Software Engineering

2
Software Processes (2)
Agile Software Development

School of Computer Science
Prof. Euijong Lee

Topics covered

01 Background of agile method

02 Agile methods

03 Agile development techniques

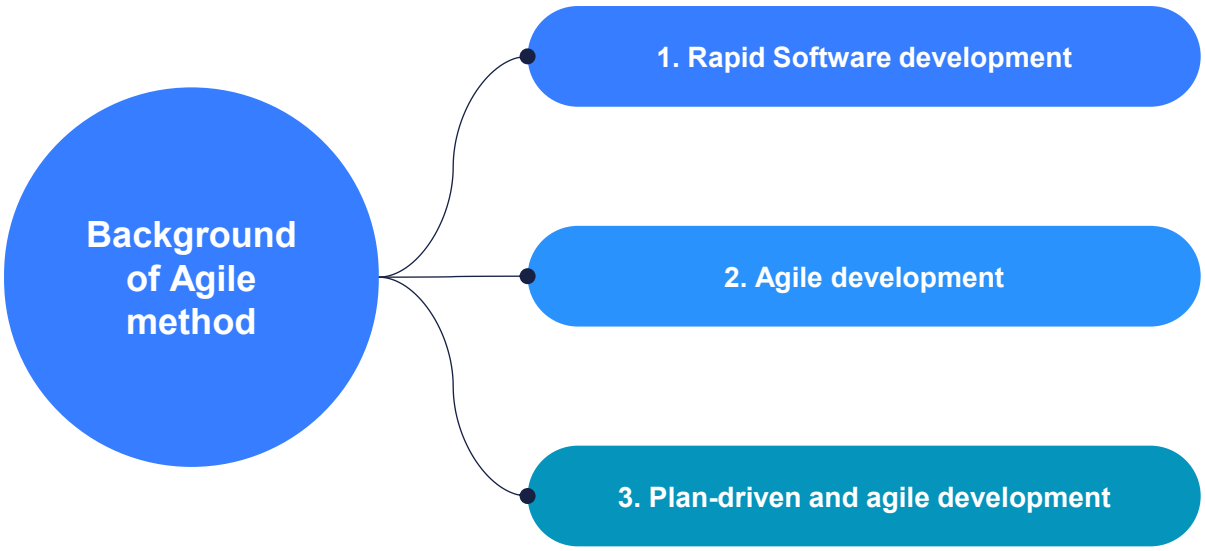
04 Agile project management

05 Scaling agile methods

Agile?

- Agile success cases
  - https://hrbulletin.net/casestudy/%EB%A0%88%EA%B3%A0%EC%9D%98-%EC%95%A0%EC%9E%90%EC%9D%BC-%EC%A0%84%ED%99%98-1%EB%85%84%EC%9D%98-%EC%84%B1%EA%B3%BC/
  - https://news.skhynix.co.kr/post/Meeting-of-SK-hynix-and-Agile

2-1: Background of agile method



Background of agile method

[Rapid software development : 신속한 소프트웨어 개발]

- **Rapid development and delivery is now often the most important requirement for software systems**
  - Businesses operate in a fast –changing requirement and it is practically impossible to produce a set of stable software requirements
  - Software has to evolve quickly to reflect changing business needs.
- **Plan-driven development**
  - Essential for some types of system but does not meet these business needs.
- **Agile development methods**
  - Emerged in the late 1990s whose aim was to radically reduce the delivery time for working software systems

Background of agile method

Agile development

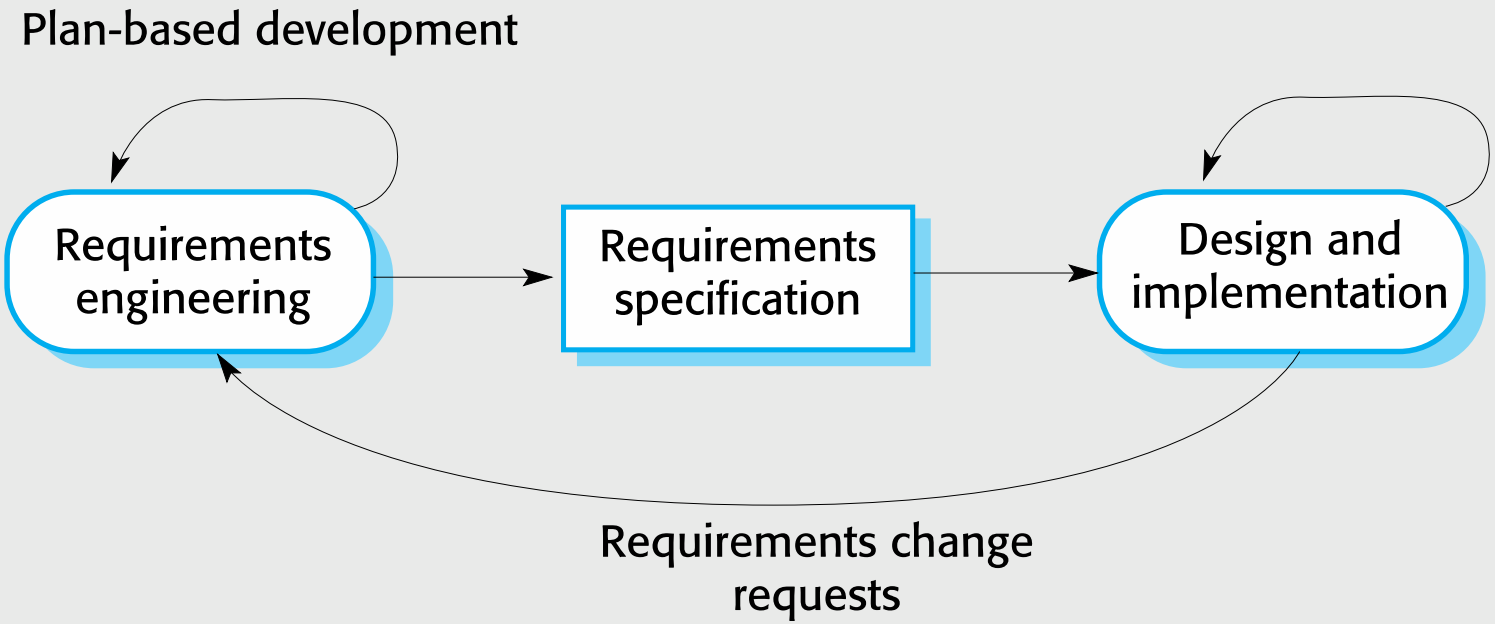
- Program specification, design and implementation are **inter-leaved**
- The system is developed **as a series of versions or increments** with stakeholders involved in version specification and evaluation
- **Frequent delivery** of new versions for evaluation
- **Extensive tool support** (e.g. automated testing tools) used to support development.
- **Minimal documentation** – focus on working code

Plan-driven and Agile development (1)

- **Plan-driven** development
  - **A plan-driven approach** to software engineering is based around **separate development stages** with the **outputs** to be produced at each of these stages planned in advance.
  - Not necessarily waterfall model – plan-driven, incremental development is possible
  - Iteration occurs within activities.

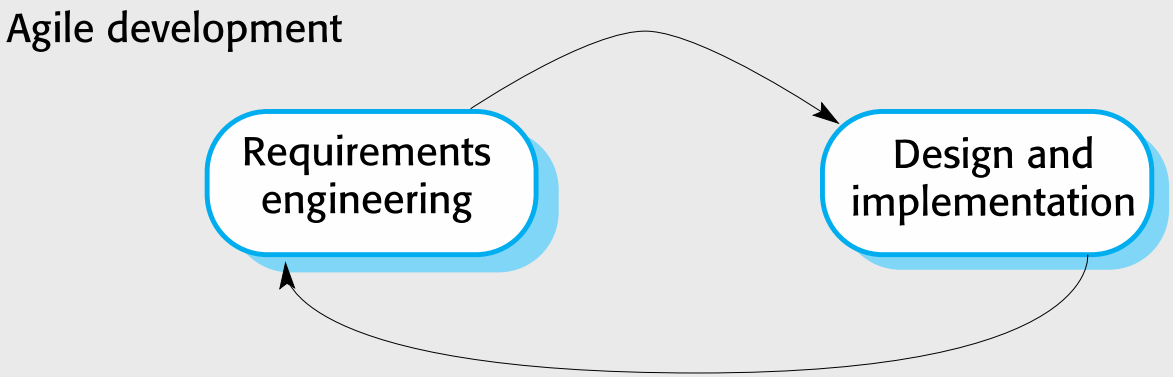
Plan-driven and Agile development (1)

- **Plan-driven** development

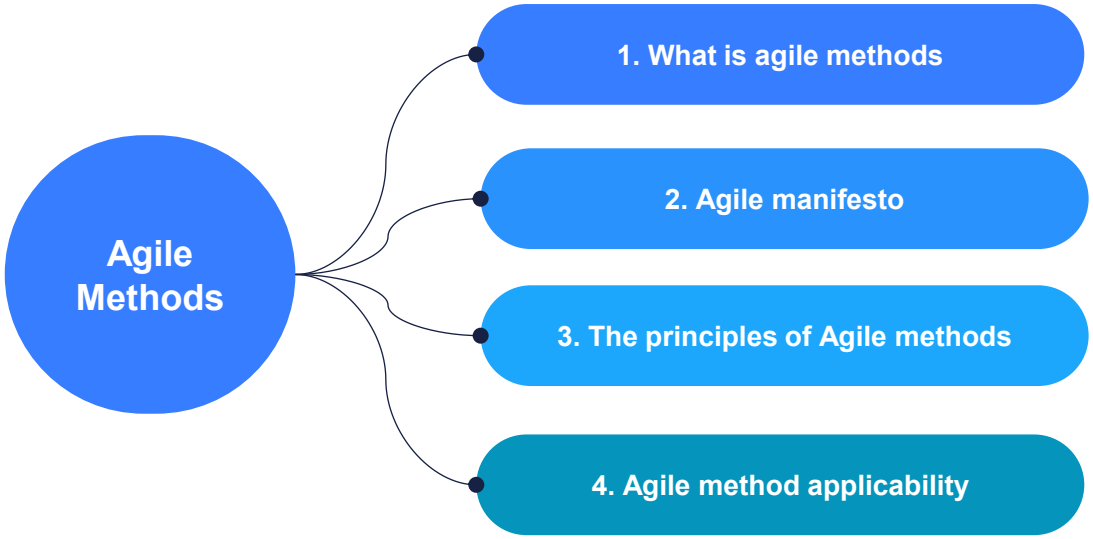


Plan-driven and Agile development (2)

- **Agile** development
  - Specification, design, implementation and testing are **inter-leaved**
  - The **outputs** from the development process are decided through a **process of negotiation** during the software development process.



2-2: Agile methods



What is Agile methods

[What is agile methods?]

Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods.

- Characteristics of agile methods
  - **Focus on the code** rather than the design
  - Are based on an **iterative approach** to software development
  - Are intended to deliver working software **quickly** and **evolve** this quickly to meet changing requirements.

- The aim of agile methods
  - **reduce overheads** in the software process (e.g. by limiting documentation)
  - able to respond **quickly to changing** requirements without excessive rework.

Agile Manifesto

[Agile manifesto (http://agilemanifesto.org)]

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.

