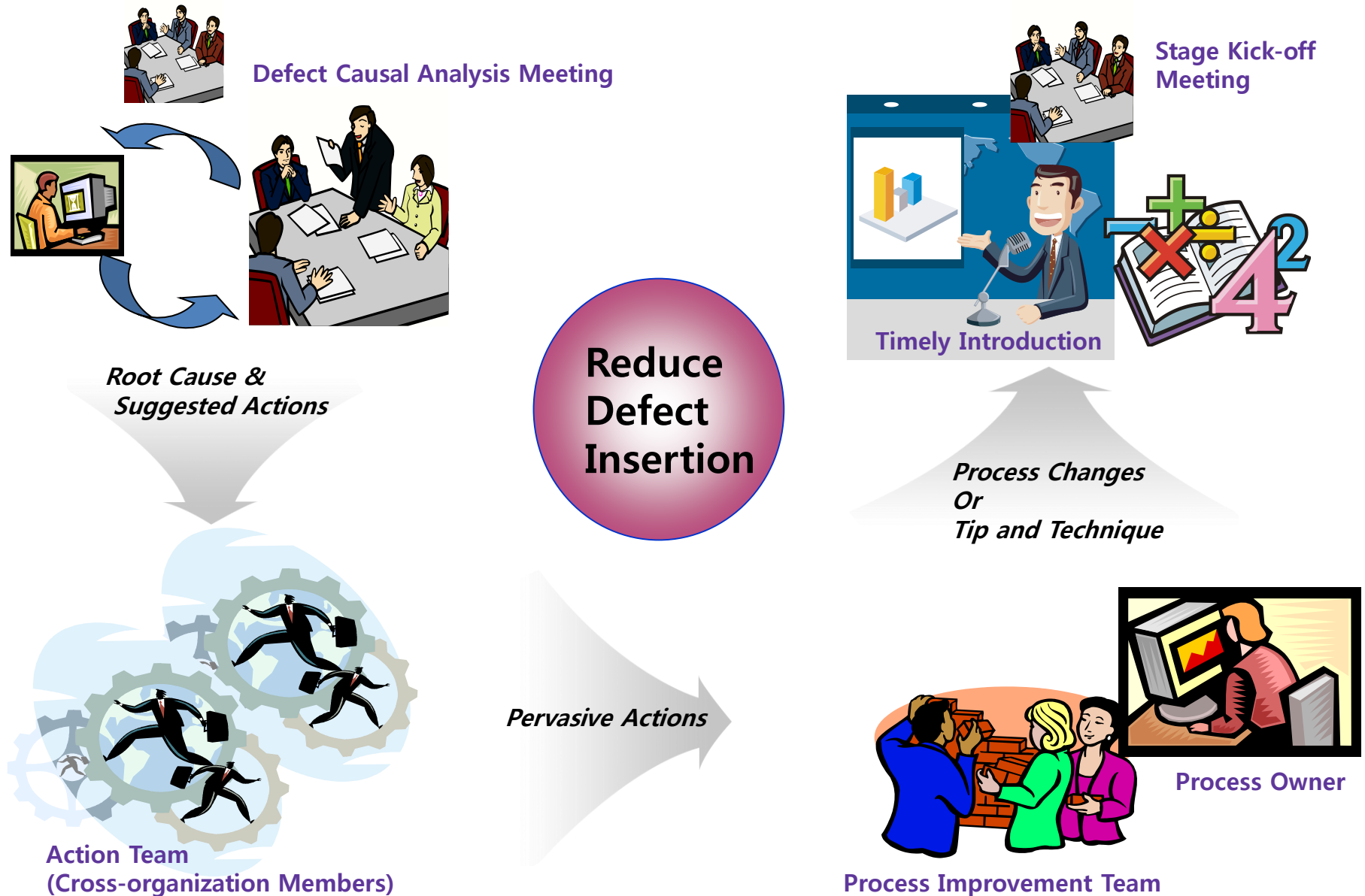
## ❖ 품질의 내재화

- 결함 발생 전 예방을 위한 테스트를 통한 품질의 내재화
  - ✓ 결함 수정을 위한 테스트가 아님
- 품질 내재화의 대표적인 방법
  - ✓ TDD, Continuous Integration, Pair Programming, Manage Trade-offs



