Introduction of Agile

Technologic Arts Incorporated

Purpose of the Course

What to Learn

- The Agile Way of Thinking
- Overview of Scrum
- Principles and Practices of XP

Prerequisite

- knowledge of general software development process

Course Schedule

- 1day

Learning Content

- You will get general knowledge of Agile development method, understand overview of Scrum which is the most used among them, and master principles and practices of XP.

Agenda

Chapter 1. Overview

- 1.1. What is Agile
- 1.2. Background of Agile
- 1.3. Emergence of Agile
- 1.4. Summary

Chapter 2. Scrum and Stories

- 2.1. Overview of Scrum
- 2.2. What is Story
- 2.3. Summary

Chapter 3. Overview of XP

- 3.1. What is XP
- 3.2. Practices of XP
- 3.3. Summary

Chapter 4. Team Exercise

Chapter 1. Overview

What to learn in this chapter

- Definition of Agile Software Development
  - Manifesto for Agile Software Development
  - A variety of agile software development methods
  - Well-known agile software development methods
  - Approaches for adopting agile methods

- Background of Agile Software Development
  - Factors for Emergence of Agile

- Characteristics of Agile Software Development
  - Difference from traditional software development methods

Chapter 1. Overview

- 1.1 **What is Agile**
- 1.2 Background of Agile
- 1.3 Emergence of Agile
- 1.4 Summary

What is Agile

- Background
  - Lightweight software development methods evolved in the late 1990s as a reaction against traditional process-oriented heavyweight methods.

- What is Agile
  - Agile is the general term which represents such lightweight software development methods.
  - Agile is also the term which represents common ideas and philosophy among those software development methods.

It is difficult to define Agile concretely, and does not make sense.

What is Agile Menifesto

There are a variety of Agile approaches, so it is difficult to define in general.

However, there is a manifesto which typifies the agile way of thinking.

Manifesto for Agile Software Development (Agile Manifesto)

- Lightweight software development methods evolved in the late 1990s as a reaction against traditional process-oriented heavyweight methods.
- Authors of lightweight software development methods gathered at Utah on February 2001, to discuss what is common among their approaches.
- The conclusion (which they had agreed) was published as Manifesto for Agile Software Development.

Manifesto for Agile Software Development http://www.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.

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

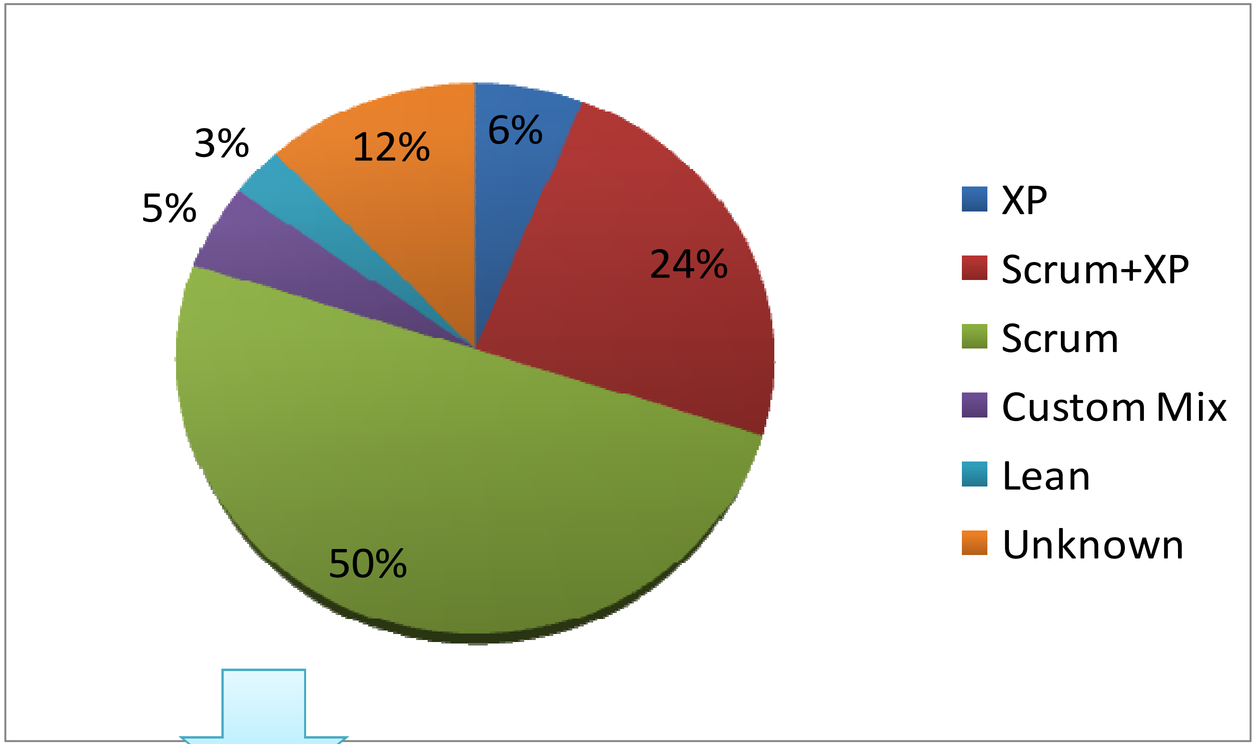
A variety of agile software development methods

- Kinds of agile software development methods
  - **XP (1996)**
    - Kent Beck, Ward Cunningham, Ron Jeffries
  - **Scrum**
    - Ken Schwaber, Jeff Sutherland, Mike Beedle
  - **Crystal (1986)**
    - Alistair Cockburn
  - **ASD (Apaptive Software Development)**
    - Jim Highsmith
  - **FDD (Feature Driven Development)**
    - Peter Code, Jon Kern
  - **DSDM**
    - Arie van Bennekum
  - **Pragmatic Programmer**
    - Andrew Hunt, Dave Thomas
  - **Executable UML**
    - Stephan Mellor

etc

Scrum + XP

- What is the most used agile process?
  - 80 % of it was concentrated in XP and Scrum.



This course deals with XP and Scrum.

(Source: Version One State of Agile Development Survey 2009)

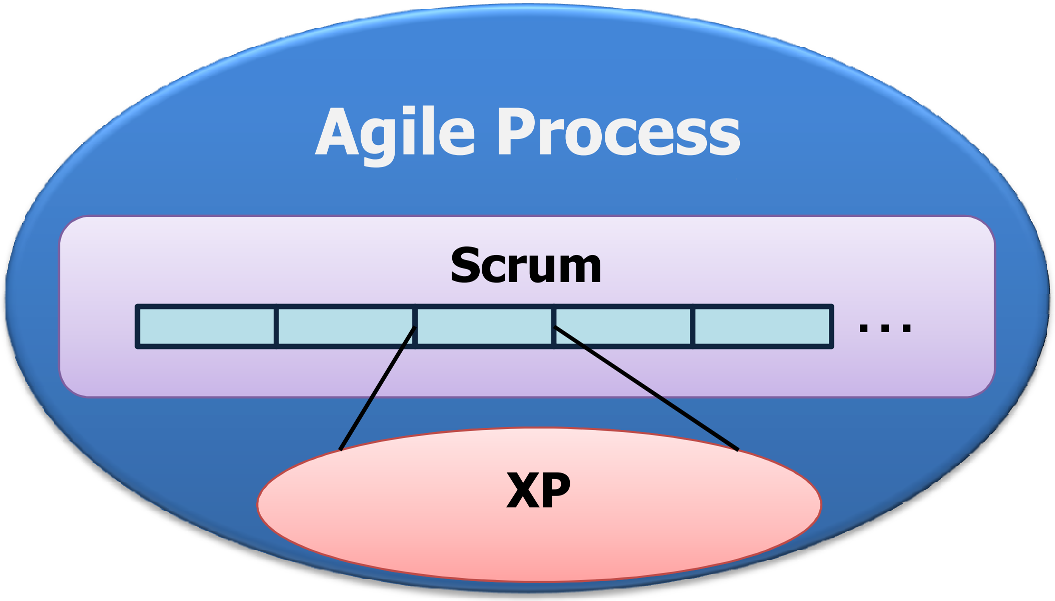
Relationship between XP and Scrum

- Scrum
  - It defines only the structure of process such as time box, organization, and way of task management.
  - ➢ It does not specify concrete development approaches, so it can be combined with various methods.

- XP
  - It is collection of practices which any engineer should put into.
  - ➢ Individual practices can be combined with other development processes.

⇩

We can use Scrum and XP together.



Key ideas for adopting agile

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

- Self Organization
  - It does not mean you have only to follow what Agile software development methods instruct.
  - Best practices of the team is established through organic relationship of team members and their proactive involvement with the process.
- Continuous Improvement
  - Team process is improved continuously.
  - Agile has “mechanisms” for improvement.

Chapter 1. Overview

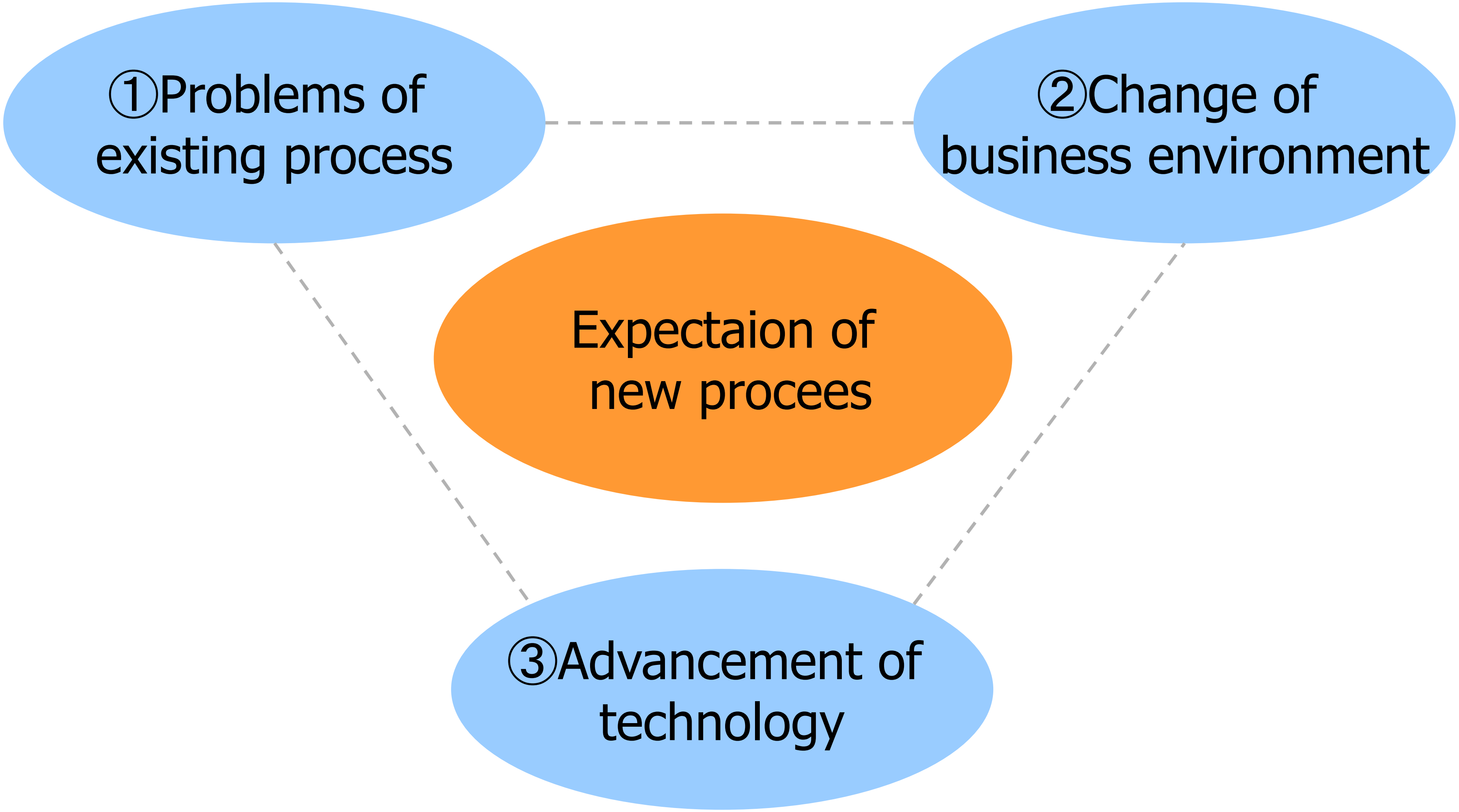
1.1 What is Agile

1.2 **Background of Agile**

1.3 Emergence of Agile

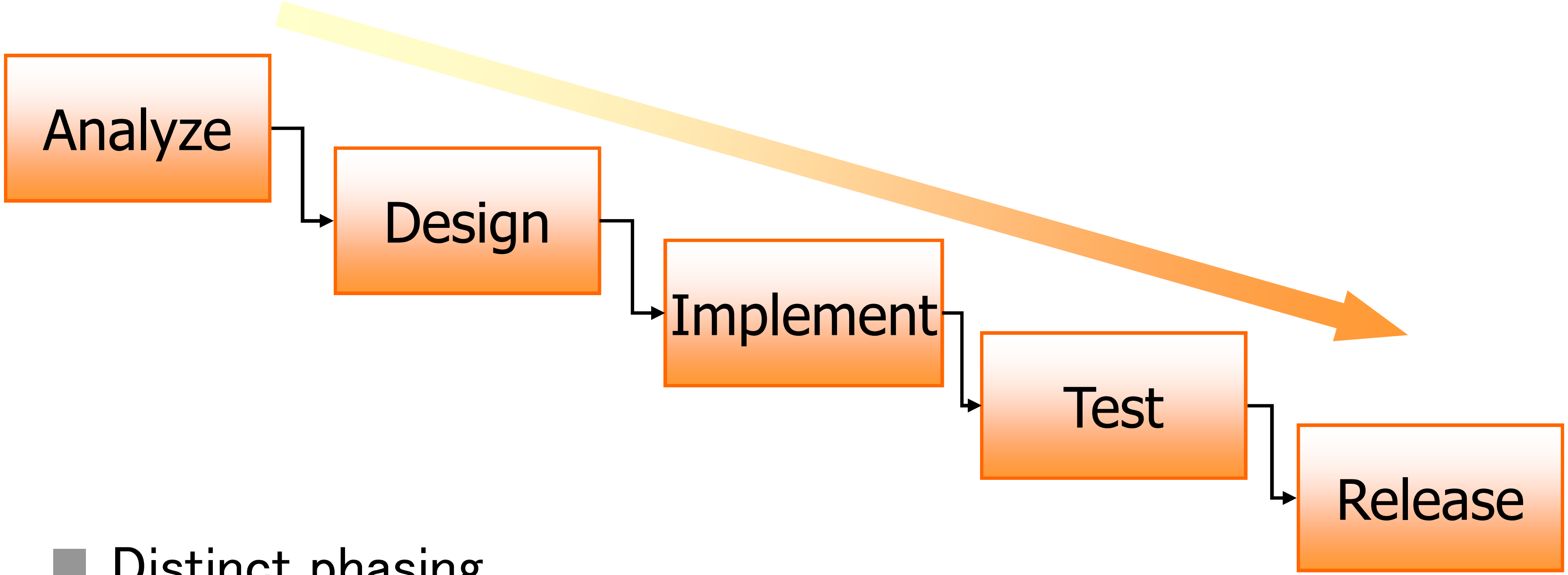
1.4 Summary

Background which Agile process has appeared



①Problems of existing process
②Change of business environment
Expectaion of new procees
③Advancement of technology