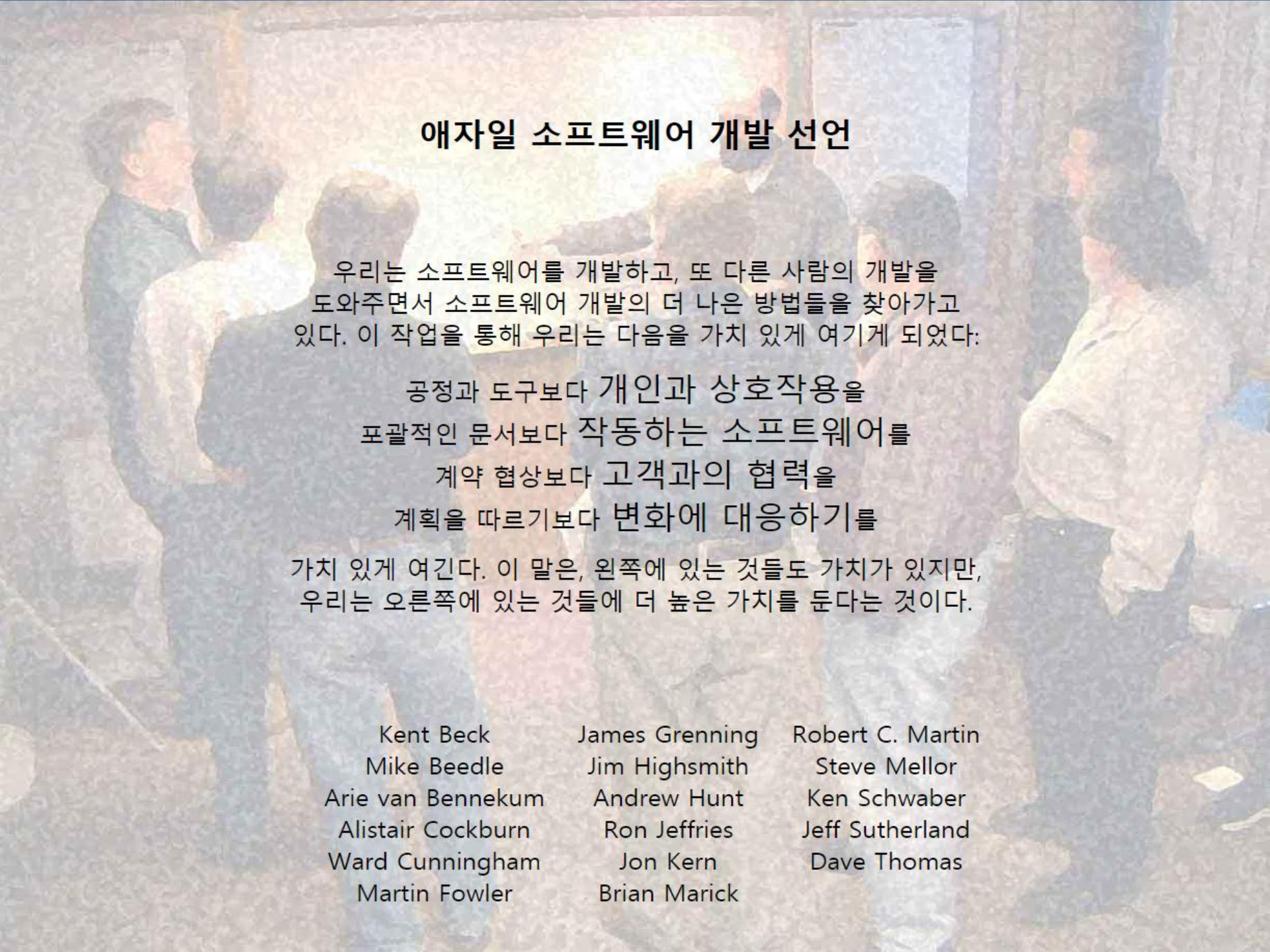
Manifesto for Agile Software Development

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



애자일 소프트웨어 개발 선언

우리는 소프트웨어를 개발하고, 또 다른 사람의 개발을
도와주면서 소프트웨어 개발의 더 나은 방법들을 찾아가고
있다. 이 작업을 통해 우리는 다음을 가치 있게 여기게 되었다:

공정과 도구보다 **개인과 상호작용**을
포괄적인 문서보다 **작동하는 소프트웨어**를
계약 협상보다 **고객과의 협력**을
계획을 따르기보다 **변화에 대응하기**를

가치 있게 여긴다. 이 말은, 왼쪽에 있는 것들도 가치가 있지만,
우리는 오른쪽에 있는 것들에 더 높은 가치를 둔다는 것이다.

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 (1)

Customer involvement (고객의 참여)

- Customers should be closely involved throughout the development process.
- Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.

Incremental delivery (점증적 인도)

- The software is developed in increments with the customer specifying the requirements to be included in each increment.

The principles of Agile methods (2)

Embrace change (변화의 수용)

- Expect the system requirements to change and so design the system to accommodate these changes.

Maintain simplicity (단순성 유지)

- Focus on simplicity in both the software being developed and in the development process.
- Wherever possible, actively work to eliminate complexity from the system.

The principles of Agile methods (3)

People not process (프로세스가 아닌 사람)

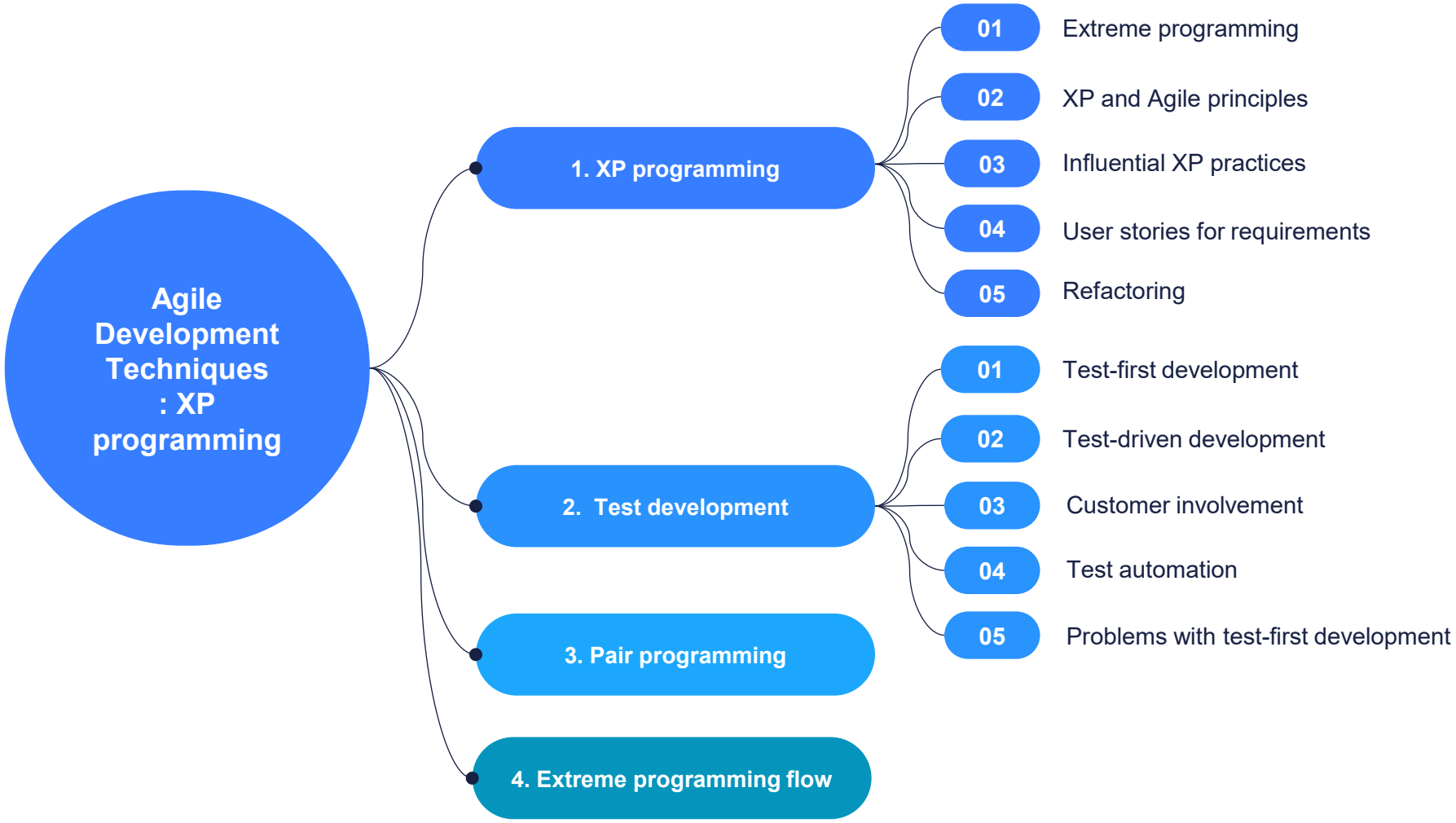
- The skills of the development team should be recognized and exploited.
- Team members should be left to develop their own ways of working without prescriptive processes.

Agile method applicability : 적용가능 분야

- **Product development**
  - a software company is developing a **small or medium-sized** product for sale.
  - Virtually all software products and apps are now developed using an agile approach

- **Custom system development within an organization**
  - there is a **clear commitment** from the **customer to become involved** in the development process
  - there are few external rules and regulations that affect the software.

2-3: Agile development techniques : XP programming



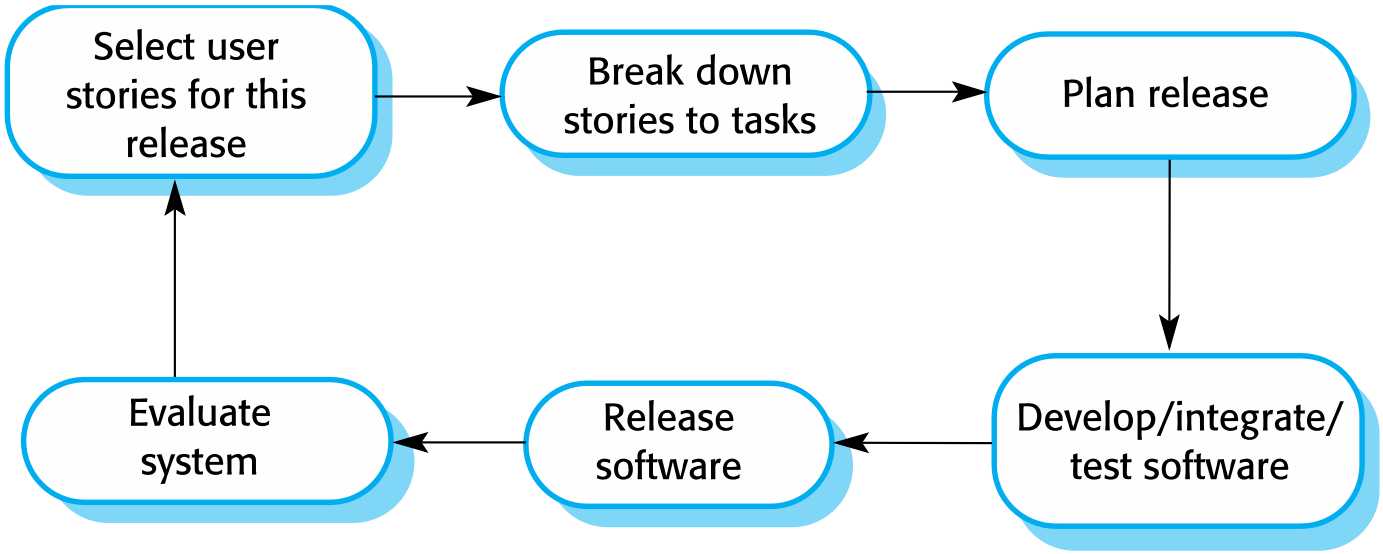
XP programming

[Extreme programming]

- A very influential agile method, developed in the late 1990s, that introduced a range of agile development techniques.

- Extreme Programming (XP) takes an '**extreme** : 극단적' approach to iterative development.
  - **New versions** may be built **several times per day**;
  - **Increments** are delivered to customers **every 2 weeks**;
  - All **tests must be run for every build** and the build is only accepted if tests run successfully.

XP programming



XP programming

[Extreme programming]

Incremental planning (점진적인 계획)

- Requirements are recorded on **story cards** and the stories to be included in a release are determined by the time available and their relative priority.
- The developers **break these stories** into development '**Tasks**'.

Small releases (소규모 릴리즈)

- The **minimal useful set of functionality** that provides business value is developed **first**.
- **Releases** of the system are **frequent** and **incrementally** add functionality to the first release.

