Agile Process

Agile Manifesto[선언문] (2001)

맨날 시험에 내는 4가지

* source) http://wwwagileallianceorg

12 Agile Principles safe?

1. Customer satisfaction by rapid delivery of useful software 빅뱅의 단점을 건드림
2. Welcome changing requirements, even late in development 재사용
3. Working software is delivered frequently (weeks rather than months)
4. Working software is the principal measure of progress
5. Sustainable development, able to maintain a constant pace
6. Close, daily co-operation between business people and developers
7. Face-to-face conversation is the best form of communication (co-location)
8. Projects are built around motivated individuals, who should be trusted
9. Continuous attention to technical excellence and good design
10. Simplicity
11. Self-organizing teams
12. Regular adaptation to changing circumstances

1. 고객 만족
2. 요구 사항 변경 가능
3. 빈번한 소프트웨어 작동
4. 원칙 준수
5. 유지 보수
6. 상호 작용
7. 만나서 의사소통
8. 구성원
9. 집중력
10. 단순성
11. 팀 조직
12. 변화에 적응

Agile 방법론

- 기존의 전통적 방법론에서 중시하는 절차와 산출물보다는 고객과의 협력과 동작하는 소프트웨어를 중점으로 하는 방법론
- 비즈니스 시장 변화에 유연하게 대응하기 위한 경량 개발 방법론

- 주요 Agile 방법론
  - ❑ XP
  - ❑ **SCRUM**

Lean Development

시험 x
참고로 배움

The Concept of Lean Development

- Toyota Production System (TPS)
  - ❑ Lean development at Toyota
- Implementation of lean manufacturing principles into a software development model
- Lean manufacturing focused solely on the production process and focused on **removing waste** from the production process.
- **Waste** is defined as everything that does not contribute to the creation of value for the customer
- Example in SW
  - ❑ Partially done work
  - ❑ Extra processes
  - ❑ Defects

-> 낭비를 줄이자

Goal: Reduce the waste in a system and produce a higher value for the final customer

Basics of Lean Development

- Value
  - What do customers really need?
- No product is made until the customer requests it
  - What the customer wants the product to do
  - 45% of all software features go unused
- Just-In-Time (JIT)
  - a principle aimed at reducing waste by producing and delivering goods or services only when needed.
- Pursuit of perfection
  - After a project flows… keep improving it
  - Iterative cycles
  - Feedback vs Forecast

Lean principles

1. Eliminate waste
2. Amplify learning, Create Knowledge
3. Decide as late as possible, Delay in making decisions
4. Deliver as fast as possible, Fast Delivery 배포는 빠르게
5. Empower the team, Empower your team 팀에게 힘을
6. Build integrity in, Best Quality In
7. See (Optimize) the whole 전체를 보고 최적화