①Problems of existing process ～ Traditional approach



- Distinct phasing
  （Analyze・Design・Implement・Test・Release）
- Tasks are completed in each phase.
- In principle, you can not go back to prior phases.
- Communicate by documents

① Problems of existing process 〜 Pros and Cons

Background

- Big project with large size and many people
- Change of requirement is small and there is enough time until the time of delivery.
- Feature development with procedural languages

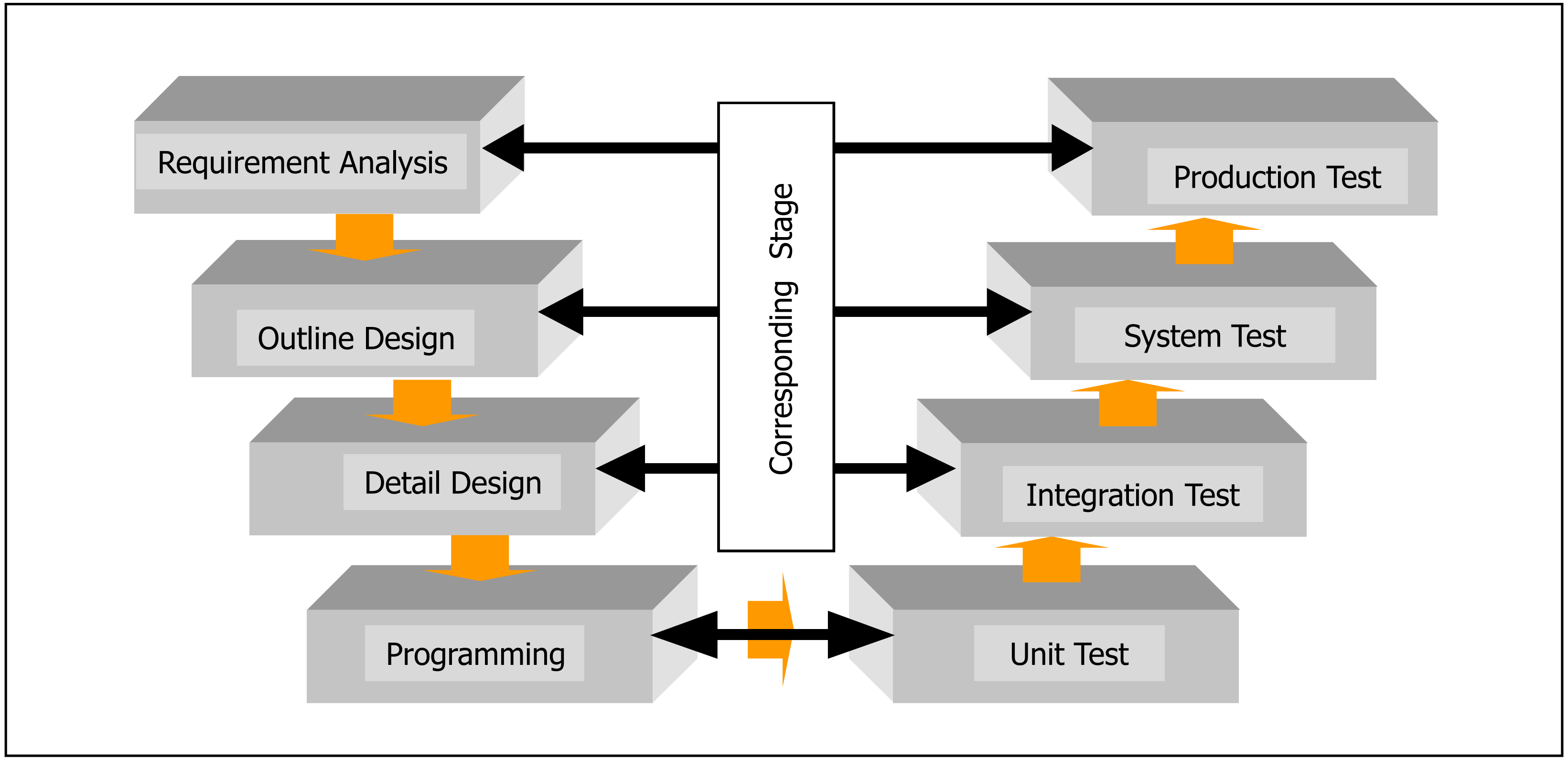
Pros

- Easy to manage (people, time, artifacts) → Suitable for off-shore development
- Easy to make long-term plans

Cons

- Sensitive to changes, slow for release
- Separation of stages of work, difficult to finish each stage

① Problems of existing process　～　V Curve



Frequent happenings

- Never ending project(death march)
- Schedule slip, and eventual delay of release

②Change of business environment　～　Business requires Agility

- Any management strategy is only valid for two years and a half?

Top 50 Best selling companies 1994-2001 worldwide kept their position for 4.8 years in average.In the first half, they would enter the new market and increase their sales, so they reached their peak in two years and a half.

（Source：IT Architect Vol.12 2007）

- Why does business require IT to be agile?

1) responding rapidly to changing market conditions and customer needs

2) early launch of new products and services by business cooperation

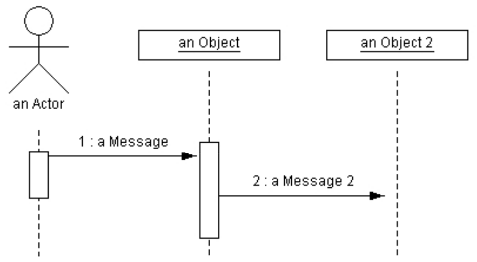
3) early realization of synergy in mergers and acquisitions

4) pursuit of sustainability

③Advancement of technology ～ Object Oriented

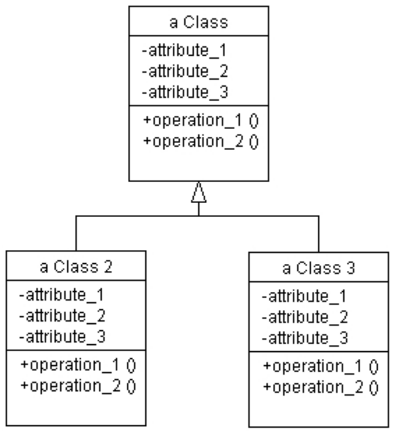
Seamless

- Design documents are refined with proceeding stages
- Design is changed and optimized any time with proceeding stages



Reusable

- Reusable implementation（class library）
- Reusable design（e.g. design patterns）
- Reusable analysis（e.g. domain analysis）



Quality（Maintainability）

- Intuitive structure closer to real world
- More readable code

③ Advancement of technology　～　Prevalence of the Internet

■ Open information

- ● Know-how is spread over the internet.
- ● Libraries and frameworks are distributed freely and standardized.

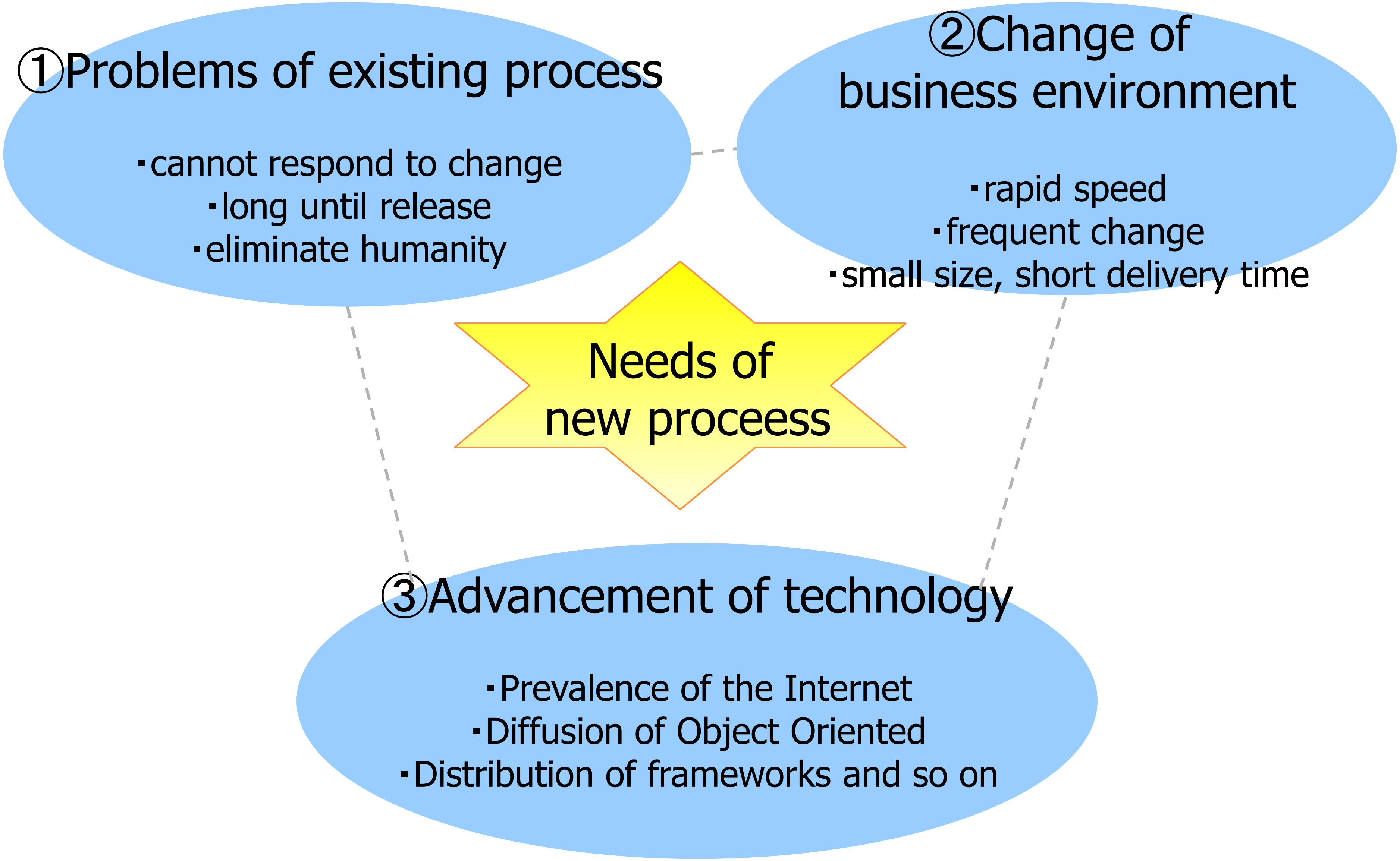
■ WWW

- ● Easy browse and HTML
- ● System running within Web browsers
- ● Intersystem Coordination（e.g. SOA）

■ Speed, rapid change

- ● Realize niches faster
- ● Interest with newer things make changes rapider.

Summary



①Problems of existing process

・cannot respond to change
・long until release
・eliminate humanity

②Change of business environment

・rapid speed
・frequent change
・small size, short delivery time

Needs of new proceess

③Advancement of technology

・Prevalence of the Internet
・Diffusion of Object Oriented
・Distribution of frameworks and so on

