Software Engineering

(8) Agile Software Development

Sokendai / National Institute of Informatics

Fuyuki Ishikawa / 石川　冬樹

f-ishikawa@nii.ac.jp / @fyufyu

http://research.nii.ac.jp/~f-ishikawa/

目次

Critical Review on “Traditional” Approaches

- Biased too much to plans and templates/routines
  - Do not consider changes or adaptations by assuming and following feasible and useful plans
  - Need long time, half a year or a year, to obtain and validate the value by the working
  - Have little support individuals and teams, including mental and social aspects

➡ Agile Manifesto in 2001

Agile Manifesto

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right,
we value the items on the left more.

[http://agilemanifesto.org/]

Also check the principles!
[http://agilemanifesto.org/principles.html]

Principles behind Agile Manifesto (1)

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.

[http://agilemanifesto.org/principles.html]

Principles behind Agile Manifesto (2)

- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

[http://agilemanifesto.org/principles.html]

Principles behind Agile Manifesto (3)

- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

[http://agilemanifesto.org/principles.html]

Agile Software Development

- Agile Software Development （アジャイルソフトウェア開発）
  - A wide term that refers to various approaches based on the manifesto
- Iterative and Incremental Development （反復的・漸進的開発）
  - Iterates cycles of 2-3 weeks or 2-3 months
  - Repeat: “work on the minimum valuable part, and then decide the next by considering the feedback”

Typical "Agile" Approaches (1)

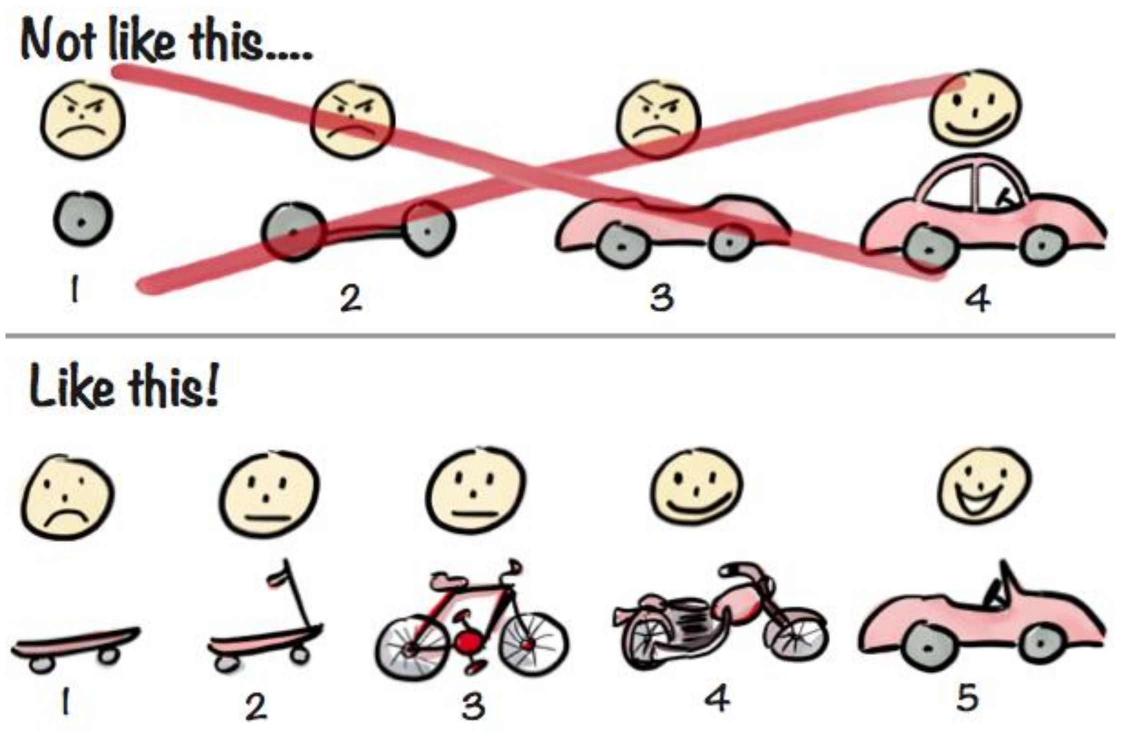
- Decide the target and way of progress on a case-by-case basis
  - Manage goals, TODOs, and their priorities in periodical meetings including the customer
- Have the working integrated code
  - Avoid "only component" states or "do not work if integrated"
  - Always run the tests that represent the value for the customer and maintain the system to pass them
  - Need use of tentative code (mock) and automated testing