가장 늦게 결정해라

2. Kanban

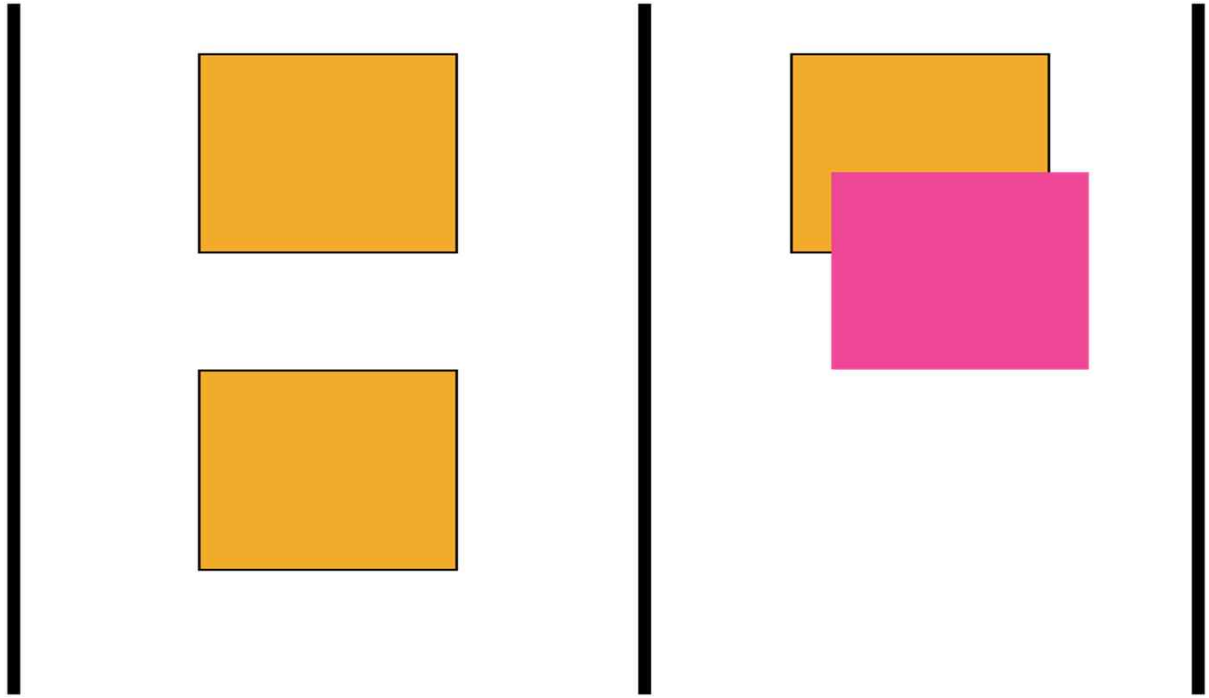
칸반(Kanban) : 시각적 프로젝트 관리 방법

- 칸반(Kanban)은 반복적인 프로세스의 단계를 명확히 나누는 프로젝트 관리 방식
- 모든 업무 단계를 시각적으로 표현, 모든 업무의 진행 상황을 매일 확인
- 팀원은 이 보드를 보며 모든 작업의 상태를 명확히 파악, , 문제가 발견되면 즉시 해결
- 일본어인 칸반(Kanban)은 우리말로 '간판(看板)'

