SWE3002-42: Introduction to Software Engineering

Lecture 3 – Agile Software Development

Sooyoung Cha

Department of Computer Science and Engineering

Topics covered

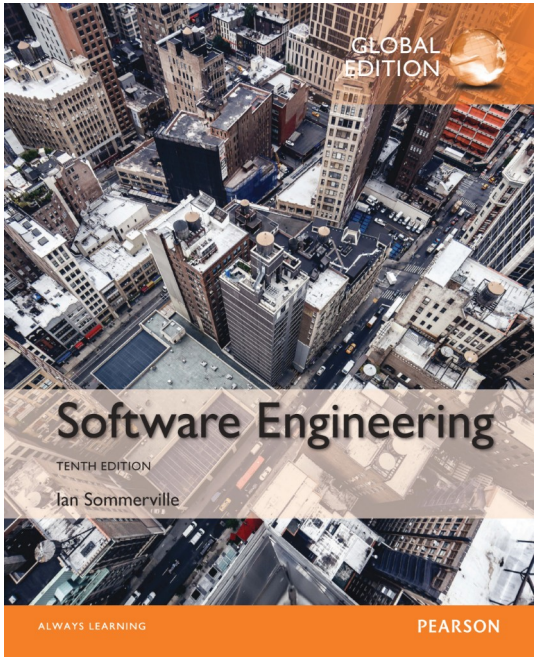
01 Background

02 Agile methods

03 Agile development techniques

04 Agile project management

05 Scaling agile methods



Chapter 3. (p.72 ~ p.100)

Background of Agile method

[Rapid software development]

- **The most critical requirement for software systems**

Early SW Deployment **vs** Software Quality

A complete set of stable SW requirements ??

Galaxy Z Flip4[1]

- **Agile methods** > Plan-driven methods
  - The need for rapid SW development that can handle changing requirements.
  - ex) Extreme Programming(1999), Scrum(2001), …

1: https://www.samsung.com/sec/smartphones/galaxy-z-flip4/

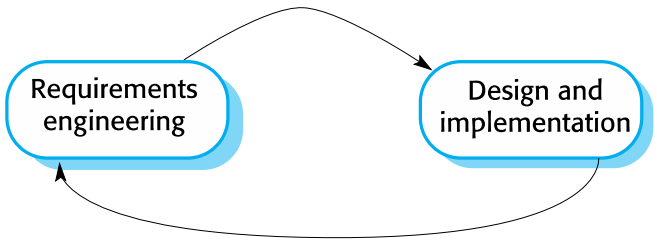
Agile development

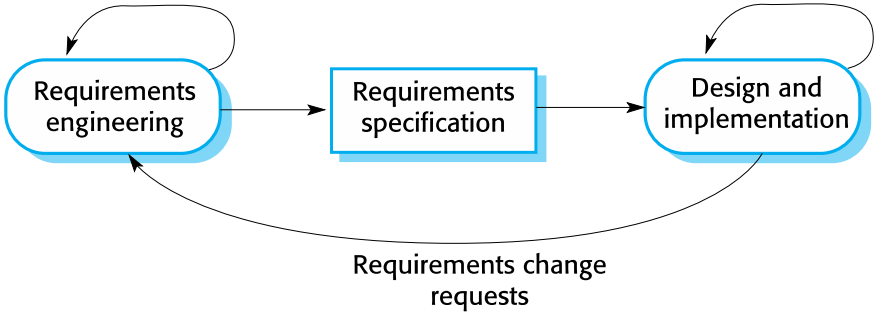
[Common characteristics of agile methods]

- Program specification, design and implementation are interleaved.
  - No detailed system specification, minimized design documentation
- The system is developed as a series of versions or increments.
  - Stakeholders propose new requirements for a later version of the system.
- Extensive tool support is used to support the SW development.
  - Automated testing tools, automated user interface production tools, ...

Plan-driven and Agile development

[Plan-driven approach vs Agile approach]