Typical “Agile” Approaches

- Let the team self-organized
  - Do not employ a manager who makes commands
  - Request each team member to understand the project status and given them the right of decision for their work
- Have minimum software development
  - YAGNI : You Ain’t Gonna Need It
  - Not confused “functions that may be used sometime”
  - Consider documents/models as means, not goals

Famous Metaphor on MVP

■ Minimal Viable Product (MVP)



1. Most important goal: "allow to run"
   → find stability is important
2. "Allow to run in a stable way"
   → hard to go over tens of meters
3. "Allow more efficient run"
   → we may find this is already enough!
4. …

[https://blog.crisp.se/2016/01/25/henrikkniberg/making-sense-of-mvp]

Popular Terminology (1)

- Product owner
  - Participants from the customer side
- User story
  - Requirements stated in a way to clarify who/what/why
  - Ishikawa, a lecturer, wants to check who joined each week so that he can give proper scores for his lecture…
- Coach, scrum master
  - Role responsible for facilitation inside the team and with outside
  - No power for managing the project

Popular Terminology (2)

- Iteration, sprint
  - Unit of iteration, usually one week – one month at most
- Backlog
  - Set of “what we want to do/what we should do”
  - Different from “what we decided to do” or “what we are doing”
  - Product-level or iteration-level
- Velocity
  - Speed of development (estimation and actual)
  - Necessary for adaptive planning