Chapter 1. Overview

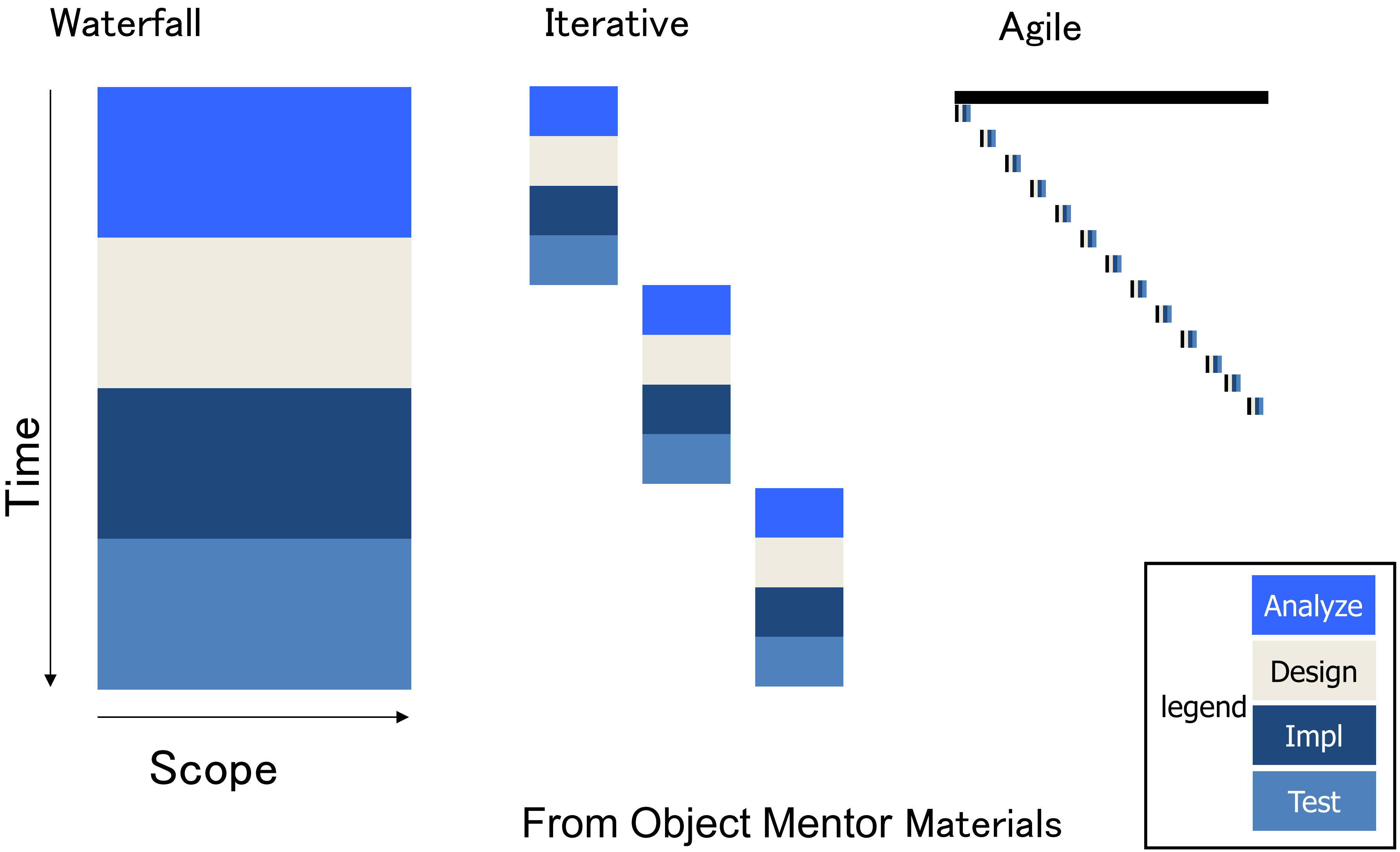
1.1 What is Agile

1.2 Background of Agile

1.3 **Emergence of Agile**

1.4 Summary

Characteristics of Agile ：Short time iterative development

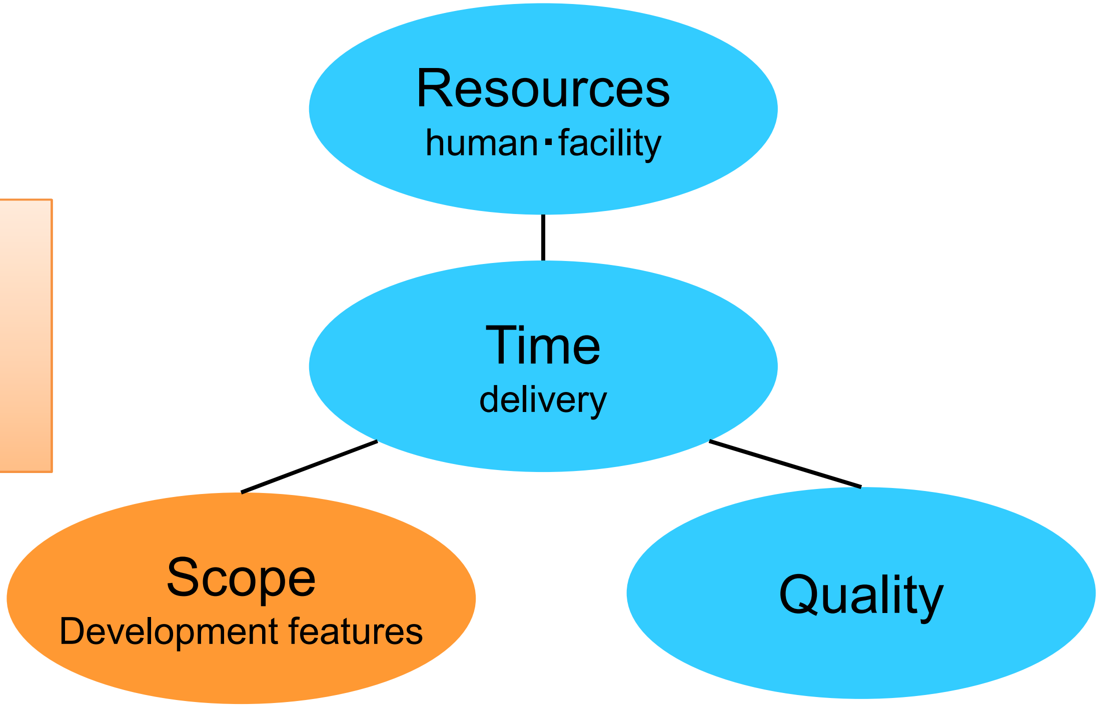


From Object Mentor Materials

Characteristics of Agile：Scope manipulation

- Manipulate scope
- Cause
  - Resource manipulation is invisible with its effect.
  - Time manipulation is often against business needs.
  - Quality manipulation is always dangerous.

Iteration span is considered as “time box”, and realizable features within it must be planned and developped.



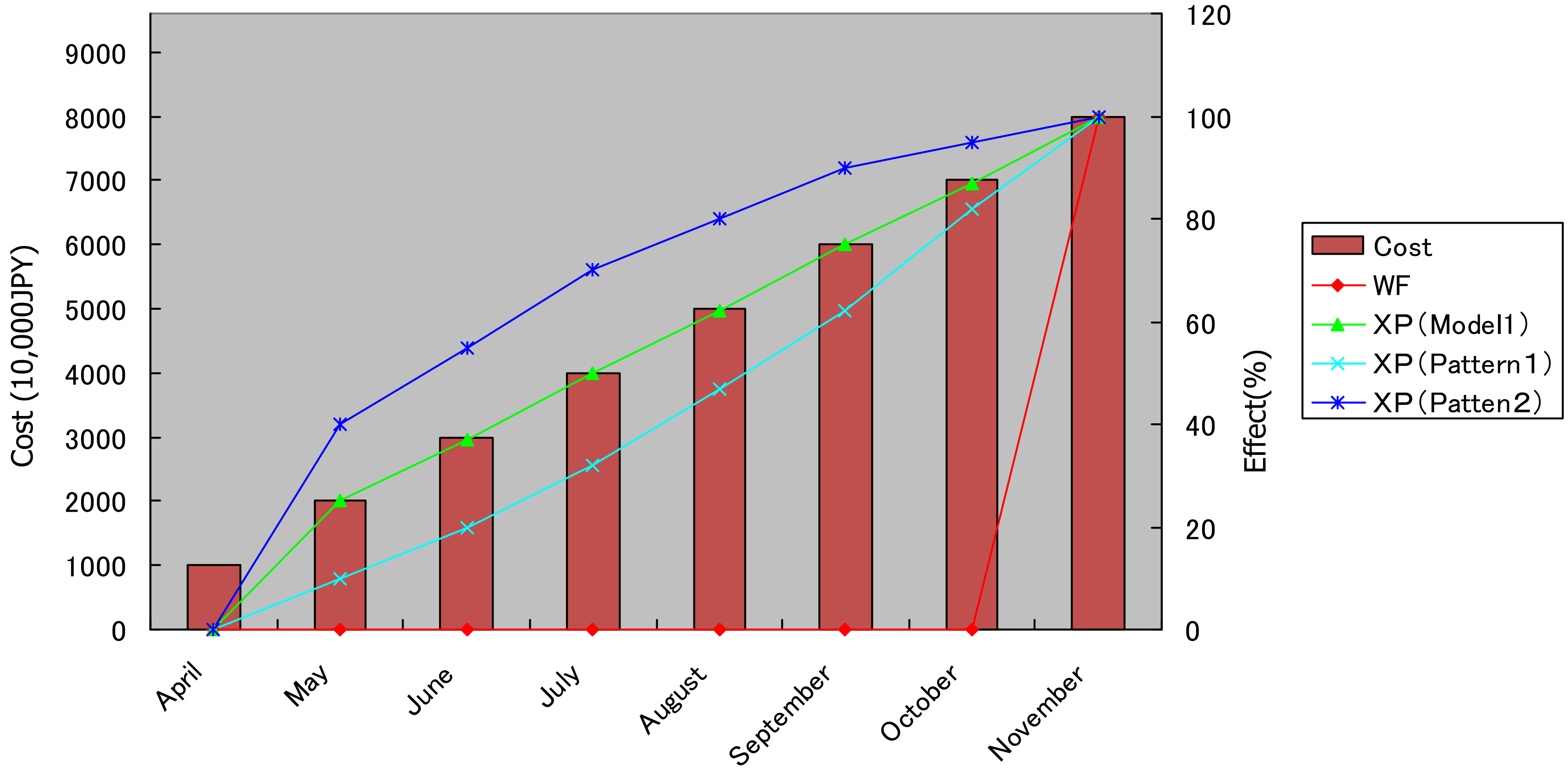
Compare with existing development

＜existing development＞		＜agile process＞
Eliminate humanity	⟷	Leverage humanity
Hate change	⟷	Welcome change
Long until release	⟷	Short until release
foreseeable	⟷	adaptive

Compare ways of thinking

	Existing development	Agile development
Requirement	Draw up to fix	Respond any time since it always changes
Purpose of system development	Construct as the user required at first time	User satisfaction
Communication	Formal documents and meetings	Face to Face
State of “developing”	Main state during development	Temporary state. Release as soon as possible to get feedback.

Cost Model



Agile process makes stepped cost recovery possible

Chapter 1. Overview

- 1.1 What is Agile
- 1.2 Background of Agile
- 1.3 Emergence of Agile
- 1.4 **Summary**

What to learn in this chapter (again)

- Definition of Agile Software Development
  - Manifesto for Agile Software Development
  - A variety of agile software development methods
  - Well-known agile software development methods
  - Approaches for adopting agile methods

- Background of Agile Software Development
  - Factors for Emergence of Agile

- Characteristics of Agile Software Development
  - Difference from traditional software development methods

Summary of this chapter

- **Agile has appeared from the background below:**
  - Problems of existing process
  - Change of business environment
  - Advancement of technology

- **What is agile**
  - Agile is the general term which represents lightweight software development methods whose authors took part in Agile Manifesto.
  - Agile is way of thinking for software development process.

- **Difference from existing development**
  - Repeat short iteration cycles to release features which customers want to have at this point.
- **To adopt agile process, team needs :**
  - Self Organization
  - Continuous Improvement

Chapter 2. Scrum and Stories

What to learn in this chapter

- Overview of Scrum
  - Scrum way of thinking
  - Organization and roles of Scrum
  - Overview of Process
- Stories
  - What are stories
  - Estimate of stories

Chapter 2. Scrum and Stories

2.1 **Overview of Scrum**

2.2 What are stories

2.3 Summary

What is Scrum

- Scrum
  It is derived from “scrum” of rugby football.

- Proponents
  Ken Schwaber, Jeff Sutherland, Mike Beedle

- Origin
  It has been tried since 1996, and refined until now.

- Characteristics
  - It has rich set of management practices.
  - It does not refer to technical issues.
  - Therefore, it can be used to combine other methods(such as XP).

Scrum way of thinking

Purpose

- Adaptive process responding changes

To realize it

- Accurate (grow up) working software

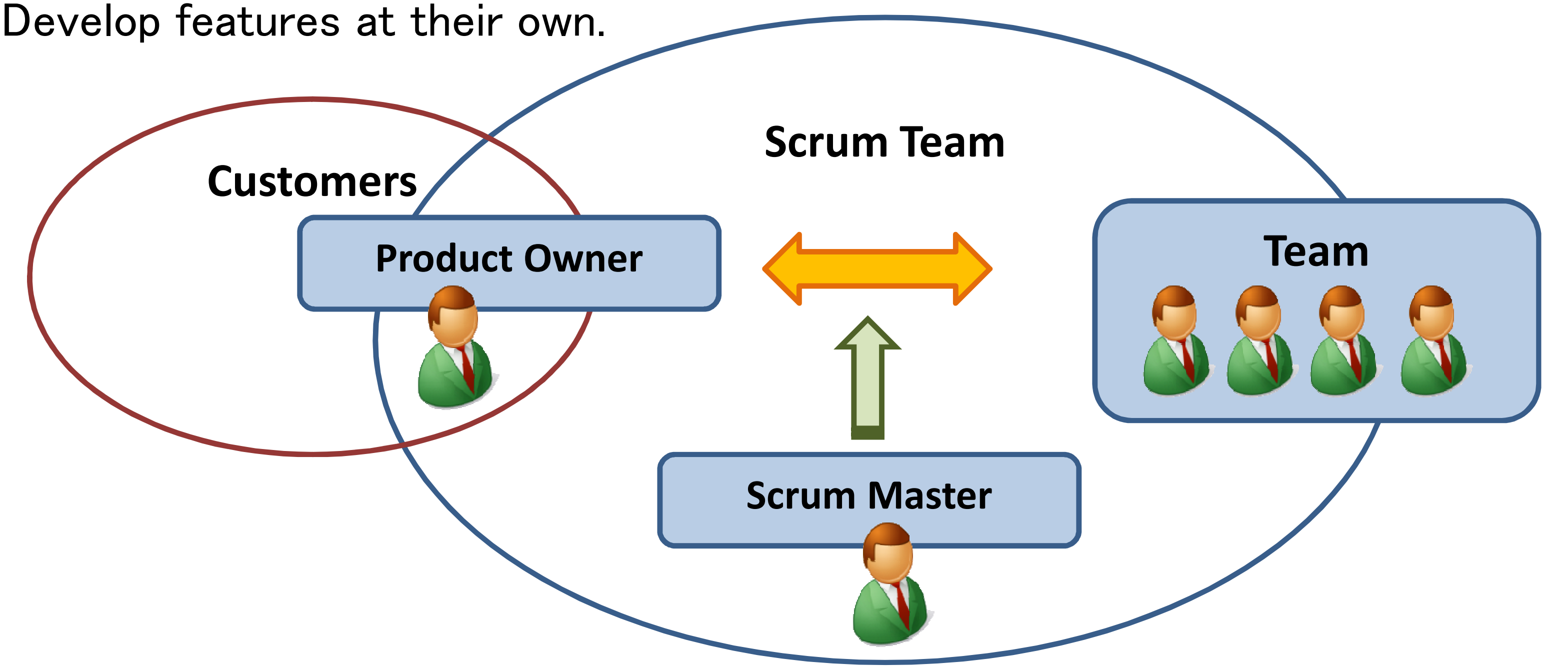
Mechanisms to grow up in stages

- Ensure frequently the software is always correct.
- Check frequently whether there is abnormal things during development.

Perspective of “feedback”

Organization and roles of Scrum （1）

- Simple roles
  - Product owner
    - Representative of customers who decides the specification.
  - Scrum Master
    - Help the project to proceed smoothly.
  - Team
    - Develop features at their own.



Organization and roles of Scrum(2)

- Characteristics
  - Product owner and Team communicate directly and deeply.

→ **Less noise, rapid feedback**

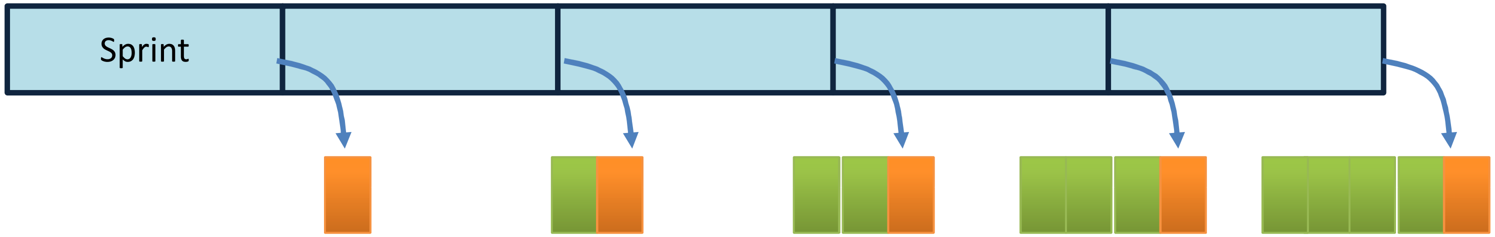
- Ideal Product owner
  - Only product owner can decide priorities among specifications.
  - Coordinate with stakeholders to decide specifications of the product as the representative.