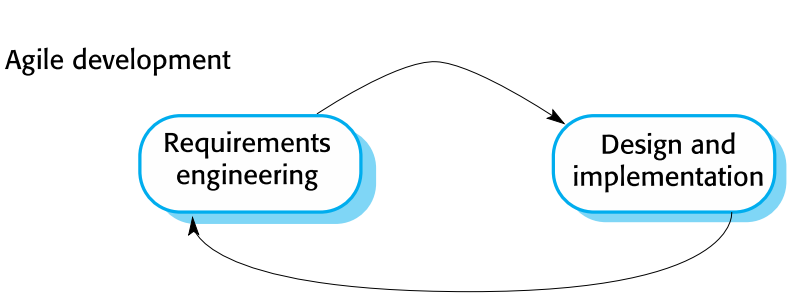


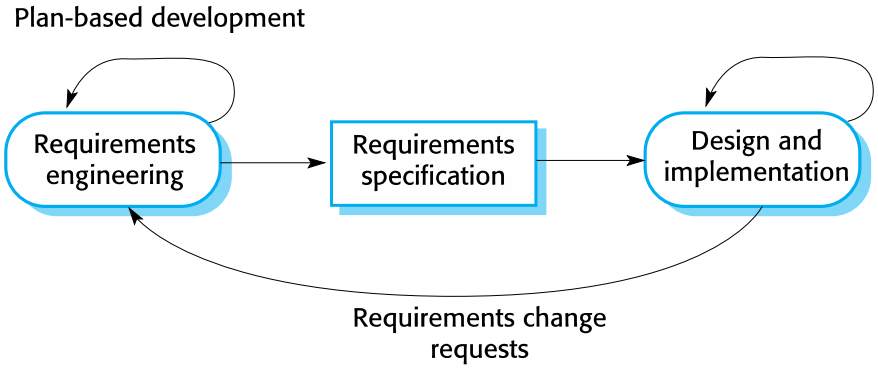


Plan-driven and Agile development

[Plan-driven approach]

- Iteration occurs within activities.
- Formal documents are used to communicate between stages of the process.





[Agile approach]

- Iteration occurs across activities.
- The requirements and the design (implementation) are developed together.

What is Agile methods

[Plan-driven approach vs Agile approach]

- Motivation (in the 1980s and early 1990s)
  - Dissatisfaction with this heavyweight, plan-driven development approach.
  - Time: documentation and design > SW development and testing.

- Agile methods (in the late 1990s)
  - Focus on the software itself rather than on its documentation.
  - Deliver working SW quickly.
  - Evolve SW quickly to meet changing requirements.

Agile Manifesto

https://agile-lounge.com/18-years-of-agile-manifesto-for-software-development/

Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

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.

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

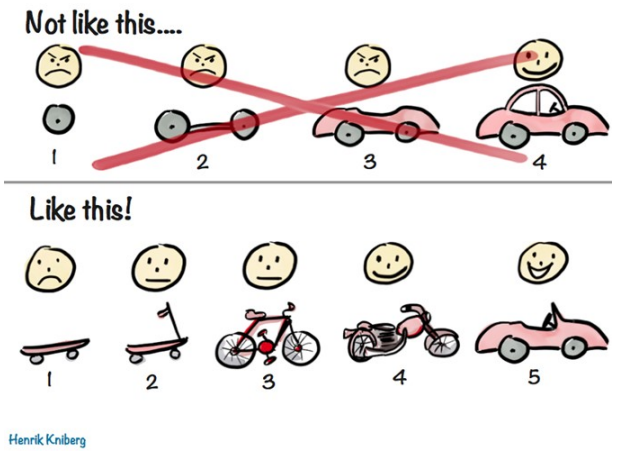
The principles of Agile methods

[All agile methods share a set of principles.]

- Customer involvement
  - Customers are closely involved throughout the development process.

    (Providing and prioritizing new system requirements and evaluating the iterations of the system.)

- Incremental delivery
  - The SW is developed in increments.



The principles of Agile methods

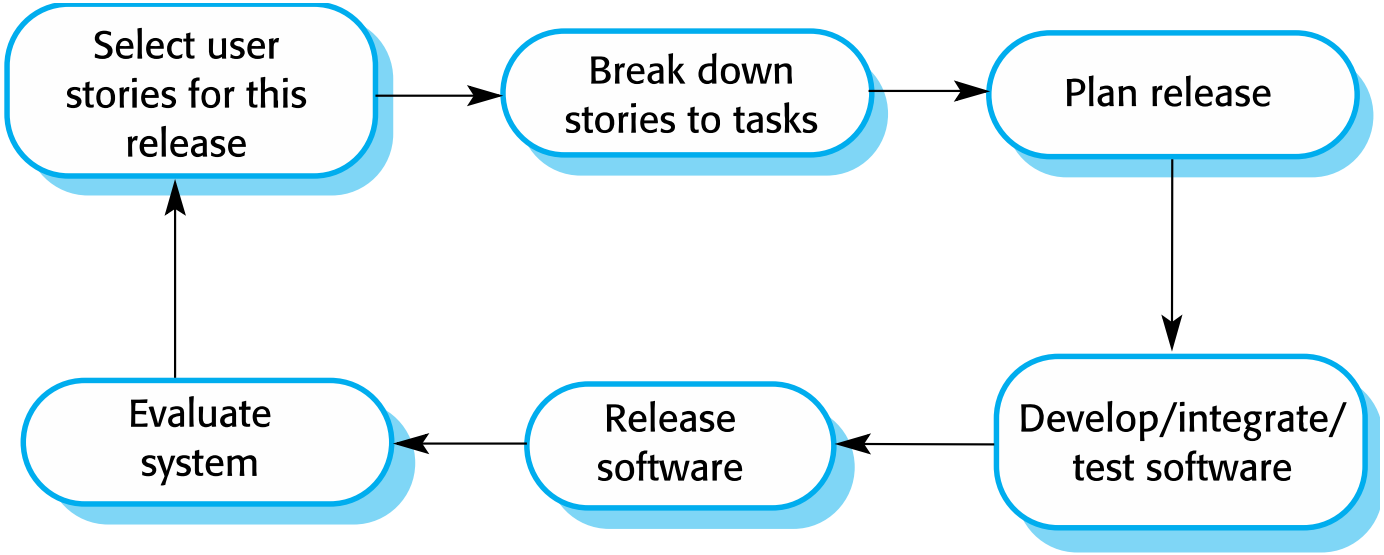
[All agile methods share a set of principles.]