# Defect Prevention Process

Time goes now  
[Red] [Orange] [Yellow] [Green] [Blue]

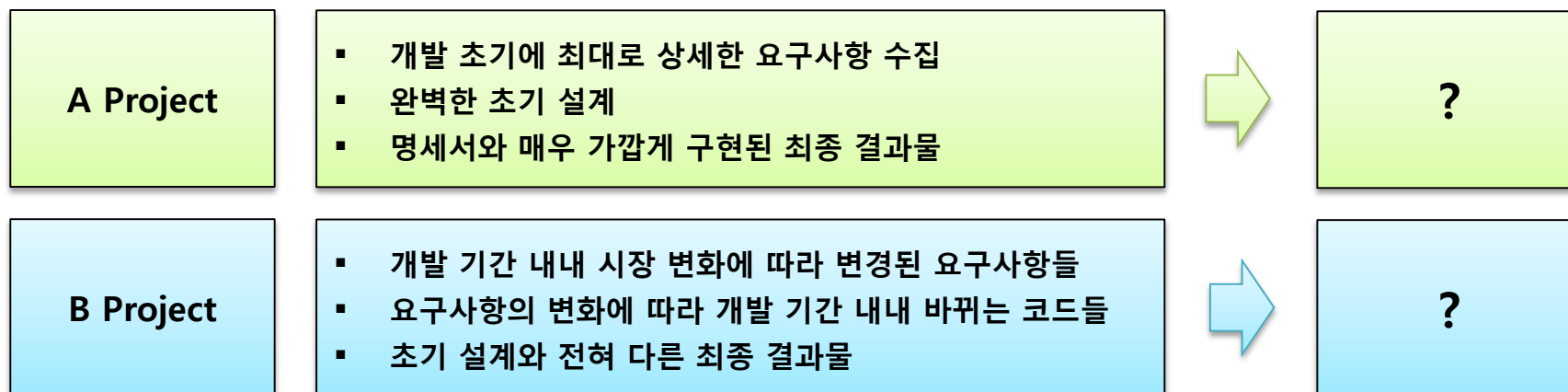


What's your point?



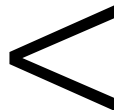
## ❖ 소프트웨어 개발 = 지식 창출 프로세스

- “요구사항 → 설계 → 개발 → 테스트 → 고객 피드백”이 진행될수록 구체적인 경험(지식) 창출
- 지식을 창출하는데 도움이 되는 것들
  - ✓ Pair Programming, Code Reviews, Documentation
  - ✓ Wiki, Knowledge sharing sessions, Training, Use Tools for Manage
- 지식 창출 프로세스의 방해 요소
  - ✓ 요구 사항과 코딩의 분리 : 개발된 코드를 사용하는 고객 피드백이 제품에 미반영
  - ✓ 초기에 확정된 설계 : 초기 설계를 구현하는 중 얻는 지식을 코드에 반영하지 못함
- 어떤 프로젝트가 좋은 프로젝트인가?



## ❖ 왜 확정을 늦추는가?

- 결정을 미루라는 것이 아니다.
  - ✓ 예측 불가능하고 위험한 상황 : 정보를 최대한 많이 가지고 있을 때의 결정이 유리
- S/W 개발 환경 = 대부분 예측이 불가능한 상황
  - ✓ 완벽한 명세서를 가지고 개발을 시작하는 것 = 미신
- 더 많은 지식을 가진 상태에서 결정을 하라.
  - ✓ 결정은 하되 변화를 수용할 수 있어야 한다.
    - 의존성을 깨뜨려라
    - 옵션을 유지하라
- 돌이킬 수 없는 결정은 마지막 결정의 순간에 하라
  - ✓ 결정 전 가능한 많이 학습 : 설계 → 구현 → 피드백의 반복



## ❖ 빠른 인도의 장점

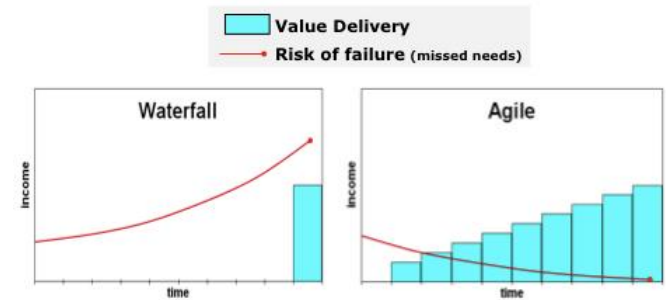
- 고객의 불확실성 감소 → 사실에 기반한 결정
- 불필요한 개발로 인한 낭비 감소 → 비용 우위