best practice

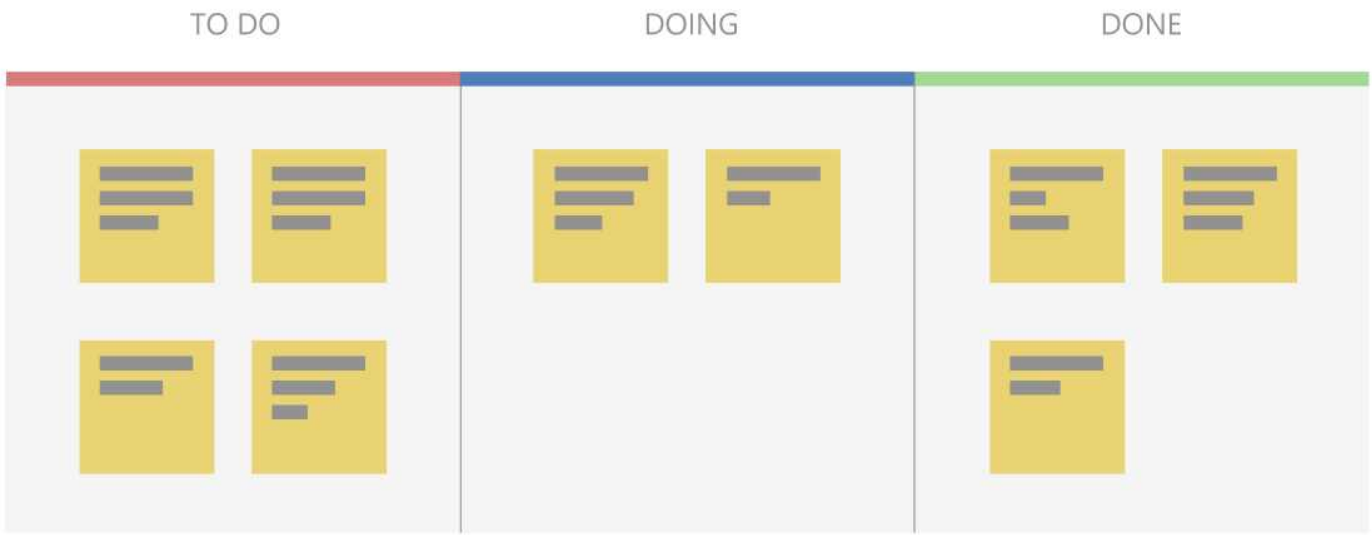
칸반 카드

- 칸반 카드는 완료해야 하는 작업 조각을 나타내는 카드
- 각 카드에 작업을 요청하는 사람, 작업 기한, 작업 담당자 등 나열
- 카드에는 간략한 작업 설명, 지원 문서 링크 또는 첨부 파일, 원활한 협업을 위한 주석 등을 포함



칸반 보드

- 기본 칸반 보드에는 할 일, 진행 중, 완료라는 3개의 열
  - ❑ 필요에 따라 '요청됨', '백로그', '분석 중', '개발 중', '테스트 중', '완료' 등 추가
- 반복되는 프로젝트 단계가 칸반 보드의 열이 됨



시험에 나옴

3. XP (eXtreme Programming)

Overview

- Definition
  - ❑ A software development methodology which is intended to improve responsiveness to changing customer requirements
    - ✓ A lightweight, efficient, low-risk and flexible way to develop software
- XP Philosophy
  - ❑ You need to improve **Communication**,
  - ❑ You need to seek **Simplicity**,
  - ❑ You need to get **Feedback** on how well you are doing
  - ❑ You need to always proceed with **Courage**

take commonsense principles and practices to extreme levels

peer review(동료 리뷰?)를 강화한다

pair 쌍으로

1. The whole team including On-site Customer
2. Planning game
3. Small Releases
4. Testing (Test driven development) TDD 모듈을 개발할 때 test을 만들고 시작
5. Metaphor
6. Simple Design
7. Pair Programming
8. Design Improvement –Refactoring

8, 9번이 맞물림

9. Collective Code Ownership
10. Continuous Integration
11. Sustainable Pace-40 Hour Work Week
12. Coding Standards

1. The whole team including On-site Customer

- A real user as a develop team member
- Customer's task
  - ❑ answer questions
  - ❑ resolve disputes
  - ❑ set small-scale priorities & scope
  - ❑ write functional tests
- Quick and concrete feedback is possible

2. Planning game

- Players
  - ❑ Developer
  - ❑ Customer

- Pieces
  - ❑ story cards

- Moves (cyclic)
  - ❑ exploration phase
    - ✓ Give both players an appreciation for what all the system should eventually do
  - ❑ commitment phase
    - ✓ choose the scope and date of release (Business)
    - ✓ commit to delivering it confidently (Development)
  - ❑ steering phase
    - ✓ update the plan based on the learned

3. Small Releases

- As small as possible while still delivering enough business value
- Advantages
  - ❑ provide value to the customer early
  - ❑ get concrete feedback from the customer
- A month or two would be good

4. Testing (Test driven development)

- Any program without an automated test simply doesn't exist
- Two kinds: unit test and functional test
- Why automatic testing
  - ❑ tell you when you are done
  - ❑ keep the program alive longer and more capable of accepting changes
  - ❑ give confidence in your code
  - ❑ reduce development time
- Unit Test
  - ❑ Written by developers
  - ❑ Method-by-method
  - ❑ Write tests before code
  - ❑ Always run at 100%
  - ❑ For developers' confidence
  - ❑ Used when
    - ✓ unit test
    - ✓ integration test
    - ✓ regression test

- Functional Test
  - ❑ Written by Business (supported by dedicated testers in Development)
  - ❑ Story-by-story
  - ❑ For customers' confidence
  - ❑ Not necessarily run at 100% any time
  - ❑ Black-box test
  - ❑ Also called "Acceptance Test"
  - ❑ Simplify/enhance code without changing its observable behavior
  - ❑ Major tool for "simple design"
    (cf. "big design up front")
  - ❑ When to start refactoring
  - ❑ bad smells in code (eg. long method)
  - ❑ "Refactoring: Improving the Design of Existing Code" by Martin Fowler

5. The system metaphor

- A story that everyone - customers, programmers, and managers - can tell about how the system works.