- Embrace change
  - Expect the system requirements to change and design the system to accommodate these changes.
- Maintain simplicity
  - Focus on simplicity in the SW and in the development process.
  - Eliminate the complexity from the system actively.
- People, not process
  - Team members should be left to develop their own ways of working without prescriptive processes.

Extreme programming

[Extreme Programming (XP)]

- Push an iterative development to "extreme" levels.
  - Several versions of a system are developed, integrated, and tested in a day.



"The XP release cycle"

Extreme programming

[Extreme Programming Practices]

1. Collective ownership
2. Continuous integration
3. Incremental planning
4. On-site customer
5. Pair programming
6. Refactoring
7. Simple design
8. Small releases
9. Sustainable pace
10. Test first development

In practice, companies choose the XP practices selectively.

Extreme programming

[Extreme Programming Practices]

- Collective ownership
  - The pairs of developers work on all areas of the system.
  - All the developers take responsibility for all of the code.
- Continuous integration
  - As soon as the work is complete, it is integrated into the whole system.
  - After the integration, all the unit tests in the system must pass.
- Incremental planning
  - Requirements (e.g., story cards) are determined by the time available and their relative priority.

Extreme programming

[Extreme Programming Practices]

- On-site customer
  - The customer, a member of the development team, brings system requirements to the implementation team.
- Pair programming
  - Developers work in pairs and check each other's work.
- Refactoring
  - Refactor the code continuously for keeping the code simple.
- Simple design
  - For the design, it is enough to meet the current requirements only.

Extreme programming

[Extreme Programming Practices]

- Small releases
  - The minimal useful set of functionality is developed first.
  - Releases of the system incrementally add functionality to the first release.
- Sustainable pace
  - Large amounts of overtime are not considered acceptable.
- Test first development
  - Write tests for a new piece of functionality before its implementation.

Let's discuss the most important agile development practices.

Extreme programming

[User Stories]

- A scenario of use that might be experienced by a system user.

  1) Develop a "story card" that briefly describes a story that encapsulates the customer needs.

  2) Break the story card down into tasks.

  3) Estimate the effort and resources required for implementing each task.

  4) Prioritize the stories for implementation that can be used to deliver useful business support.

- Pros and Cons

  (+) Easy understanding of the story.

  (-) Incompleteness of the story.

A ‘prescribing medication’ story

Prescribing medication

Kate is a doctor who wishes to prescribe medication for a patient attending a clinic. The patient record is already displayed on her computer so she clicks on the medication field and can select ‘current medication’, ‘new medication’ or ‘formulary’.

If she selects ‘current medication’, the system asks her to check the dose; If she wants to change the dose, she enters the new dose then confirms the prescription.

If she chooses ‘new medication’, the system assumes that she knows which medication to prescribe. She types the first few letters of the drug name. The system displays a list of possible drugs starting with these letters. She chooses the required medication and the system responds by asking her to check that the medication selected is correct. She enters the dose then confirms the prescription.

If she chooses ‘formulary’, the system displays a search box for the approved formulary. She can then search for the drug required. She selects a drug and is asked to check that the medication is correct. She enters the dose then confirms the prescription.

The system always checks that the dose is within the approved range. If it isn’t, Kate is asked to change the dose.