XP programming

[Extreme programming]

Simple design (단순한 설계)

- **Enough design** is carried out to meet the **current requirements** and no more.

Test-first development (테스트 우선 개발)

- An automated unit **test** framework is used to write tests for a new piece of functionality **before that functionality itself is implemented**.

Refactoring (리팩토링)

- All developers are **expected to refactor the code** continuously as soon as possible code **improvements are found**.
- This keeps the code simple and maintainable.

XP programming

[Extreme programming]

Pair programming (짝 프로그래밍)

- Developers **work in pairs**, checking each other's work and providing the support to always do a good job.

Collective ownership (공동 소유권)

- The pairs of **developers** work on **all areas** of the system, so that no islands of expertise develop, and all the developers take **responsibility for all of the code**.
- **Anyone** can **change** anything.

Continuous integration (연속적 통합)

- As soon as the work on a task is complete, it is **integrated** into the whole system.
- After any such integration, all the unit tests in the system must pass.

XP programming

[Extreme programming]

Sustainable pace (유지할 수 있는 속도)

- Large amounts of **overtime are not considered** acceptable as the net effect is often to reduce code quality and medium-term productivity

On-site customer (고객의 참여)

- A representative of the **end-user** of the system (the customer) should be available **full time** for the use of the XP team.
- In an extreme programming process, the **customer** is a member of the **development team** and is responsible for bringing system requirements to the team for implementation.

XP programming

[XP and agile principles]

- **Incremental development** is supported through small, frequent system releases.
- **Customer involvement** means full-time customer engagement with the team.
- People not process through **pair programming**, **collective ownership** and a process that **avoids long working hours.**
- **Change supported** through regular system **releases**.
- Maintaining **simplicity** through constant **refactoring** of code.

XP programming

Influential XP practices

- Extreme programming has a technical focus and is not easy to integrate with management practice in most organizations.

- Consequently, while agile development uses practices from XP, the method as originally defined is **not widely used**.

- Key practices
  - User stories for specification
  - Refactoring
  - Test-first development
  - Pair programming

XP programming

User stories for requirements

- In XP, a **customer** or user is part of the **XP team** and is responsible for making **decisions on requirements**.
- User **requirements** are expressed as **user stories or scenarios**.
- These are written on cards and the development team **break them down** into implementation **tasks**.
- These **tasks** are the **basis** of **schedule** and **cost** estimates.
- The customer **chooses** the **stories** for inclusion in the **next release** based on their **priorities** and the **schedule** estimates.

XP programming - A ‘prescribing medication’ story

Prescribing medication

The record of the patient must be open for input. Click on the medication field and select either ‘current medication’, ‘new medication’ or ‘formulary’.

If you select ‘current medication’, you will be asked to check the dose; If you wish to change the dose, enter the new dose then confirm the prescription.

If you choose, ‘new medication’, the system assumes that you know which medication you wish to prescribe. Type the first few letters of the drug name. You will then see a list of possible drugs starting with these letters. Choose the required medication. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

If you choose ‘formulary’, you will be presented with a search box for the approved formulary. Search for the drug required then select it. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

In all cases, the system will check that the dose is within the approved range and will ask you to change it if it is outside the range of recommended doses.

After you have confirmed the prescription, it will be displayed for checking. Either click ‘OK’ or ‘Change’. If you click ‘OK’, your prescription will be recorded on the audit database. If you click ‘Change’, you reenter the ‘Prescribing medication’ process.

XP 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.

XP programming

Refactoring (1)

- Conventional wisdom in software engineering is to design for change.
- It is worth spending time and effort anticipating changes as this reduces costs later in the life cycle.
- XP, however, maintains that this is not worthwhile as changes cannot be reliably anticipated.
- Rather, it proposes **constant code improvement (refactoring)** to make changes easier when they have to be implemented.

XP programming

Refactoring (2)

- Programming team look for **possible software improvements** and make these improvements even where there is **no immediate need for them**.
- This **improves the understandability** of the software and so **reduces** the need for **documentation**.
- **Changes** are easier to make because the code is **well-structured and clear.**
- However, **some changes** requires **architecture refactoring** and this is much more **expensive**.

XP programming

Examples of refactoring

- Re-organization of a class hierarchy to remove duplicate code.
- Tidying up and renaming attributes and methods to make them easier to understand.
- The replacement of inline code with calls to methods that have been included in a program library.

Test development

Test-first development

- Testing is central to XP and XP has developed an approach where the program is **tested after every change** has been made.

- **XP testing features:**
  - Test-first development.
  - Incremental test development from scenarios.
  - User involvement in test development and validation.
  - Automated test harnesses are used to run all component tests each time that a new release is built.

Test development

Test-driven development

- Writing **tests before code** clarifies the requirements to be implemented.
- Tests are written as programs rather than data so that they can be executed automatically.
- The test includes a check that it has executed correctly.
  - Usually relies on a testing framework such as Junit (https://junit.org/junit5/).
- All previous and **new tests** are **run automatically** when new functionality is added, thus checking that the new functionality has not introduced errors.

Test development

Customer involvement

- The **role of the customer** in the testing process is to help develop **acceptance tests** for the stories that are to be implemented in the next release of the system.

- The **customer** who is part of the team writes tests as development proceeds.
  - All new code is therefore validated to ensure that it is what the customer needs.

- However, people adopting the **customer** role have **limited time** available and so cannot work full-time with the development team.
  - They may feel that providing the requirements was enough of a contribution and so may be reluctant to get involved in the testing process.

Test development

[Test case description for dose checking]

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.
2. Test for inputs where the single dose is too high and too low.
3. Test for inputs where the single dose * frequency is too high and too low.
4. Test for inputs where single dose * frequency is in the permitted range.

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

Test development

Test automation

- **Test automation** means that tests are **written as executable components** before the task is implemented
  - These testing components should be stand-alone, should simulate the submission of input to be tested and should check that the result meets the output specification.
  - An automated test framework (e.g. JUnit) is a system that makes it easy to write executable tests and submit a set of tests for execution.

- As testing is automated, there is **always a set of tests** that can be **quickly** and **easily** executed
  - Whenever any functionality is added to the system, the tests can be run and problems that the new code has introduced can be caught immediately.

Test development

Problems with test-first development

- Programmers prefer programming to testing and sometimes they **take short cuts** when writing tests.
  - For example, they may write incomplete tests that do not check for all possible exceptions that may occur.

- Some tests can be **very difficult** to write incrementally.
  - For example, in a complex user interface, it is often difficult to write unit tests for the code that implements the ‘display logic’ and workflow between screens.

- It **difficult** to judge the **completeness** of a set of tests.
  - Although you may have a lot of system tests, your test set may not provide complete coverage.

Pair programming (1)

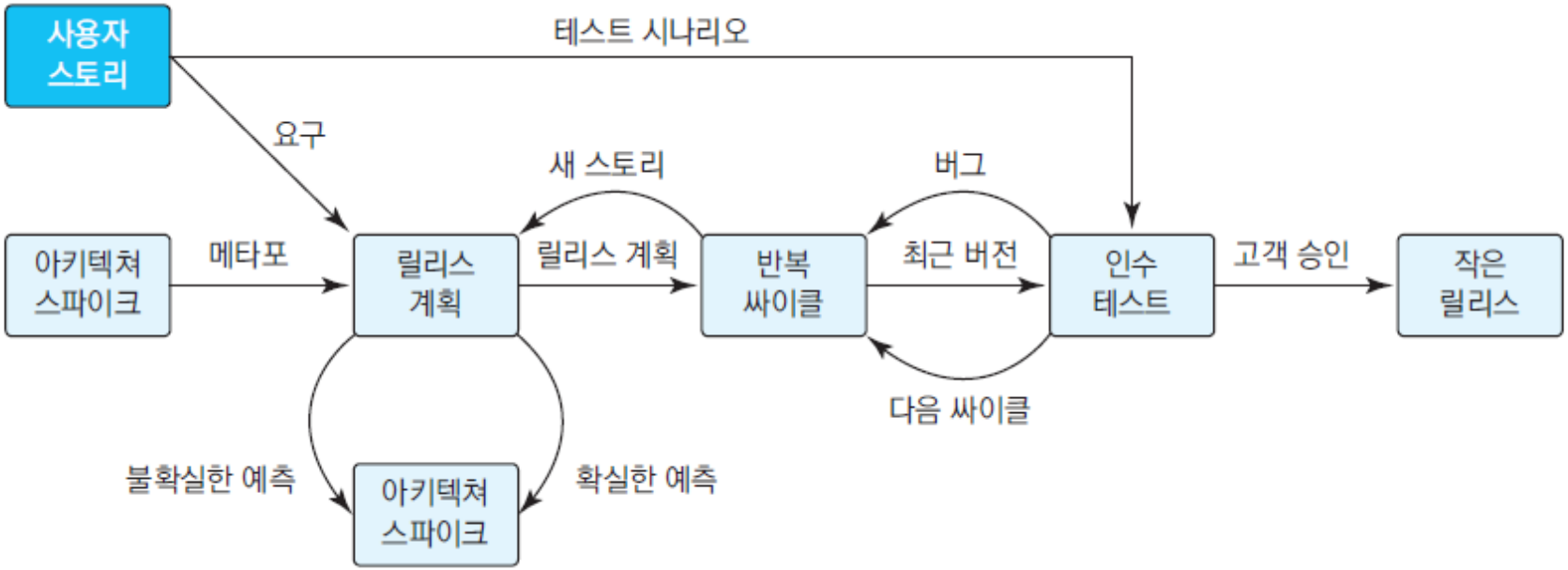
- Pair programming involves **programmers** working in **pairs**, developing code together.
- This helps develop **common ownership of code** and **spreads knowledge** across the team.
- It serves as an informal **review process** as each line of code is looked at by more than 1 person.
- It **encourages refactoring** as the whole team can benefit from improving the system code.

Pair programming (2)

- In pair programming, programmers sit together at the **same computer to develop** the software.
- Pairs are created dynamically so **that all team members work with each other** during the development process.
- The **sharing of knowledge** that happens during pair programming is very important as it reduces the overall risks to a project when team members leave.
- Pair programming is not necessarily inefficient and there is some evidence that suggests that a pair working together is more efficient than 2 programmers working separately.
- But, sometimes, it is inefficient if high-level programmers work together