- It's a naming concept for classes and methods that should make it easy for a team member to guess the functionality of a particular class/method, from its name only.

6. Simple Design

- Today's problems today, and tomorrow's problems tomorrow
- What is the simple design?
  - ❑ run all the tests
  - ❑ contain no duplicated logic
    (be wary of hidden duplication)
  - ❑ state every intention important to the programmer
  - ❑ contain the fewest possible classes and methods

7. Pair Programming

- Pairs of developers write all production code
- Benefits
  - ❑ All design decisions involve at least two brains
  - ❑ At least two people are familiar with every part of the system
  - ❑ Spread knowledge throughout the team
  - ❑ Code is always being reviewed
  - ❑ Sometimes more productive

8. Design Improvement –Refactoring

- Simplify/enhance code without changing its observable behavior
- Major tool for “simple design”
  (cf. “big design up front”)
- When to start refactoring
- bad smells in code (eg. long method)
- “Refactoring: Improving the Design of Existing Code” by Martin Fowler

9. Collective code ownership

- (also known as "team code ownership" and "shared code") means that everyone is responsible for all the code; therefore, everybody is allowed to change any part of the code.
- Everybody is responsible for all code
- Make refactoring work better by exposing the team to more opportunities of big refactoring
- Tend to be familiar with most of the system
- Backed up by pair programming and unit test

- Do integration:
  - ❑ several times a day
  - ❑ after getting all the unit tests to run
- No big-bang integration
  ⇒ no integration night-mares
- Easy to identify integration errors
- Use one designated integration machine

11. Sustainable Pace-40 Hour Work Week

- Discourage “overtime work”
- Motto
  - ❑ Be fresh and eager every morning
  - ❑ Be tired and satisfied every night
  - ❑ Think about something other than work on weekend
  - ❑ Come in full of fire and ideas on Monday
- Overtime is a symptom of a serious problem on the project

12. Coding Standards

- Code to a common standard
- Necessary because of
  - ❑ swapping partners
  - ❑ refactoring each other's code constantly
- Should emphasize communication
- Must adopted voluntarily by the team

4. SCRUM

- 애자일 소프트웨어 개발 방법론
  - 반복적이고 점진적인 개발 방법
  - 다기능 협업팀(Cross-Functional Team) 기반의 프로젝트 수행
  - 애자일의 핵심 원칙인 지속적 개선에 중점

- 경험주의와 린(Lean) 사고 기반
  - 경험주의는 경험을 통해 지식을 얻고 관찰한 것을 기반으로 결정을 내리는 것
  - 린 사고는 불필요한 것을 줄이고 필수 사항에 집중

3 Roles

- 제품 책임자(Product Owner)
  - ❑ 제품 백로그(Product Backlog)라는 '해야할 일들의 목록을 작성하고,
  - ❑ 스프린트(Sprint)라고 정의되는 개발 주기로 나눈다.

- 다기능 협력팀(Cross Functional Team)
  - ❑ 스프린트에 할당된 작업들을 수행하고,
  - ❑ 기능 수행 결과물(Increment)를 완성한다.

- SCRUM 마스터
  - ❑ 팀의 활동을 지원한다.

Product Owner(제품 책임자)

- 비즈니스 목표를 충족시키는 제품을 만들기 위해 제품 백 로그를 관리하고 제품을 검토
- Business Oriented
- 제품의 기능 식별 및 우선순위 부여 및 목록작성
- 제품에 대한 최종 결정권을 가짐
- 팀원 전체와 협력

Scrum Master (스크럼마스터)

- Manages the Scrum process, rather than the Scrum Team.
- 팀이 SCRUM 프로세스를 따르도록 지원
- 팀을 위해 봉사, 보호, 교육, 안내함
- 팀의 방해요소 제거

Development Team (개발팀)

- Typically 10 or fewer people
- Being capable of doing the A to Z of the creation of each Product Backlog item.
- Self organizing
- Cross-functional
- .