After Kate has confirmed the prescription, it will be displayed for checking. She either clicks ‘OK’ or ‘Change’. If she clicks ‘OK’, the prescription is recorded on the audit database. If she clicks on ‘Change’, she reenters the ‘Prescribing medication’ process.

Extreme programming

[Examples of task cards for prescribing medication]

Task 1: Change dose of prescribed drug

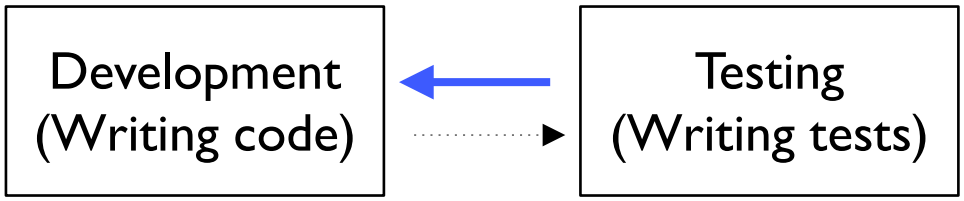
Task 2: Formulary selection

Task 3: Dose checking

Dose checking is a safety precaution to check that the doctor has not prescribed a dangerously small or large dose.
Using the formulary id for the generic drug name, lookup the formulary and retrieve the recommended maximum and minimum dose.
Check the prescribed dose against the minimum and maximum. If outside the range, issue an error message saying that the dose is too high or too low.
If within the range, enable the 'Confirm' button.

Test-first development

[Test-first development]



- Key features of testing in XP
  - Test-first development.
  - Incremental test development from scenarios.
  - User involvement in the test development and validation.
  - The use of automated testing frameworks.

Test-first development

[Test-first development]

- Assume that user stories have been developed, and these have been broken down into a set of task cards.

Test 4: Dose checking

Input:
1. A number in mg representing a single dose of the drug.
2. A number representing the number of single doses per day.

Tests:

?

Output:
OK or error message indicating that the dose is outside the safe range.

Test-first development

[Test-first development]

- Assume that user stories have been developed, and these have been broken down into a set of task cards.

Test 4: Dose checking

Input:
1. A number in mg representing a single dose of the drug.
2. A number representing the number of single doses per day.

Tests:
1. Test for inputs where the single dose is correct but the frequency is too high or too low.
2. Test for inputs where the frequency is in the permitted range but the single dose is too high or too low.
3. Test for inputs where both the single dose and frequency are too high or too low.
4. Test for inputs where both single dose and frequency are in the permitted range.

Output:
OK or error message indicating that the dose is outside the safe range.

Test-first development

[Problems in test-first development]

- Programmers sometimes take short cuts when writing tests?
- Some tests can be very difficult to write incrementally.
- It is difficult to judge the completeness of a set of tests.
  - Complete coverage ?

Pair programming

[What is pair programming?]

- Programmers work in pairs to develop the software.
- Pairs are created dynamically so that all team members work with each other during the development process.

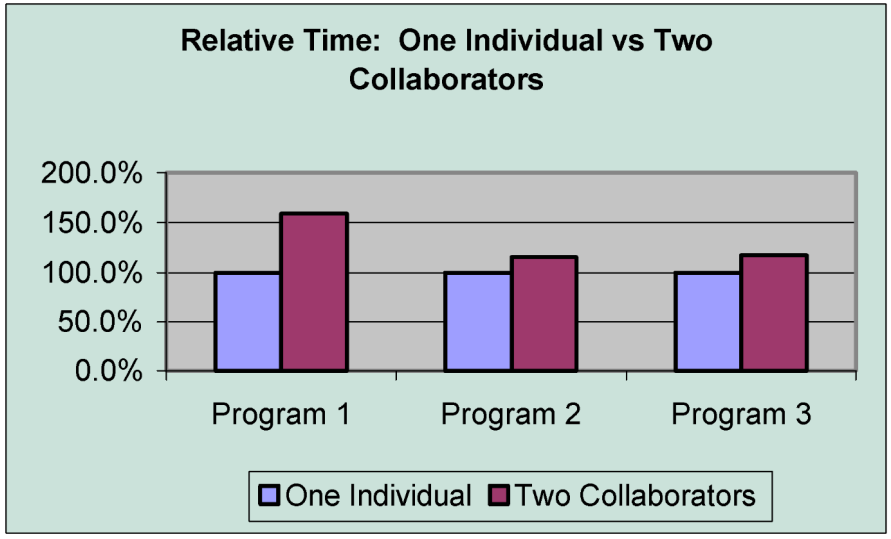
[Advantages]