- Ideal Team
  - All members talk with the product owner directly to develop the best product.
  - All members solve problems of Scrum Team.

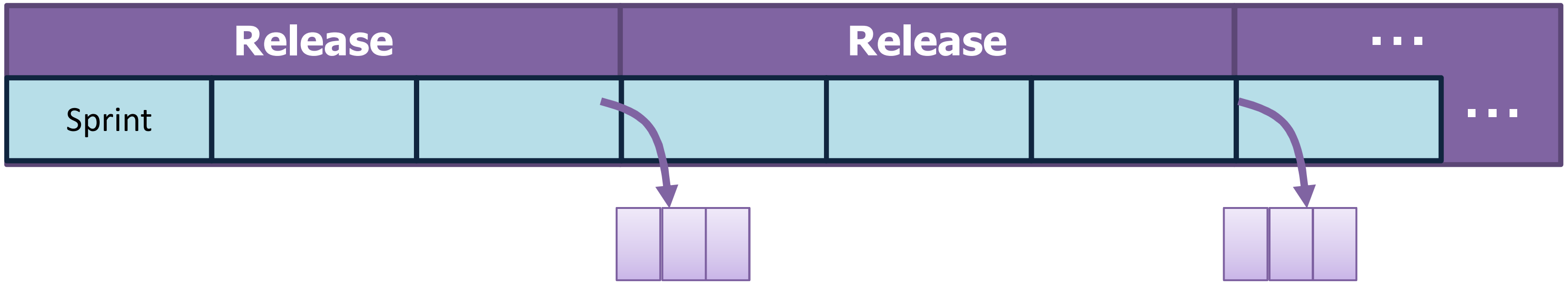
- Ideal Scrum Master
  - Ensure that process proceed in a good state.
  - Encourage communication between Product owner and Team.
  - When there is a problem in Team, **help** them to solve the problem.
  - Not team leader or project manager who manages from the top.

Process overview of Scrum (1)

- Repeat Sprints
  - Fixed development time called sprint is repeated many times.
  - At the end of any sprint, team delivers software artifacts to their customers.
  - As sprits proceed, the customer gets richer software artifacts.



- Concept of Release
  - Several sprints can be gotten together to consist one release.
  - At the end of any release, team releases software products.



Process overview of Scrum (2)

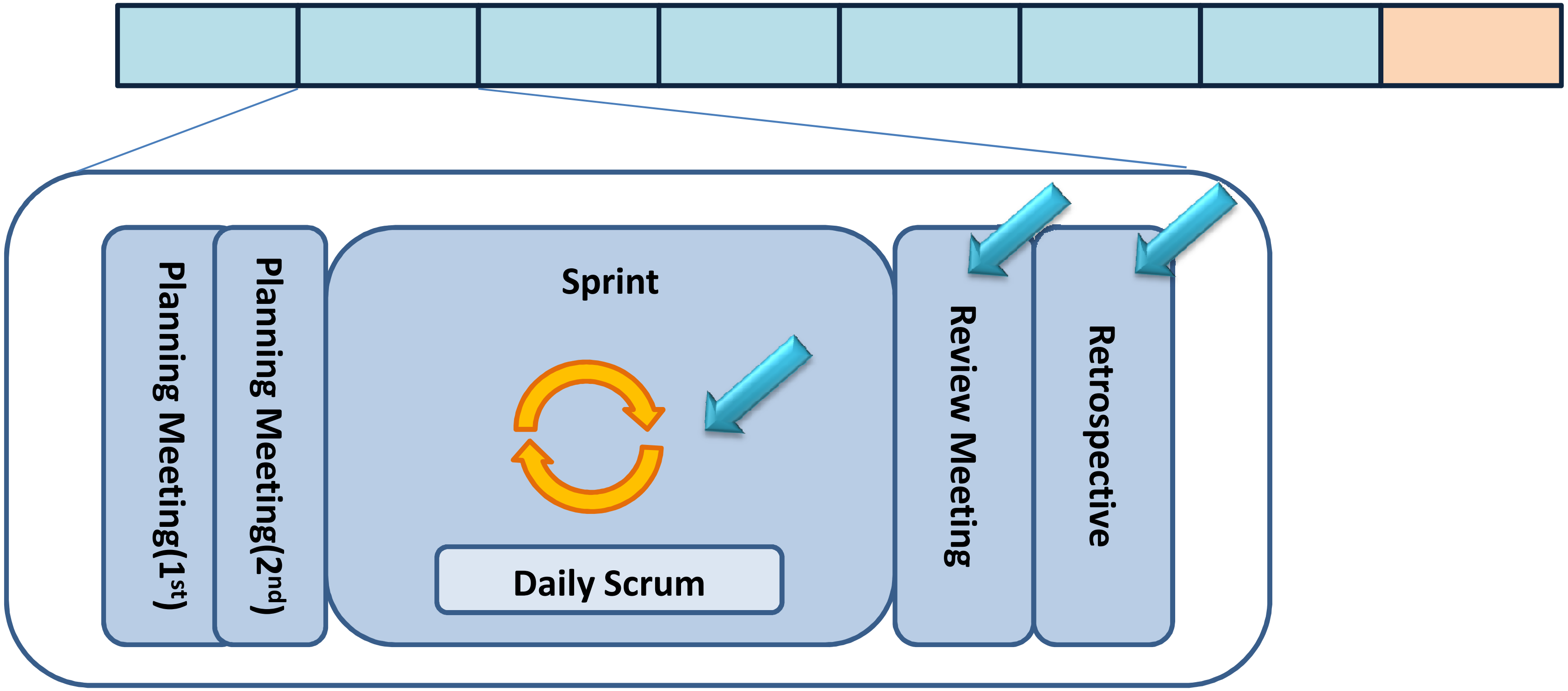
Meaning of iterations(sprints)

- Safe software development
  - It makes problems to solve smaller, to make the unit of software development smaller.
  - Accumulating reliable software artifacts, proceed software development step by step safely.
- Frequent feedback
  - Delivering software artifacts to the customers frequently, validate original requirements and understanding of the requirements.
  - Give the customers opportunities to decide (or change) their intention.
- Produce a development rhythm
  - Setting goals with fixed intervals, experiments with achievement are repeated.
  - Proceeding development with a fixed rhythm, makes estimate of tasks more accurate.

Inside sprint

- Frequent feedback by chains of repetition
  - Feedback from the customers
  - Daily feedback inside the team
- Fixed length time box

Rapid and frequent feedback

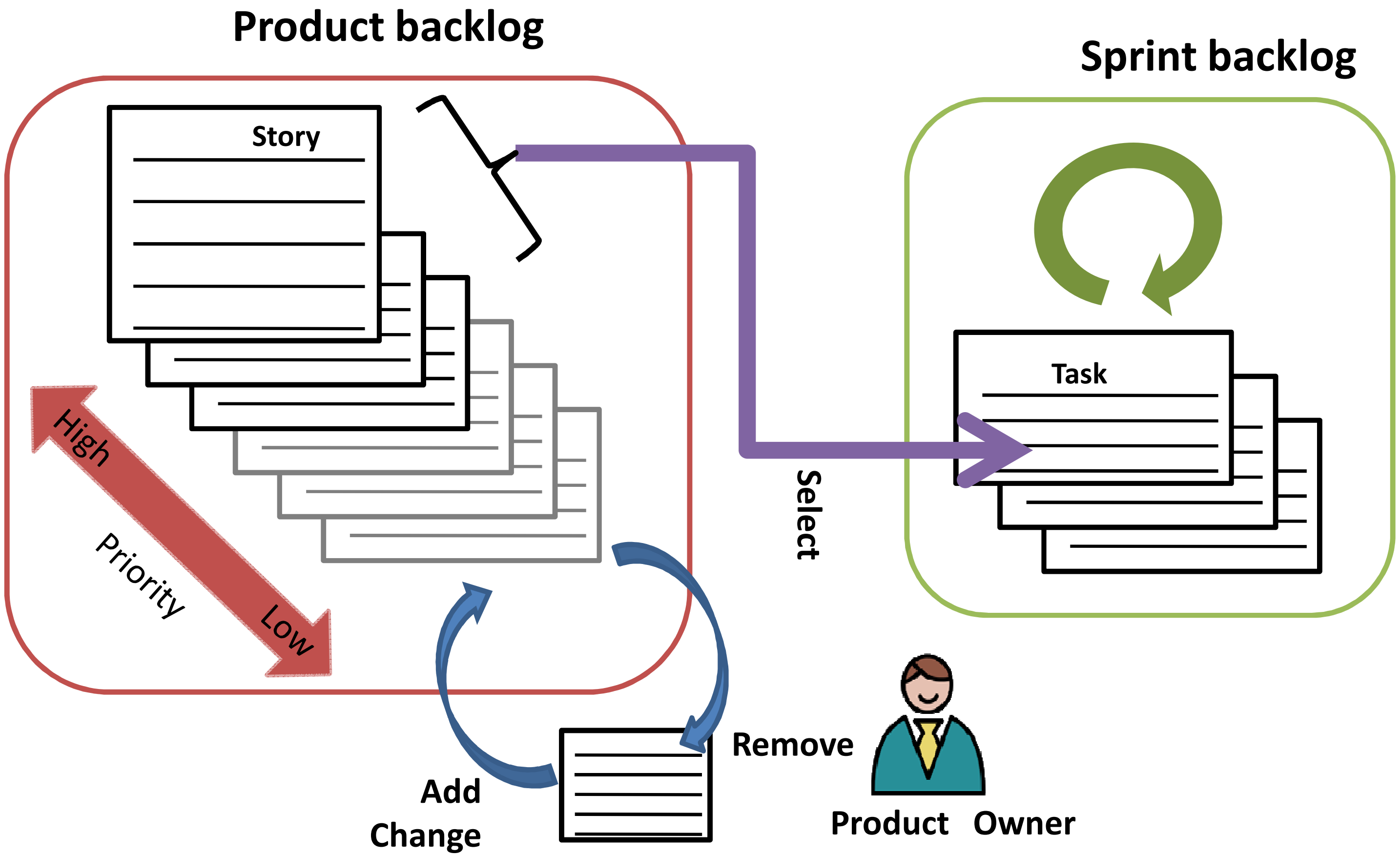


Meetings defined in Scrum

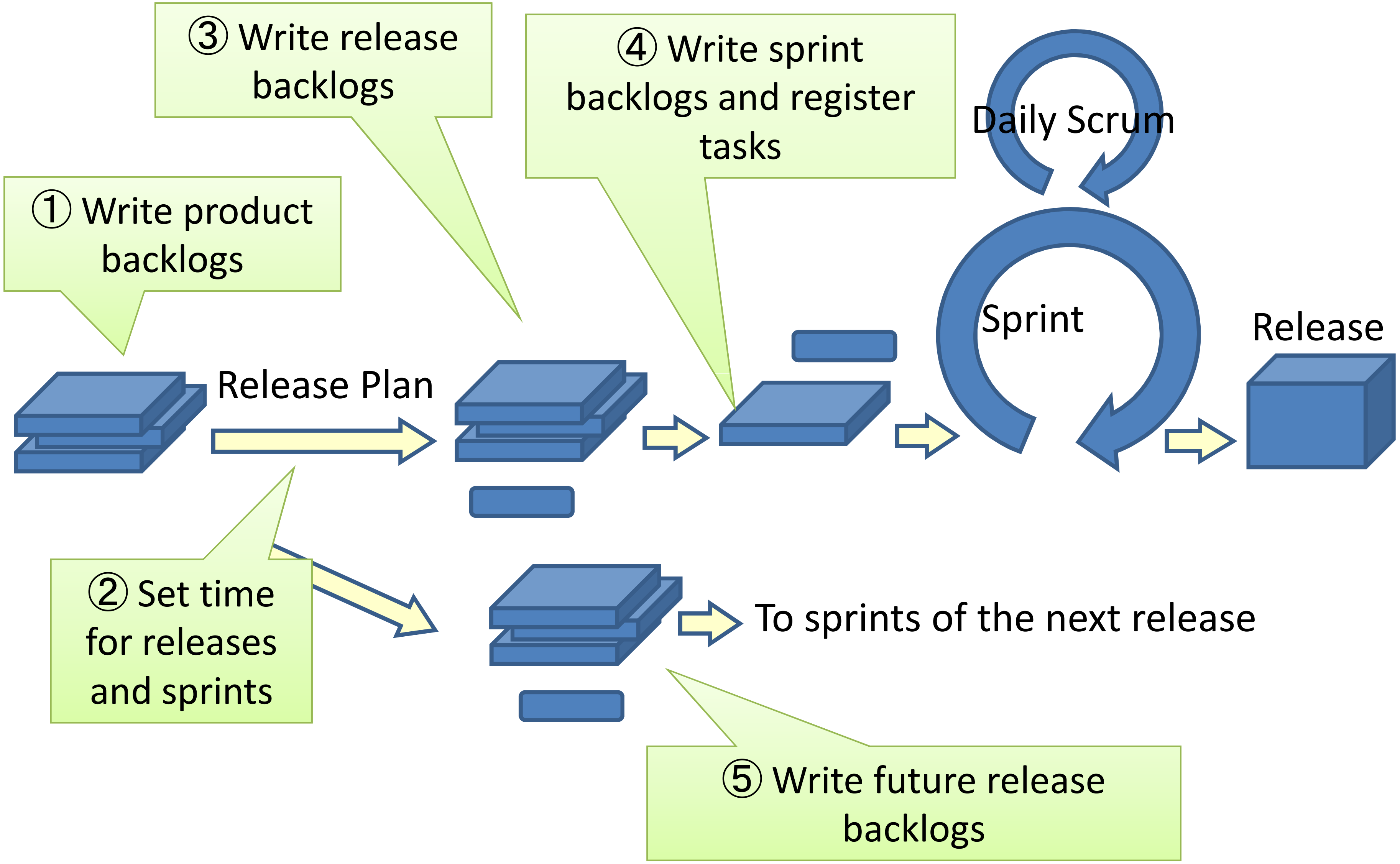
- Daily Scrum
  - Every day, held at same time, at same place(about 15 minutes)
  - Scrum master hosts it , and all team member participate.
    - Things done since the last meeting.
    - Things to do until the next meeting.
    - Obstacles against current tasks.
    - “Hens and hogs”
- Sprint Planning Meeting
  - All stakeholders decide the goal and features of the next sprint (1st).
  - Team reveals necessary tasks for the goal and features (2nd).
- Sprint Review Meeting
  - Review things constructed during the spring.
- Retrospective
  - Meeting inside Team to think backward and forward.

Task Management of Scrum

- Task Management by Backlogs
  - Manage product backlog items(stories) written in users’ language.



Scrum Process General View (1)



③ Write release backlogs
④ Write sprint backlogs and register tasks
Daily Scrum
① Write product backlogs
Sprint
Release
Release Plan
② Set time for releases and sprints
To sprints of the next release
⑤ Write future release backlogs

Scrum Process General View (2)

① Write product backlogs

Write a list of requirements for the product (**product backlogs**).