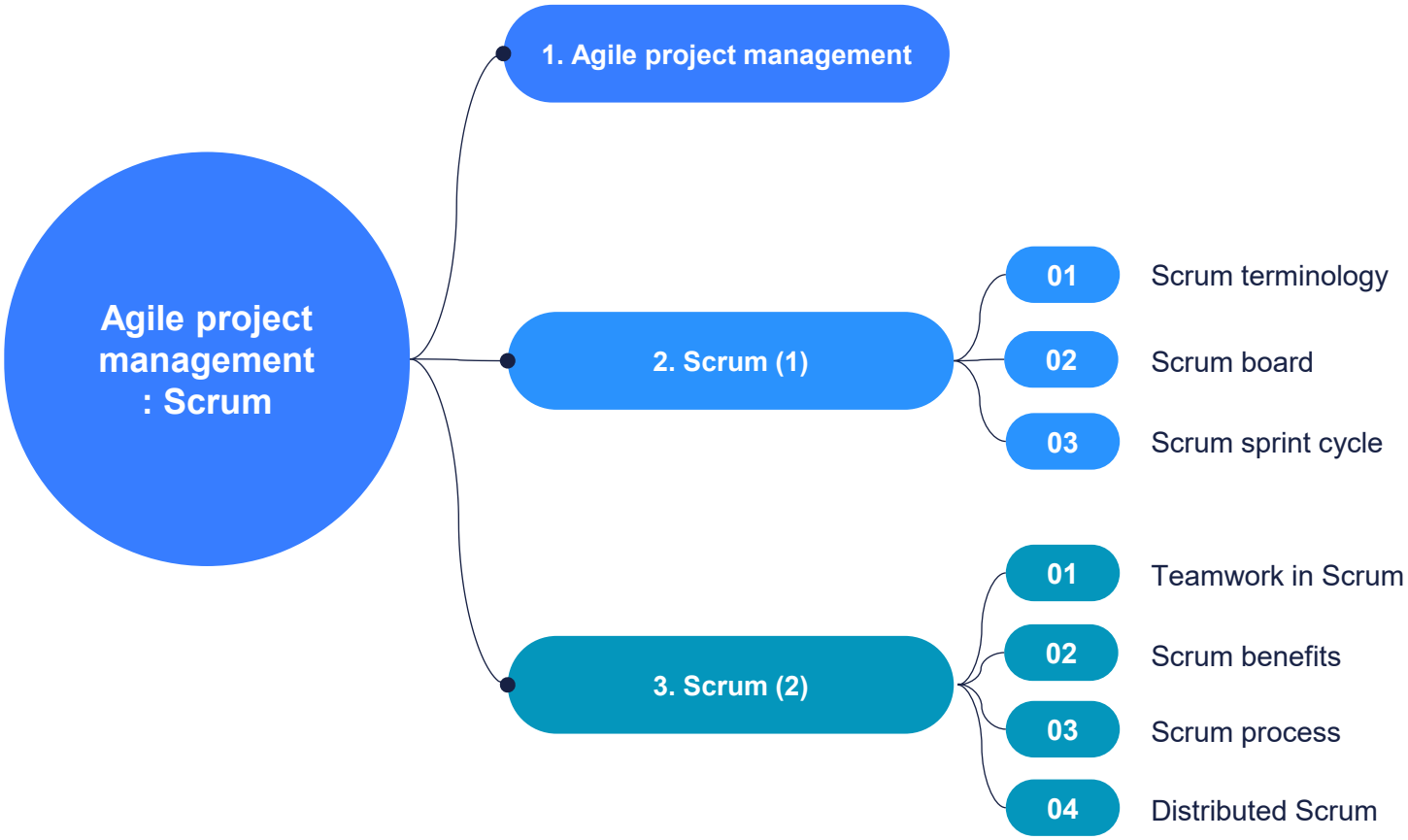
Extreme programming flow

[Extreme programming flow (as Korean)]



Source: <All about software engineering>. 생능출판사. 최은만

2-4: Agile project management : Scrum



Agile project management

- The principal responsibility of software project managers is to manage the project so that the software is **delivered on time** and within the **planned budget** for the project.
- The standard approach to project management is plan-driven.
- Managers draw up a plan for the project showing what should be delivered, when it should be delivered and who will work on the development of the project deliverables.
- **Agile project management** requires a different approach, which is adapted to **incremental development** and the practices **used in agile methods.**

Scrum (1)

Scrum

Scrum is an **agile method** that focuses on managing **iterative development** rather than specific agile practices.

There are three phases in Scrum.

- **The initial phase** is an **outline planning** phase where you establish the general **objectives** for the project and **design** the software architecture.
- This is followed by a series of **sprint cycles**, where each cycle develops an **increment** of the system.
- The **project closure phase** wraps up the project, **completes** required documentation such as system help frames and user manuals and assesses the lessons learned from the project.

Scrum (1)

Scrum terminology

- **Development team** : 개발팀
  - A **self-organizing group** of software developers, which should be no more than 7 people.
  - They are responsible for **developing the software** and other **essential project documents**.

- **Sprint** : 스프린트
  - A development iteration.
  - Sprints are usually 2-4 weeks long.

- **Scrum** : 스크럼
  - A daily meeting of the Scrum team that reviews progress and prioritizes work to be done that day.
  - Ideally, this should be a short face-to-face meeting that includes the whole team.

Scrum (1)

Scrum terminology

- **Scrum Master** : 스크럼 마스터
  - The Scrum Master is **responsible** for ensuring that the **Scrum process** is followed and **guides** the team in the effective use of Scrum.
  - He or she is responsible for **interfacing** with the rest of the company and for ensuring that the Scrum team is not **diverted by outside interference.**
  - The Scrum developers are **adamant** that the ScrumMaster should not be thought of as a project manager.
  - Others, however, may not always find it easy to see the difference.

Scrum (1)

Scrum terminology

- **Potentially shippable product increment** : 잠재적 전달 가능한 제품 증가분
  - The **software increment** that is delivered from a sprint.
  - The idea is that this should be 'potentially shippable' which means that it is in a **finished state and no further work**, such as testing, is needed to incorporate it into the final product.
  - In practice, this is not always achievable.

- **Product backlog** : 제품 백로그
  - This is a list of **'to do' items** which the Scrum team must tackle.
  - They may be feature definitions for the software, software requirements, user stories or descriptions of supplementary tasks that are needed, such as architecture definition or user documentation.

Scrum (1)

Scrum terminology

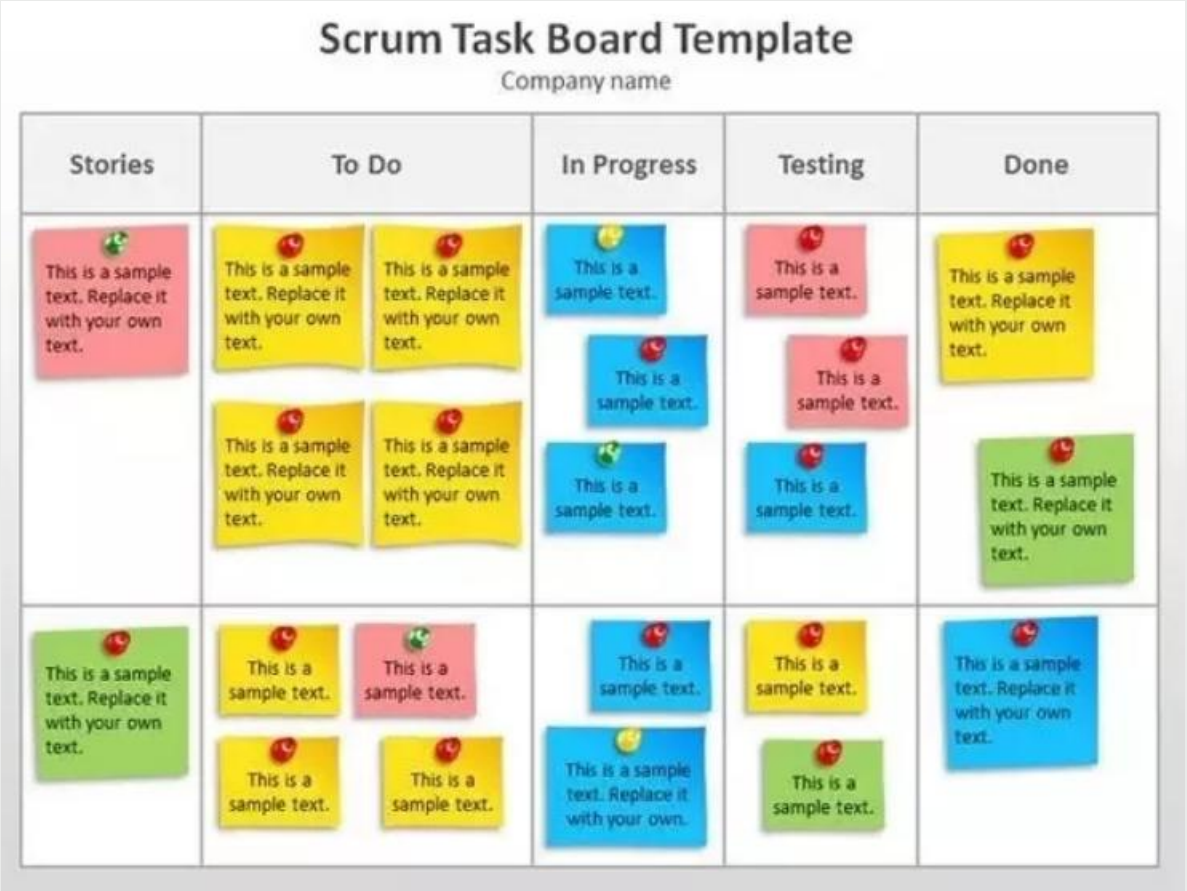
- **Product owner** : 제품 소유권자
  - An individual (or possibly a small group) whose **job** is **to identify product features or requirements**, **prioritize** these for development and continuously **review the product backlog** to ensure that the project continues to meet critical business needs.
  - The Product Owner can be a **customer** but might also be a **product manager** in a software company or **other stakeholder** representative.

- **Velocity** : 속도
  - An **estimate** of how much product backlog effort that a team can cover in a **single sprint**.
  - Understanding a team's velocity helps them estimate what can be covered in a sprint and provides a basis for measuring improving performance.

Scrum (1)

Scrum board

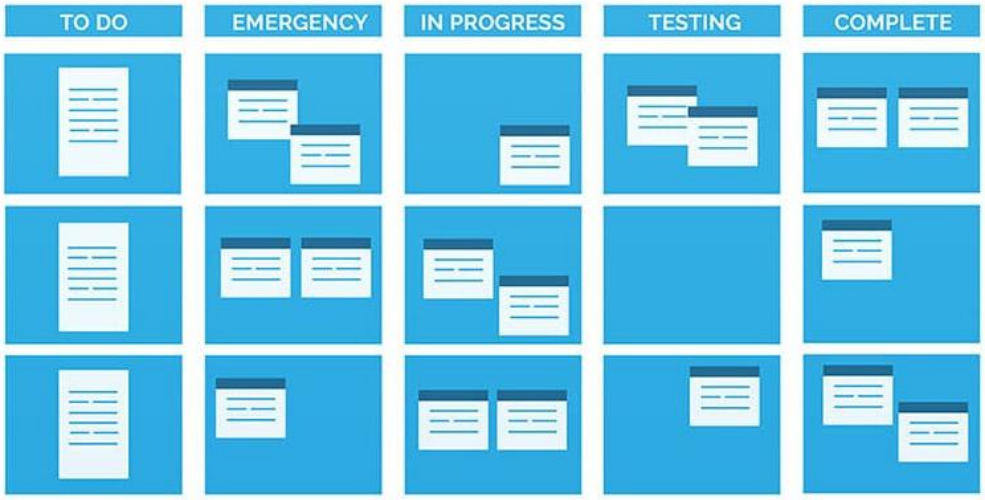
- Story (backlog)
- To do
- In progress
- Testing
- Done



Source: <https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=ijoos&logNo=221463849700>

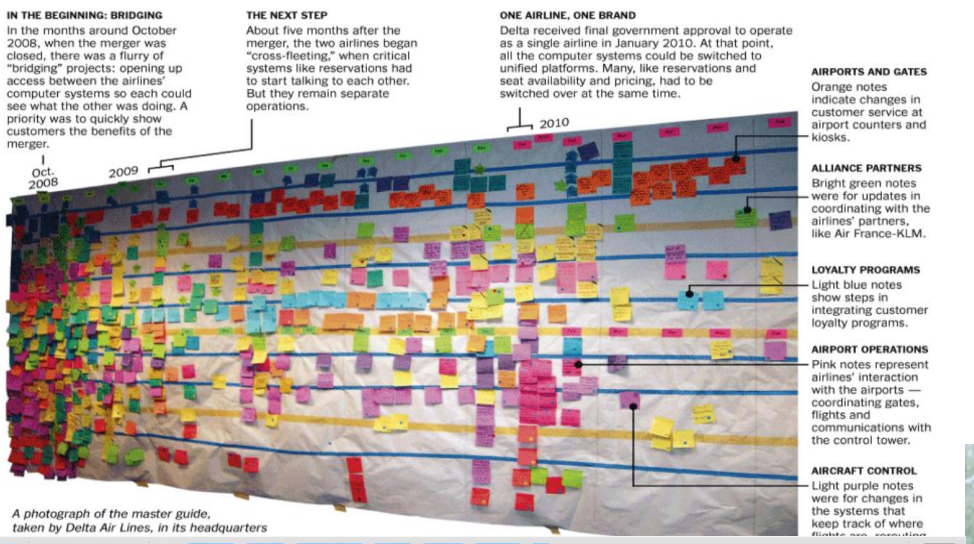
Scrum (1)