## ❖ 빠른 인도를 위한 필요조건

- Have the right people, Keep It Simple, Work as a Team
- 낭비의 제거(Eliminate Waste), 품질의 내재화(Build Quality In)
- 일의 양 제한 : 안정적이고 반복 가능한 속도 유지를 위한 WIP 제한

## ❖ 대기행렬이론을 개발에 적용

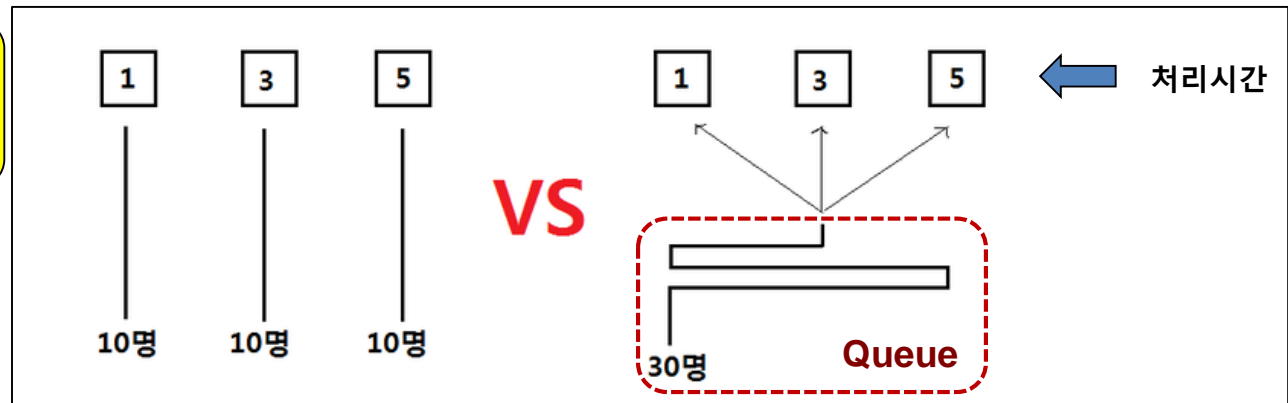
- 가동률을 강조하면 오히려 정체를 일으켜 가동률이 떨어진다
- 배치 크기를 작게하고 진행중인 작업량을 줄여 주기시간을 줄여라



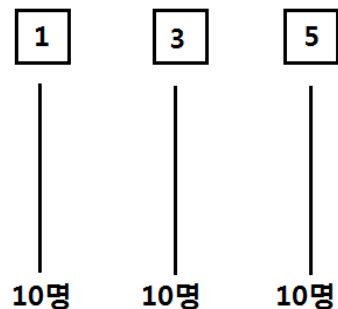
↑  
권장 출시 주기 : 2주



어떤 시스템이 더 빠르게 일을 처리하는가?



## ❖ 각 줄 서기



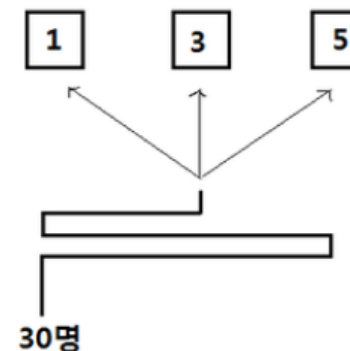
$$\sum_{N=1}^9 N = 45$$

$$\sum_{N=1}^9 3N = 135$$

$$\sum_{N=1}^9 5N = 225$$

총 405분

## ❖ 한 줄 서기



15명	5명	3명	15분
1명			16분
1명			17분
1명	1명		18분
1명			19분
1명		1명	20분

$$\sum_{N=1}^{19} N = 190$$

$$\sum_{N=1}^9 3N = 45$$

$$\sum_{N=1}^3 5N = 30$$

총 265분

## ❖ 경청

의사 선생,  
팔을 머리 위로 들 때면,  
두통이 심해져요



그러면, 절대로  
팔을 머리 위로  
올리지 마세요

## ❖ 피드백



[ 잘못된 피드백 ]



좋은 피드백을 위해서는 ..