② Set time for releases and sprints

Determine release date and iteration (**sprint**) span (normally 30 days).

③ Write release backlogs

From the product backlogs, select ones developed for this release, make them **release backlogs**.

④ Write sprint backlogs and register tasks

Team reveals necessary **tasks** to realize the target of this sprint and register the tasks as **sprint backlogs**.

⑤ Write future release backlogs

Product backlogs which are not developed for this release are saved for the future.

Terms of Scrum

- Scrum has a lot of characteristic terms.
- In Japanese agile development, XP terms are often used.

Scrum Terms	General Terms	Description
Sprint	Iteration	Development cycle to be repeated
Product backlog	Story	Description of features which customers select for their product.
Sprint backlog	Task	Work items development team do.
Scrum master	Agile coach	Facilitator for agile process

This course use only general terms after that.

Chapter 2. Scrum and Stories

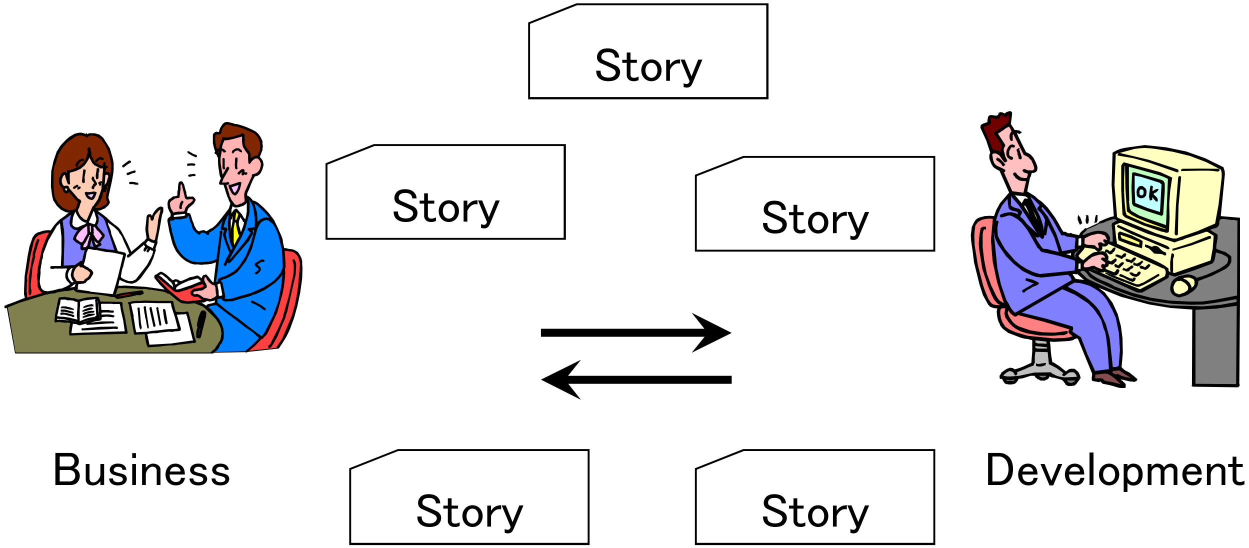
2.1 Overview of Scrum

2.2 **What are stories**

2.3 Summary

Story

- Features visible from users
- Concepts, not detail specification
- Write in users’ language
- Succinctly in one or two lines
- Moderate size to be estimated (if possible)



Story

Story

Story

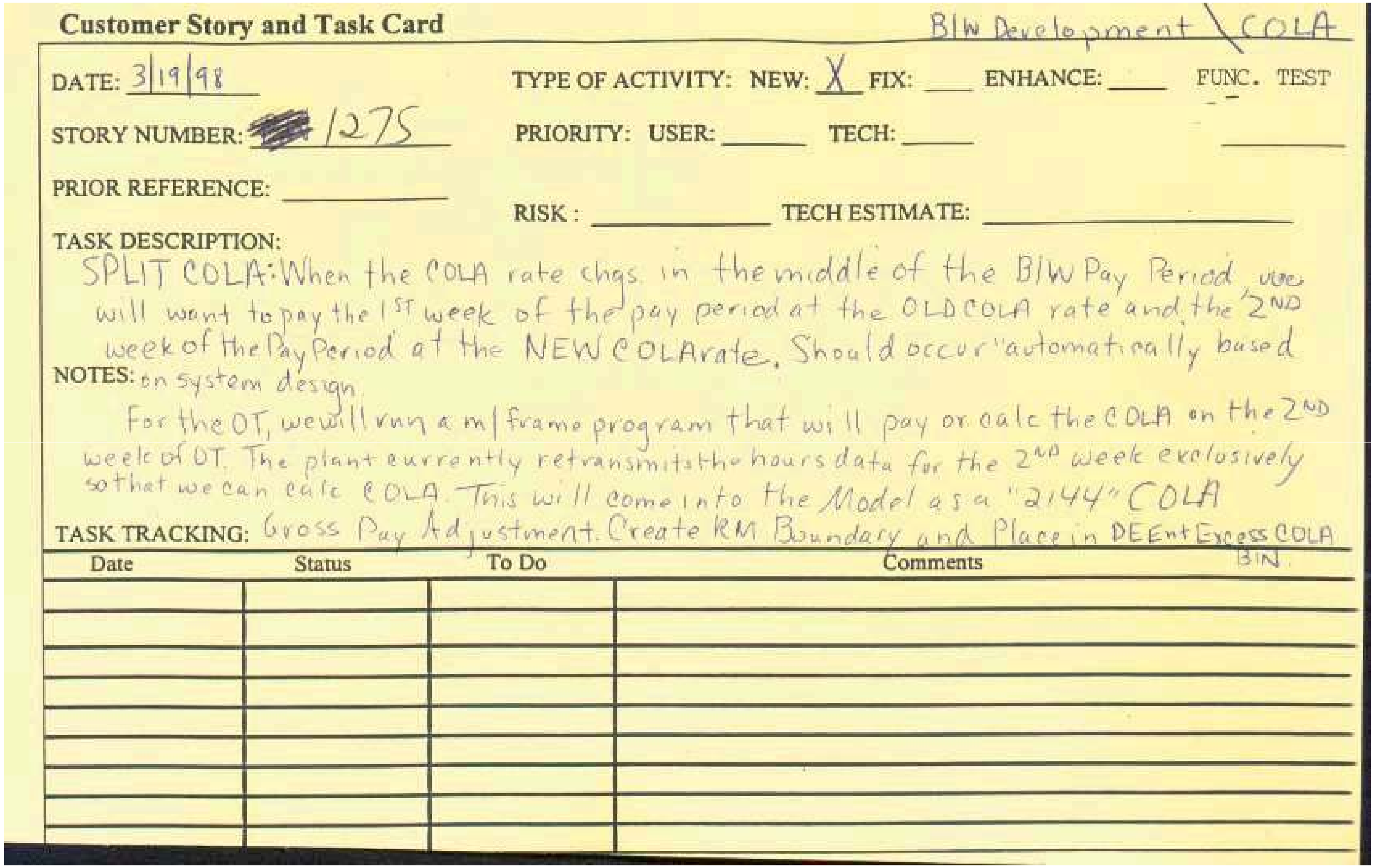
Business

Story

Story

Development

Story Card



Customer Story and Task Card B|W Development \ COLA

DATE: 3|19|98

TYPE OF ACTIVITY: NEW: X FIX: ___ ENHANCE: ___ FUNC. TEST ___

STORY NUMBER: 1275

PRIORITY: USER: ___ TECH: ___

PRIOR REFERENCE: ___

RISK : ___ TECH ESTIMATE: ___

TASK DESCRIPTION:

SPLIT COLA: When the COLA rate chgs. in the middle of the B|W Pay Period, we will want to pay the 1ST week of the pay period at the OLD COLA rate and the 2ND week of the Pay Period at the NEW COLA rate. Should occur "automatically based on system design.

NOTES:

For the OT, we will run a m/frame program that will pay or calc the COLA on the 2ND week of OT. The plant currently retransmits the hours data for the 2ND week exclusively so that we can calc COLA. This will come into the Model as a "2144" COLA

TASK TRACKING: Gross Pay Adjustment. Create RM Boundary and Place in DE Ent Excess COLA BIN

Date	Status	To Do	Comments

From "XP explained"

Contents of story cards

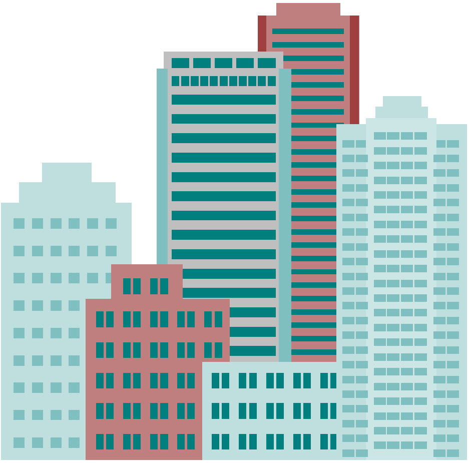
- One story in one card
- Prepare fields for priority, estimated time, and actual development time.
- You can attach materials which help developers’ understanding.
- Considerations are written in notes fields.
- When a card get unnecessary because its story has been split, shrunk or rewritten, tear it down.

Estimate of stories

- Story Points
- Planning Poker
- Yesterday’s weather
- Velocity
- Spike

Story Points

- It is difficult to anticipate with an absolute scale
  - It can be estimated with a relative scale.
  - “How high is the highest building?”
- Story Points（SP）
  - Determine a story to be a reference.
  - Compare other stories with the reference story.
  - Avoiding influence of absolute values, use a virtual scale(SP).



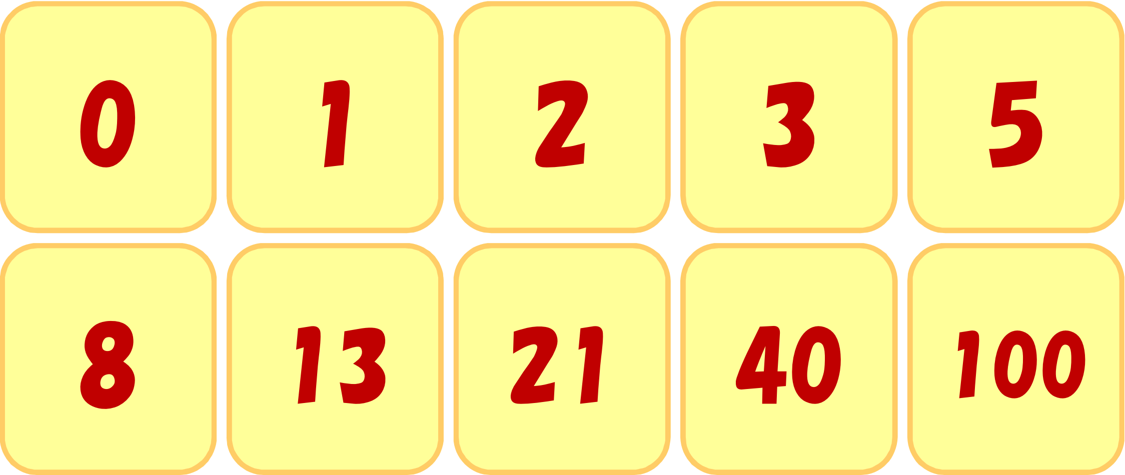
Planning Poker

Planning Poker

- One of tools to estimate with story points.
- All members show a card simultaneously.
  - If all cards are same, pick the number.
  - If difference is small, pick the bigger one.
  - If difference is large, talk together.

(It seems members have different assumption.)

Numbering is slightly different with kinds of cards, but mostly Fibonacci sequence is used.

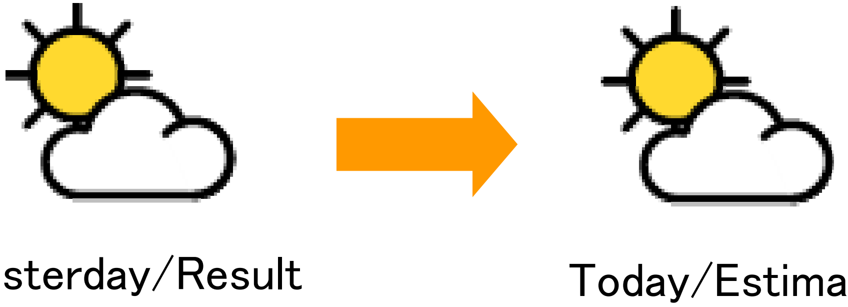


0 1 2 3 5

8 13 21 40 100

Yesterday’s weather

- Yesterday’s weather
  - Today’s weather is generally same with yesterday’s one.
  - Predicting tomorrow’s weather with more cost, the result is not different for the most time.
- “Amount done on yesterday will be amount to do on today.”
  - Based on “actual results”, not “expectation”.
  - Make automatic estimation possible.
- Key points
  - Keep estimate cost lower.
  - Don’t mind if estimation will turn out to be wrong. It is important to adapt the situation in a agile way.



Velocity

- Velocity ＝ Amount developed last time

- For example:
  - When 10 story points were developed during the last iteration,
  - Velocity ＝ 10 story points

Spike

- Research or development until some estimation can be made.
  - When estimation can be made, spike work will be stopped.

- For example:
  - Implement with unused technologies
  - Implement with new technologies

Planning Poker

Planning Poker Exercise

- Estimate necessary time to do the below things by team.
  1. Cook sweet-and-sour pork.
  2. Clean up a house with 2LDK.
  3. Go from Sapporo to Naha.
  4. Write a report attending this course.

  Unit: Hours

- Show a card simultaneously in all.
  - If all cards are same, pick the number.
  - If difference is small, pick the bigger one.
  - If difference is large, talk together.（It seems assumptions are different. ）
  - All estimation from 1 to 4 must be finished in total 15 minutes.

- Present the results by team.
  - Estimated hours can be different from numbers on cards.