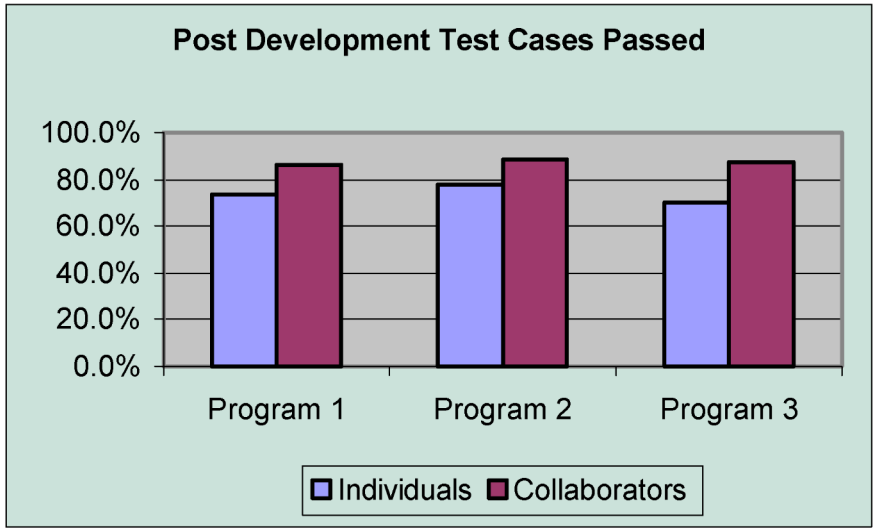
- Support collective ownership and responsibility for the system.
- Act as an informal review process because each line of code is looked at by at least two people.
- Encourage refactoring to improve the software structure.

Pair programming

[The Costs and Benefits of Pair Programming]

- At the University of Utah, the authors investigated the economics of pair programming. (Alistair Cockburn and Laurie Williams, 2000)

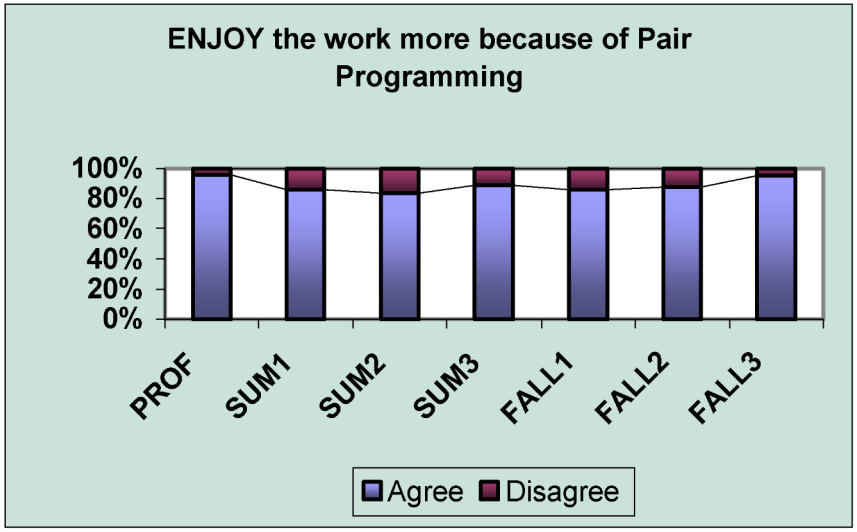


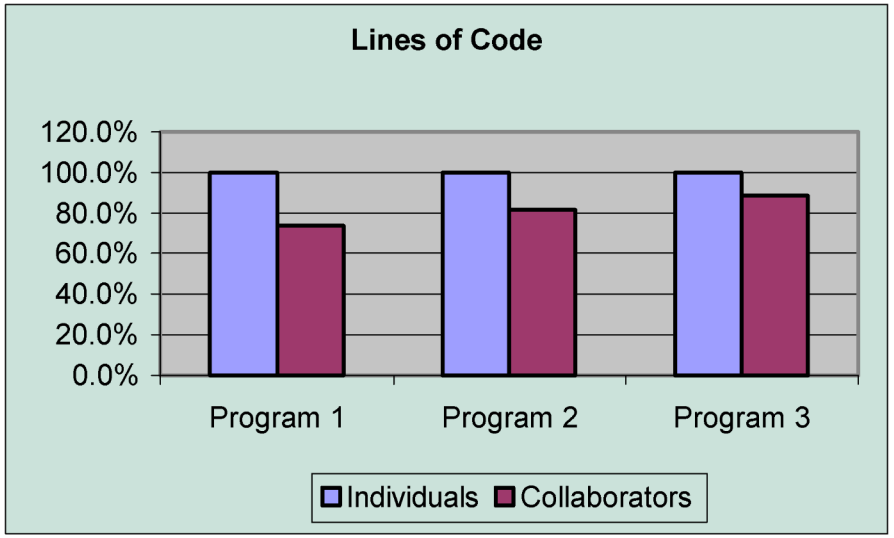


Pair programming

[The Costs and Benefits of Pair Programming]

- At the University of Utah, the authors investigated the economics of pair programming. (Alistair Cockburn and Laurie Williams, 2000)





Agile project management: Scrum

[Motivation]

- The informal planning and project control clashed with this business requirement for visibility.
- Agile development teams should manage the time and resources available to the team effectively.