[Example of scrum board]

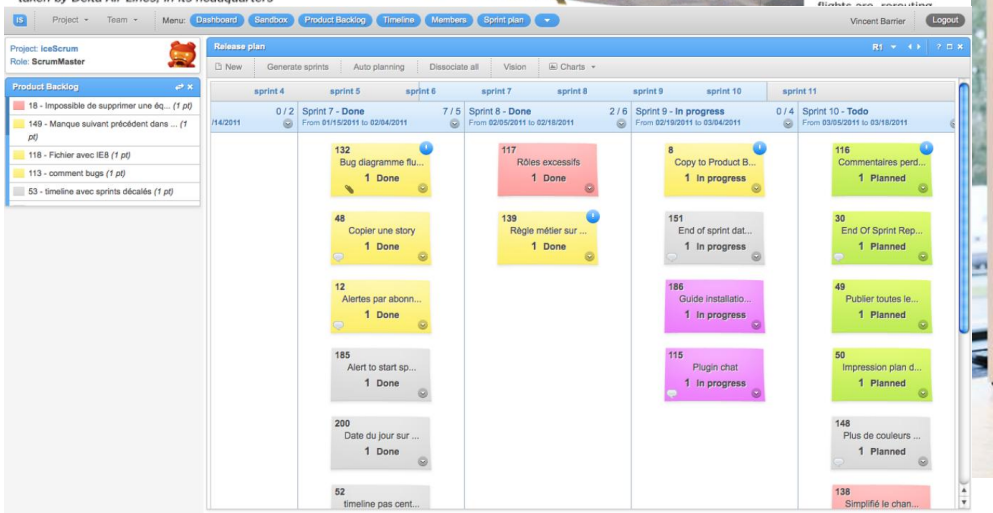


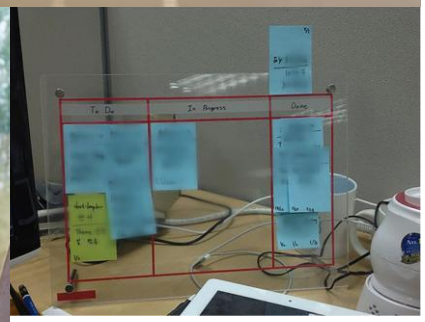
Source: <https://scrumexplainer.com/scrum/scrum-board/>

Scrum (1)



A photograph of the master guide, taken by Delta Air Lines, in its headquarters



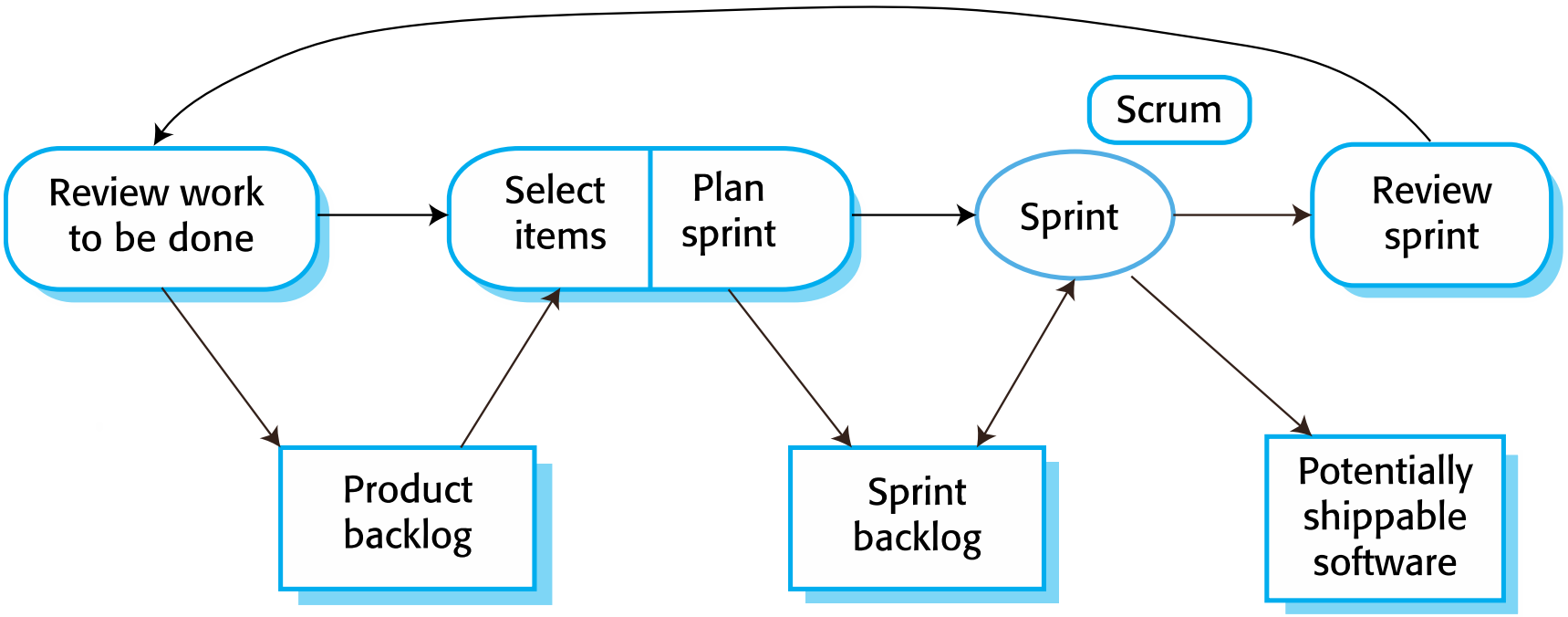


https://startinfinity.com/blog/scrum-software

Scrum (1)

initial phase

[Scrum sprint cycle]



Scrum (1)

Scrum sprint cycle (cont.)

- Sprints are fixed length, normally **2–4 weeks**.
- The **starting point** for planning is the **product backlog**, which is the list of work to be done on the project.
- The **selection phase** involves all of the project team who work **with the customer** to **select the features** and **functionality** from the product backlog to be developed during the sprint.
- Once these are agreed, the team **organize** themselves to **develop** the software.

Scrum (1)

Scrum sprint cycle (cont.)

- During this stage the team is isolated from the customer and the organization, with all **communications** channelled through the so-called '**Scrum master**'.
- The role of the **Scrum master** is to **protect** the development team from external distractions.
- At the end of the sprint, the **work done is reviewed** and presented to stakeholders.
- The next sprint cycle then begins.

Scrum (2)

Teamwork in Scrum

- **The 'Scrum master' is a facilitator**
  - arranges daily meetings
  - tracks the backlog of work to be done
  - records decisions
  - measures progress against the backlog and communicates with customers and management outside of the team.

- **The whole team attends short daily meetings (Scrums)**
  - all team members share information
  - describe their progress since the last meeting
  - problems that have arisen
  - what is planned for the following day.
  - everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them.

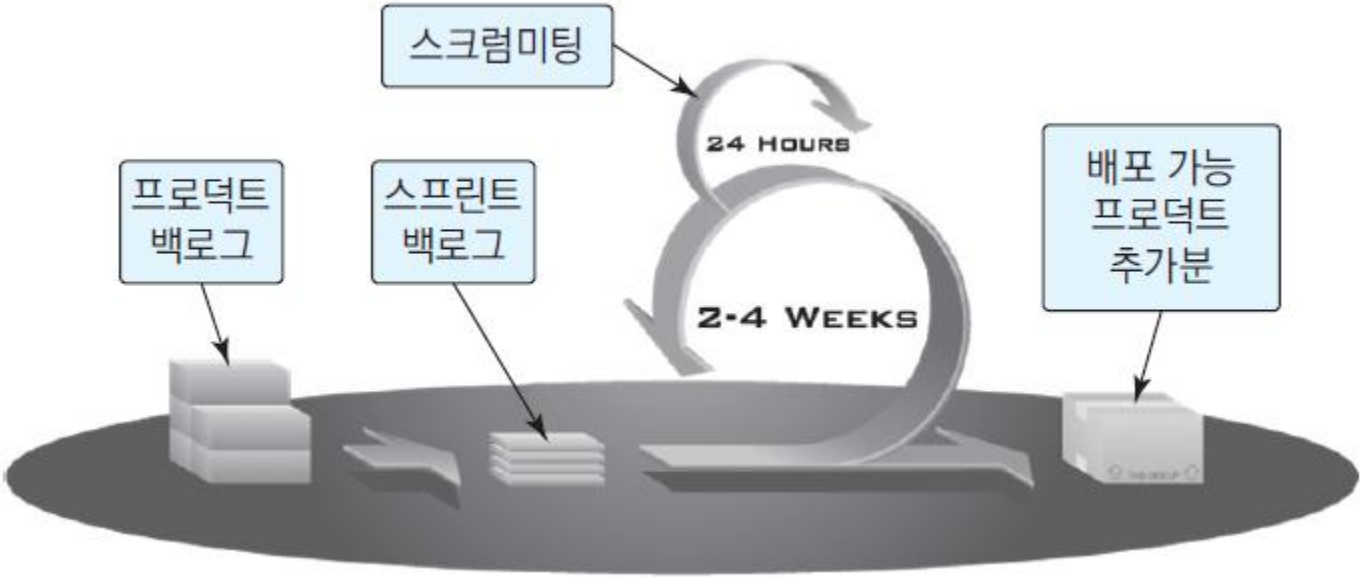
Scrum (2)

Scrum benefits

- The product is **broken down** into a set of **manageable and understandable chunks**.
- **Unstable requirements** do **not hold** up progress.
- The whole team have **visibility of everything** and consequently team **communication is improved**.
- **Customers** see **on-time** delivery of **increments** and gain **feedback** on how the product works.
- **Trust between customers and developers** is established and a positive culture is created in which everyone expects the project to succeed.

Scrum (2)

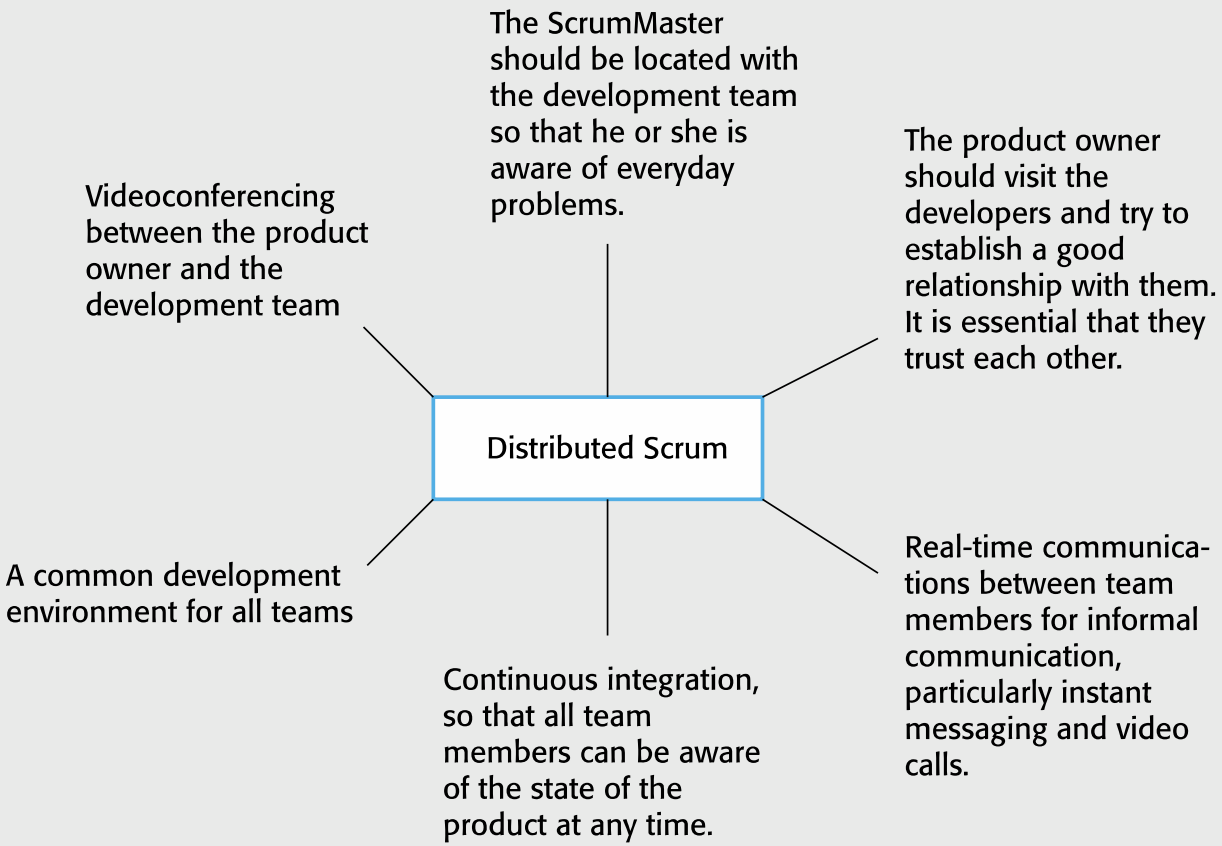
[Scrum process (as Korean)]