Chapter 2. Scrum and Stories

2.1 Overview of Scrum

2.2 What are stories

2.3 **Summary**

What to learn in this chapter(again)

- Overview of Scrum
  - Scrum way of thinking
  - Organization and roles of Scrum
  - Overview of Process

- Stories
  - What are stories
  - Estimate of stories

Summary of this chapter

- Scrum
  - Embrace changes and adapt
  - Stepped growing up of software

- Simple Roles
  - Product Owner
  - Scrum Master
  - Team

- Process Overview
  - Sprint
  - Release
  - Meetings

- Stories
  - Description of specification in users' language
  - Estimate（Story points･Yesterday's weather･Velocity･Spike）

Chapter 3. Overview of XP

What to learn in this chapter

- Overview of XP
  - Position of XP
  - 4 Values of XP

- Practices of XP
  - 3 Perspectives
  - Project Management Practices
  - Team Operation Practices
  - Development Practices

Chapter 3. Overview of XP

3.1 **What is XP**

3.2 Practices of XP

3.3 Summary

What is XP

- eXtreme Programming

  do things considered good extremely

  do nothing considered unnecessary at all

- One of Agile Software Development Methods

- Values XP provides
  - Risk reduction of projects
  - Adapt changes of business rapidly
  - Improve productivity

The Three Extremos

Proponents of XP

- Ward Cunningham - the inventor
- Kent Beck - the articulator
- Ron Jeffries - the realizer

＋

- Martin Fowler

4 Values XP represents

- Communication
- Simple
- Feedback
- Courage

※ Respect

Communication

- Cause of many problems

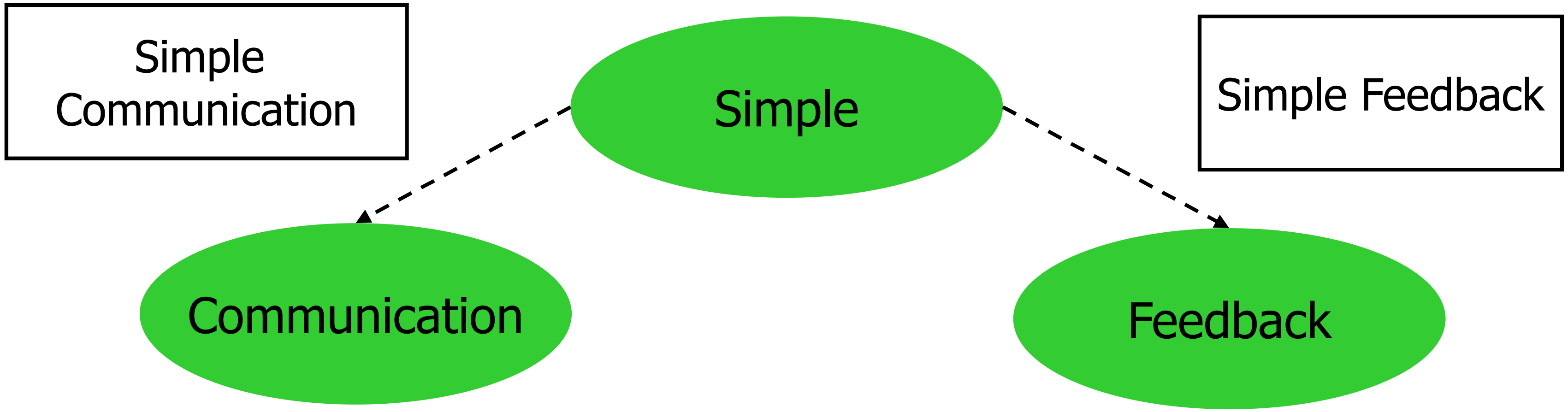
- Among end users, programmers, managers
  - System requirements
  - Intention of design
  - Failure, trouble
  - Progress

- Many practices of XP cannot be done without communication
  - Pair Programming
  - Whole Team etc

Simple

- Make a simple implementation for today
  - If something is necessary tomorrow, implement it tomorrow.
  - Complex functions implemented today may not be used tomorrow.
  - If system is simpler, it is easier to communicate.

- What is the simplest way of communication?
  - Face to face communication with users



Feedback

- The most informative feedbacks are from system under production.
  - Requirements：As users operate the system
  - Design：Read real code and add some functions
  - Quality：Test the system under production
  - Progress：See working functions and actual results

Courage

- Courage is extremely valuable combined with communication, simple, feedback.
  - Courage to fix defects, and throw code away
  - Courage to keep system simple
  - Courage to communicate honestly
  - Courage to accept feedbacks from other members

- A proof XP considers humans as the most important element.

Respect

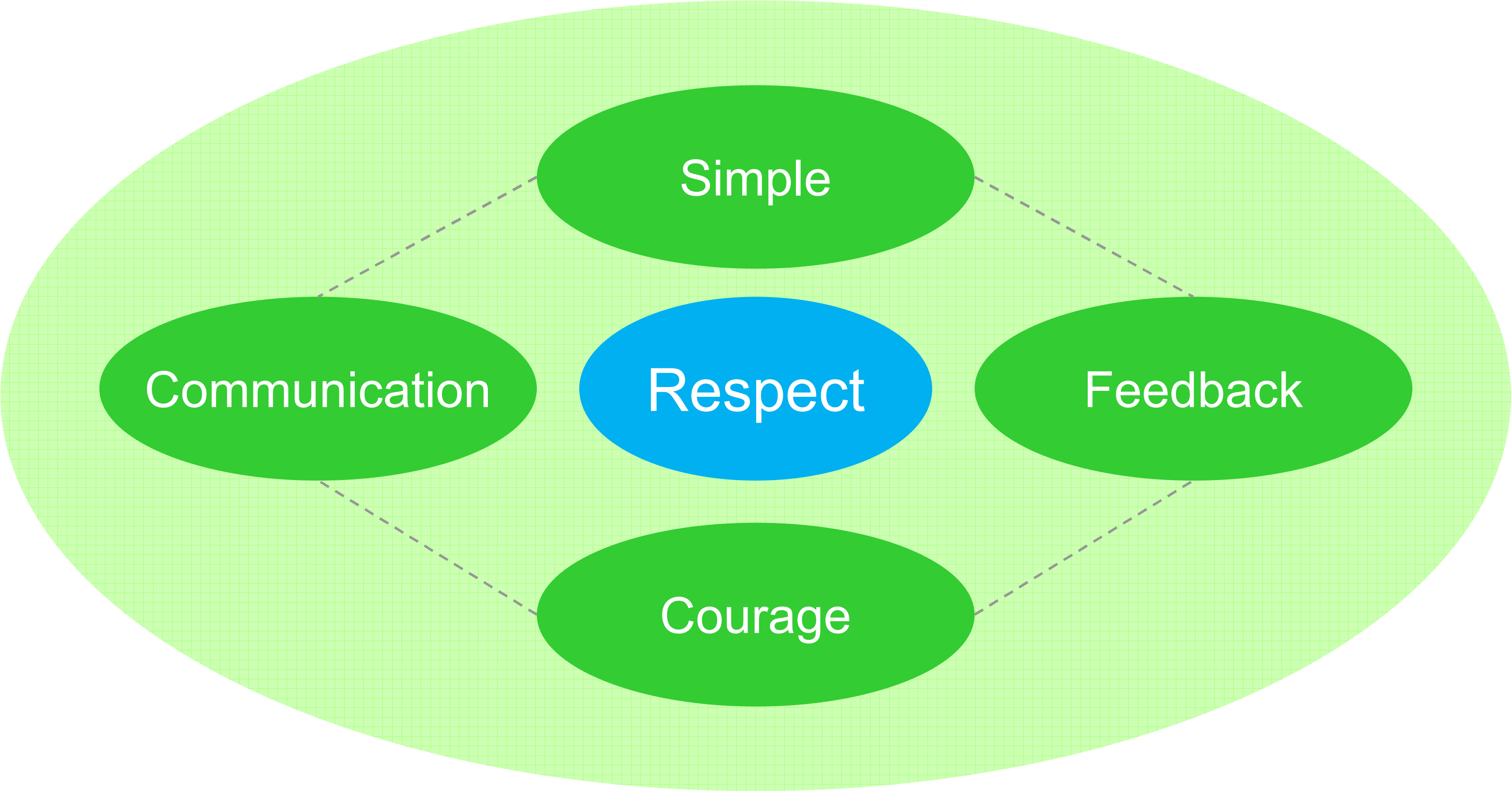
- The value behind 4 values

- Team members respect each other.
  - In agile development, cooperation of team is more important.

- Members respect the value of the project itself.
  - All members must understand the value realized by the project.
  - Show the goal explicitly, and trust members to realize it.

Meaning of 4 values

- Prerequisite of anything(practices, process, and so on) in X P
  - Guide to make a difficult decision



Chapter 3. Overview of XP

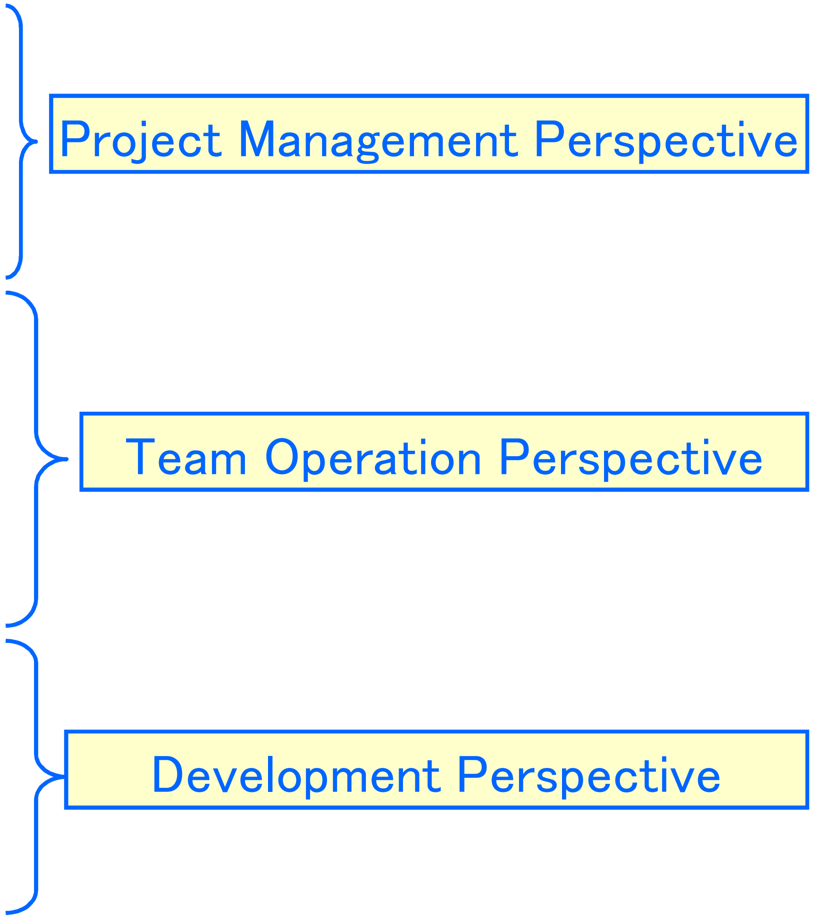
3.1 What is XP

3.2 **Practices of XP**

3.3 Summary

Practices of XP

- Small Releases
- Whole Team
- Customer Tests
- Planning Game
- Collective Ownership
- Continuous Integration
- Coding Standard
- Metaphor
- Sustainable Pace
- Simple Design
- Refactoring
- Test-Driven Development
- Pair Programming

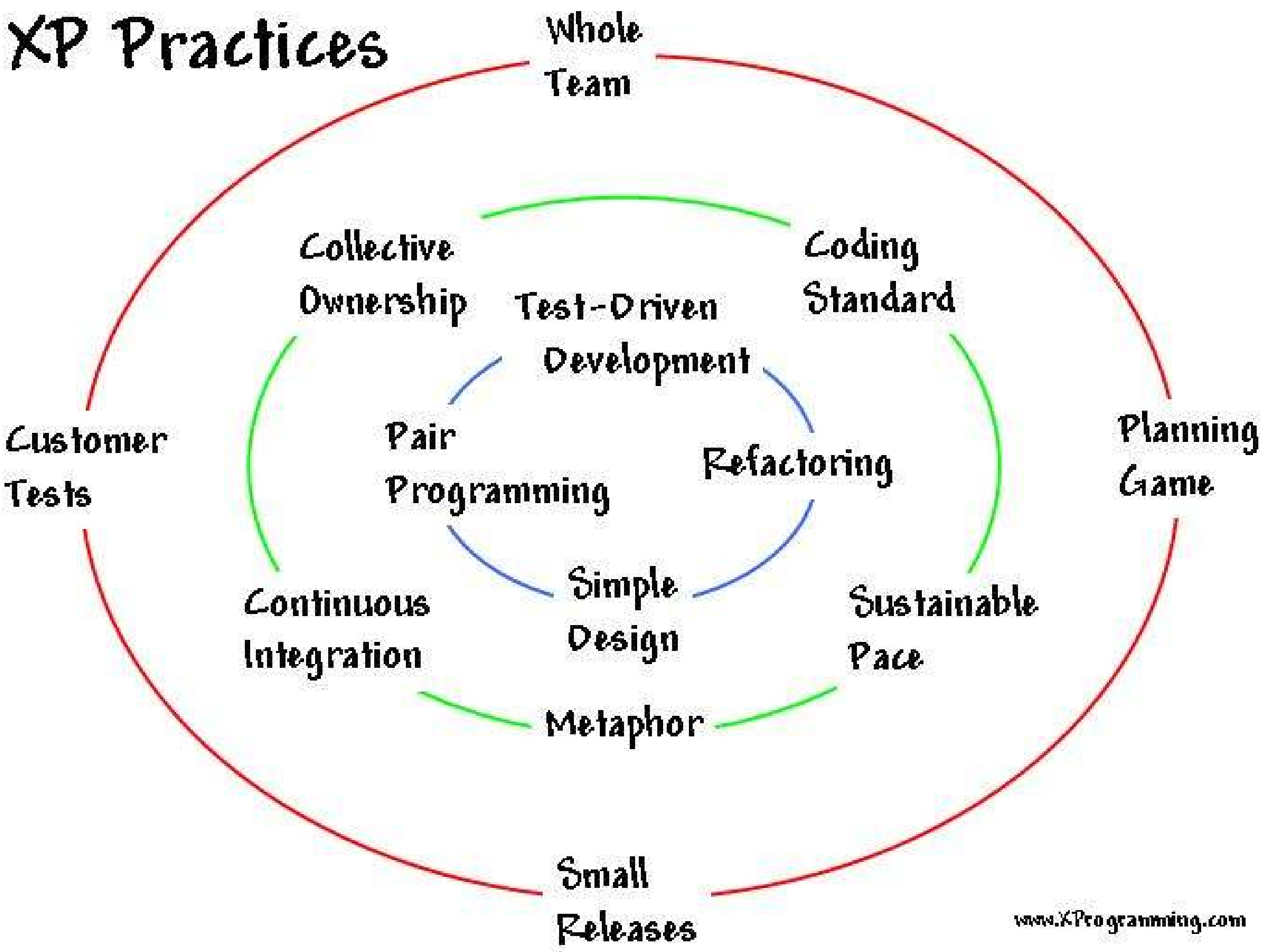


Project Management Perspective

Team Operation Perspective

Development Perspective

Practices of XP



Practices of XP

Here!

Project Management Perspective

Team Operation Perspective

Development Perspective

Practices from Project Management Perspective

- Small Releases
  - Make the time until the next release as small as possible
- Whole Team
  - Users and developers are both in the same team.
- Customer Tests
  - Users in the team perform acceptance tests.
- Planning Game
  - Following rules, estimate and make plans.

