

Analysis of the interaction between practices for introducing XP effectively

Osamu Kobayashi

SRA (Software Research Associates)

Mitsuyoshi Kawabata

Agileware

Makoto Sakai

SRA Key Technology Laboratory

Eddy Parkinson

*Department of Computer Science,
Osaka University*

Agenda

- Background
- Analysis Methods
- Case Studies
- Results
- Discussions
- Summary

Background: How we began

- First meeting of we authors:
 - At a small forum of SEA (Software Engineer's Associates) in Osaka.
 - The forum's Subjects: eXtreme Programming
- SEA (Software Engineer's Associates)
 - An organization of active software engineers
 - Engineers of different background and from different organizations gather and discuss about technologies and process.

3

Background: How we began

- Background of authors:
 - Kawabata:
 - a very young, independent programmer
 - Convinced of values of XP.
 - Sakai:
 - an experienced researcher, has wide knowledge and interests
 - has critical points of views (for everything)
 - Kobayashi:
 - a working developer (a little bit old fashioned ?)
 - is interested in new approaches but hesitate to adopt them
 - Parkinson:
 - a researcher from England
 - has a free point of views for this work

4

Background : XP (eXtreme Programming)

- What is XP ?
 - eXtreme Programming
 - Four Values
 - Communication
 - Simple
 - Feedback
 - Courage
- Why XP ?
 - Adapt to change, flexibly
 - Agile development, avoiding waste
 - High Quality, maintainability, productivity

5

XP Practices

- What are the Practices?
 - Practices for to realize the four values
 - Whole Team
 - Planning Game
 - Small Releases
 - Customer Tests
 - Simple Designs
 - Pair Programming
 - Test Driven Development
 - Refactoring
 - Continuous Integration
 - Collective Code Ownership
 - Coding Standards
 - Metaphor
 - Sustainable Pace

Often, we fail because we cannot use all of these practices.

6

Using a subset of practices

- Why can't we use all practices?
 - External problems
 - Customers refuse *E.g. Planning Game, Customer Test*
 - Conventional Contracts *E.g. Small Releases*
 - Distributed development across sub-contractors *E.g. Whole Team*
 - Lack of resources
 - Experts *E.g. Refactoring*
 - Development environments *E.g. Test Driven Development*
 - Champions *E.g. Pair Programming*

7

Problems of using not all practices

- Not Effective Enough
 - Concentration is needed for “Pair Programming”, thus, we have to keep to a “Sustainable Pace”.
 - To keep a “Sustainable Pace”, we have to drive plans using the “Planning Game”
- We have to understand interactions between practices.

We identify interactions between practices based on case studies.

8

Analysis Methods

1. Interview project members
First, with free form answers to questions.
2. Define notations for the results of interviews
We defined graphical notations.
3. Interview again, model interactions
This time, using defined notations.
4. Analyze interactions between practices
Categorized effects of practices, analyzed dependencies.

9

Case Study1 Web site workload analyzer

| Item | Data | Practices | Adoption |
|----------------------|--|---------------------------|----------|
| Team | 1 Manager 3 Developers 2 Users | Whole Team | |
| | | Planning Game | |
| | | Small Releases | |
| | | Customer Tests | |
| Period | 5 Months (XP was used during last 3 months) | Simple Design | |
| | | Pair Programming | |
| | | Test Driven Development | |
| Language | C#, Stored Procedure | Refactoring | |
| | | Continuous Integration | |
| Number of Releases | 3 | Collective Code Ownership | |
| | | Coding Standards | |
| Number of Iterations | 4 | Metaphor | x |
| | | Sustainable Pace | |

10

Case Study1 Web site workload analyzer

- Practices which were adopted partially:
 - Whole team
 - We could not keep customers on site.
 - The manager and developers worked together
 - Planning game
 - Without customers on site, the manager took the role of the customer in planning games.
- Practices which were not adopted:
 - Metaphor
 - The architecture of the system was simple, so we did not need metaphor to understand it.

11

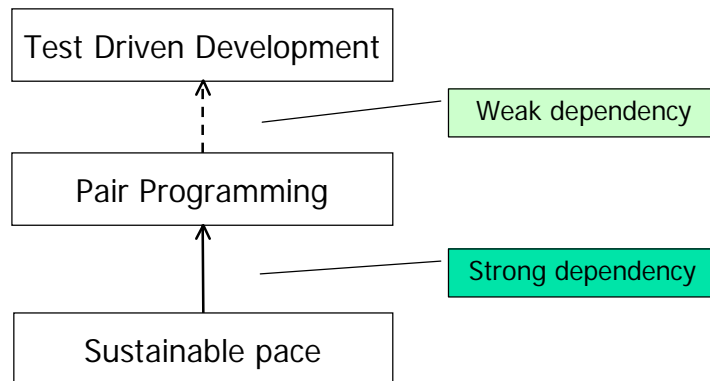
Results of Interviews (Examples)

- Pair programming and sustainable pace
 - Because of schedule delay they had to work at an unsustainable pace.
 - But then, they got tired and could not think well in the morning.
 - Without healthy concentration, pair programming lost productivity.
- Pair programming and productivity and quality
 - “Solo programming” introduced bugs and misunderstandings of specifications, resulting low productivity.
 - Pair programming is a “real time code review”. It raised the quality of code from the beginning and minimized rework, thus resulting high productivity.