[Scrum]

- A framework for organizing agile projects and providing external visibility of what is going on at least.
- Using new terminologies to differentiate it from conventional project management. (e.g., ScrumMaster, Sprint, Product backlog)

Agile project management: Scrum

[Scrum terminology]

- Development team
  - A self-organizing group of software developers. (< 7 people)
  - Developing the software and other essential project documents.
- Sprint
  - A development iteration. (2 ~ 4 weeks)
- Scrum
  - A daily meeting that reviews progress and prioritizes work to do that day.
  - A short face-to-face meeting that includes the whole team. (Ideal case)

Agile project management: Scrum

[Scrum terminology]

- Scrum Master (= Project manager ???)
  - Checking that the Scrum process is followed.
  - Guiding the team in the effective use of Scrum.
  - Interfacing with the rest of the company.

- Product backlog
  - A list of "to do" items which the Scrum team must tackle.
  - (Ex) User stories, SW feature definitions, SW requirements, ...

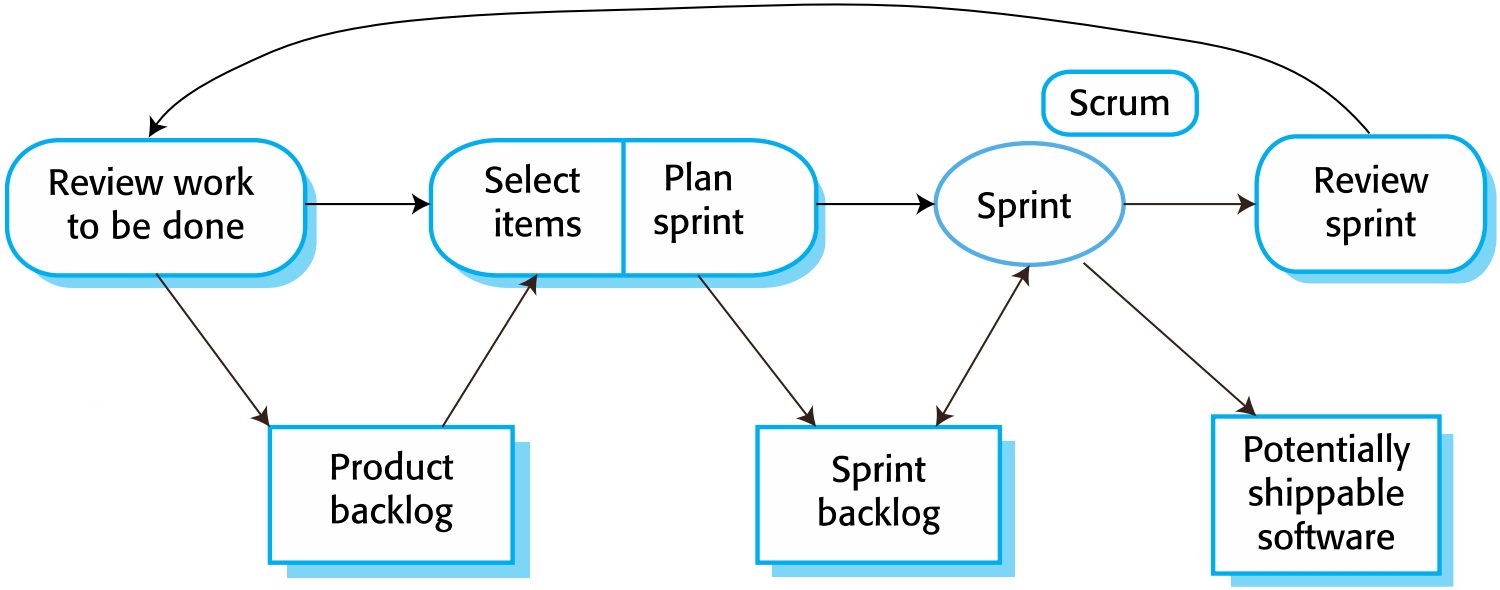
Agile project management: Scrum

[Scrum terminology]

- Product owner
  - Identifying product features or requirements.
  - Reviewing the product backlog to ensure that the project continues to meet critical business needs.
  - Ex) A customer, product manager, other stakeholder representative.
- Velocity
  - Estimating how much backlog can be covered by a team in a single sprint.