Artifacts

- Product Backlog
  - ❑ An ordered list of everything (aka stories) that might be needed in the final product.
- Sprint Backlog :
  - ❑ Development Team pulls the highest ordered items from the Product Backlog into the Sprint Backlog.
  - ❑ Selected items (stories) from the Product Backlog to be delivered through a Sprint, along with the Sprint Goal and plans for delivering the items and realizing the Sprint Goal
- Product Increment
  - ❑ The "Done" portion of the product completed during a Sprint.
  - ❑ The set of all the Product Backlog items completed so far in the project (up to the end of a certain Sprint)

Vision Statement

- 프로젝트를 통해 달성하고자 하는 것
  - ❑ 프로젝트의 존재이유
- 구성
  - ❑ 잠재 고객 파악 : 누가 Product를 구매하는가?
  - ❑ 고객의 Needs 파악
  - ❑ 제품의 Value
  - ❑ 유사 제품보다 우월한점

backlog 안에 포함

Product Backlog

- User Story를 기반으로 작성
- Product Owner가 책임
  - 개발 팀과 함께 의논하며 작업
  - 새로운 항목 추가 또는 수정
  - 우선순위 설정
- An ordered list of everything (aka stories) that might be needed in the final product
  - 기능, 비기능 Features
  - 기술적, 관리적 업무
  - 개선사항
  - 오류수정
- 프로젝트를 수행하는 동안 수정이 가능

User Story

- 고객의 요구사항을 Story로 표현
- 사용자 관점에서 표현
- Who, Why, What을 명확하게 명시

Story Point 결정

- 기준이 되는 스토리를 1로 정하고, 몇배의 중요도 인가를 결정
- 1pt : (예) 1일 6 hours
- 주로 피보나치 수열의 변형을 사용. 0, 1, 2, 3, 5, 8, 13, 20, 100 …

User story Point 부여 방법 : Planning Poker

- Product Owner, Customer, 그리고 개발 팀이 참여
- Story에 대해 설명하고 질문과 토론시간

1. 각 참여자들은 자신의 추정치를 제시
2. 가장 높은 추정치와 가장 낮은 추정치를 적은 참여자가 추정의 논리를 설명
3. 질문과 토론 시간
4. 새로운 추정치 제시

- 약 3회정도 이 과정을 반복하면 추정치가 수렴함. (평균으로 하기도함)

Release Plan

- 개발 팀의 역량을 기반으로 전체 일정 계획을 수립하는 것
- 전체 일정 계획 : Sprint들로 구성
- 각 Sprint에는 Story의 우선순위, 개발 선후 관계, 의존관계등을 고려하여 Story를 할당
- Product의 전체 Story Point와 Velocity(개발속도)를 기반으로 함
- Velocity : Sprint 동안 완성할 수 있는 Story Point

Sprint Plan

- Sprint
  - Each Scrum project is a set of Sprints.
  - Each Sprint lasts - at most - **one calendar month**, and shorter Sprints are extremely common.
- Sprint Plan
  - Plan what will go into a Sprint.
  - The Sprint Planning meeting is timeboxed to **eight hours for a one-month** Sprint.
  - The team crafts a Sprint Goal, a high-level objective that will be accomplished by delivering the selected Product Backlog items.
  - Create the Sprint Backlog
- Sprint Backlog
  - Sprint Backlog is a list of all stories that will be developed.
  - The Team breaks down (expands) these stories into tasks.
- Task
  - 개발자 관점에서 개발자가 Sprint 동안에 수행할 작업을 기술

Burndown Chart (번다운차트)

- The Scrum Burndown Chart is a visual measurement tool that shows the completed work per day against the projected rate of completion for the current project release.