12

Notations

- We describe interactions between practices using 2 types of relationship.



13

Notations

- Strong dependency
 - Denoted by solid line arrows \longrightarrow
 - $A \longrightarrow B$ means:
Without A, B has little or no impact.
- Weak dependency
 - Denoted by dotted line arrows $- - \longrightarrow$
 - $A - - \longrightarrow B$ means:
By adopting A at the same time as B, B is more effective. (Without A, B is still partially effective.)

14

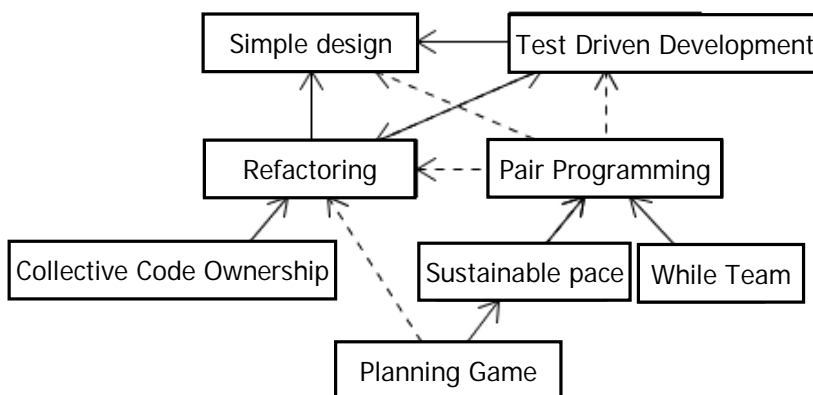
Second interviews and modeling

- We categorized the results of second interviews into 5 quality categories.
 - Communications
 - Productivity
 - Quality of Code
 - Quality of requirement
 - Maintainability