➡ Unify users and developers

Small Release

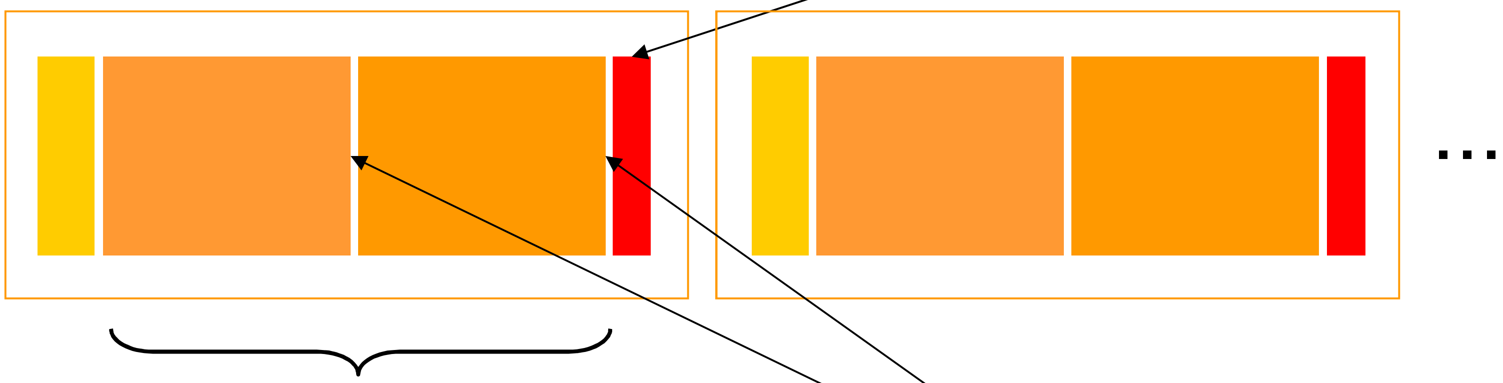
- Released software can be put into production
  - Deliver working software as soon as possible
  - Communication with actual system

- Reflect feedbacks from users to next development

↓

  - Users need courage to accept feedbacks from developers.
  - Developers needs courage to accept feedbacks from users.
  - Communication can be established only when they respect each other.

Small Release（Whole Plan）



Release to end users
（most frequently without confusing）

Releases within developers
（every day /needs integration environment）

Releases to users
（for features the users have selected for the iteration／needs evaluation environment）

Roles within XP Team

- Users
  - Have domain knowledge of targeted area.
  - Decide priorities of stories to be realized.
  - Understand business value provided by the system under development.
- Programmers
  - All developers
- Coaches and Managers
  - Eliminate difficulties of process and communication within the team.
  - Facilitation
  - Do not violate rights of users and programmers.
- Trackers
  - Act as sensors for team progress

Whole Team

- Development in the past
  - Users, managers, developers belong to other teams and have different interests.
  - Naturally, they work in different offices.

	Cost	Time	Quality
Users	Lower	Earlier	Higher
Managers	Higher	Earlier	Higher
Developers	Higher	Slower	Higher

Whole Team（Merits of XP）

Development in XP

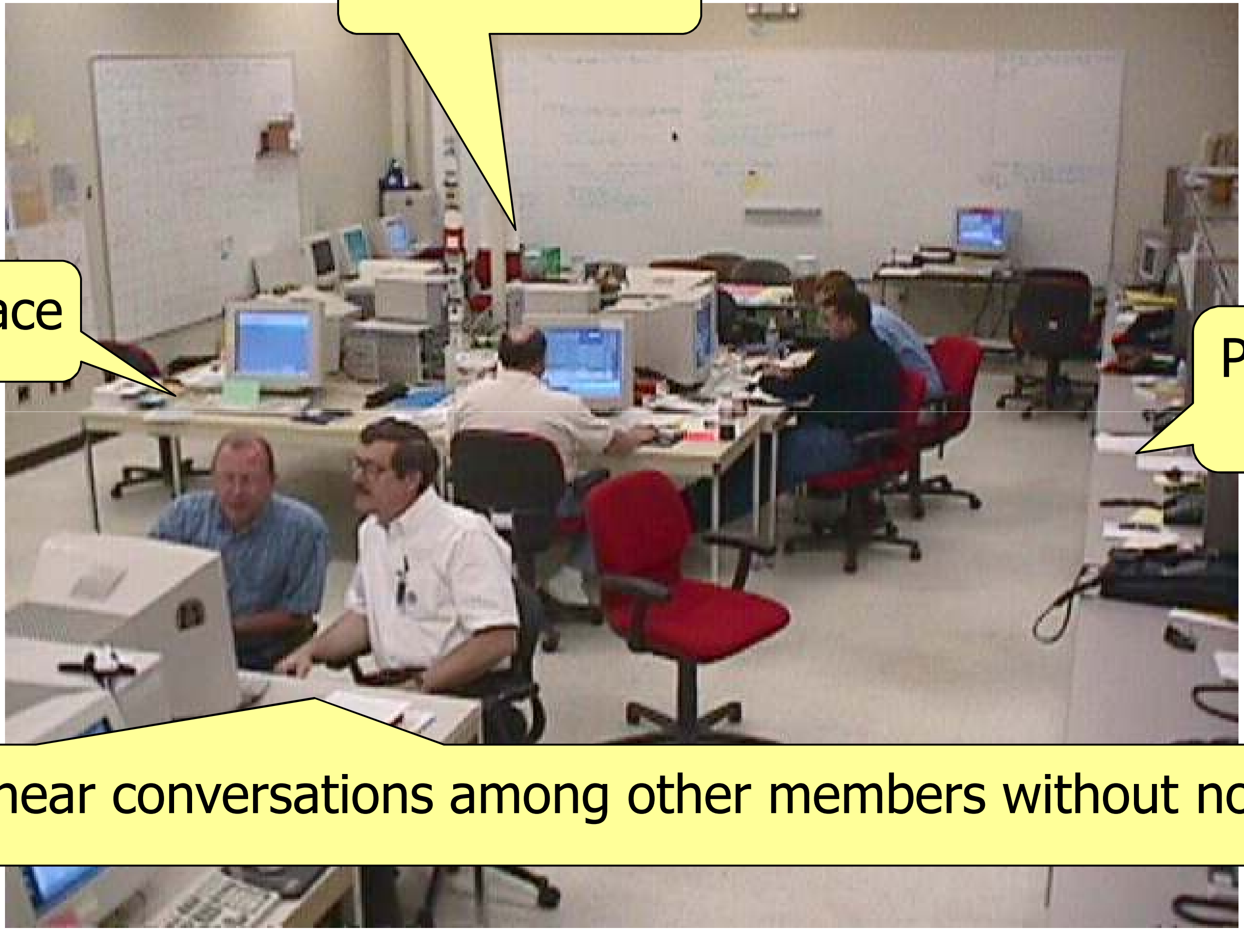
- Users, managers, developers are team members who have save goals. Users are too important to put outside of the team.
- Sit Together
- When the team is organized, consider following things.

	Cost	Time	Quality
XP Team	Optimized	Optimized	Higher

- Encourage communication and feedbacks.
- Realize simple communication（Face to Face）.
- Communicate honestly with courage
- Share the big goal of the team

Facility Strategy

- C3 Team



Rest Space

Common Space

Private Space

You can hear conversations among other members without notice.

Customer Test

- Way to confirm whether purpose of users has been achieved.
  - Communicate and feed back by tests.
  - Not communication or feedbacks by specification documents
  - It is not the goal that system gets matched with its specification documents.

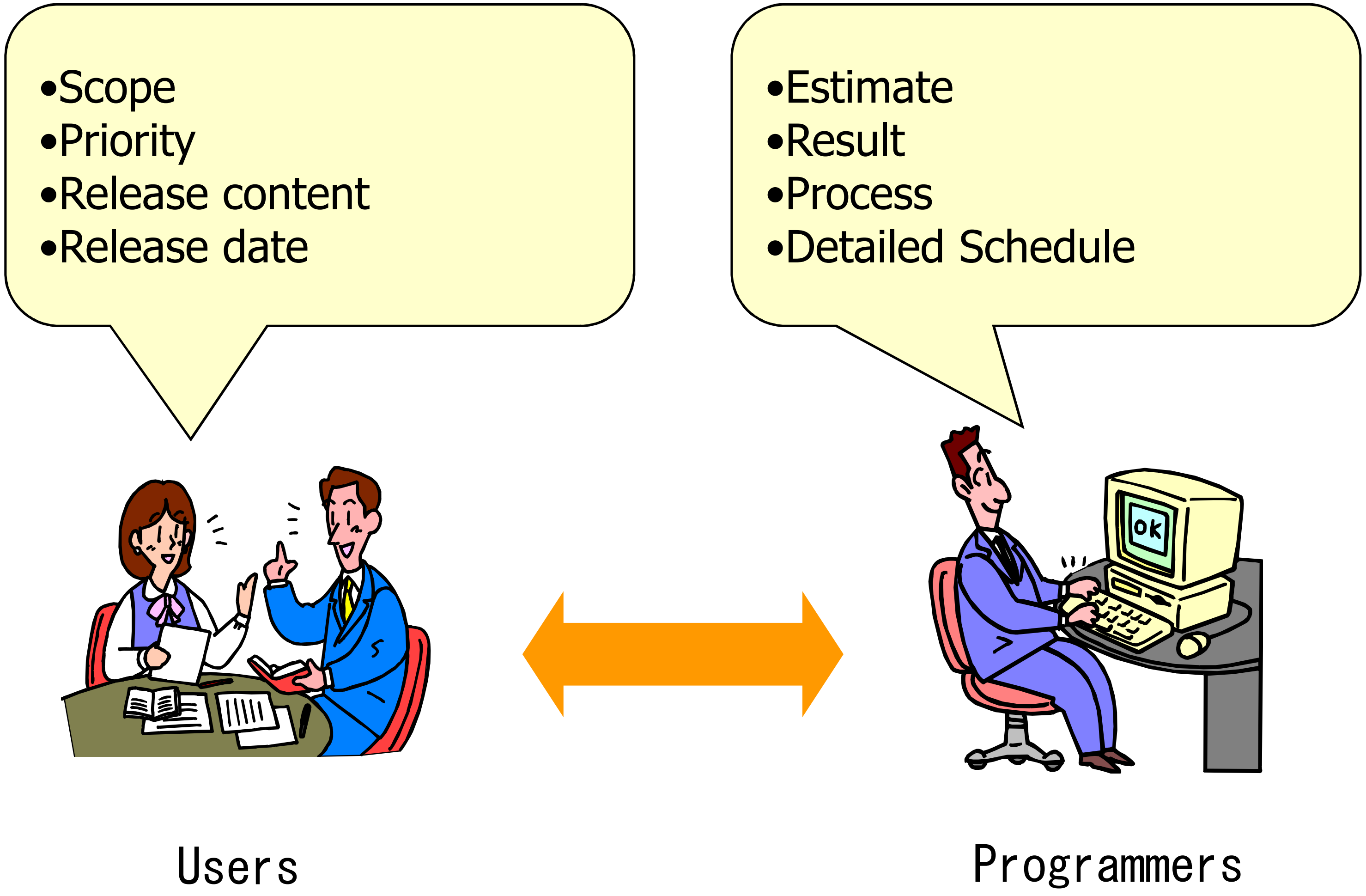
- How to do
  - Customers define tests based on stories.
  - Customers perform the tests.

⇒ New discovery of customers becomes feedbacks to the team.

Planning Game

- Roles and procedures for planning are explicated by rules of a game.
  - Players
    - Users
    - Programmers
  - Rules
    - Any players play their role.
    - Must not interfere with a role of other players. （They respect each other）
- “Cooperative work” to achieve a maximum effect in the project
  - Talk about what users will get next among team members.
  - Users and programmers communicate to explore the most effective stories among realizable stories.
- What they have to plan
  - Estimate what can be achieved until a deadline.
  - Decide what to do next.

Planning Game – What to decide



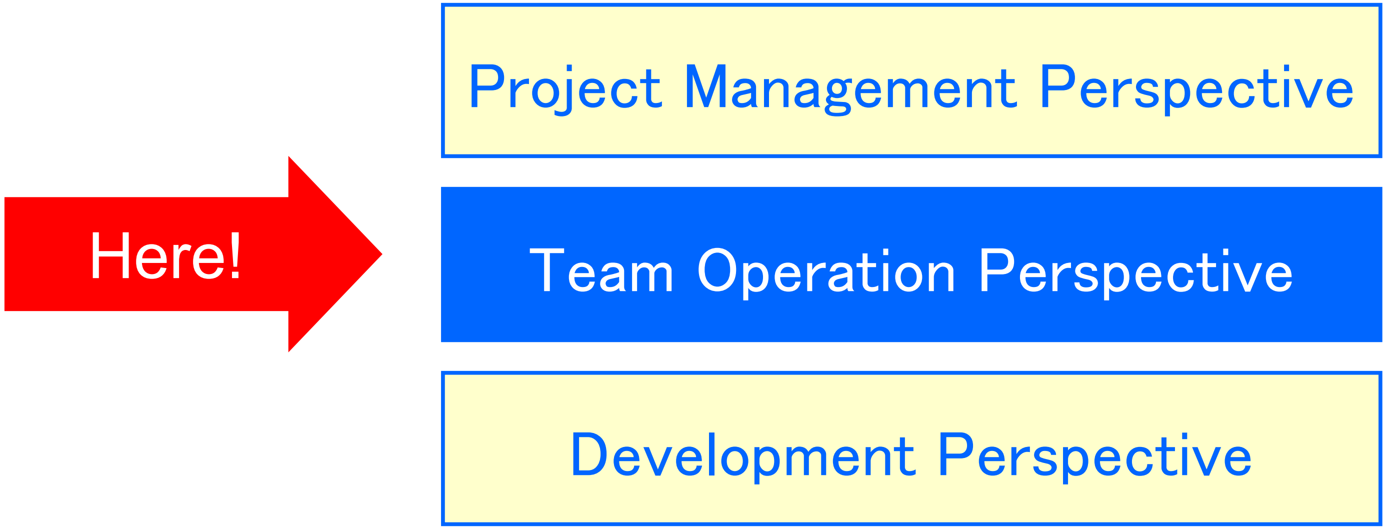
- Scope
- Priority
- Release content
- Release date

- Estimate
- Result
- Process
- Detailed Schedule

Users

Programmers

Practices of XP



Here!