- 구체적으로 이야기하고,
- 상대의 인격을 비하하면 안되고,
- 부정적인 피드백을 피하고,
- 단순한 언어를 사용해야 한다.

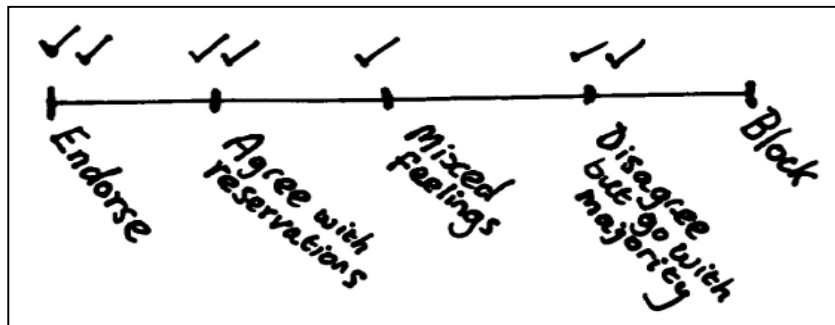
## ❖ 대화

**Respect**  
You Give It - You Get It

"네가 어제 '어머니 애는 암것도 못해요'라고 말했을 때 (관찰) 엄청 화가 났어 (느낌), 시댁에 가면 안그래도 긴장하고 불편한데 네가 그 말 하니까 완전 당황스럽더라 (느낌 again), 난 시댁에서 너랑도 시어머니랑도 즐겁고 재미있게 잘 지내고 싶어, 너랑 시어머니한테 사랑 받고 싶기도 하고 (필요/욕구), 그러니까 다음부터는 시댁에서는 가벼운 말이라도 한번만 더 생각하고 말해주었으면 좋겠어 (요청/부탁)"



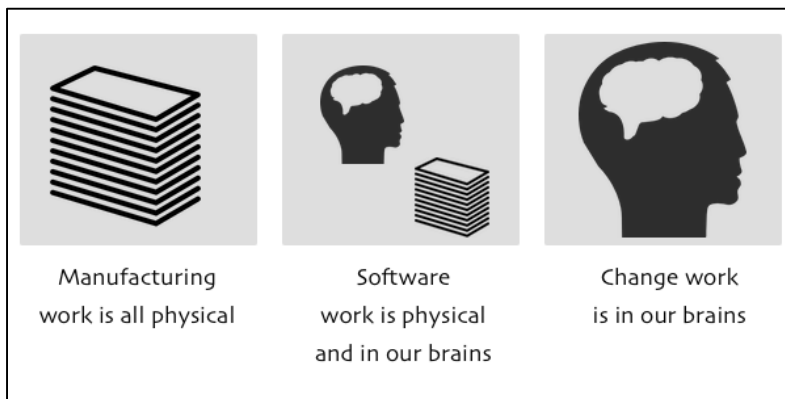
## ❖ 합의



사람마다 생각하는 합의의 수준이 다르다

## ❖ 문제의 근원을 생각하고 전체적인 관점에서 최적화

- 가치 흐름에 초점을 맞춰라
- 완전한 제품을 인도하라
  - ✓ 소프트웨어만이 아닌 완전한 제품을 개발
  - ✓ 완전한 제품은 완전한 팀에 의해 만들어진다



- 측정을 통한 개선
  - ✓ 주기적으로 프로세스 역량을 측정
  - ✓ 인도되는 비즈니스 가치로 팀의 역량을 평가
  - ✓ 고객 만족을 평가

- 최적화의 결과는 전체로서 평가된다
- 80 : 20 법칙 (Pareto Principle)
  - 20%의 코드가 80%의 수행시간을 차지한다
  - 바꿔말하면: 나머지 80%의 코드는 바꿔봤자 보람이 없다
- 나무보다 숲을 볼 수 있어야 한다
- bottleneck은 생각하지 못한 곳에 있곤 한다.
- 전체에 대한 부분의 영향은 직관적으로 알기 힘들다.

**We should forget about small efficiencies,  
say about 97% of the time:**  
Premature optimization is the root of all evil.

Ref: <http://www.slideshare.net/innover/ndc08-20109236>