15

Interactions between practices

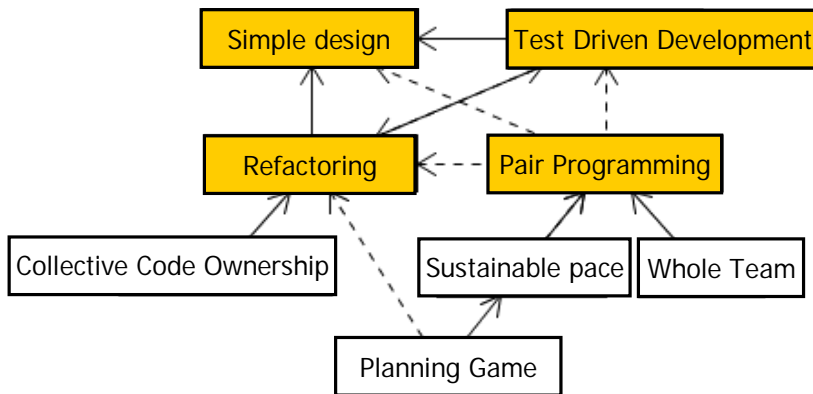
- Practices in “Productivity” category



16

Interactions between practices

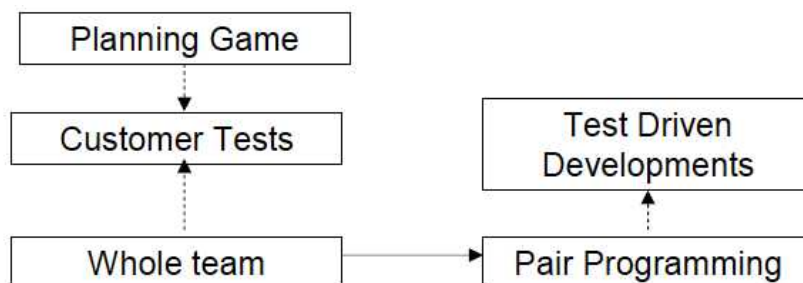
- Practices in “Productivity” category



17

Interactions between practices

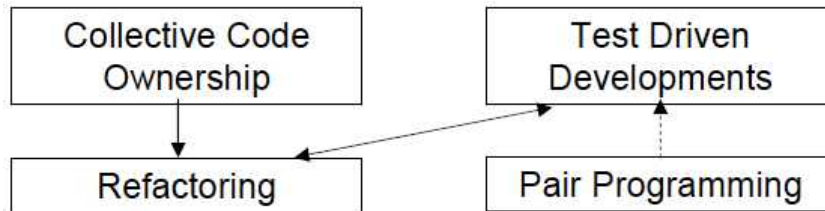
- Practices in “Communication” category



18

Interactions between practices

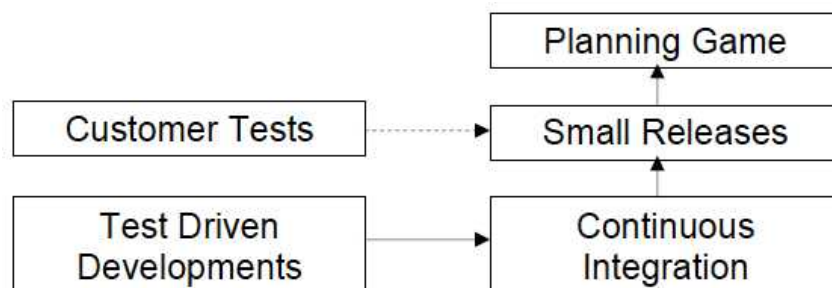
- Practices in “Quality of Code” category



19

Interactions between practices

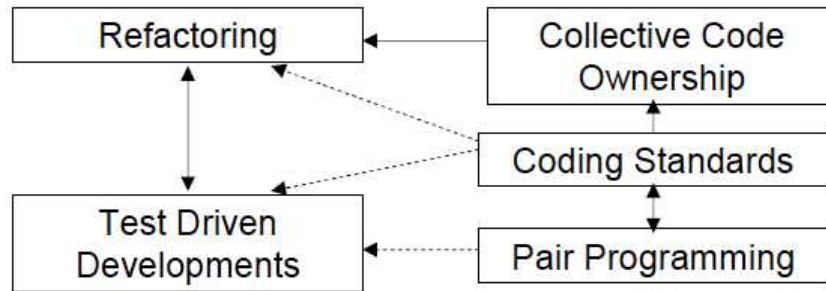
- Practices in “Quality of Requirement” category



20

Interactions between practices

- Practices in “Maintenance” category



21

Discussions (Effects)

| | Effects | communication | productivity | quality of code | quality of requirements | ease of maintenance | dependency | contribution | ↔ | ← | → | ←-- | →-- |
|-------------------------------|---------|---------------|--------------|-----------------|-------------------------|---------------------|------------|--------------|----|-------|----------|-----|----------|
| Small Releases(SR) | 1 | | | | v | | 1.5 | 1 | | CI | PG | | CT |
| Continuous Integration(CI) | 1 | | | | | v | 1 | 1 | | TD | SR | | |
| Customer Tests(CT) | 2 | v | | | v | | 1 | 0.5 | | | | WT | PG SR |
| Planning Game(PG) | 3 | v | v | | v | | 1 | 1.5 | | SR | SP | | CT |
| Simple Design(SD) | 1 | | v | | | | 2.5 | 0 | | TD R | | PP | |
| Sustainable Pace(SP) | 1 | | v | | | | 1 | 1 | | PG | PP | | |
| Coding Standards(CS) | 1 | | | | | v | 1 | 2 | PP | | CC | | R TD |
| Whole team(WT) | 2 | v | v | | | | 0 | 1.5 | | | PP | | CT |
| Refactoring(R) | 3 | v | v | v | v | | 3.5 | 2 | TD | CC | SD | | CS PP PG |
| Collective Code Ownership(CC) | 3 | v | v | v | | | 2.5 | 1 | | CS | R | | CS |
| Test Driven Developments(TD) | 4 | v | v | v | v | | 1.5 | 4 | R | | SD CC CI | PP | |
| Pair Programming(PP) | 4 | v | v | v | v | | 3 | 2.5 | CS | SP WT | | | SD R TD |

22

Discussions (Effects)

We can identify subsets of practices for each of the quality categories.

| | Effects | communication | productivity | quality of code | quality of requirements | ease of maintenance |
|-------------------------------|---------|---------------|--------------|-----------------|-------------------------|---------------------|
| Small Releases(SR) | 1 | | | v | | |
| Continuous Integration(CI) | 1 | | | | v | |
| Customer Tests(CT) | 2 | v | | | v | |
| Planning Game(PG) | 3 | v | v | | v | |
| Simple Design(SD) | 1 | | v | | | |
| Sustainable Pace(SP) | 1 | | v | | | |
| Coding Standards(CS) | 1 | | | | | v |
| Whole team(WT) | 2 | v | v | | | |
| Refactoring(R) | 3 | | v | v | | v |
| Collective Code Ownership(CC) | 3 | | v | v | | v |
| Test Driven Developments(TD) | 4 | v | v | v | v | v |
| Pair Programming(PP) | 4 | v | v | v | v | v |