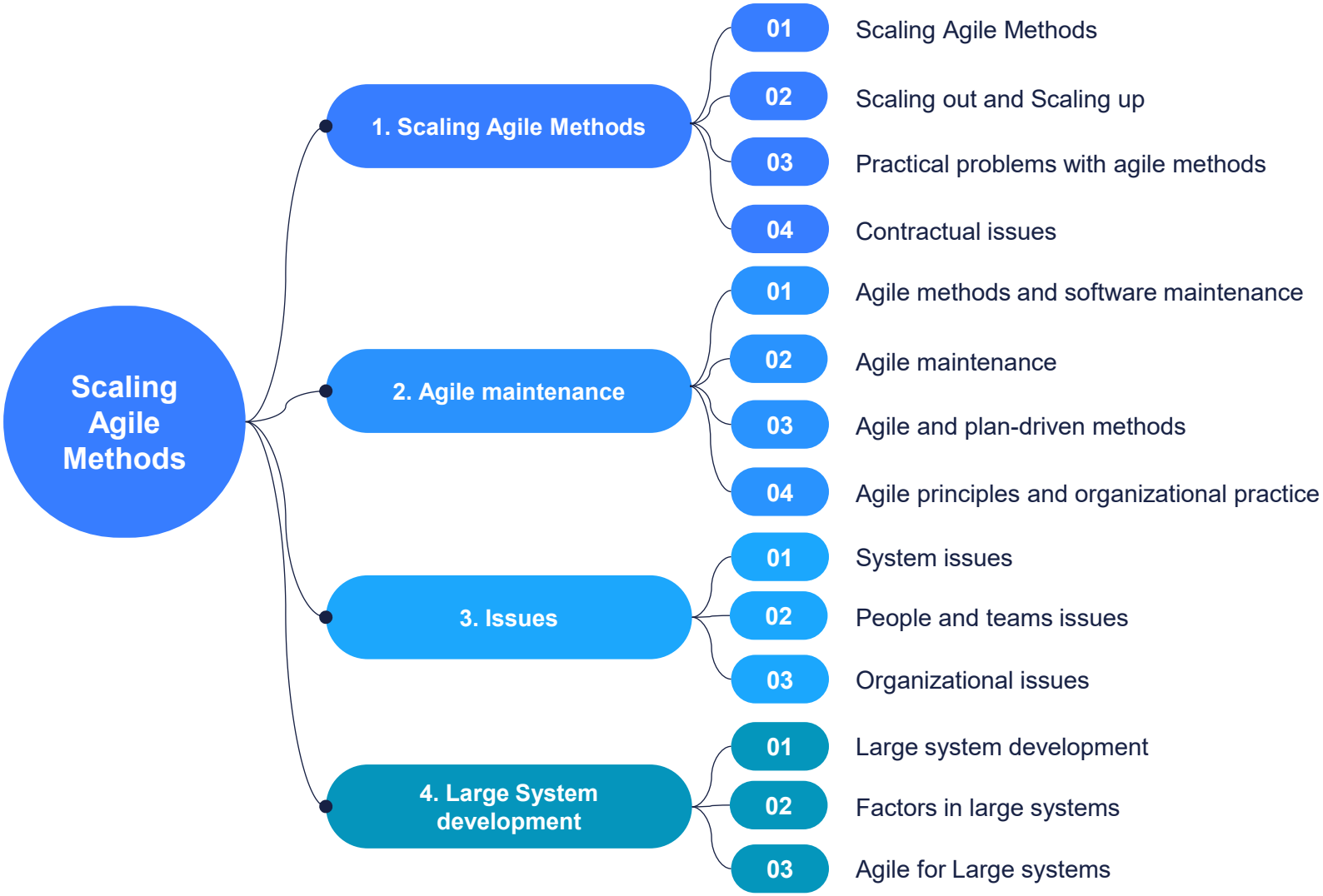
Source: <All about software engineering>. 생능출판사. 최은만

Scrum (2)

[Distributed Scrum]



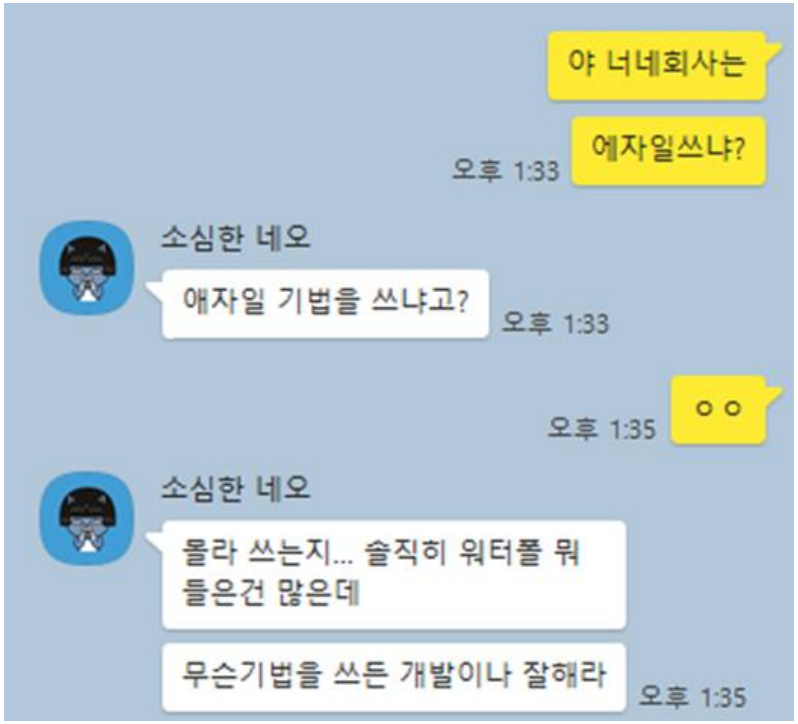
2-5: Scaling agile methods (애자일 프로그램의 규모 조정)



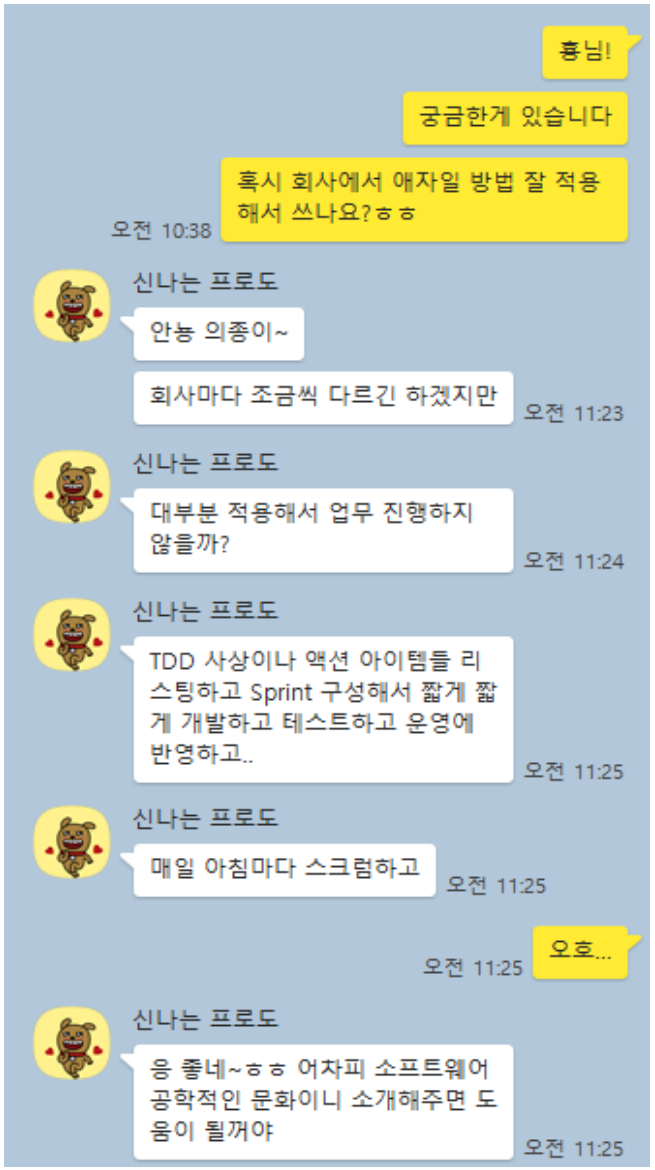
Scaling agile methods

Agile in real fields?

- 사원수 5,600여명 대기업 (금융권 / PM of PMs)

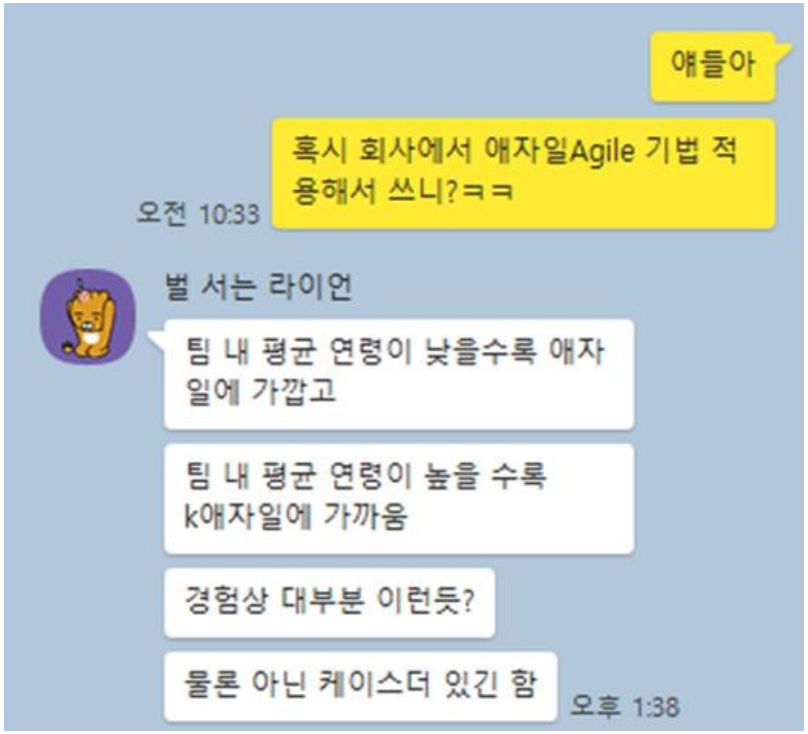


- 스타트업 (개발자)

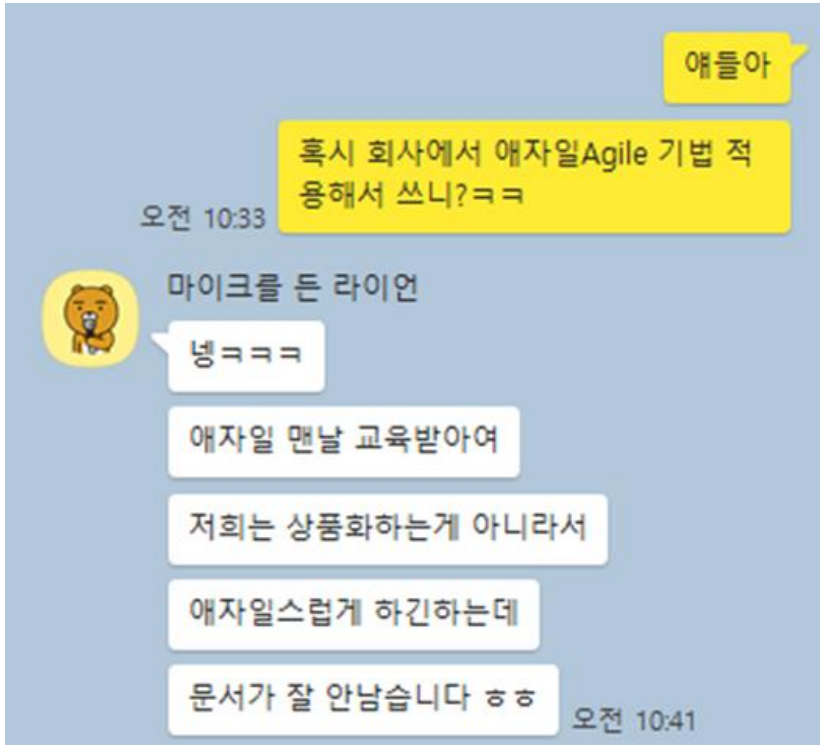


Agile in real fields?