Project Management Perspective

Team Operation Perspective

Development Perspective

Practices from Project Management Perspective

- Collective Ownership
  - Introduce SVN, VSS or another SCM tool
- Continuous Integration
  - Retrieve files from SVN to build system automatically.
- Coding Standard
  - Spontaneous rules
- Sustainable Pace
  - Keep work time within about 40 hours per week.
- Metaphor
  - Share concepts and construct an architecture

➡ Focus on efficiency of team development

Collective Code Ownership

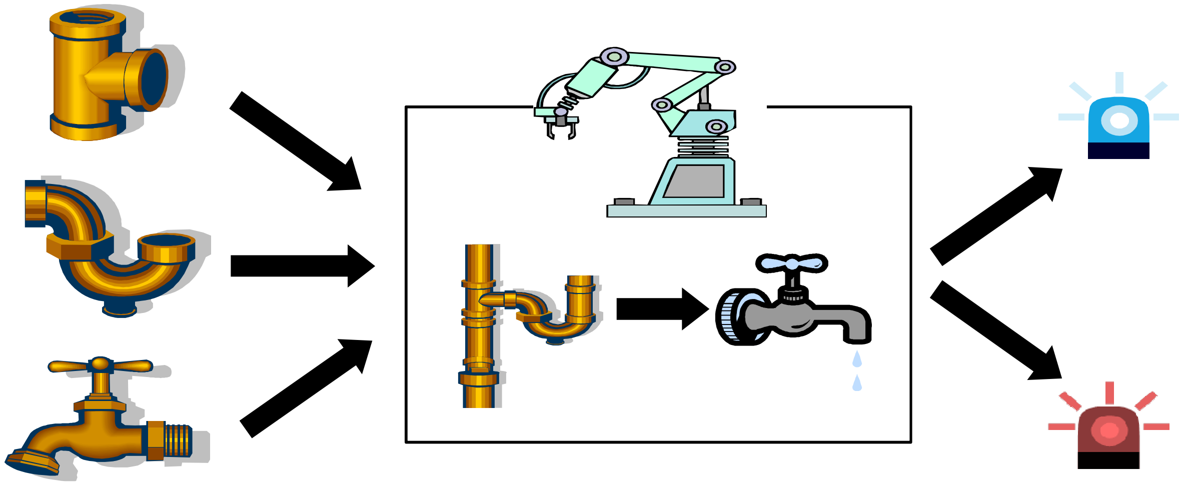
- Anyone can modify code anytime anywhere.
  (If necessary, anyone must do it.)
- There is no gatekeeper for any piece of code.
  - This is true for schema, libraries, and documents.

- Pros
  - Everyone understands every code.
  - It encourages simple design and refactoring.
- Cons
  - My code is modified by others.
  - It is likely that responsibilities of everyone become responsibilities of no one.

Continuous Integration

- In integration environment, every test should keeps passing.
- Once a task is done, integrate it immediately.
- Clearer where is defect, easier to fix.

- Every time a check in (commit) is made, build the system end to end.
- Check in (commit) frequently.
- Needs an environment to build fast.
- Needs an environment to test fast.



Coding Standard

- Promises agreed by every member（rules within the team）
- Spontaneously agreed.
- Continuously improved.
- Reinforced with tools such as Checkstyle

- Code readability
- Code anyone can modify（”Everyone owns every code”）

- Supports the below practices

  ・Collective Code Ownership

  ・Pair Programming

  ・Refactoring

  ・Continuous Integration

Sustainable Pace

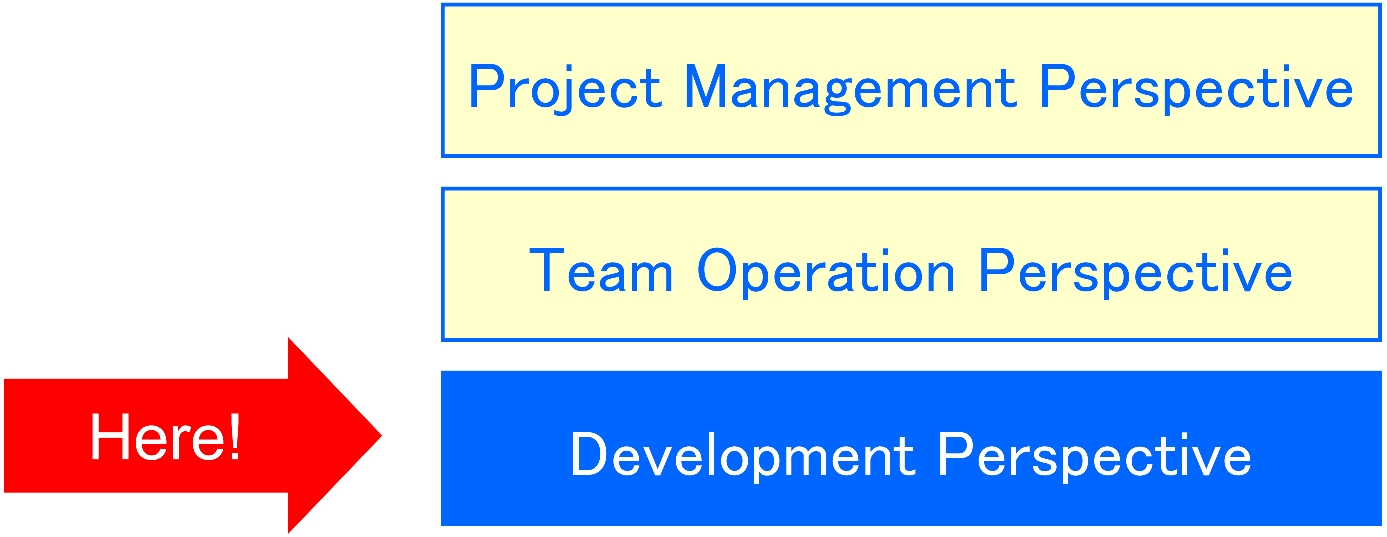
- Tired people cannot do their best.
- When they cannot do their best, they produce a mess.
- A mess slows every member down.
- Therefore any member needs rest.
- Sometimes moderate overtime can be continued about one week.

- You practice XP in order to develop the best software. You cannot go home leaving what to do the day undone.

Metaphor

- Purpose of Metaphor
  - Construct common vision
  - Share vocabulary
  - Generate ideas
  - Architecture
- (Example) Desktop screen of PC→desks at office
  - What is needed
    - Storages
    - Notebooks
    - Pencils
    - Drawers
    - Trash boxes etc
  - What is considered
    - Who throws trashes away?
    - Where can be customized, and where cannot be ?

Practices of XP



Practices from Development Perspective

- Pair Programming
  - A pair of 2 programmers performs development tasks from design until test.
- Test-Driven Development
  - Write test code first, then write production code.
- Simple Design
  - Keep designs extendable
- Refactoring
  - Design improvement from the bottom. Not adding features.

➡ Development which can adapt change

Pair Programming

- Approaches
  - Two programmers develop software side by side at one computer.
  - They type code in turn.
  - Change pairs frequently in short time.

- Having knowledge dispersed across the team
- Novices can learn by watching and listening,

- Results of pair programming
  - Development-time：increase１５%; 20 hours for one individual ＝11.5 hours for two collaborators）
  - Defect-rate　：reduce １５%
  - The pairs consistently implemented the same functionality as the individuals in fewer lines of code
  - Improve staff skills and reduce staff risk.
  - Expertise are transferred from experts to novices.
  - They enjoyed pair programming more than when they programmed alone.

The Costs and Benefits of Pair Programming－Alistair Cockburn,
Laurie Williams
http://collaboration.csc.ncsu.edu/laurie/

Test-Driven Development

- Develop by tests（Tests drive development）
  - It is not methods of testing, but methods of designing and developping.

- Tests define specifications of components.

↓

Before writing production code, write unit tests and perform automated tests.

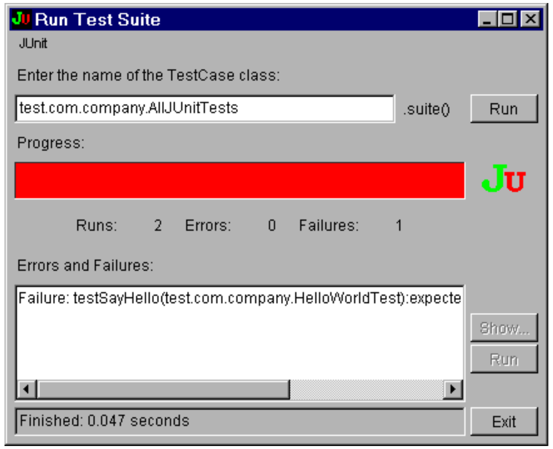
↓

You can have a completed set of implementations and tests for the feature.

Test-Driven Development – Test Cycle

Small cycle（about 5 minutes）

- Write a test
- Write code to make the test failed.
- Run the test. → ①Red
- Write code to make the test passed.
- Run the test. → ②Green
- Repeat the above until your program behaves as you expected.



Enter the name of the TestCase class:

test.com.company.AllJUnitTests .suite()

Progress:

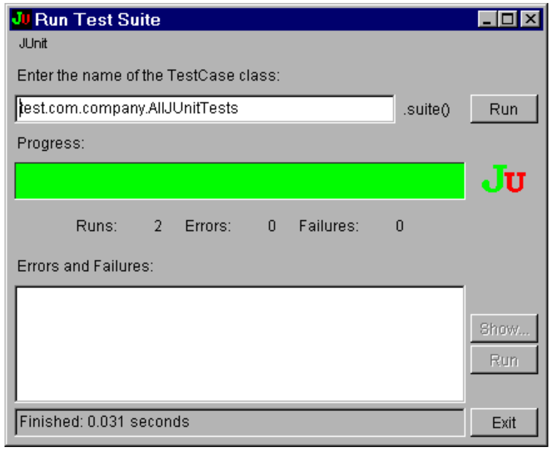
Runs: 2 Errors: 0 Failures: 1

Errors and Failures:

Failure: testSayHello(test.com.company.HelloWorldTest):expecte

Finished: 0.047 seconds

①Red



Enter the name of the TestCase class:

test.com.company.AllJUnitTests .suite()

Progress:

Runs: 2 Errors: 0 Failures: 0

Errors and Failures:

Finished: 0.031 seconds

②Green

Test-Driven Development – Test Code

- JUnit
  - Test Code

```
@Test public void firstZero() {
        assertEquals(3, add(0, 3));
}
@Test public void secondZero() {
        assertEquals(2, add(2, 0));
}
@Test public void bothInt() {
        assertEquals(5, add(2, 3));
}
```

  - Code under Test

```
int add (int first, int second) {
        return first + second;
}
```

Simple Design

- What is Simple
  - System (Code and Test) communicates information as needed.
  - System has no duplicate code.
  - System has no redundant code.

- Having simple
  - Code becomes readable.
  - Change impact is limited on a small part.

↓

Design which is easy to modify, and easy to extend.

- You aren’t going to need it. (YAGNI)
  - Implement necessary features.
  - Never implement more, it can be implemented when it becomes necessary.

Refactoring

- Once some task progresses, refactor.
  - Small improvement (about 5 minutes)
  - Assuming all tests are passed.
  - It needs to be very fast to build and test.

- To check in
  - All unit tests must be green.
  - There is no duplicates.
  - Code is as simple as possible.
  - Code represents its behavior properly.

- With comments, you write ideas on backgrounds.
  - Comments describing how it works should be as little as possible.
  - You should comment your idea which is not represented in the code.
  - There are more ways to express software richer other than comments.

Refactoring - Example

```
tax = 0.05 * price;
```

```
static final double CONSUMER_TAX_RATE = 0.05;

tax = CONSUMER_TAX_RATE * price;
```

“Refactoring: Improving the Design of Existing Code “

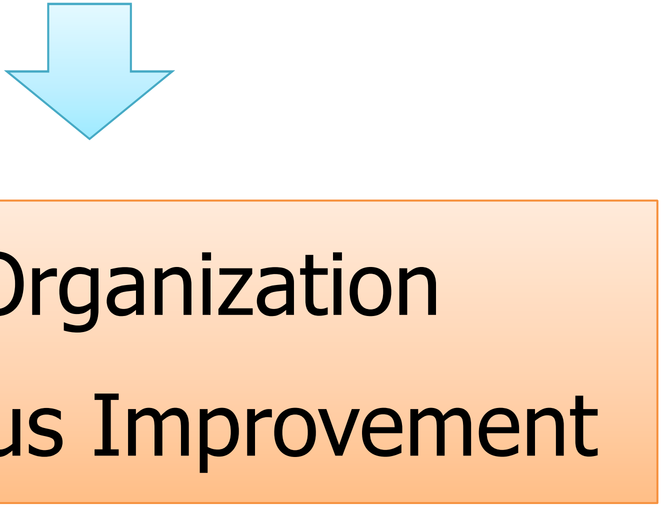
- By Martin Fowler

Addison-Wesley Professional,1999

Tailor practices into actual development sites

- Performing practices as indicated, makes sense. （It allows 4 values realize）

- However, actual development sites have various constraints, so some practices can not be done as indicated there.



Self Organization

Continuous Improvement

Chapter 3. Overview of XP

3.1 What is XP

3.2 Practices of XP

3.3 **Summary**

What to learn in this chapter(again)

- Overview of XP
  - Position of XP
  - 4 Values of XP

- Practices of XP
  - 3 Perspectives
  - Project Management Practices
  - Team Operation Practices
  - Development Practices

Summary of this chapter

- **Values of XP**
  - Communication
  - Simple
  - Feedback
  - Courage
  - Respect

- **Project Management Practices**
  - Small Releases
  - Whole Team
  - Customer Tests
  - Planning Game

- **Team Operation Practices**
  - Collective Ownership
  - Continuous Integration
  - Coding Standard
  - Metaphor
  - Sustainable Pace

- **Development Practices**
  - Simple Design
  - Refactoring
  - Test-Driven Development
  - Pair Programming

Chapter 4. Team Exercise

Team Exercise

Extreme Hour

References

References(1)

- **Extreme Programming Explained: Embrace Change**
  - Author：Kent Beck
- **Planning Extreme Programming**
  - Author：Kent Beck, Martine Fowler
- **Extreme Programming Explored**
  - Author：William C. Wake

XPエクストリーム・プログラミング入門
変化を受け入れる
第2版
Extreme Programming Explained
ケント・ベック著／シンシア・アンドレス協力
長瀬嘉秀 監訳／株式会社テクノロジックアート 訳
PEARSON Education Japan
ピアソン・エデュケーション

References(2)

- **Extreme Programming Installed**
  - Author：Ronald Jeffries, Ann Anderson, Chet Hendrickson
- **Extreme Programming in Practice**
  - Author ：James W. Newkirk, Robert C. Martin
- **Extreme Programming Examined**
  - Editor：Giancarlo Succi, Michele Marchesi

References(3)

- **Extreme Programming Applied: Playing to Win**
  - Author：Ken Auer, Roy Miller
- **Agile Software Development: The Cooperative Game**
  - Author：Alistair Cockburn
- **Agile Software Development with SCRUM**
  - Author：Ken Schwaber, Mike Beedle

References (4)

- **eXtreme Programming testing methods (in Japanese)**
  - Author：XPJUG, Supervisor: Yoshihide Nagase
  - Publisher: Shoei Sha
- **Refactoring: Improving the Design of Existing Code**
  - Author：Martin Fowler
- **The Pragmatic Programmer : From Journeyman to Master**
  - Author：Andrew Hunt, David Thomas

Introduction of Agile

Technologic Arts Incorporated

Contact
E-mail: training@tech-arts.co.jp