Agile project management: Scrum

[The Scrum process or sprint cycle]



- Sprints are never extended to take account of unfinished work. Items are returned to the product backlog.
- A Scrum board is used for the daily interactions among the teams.

Agile project management: Scrum

[The benefits of Scrum]

- The product is broken down into a set of manageable and understandable chunks that stakeholders can relate to.
- Unstable requirements do not hinder progress.
- The whole team has visibility of everything. (team communication↑)
- Customers see on-time delivery of increments and gain feedback.
- Trust between customers and developers is established.

Scaling agile methods

[Scaling agile methods]

Large software systems	VS	Small and medium-sized systems

[Practical problems with agile methods]

- Contractual issues can be a major problem when agile methods are used.
- Agile methods are most appropriate for new SW development rather than for SW maintenance.
- SW development now involves worldwide distributed teams rather than small co-located teams.

Scaling agile methods

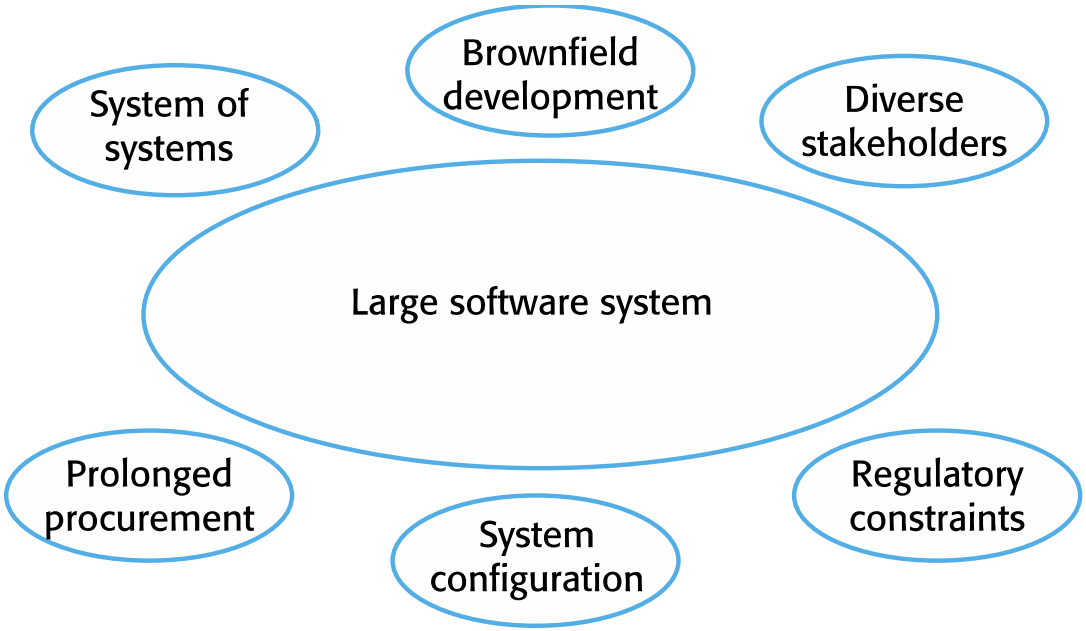
[Problems when using agile methods for SW maintenance]

- Lack of product documentation
  - There is no formal documentation, which tells the software engineer what the system is supposed to do.
- Development team continuity
  - If an agile development team is broken up, the implicit knowledge is lost.
  - It is difficult for new team members to build up the same understanding of the system and its components.
- Keeping customers involved

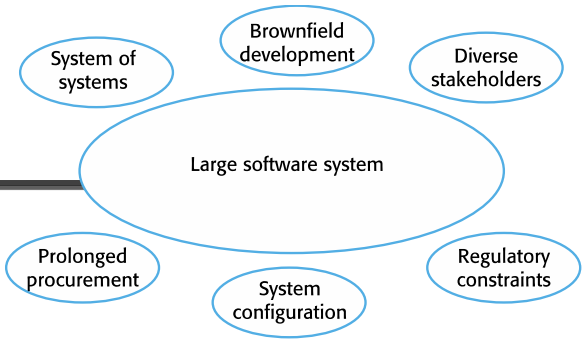
Scaling agile methods

[Agile methods for large systems]

- Agile methods have to evolve to be used for large-scale SW development.
- Six principal factors contribute to the complexity of large-scale SW systems.