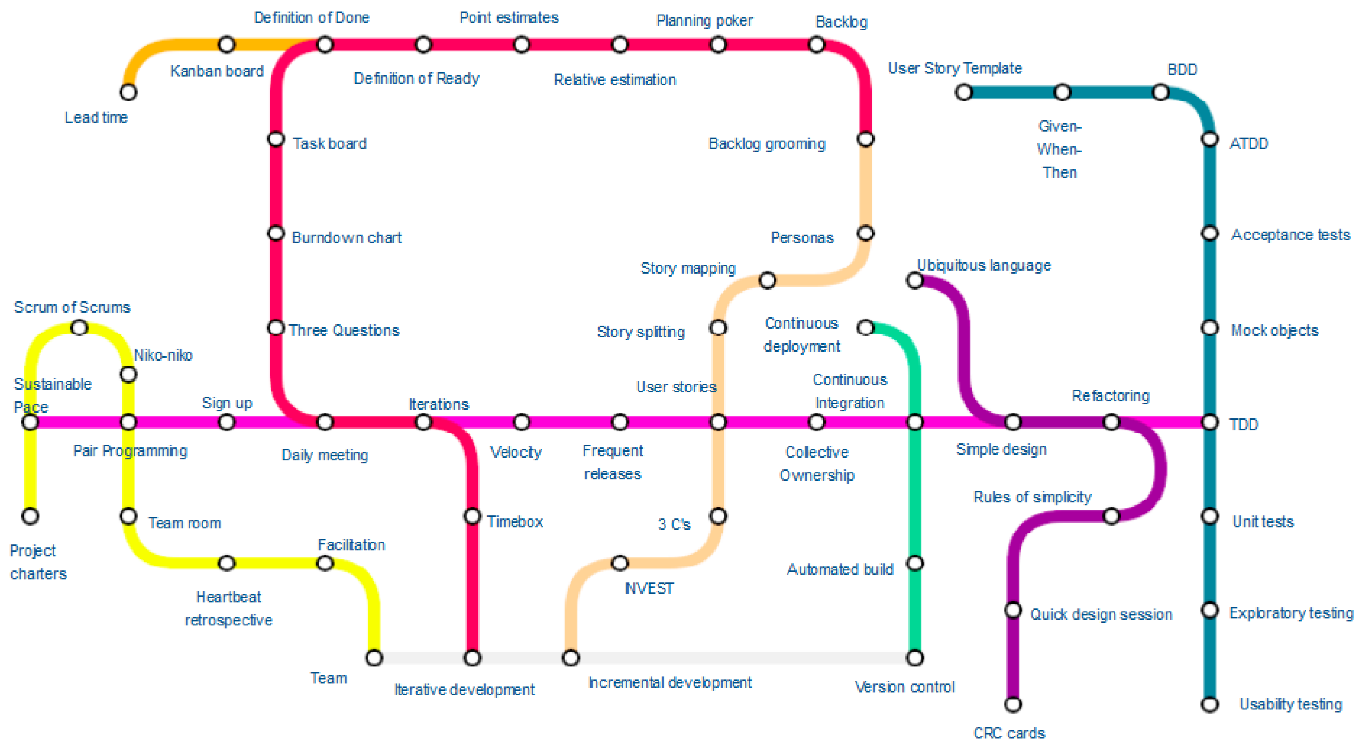
目次

Practice

- Practice（プラクティス）
  - Practical know-how or technique to realize the principles
  - In other words, patterns of development activities
  - Some methods recommend a set of practices, such as scrum, XP (eXtream Programing), etc.
    - But if you follow them blindly, it may not be the “agile” way

Practices of Agile Software Development

■Example: agile practice map



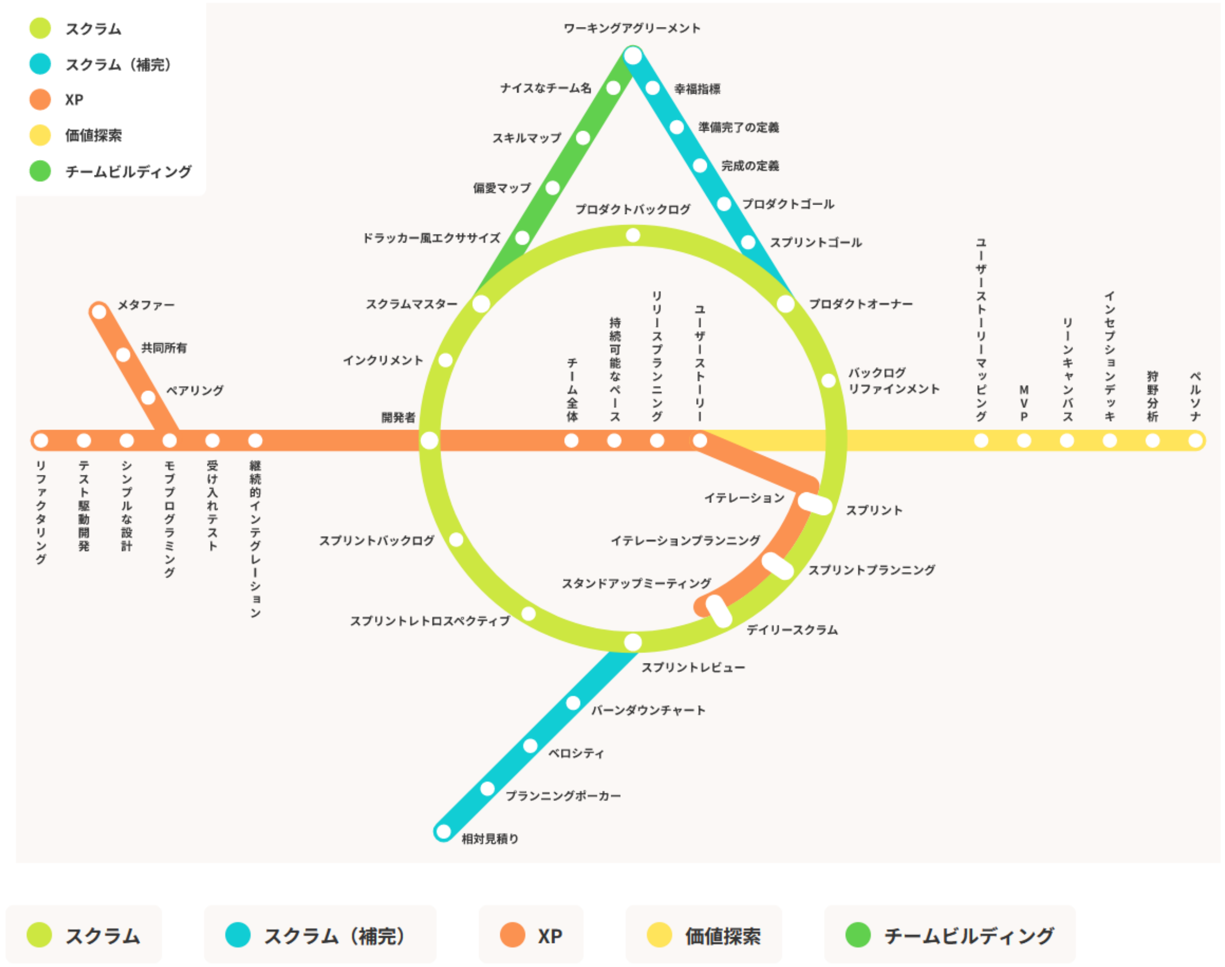
Lines represent practices from the various Agile "tribes" or areas of concern:

▬	Extreme Programming	▬	Scrum	▬	Design
▬	Teams	▬	Product management	▬	Testing
▬	Lean	▬	Devops	▬	Fundamentals

[https://www.agilealliance.org/agile101/subway-map-to-agile-practices/]

Practices of Agile Software Development

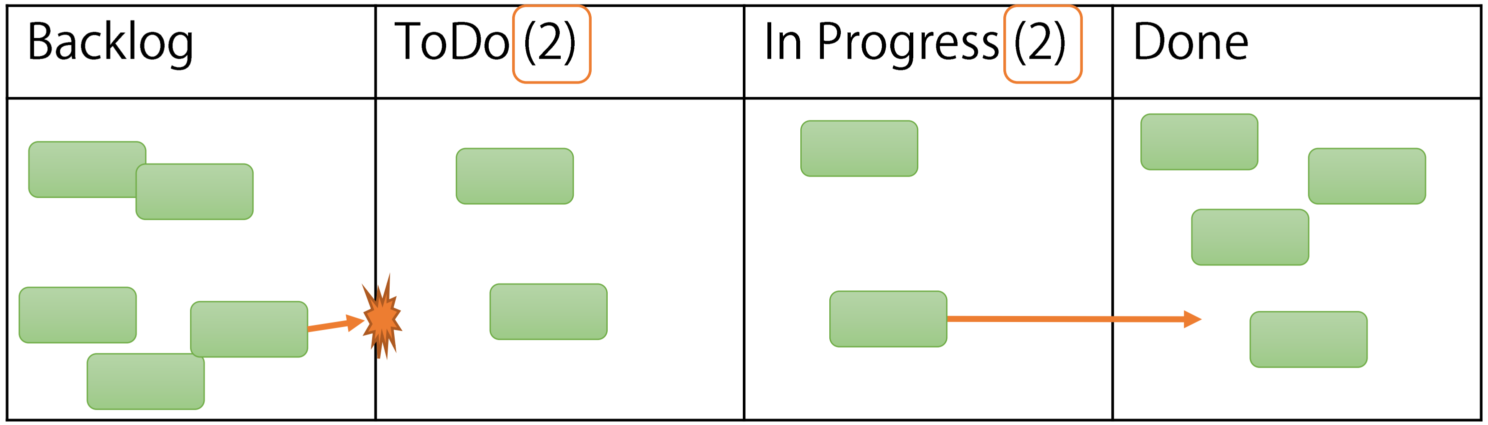
- Example: agile practice map (by a Japanese company)



[https://www.agile-studio.jp/agile-practice-map]

Practice Example: Kanban

- Kanban
  - Problem: need to control the work amount when requirements and their changes emerge in an uncertain and irregular way
  - Means: use a board to manage the status of tasks and specify the limit of WIP (Work in Progress)