- 사원수 4,200여명 대기업 (개발자)



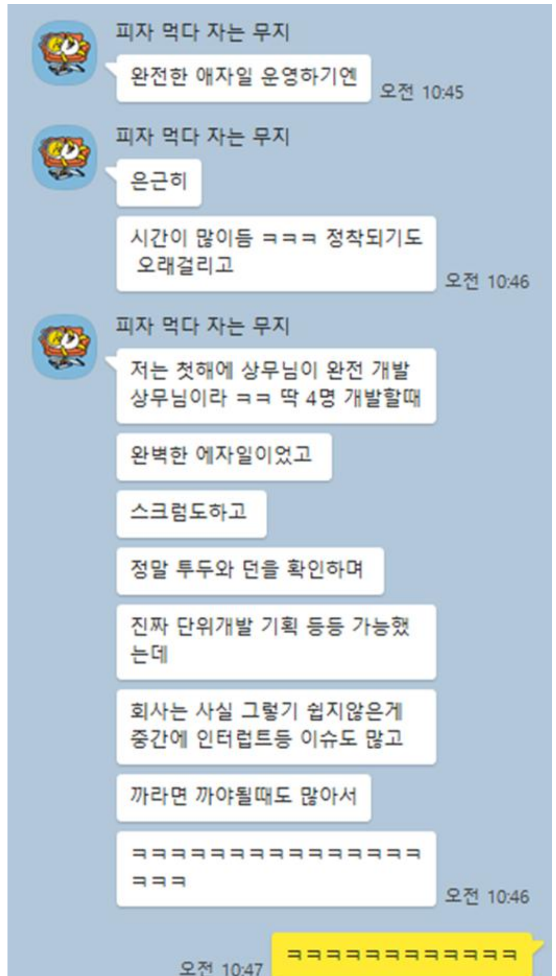
- 사원수 11,100여명 대기업 (연구직)

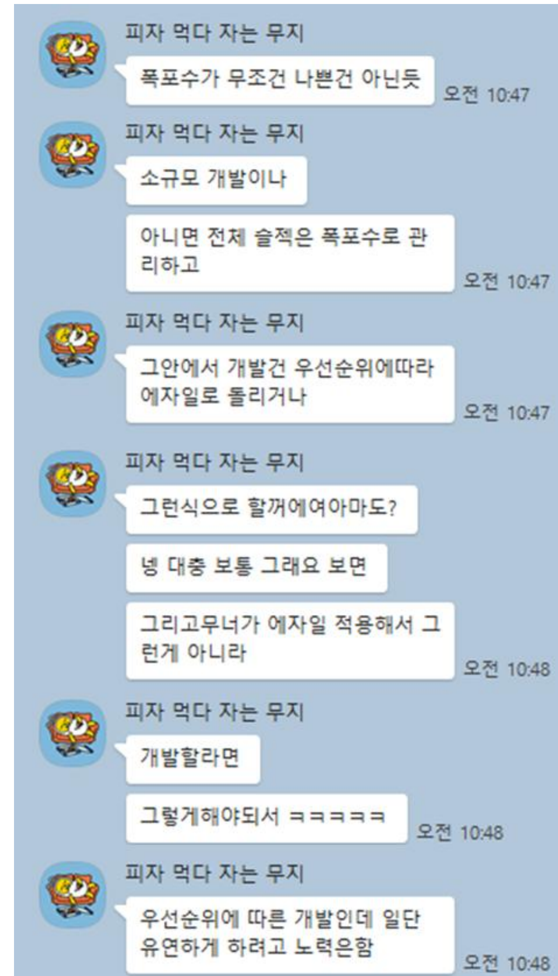


Scaling agile methods

Agile in real fields?

(사원수 22,300여명 대기업 / 연구직)

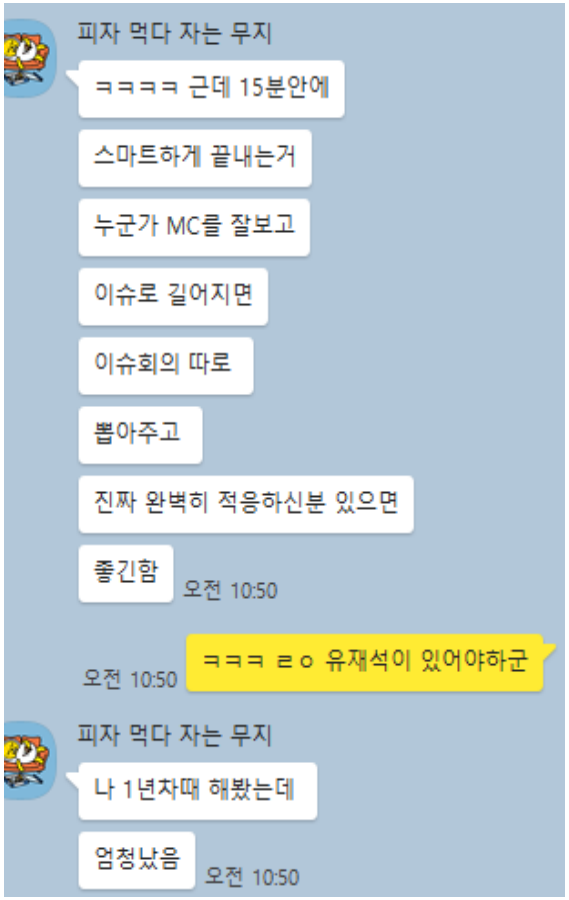


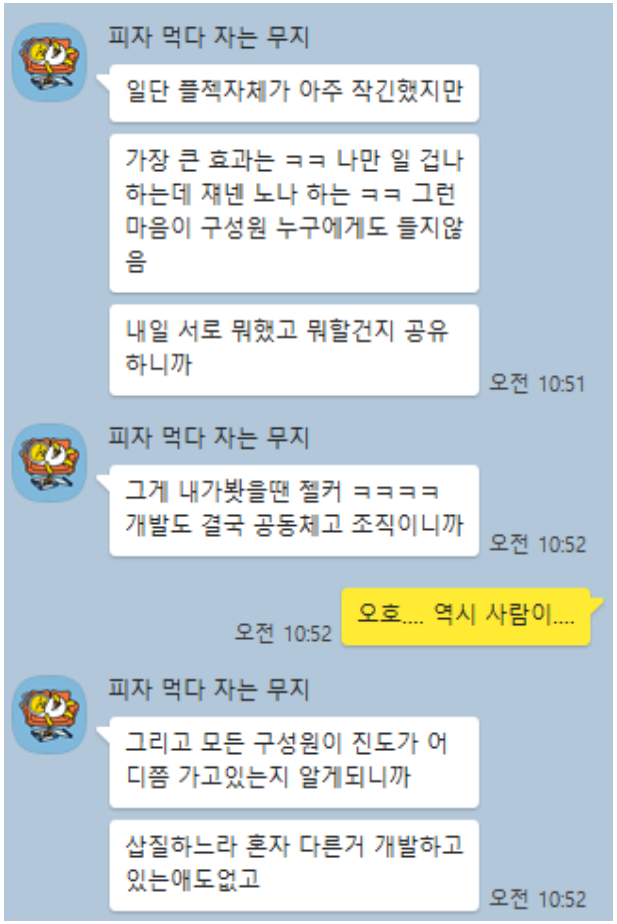


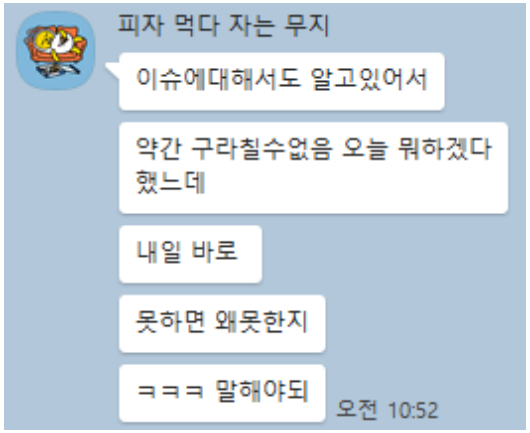
Scaling agile methods

Agile in real fields?

(사원수 22,300여명 대기업 / 연구직)







Scaling agile methods

Agile in real fields?

(사원수 1200명 중견기업)

Scaling agile methods

[Scaling agile methods]

- Agile methods have proved to be successful for **small and medium sized projects** that can be developed by a small co-located team.

- It is sometimes argued that the **success of these methods** comes because of **improved communications** which is possible **when everyone is working together**.

- Scaling up agile methods involves changing these to cope with larger, longer projects where there are multiple development teams, perhaps working in different locations.

Scaling agile methods

[Scaling out and scaling up]

- **‘Scaling up’**
  - Concerned with using agile methods for developing large software systems that cannot be developed by a small team.
- **‘Scaling out’**
  - concerned with how agile methods can be introduced across a large organization with many years of software development experience.
- **When scaling agile methods it is important to maintain agile fundamentals:**
  - Flexible planning, frequent system releases, continuous integration, test-driven development and good team communications.

Scaling agile methods

[Practical problems with agile methods]

- The informality of agile development is incompatible with the **legal approach** to contract definition that is commonly used in large companies.

- Agile methods are most **appropriate** for new software **development** rather than **software maintenance**.
  - Yet the majority of software costs in large companies come from maintaining their existing software systems.

- **Agile methods** are designed for small **co-located teams** yet much software development now involves worldwide distributed teams.

Scaling agile methods

[Contractual issues]

- **Most software contracts** for custom systems are **based around a specification**, which sets out what has to be implemented by the system developer for the system customer.

- However, this precludes **interleaving specification and development** as is the norm in agile development.

- A contract that **pays** for developer **time** rather than functionality is required.
  - However, this is seen as a high risk my many legal departments because what has to be delivered cannot be guaranteed.

Agile maintenance

[Agile methods and software maintenance]

- Most organizations **spend more on maintaining** existing software than they do on new software development.
  - So, if agile methods are to be successful, they have to support maintenance as well as original development.

- Two key issues:
  - Are systems that are developed using an agile approach maintainable, given the emphasis in the development process of **minimizing formal documentation**?
  - Can agile methods be used **effectively for evolving a system** in response to customer change requests?

- Problems may arise if original development team cannot be maintained.