23

Discussions (Effects)

| | Effects | dependency | contribution | ↔ | ← | → | ←-- | --→ |
|-------------------------------|---------|------------|--------------|---|----------|-------------|-----|---------|
| Small Releases(SR) | 1 | 1.5 | 1 | | CI | PG | CT | |
| Continuous Integration(CI) | 1 | 1 | 1 | | TD | SR | | |
| Customer Tests(CT) | 2 | 1 | 0.5 | | | | WT | PG SR |
| Planning Game(PG) | 3 | 1 | 1.5 | | SR | SP | | CT |
| Simple Design(SD) | 1 | 2.5 | 0 | | TD R | | PP | |
| Sustainable Pace(SP) | 1 | 1 | 1 | | PG | PP | | |
| Coding Standards(CS) | 1 | 1 | 2 | | PP | CC | | TD |
| Whole team(WT) | 2 | 0 | 1.5 | | | PP | | |
| Refactoring(R) | 3 | 3.5 | 2 | | TD CC | SD | | |
| Collective Code Ownership(CC) | 3 | 2.5 | 1 | | CS | R | CS | |
| Test Driven Developments(TD) | 4 | 1.5 | 4 | | R | SD CC CI PP | | |
| Pair Programming(PP) | 4 | 3 | 2.5 | | CS SP WT | | | SD R TD |

Easy to introduce by itself, highly effective

24

Discussions (Effects)

| | Effects | dependency | contribution | ↔ | ← | → | ←-- | --> |
|-------------------------------|---------|------------|--------------|----|------|----|-----|---------|
| Small Releases(SR) | 1 | 1.5 | 1 | | CI | PG | | CT |
| Continuous Integration(CI) | 1 | 1 | 1 | | TD | SR | | |
| Customer Tests(CT) | 2 | 1 | 0.5 | | | | WT | PG SR |
| Planning Game(PG) | 3 | 1 | 1.5 | | SR | SP | | CT |
| Simple Design(SD) | 1 | 2.5 | 0 | | TD R | | | |
| Sustainable Pace(SP) | 1 | 1 | 1 | | PG | | | |
| Coding Standards(CS) | 1 | 1 | 2 | PP | | | | R TD |
| Whole team(WT) | 2 | 0 | 1.5 | | | PP | | CT |
| Refactoring(R) | 3 | 3.5 | 2 | TD | CC | SD | CS | PP PG |
| Collective Code Ownership(CC) | 3 | 2.5 | 1 | | CS | R | CS | |
| Test Driven Developments(TD) | 4 | 1.5 | 4 | R | | SD | CC | CI PP |
| Pair Programming(PP) | 4 | 3 | 2.5 | CS | SP | WT | | SD R TD |

Difficult to introduce, but highly effective

25

Discussions (Effects)

| | Effects | dependency | contribution | ↔ | ← | → | ←-- | --> |
|-------------------------------|---------|------------|--------------|----|------|----|-----|---------|
| Small Releases(SR) | 1 | 1.5 | 1 | | CI | PG | | CT |
| Continuous Integration(CI) | 1 | 1 | 1 | | TD | SR | | |
| Customer Tests(CT) | 2 | 1 | 0.5 | | | | WT | PG SR |
| Planning Game(PG) | 3 | 1 | 1.5 | | SR | SP | | CT |
| Simple Design(SD) | 1 | 2.5 | 0 | | TD R | | | PP |
| Sustainable Pace(SP) | 1 | 1 | 1 | | PG | PP | | |
| Coding Standards(CS) | 1 | 1 | 2 | PP | | CC | | R TD |
| Whole team(WT) | 2 | 0 | 1.5 | | | PP | | CT |
| Refactoring(R) | 3 | 3.5 | 2 | TD | CC | SD | CS | PP PG |
| Collective Code Ownership(CC) | 3 | 2.5 | 1 | | CS | R | CS | |
| Test Driven Developments(TD) | 4 | 1.5 | 4 | R | | SD | CC | CI PP |
| Pair Programming(PP) | 4 | 3 | 2.5 | CS | SP | WT | | SD R TD |

1

26

Summary

- Model interactions between practices based on case studies.
- Categorized practices and interactions into quality categories.
- Summarized results into graphs and a table.
- Using the results, it looks possible to introduce XP effectively, even when all practices cannot be introduced at once.
- Future directions
 - More case studies.
 - Recognize patterns of practice introduction.

27

Q & A

Thank you for your attendance.
Any question please.