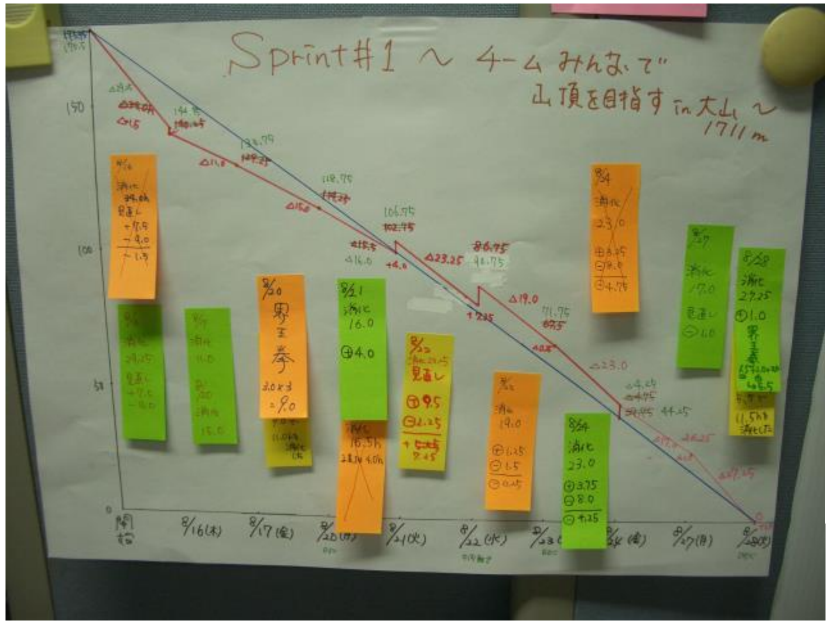


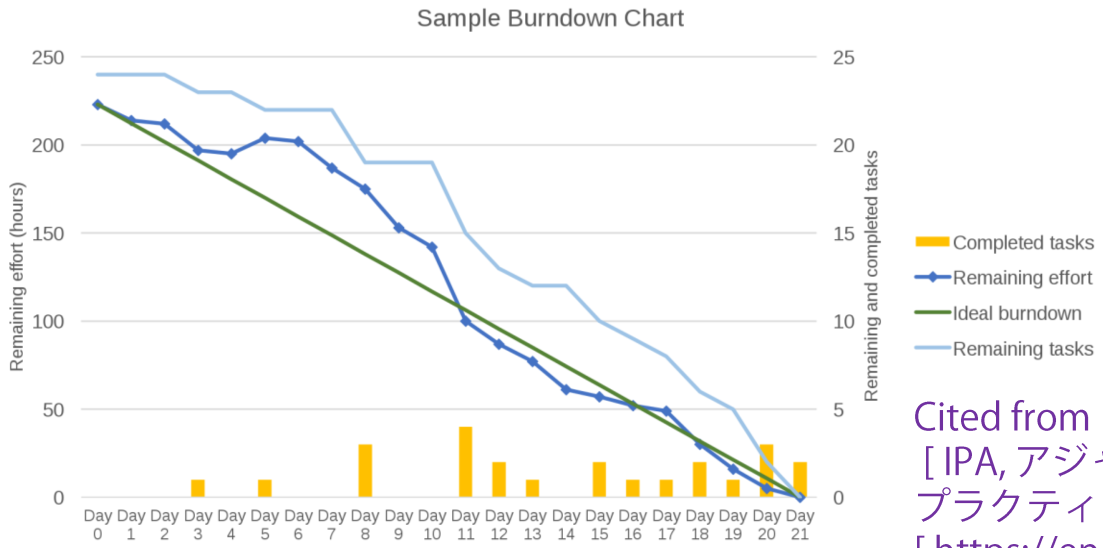
Practice Example: Velocity Measurement

- Velocity Measurement
  - Problem: cannot estimate the release date if we don't know the work amount per iteration
  - Means: record progress in each iteration by using metrics such as "story points"
    - Similar to "function points", we make scores of
  - Note: need to record velocity values for multiple iterations and combine traditional estimation if enough experience is not accumulated

Practice Example: Burndown Chart

- Burndown Chart
  - Problem: need to adaptively make decision on actions in iterations or release cycles by checking the actual progress
  - Means: visualize the progress in terms of story points over time





Cited from
［IPA, アジャイル型開発における
プラクティス活用 リファレンスガイド］
[https://en.wikipedia.org/wiki/Burndown_chart#/media/File:SampleBurndownChart.svg]

Practice Example: Inception Deck

- Inception Deck
  - Problem: sometimes objective and direction not clear among the customer and different stakeholders
  - Means: make 10 tough questions

    Why are we here?