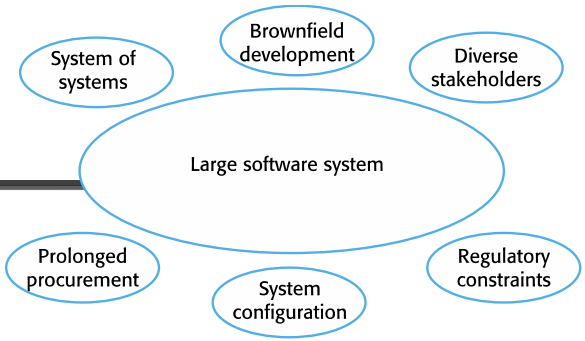
Scaling agile methods



[Agile methods for large systems]

- System of systems
  - Separate teams develop each system in different places and time zones.
  - It is impossible for each team to have a view of the whole system.
- Brownfield development
  - Include and interact with a number of existing systems.
- System configuration
  - A significant fraction of the development is concerned with system configuration rather than original code development.

Scaling agile methods



[Agile methods for large systems]

- Regulatory constraints
  - Large systems are constrained by external rules and regulations limiting the way that they can be developed.
- Prolonged procurement
  - It is difficult to maintain coherent teams who know about the system.
- Diverse stakeholders
  - It is infeasible to involve all of the different stakeholders in the development process.

Large system development

[Common features of approaches to scaling agile methods]

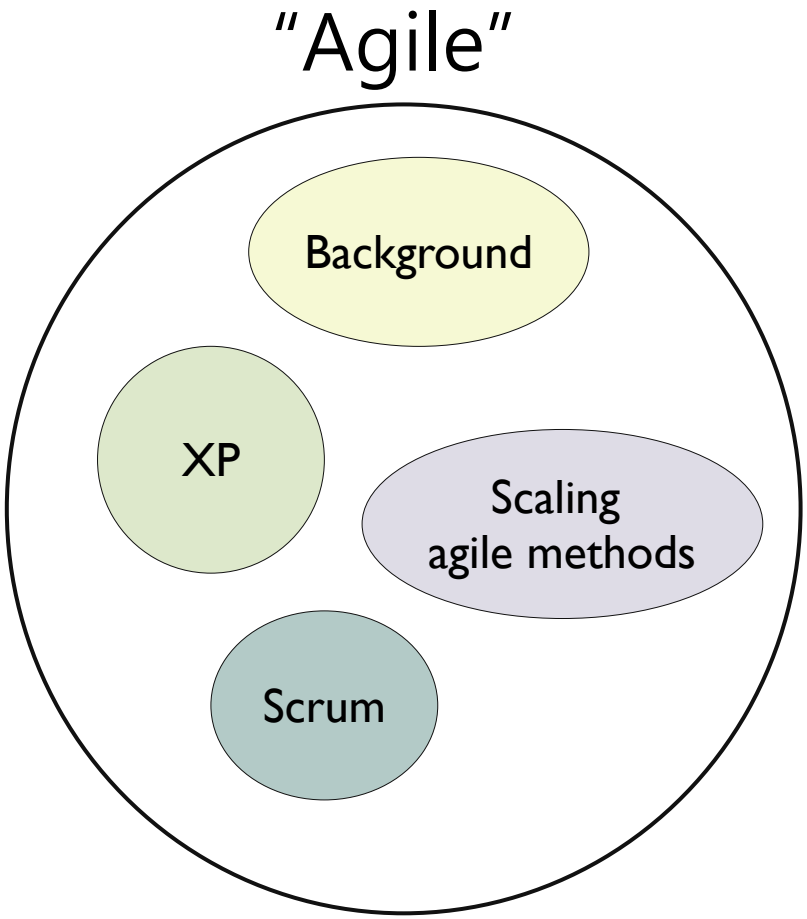
- A completely incremental approach to requirements engineering is impossible.
- There cannot be a single product owner or customer representative.
- It is not possible to focus only on the code of the system.
- Cross-team communication mechanisms have to be designed and used.
- Continuous integration, where the whole system is built every time any developer checks in a change, is practically impossible.

Large system development

[Agile for large system: Multi-team Scrum]

Role replication	Each team has a Product Owner and ScrumMaster.
Product architects	Each team chooses a product architect and these architects collaborate to design the overall system architecture.
Release alignment	The dates of product releases from each team are aligned so that a demonstrable and complete system is produced.
Scrum of Scrums	There is a daily Scrum of Scrums where representatives from each team meet to discuss progress and plan work to be done.

Summary



Thank You