Agile maintenance

Agile maintenance

- Key problems are:
  - **Lack** of product **documentation**
  - **Keeping customers involved** in the development process
  - **Maintaining** the continuity of the **development team**

- Agile development relies on the development team knowing and understanding what has to be done.

- For long-lifetime systems, this is a real problem as the original developers will not always work on the system.

Agile maintenance

Agile and plan-driven methods

- Most projects include elements of plan-driven and agile processes. Deciding on the balance depends on:
  - Is it important to have a very **detailed specification and design** before moving to implementation? If so, you probably need to use **a plan-driven approach**.
  - Is an **incremental delivery strategy**, where you deliver the software to customers and get **rapid feedback** from them, realistic? If so, consider **using agile methods**.

Agile maintenance

Agile and plan-driven methods

- Most projects include elements of plan-driven and agile processes. Deciding on the balance depends on:

  - **How large is the system** that is being developed?
    - **Agile** methods are most effective when the system can be developed with a **small co-located team** who can communicate informally.
    - Agile may not be possible for **large systems** that require larger development teams so a **plan-driven** approach may have to be used.

Agile maintenance

Agile principles and organizational practice

[Customer involvement : 고객 참여]

- This depends on having a customer who is willing and able to spend time with the development team and who can represent all system stakeholders.
  - Often, customer representatives have other demands on their time and cannot play a full part in the software development.
- Where there are external stakeholders, such as regulators, it is difficult to represent their views to the agile team.

[Embrace change : 변화 수용]

- Prioritizing changes can be extremely difficult, especially in systems for which there are many stakeholders.
- Typically, each stakeholder gives different priorities to different changes.

Scaling agile methods

Agile maintenance

Agile principles and organizational practice

[Incremental delivery : 점증적 인도 **]**

- **Rapid iterations and short-term planning** for development does **not always fit** in with the longer-term planning cycles of **business planning and marketing**.
- **Marketing managers** may need to know what product features **several months** in advance to prepare an effective marketing campaign.

[Maintain simplicity : 단순성 유지 **]**

- Under pressure from delivery schedules, team members may **not have time** to carry out desirable system **simplifications**.

[People not process : 프로세스가 아닌 사람 **]**

- Individual team members may not have suitable personalities for the intense involvement that is typical of agile methods,
- therefore may not interact well with other team members.

Issues

System issues

- **How large** is the system being developed?
  - Agile methods are most effective a relatively small co-located team who can communicate informally.
- **What type of system** is being developed?
  - Systems that require a lot of analysis before implementation need a fairly detailed design to carry out this analysis.
- **What is the expected** system lifetime?
  - Long-lifetime systems require documentation to communicate the intentions of the system developers to the support team.
- Is the system subject to **external regulation**?
  - If a system is regulated you will probably be required to produce detailed documentation as part of the system safety case.

Issues

People and teams issues

- How **good** are the **designers** and **programmers** in the development team?
  - It is sometimes argued that agile methods require higher skill levels than plan-based approaches in which programmers simply translate a detailed design into code.
- How is the **development team organized**?
  - Design documents may be required if the team is dsitributed.
- What **support technologies** are available?
  - IDE support for visualisation and program analysis is essential if design documentation is not available.

Issues

Organizational issues

- Traditional engineering organizations have a culture of plan-based development, as this is the norm in engineering.
- Is it standard organizational practice to develop a detailed **system specification**?
- Will **customer** representatives be available to **provide feedback** of system increments?
- Can informal agile development fit into the **organizational culture** of detailed documentation?

Large system development

Large system development

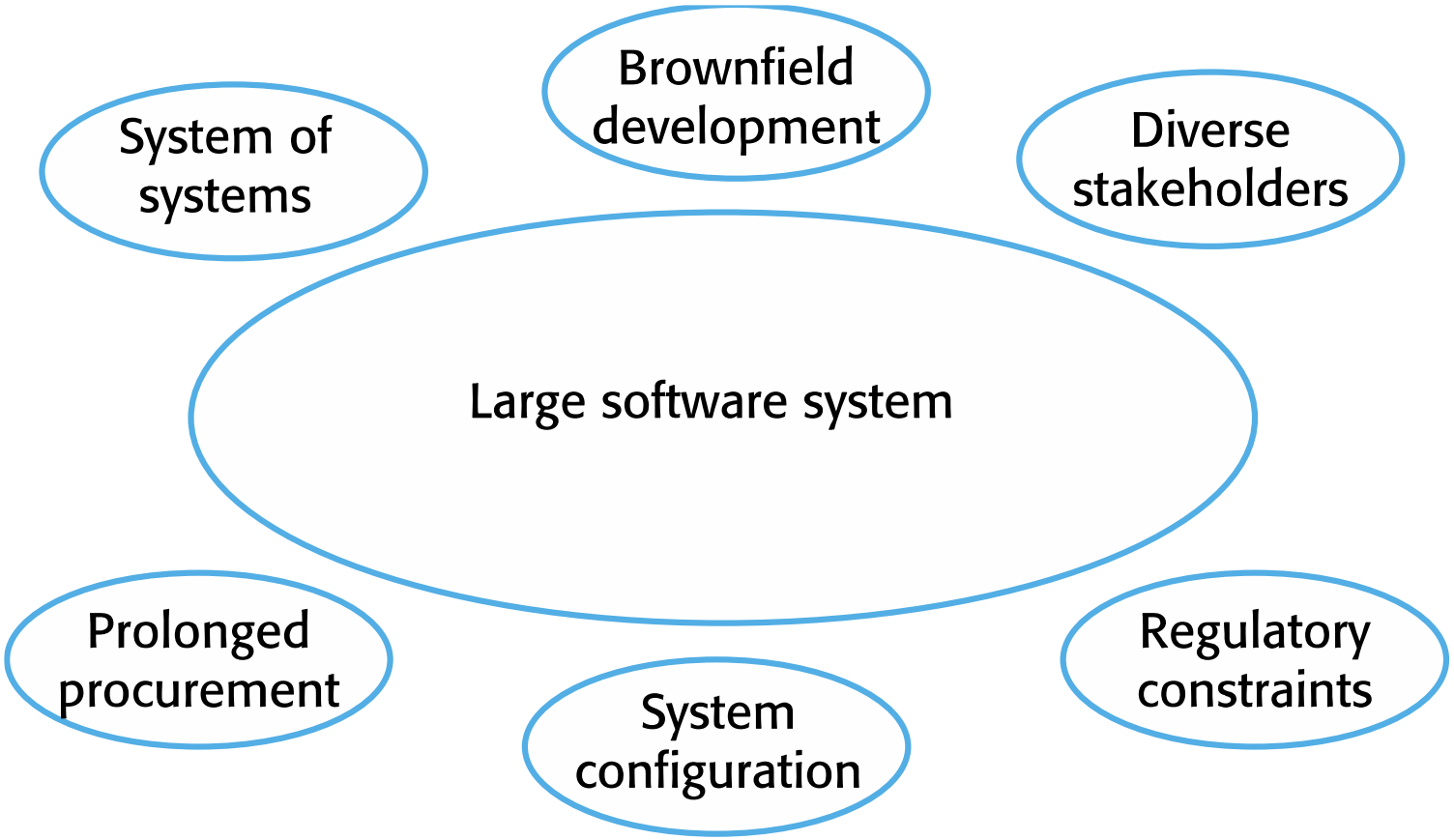
- Large systems and their development processes are often constrained by external rules and regulations limiting the way that they can be developed.

- Large systems have a long procurement and development time. It is difficult to maintain coherent teams who know about the system over that period as, inevitably, people move on to other jobs and projects.

- Large systems usually have a diverse set of stakeholders. It is practically impossible to involve all of these different stakeholders in the development process.

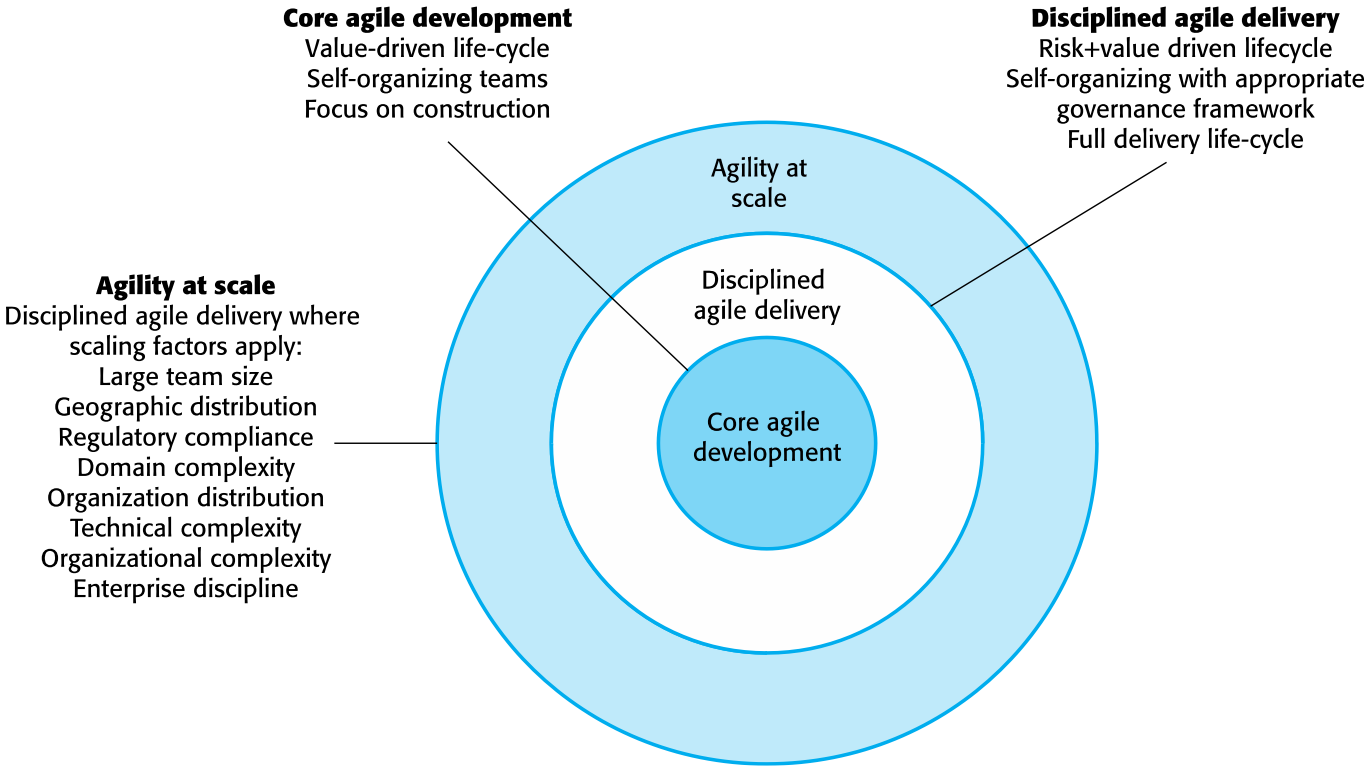
Large system development

[Factors in large systems]



Large system development

[Agile for Large system: IBM's agility at scale model]



Large system development

[Agile for large system: Multi-team Scrum]

Role replication	Each team has a Product Owner for their work component and ScrumMaster.
Product architects	Each team chooses a product architect and these architects collaborate to design and evolve 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 progressand plan work to be done.

Key points (1)

- Agile methods are incremental development methods that focus on rapid software development, frequent releases of the software, reducing process overheads by minimizing documentation and producing high-quality code.

- Agile development practices include
  - User stories for system specification
  - Frequent releases of the software,
  - Continuous software improvement
  - Test-first development
  - Customer participation in the development team.

Key points (2)

- Scrum is an agile method that provides a project management framework.
  - It is centred round a set of sprints, which are fixed time periods when a system increment is developed.
- Many practical development methods are a mixture of plan-based and agile development.
- Scaling agile methods for large systems is difficult.
  - Large systems need up-front design and some documentation and organizational practice may conflict with the informality of agile approaches.