    Meet your neighbors

    Ask what keeps us up at night

    Be clear on what's going to give

    …

Practice Example: Planning Poker

- Planning Poker
  - Problem: need to make estimation by involving knowledge of different stakeholders, especially, different experts
  - Means: make a game to let everyone to

1. Given the initial estimation, everyone shows his/her opinion at the same time by a card, e.g., “+3”
2. People with the highest/lowest values tell the reasons and everyone have discussion
3. Repeat until timeout or convergence

Practice Example: Pair Programming

- Pair Programming
  - Problem: each member has different skills, we want to develop a product that outperforms what can be done by one person, knowledge is closed inside each person
  - Means: do the programming tasks by a pair of persons (not limited to programming)
  - Variation: mob programming by more than two persons

Practice Example: Test-Driven Development

- TDD : Test-Driven Development（テスト駆動開発）
  - Problem: we easily make wrong code or break existing code if we postpone test definition and execution
  - Means: repeat the cycle of “define executable tests, develop code that passes them”
  - Often with a principle of “make (even dirty) working code and then refactoring”

Practice Example: Test-Driven Development

- (Simplistic) Example
  1. Test Case 1: (x, y, z, RESULT) = (3, 3, 5, “Isosceles”)

```
String judgeTriangle(int x, int y, int z){
  if (x==y) return "二等辺"
  else return ""
}
```

  2. Test Case 2: (x, y, z, RESULT) = (3, 5, 3, “Isosceles”)

```
String judgeTriangle(int x, int y, int z){
  if (x==y || x==z) return "二等辺"
  else return ""
}
```

(this way is effective when the problem is very difficult/complex)

Behavior Driven Development

- Behavior Driven Development（ビヘイビア駆動開発）
  - Use tests, i.e., concrete examples, as the goal of development
  - Make tests readable by the product owner and end users
  - Extend TDD, which was for unit-level engineer tasks

Specification by Test / Test as Document (in Cucumber)

```
Feature: Is it Friday yet?
Everybody wants to know when it's Friday

  Scenario: Sunday isn't Friday
    Given today is Sunday
    When I ask whether it's Friday yet
    Then I should be told "Nope"
```

Test Code

```
@Given("today is Friday")
public void today_is_Friday() {
    today = "Friday";
}

...
```

Practice Example: Continuous Integration

- Continuous Integration（継続的インテグレーション）
  - Problem: each small component of individual engineers does not work when integrated
  - Means: build and test the whole system periodically or upon each commit so that work of each engineers links to the whole systems
  - Automated by tools such as Jenkins, Circle CI, Travis CI
  - Also discussed with Continuous Delivery, including the packaging and deployment tasks (we often say CI/CD)

Practice Example: Others

- ■Daily Meeting （朝礼）
- ■Retrospective （ふりかえり）
- ■Team room（共通の部屋）
- ■Niko-niko（ニコニコカレンダー）
- ■…

Questions or Limitations of Agile (1)

- Applicability
  - Small number of people in the same place (said at most 10)
  - Multi-skilled members: everyone can do work of another; everyone works on a system-level story (not like "only network")
- Essential difficulties in changes
  - Design patterns are "preparation for a certain type of changes"
- Cost on exploration and tentative development
  - e.g., we made skateboard, bicycle, and bike before car in the MVP example

Questions or Limitations of Agile (2)

- Quality that needs careful planning and design
  - Security should be considered by design, not ad-hoc
- (In old days) unnecessary negative claims on traditional ways
  - As if all of the evils came from waterfall, documents, contracts, etc.
- "Enterprise Agile"
  - Exploration of combining traditional principles and planning/management for more stable process or large products

Summary

- Agile Software Development
  - Countermeasure to too much bias to planning and template/procedure as well as the era of rapidly changing world
  - Largest impact in