Agile Software Development

Alistair Cockburn

Addison Wesley

Three Levels of Learning

- Learning new skills
  - Following: “one procedure that works”, “at least this thing works”
  - Detaching:”when does it break down?” “learn limits of procedure”, “adapt it”, “when is it appropriate?” Survey paper.
  - Fluent:”irrelevant whether following a particular technique”, ”knowledge has become integrated”.

The Three Levels and Methodology

- Methodology: a series of related methods and techniques (Miriam-Webster)
- Level 1: processes, techniques and standards in detail. Detailed templates in RUP servel level 1 audience.

Three Levels of Methodology

- Level 2/3: The Pragmatic Programmer: identifies techniques that a practitioner uses. A useful library of ideas but the beginner finds it lacking specific rules.
- Avoid level mixup! It confuses.

Shu-Ha-Ri

- Three levels are known in other skill areas: Aikido (self defense technique)
- Shu: learn. Build technical foundation for the art. Single instructor.
- Ha: detach. Understand meaning and purpose; not just repetitive practice.
- Ri: transcend. Practitioner; original thoughts

A Cooperative Game of Invention and Communication

- A fruitful way to think about software development.
- Games used by mathematicians and corporate strategists.
- Kinds of games: zero-sum, positional, competitive, cooperative, finite, …

Software Development

- Group game
- Non-zero-sum: multiple winners and losers.
- Cooperative
- Goal-seeking
- Finite

Infinite Games

- Infinite games: organizations, corporations and countries, a person's profession.
- Do well in one game to be well positioned for the next one.

Software and Rock Climbing

- Best comparison partner
- Cooperative and goal seeking
  - How well they climbed together
  - How much they enjoyed themselves
  - Reach the top?
- Load bearing
  - Climbers must support their weight. Software must run.

Software and Rock Climbing

- Team
- Individuals with talent
- Skill sensitive
- Training
- Tools
- Resource-limited: before nightfall or the weather changes.

Software and Rock Climbing

- Plan
- Improvised

A Game of Invention and Communication

- Software development: group game which is goal seeking, finite and cooperative
- Team: sponsor, manager, usage specialists, designers, testers and writers
- Next game: maintenance, build an entirely different system

Cooperative Game of Invention and Communication

- Measure of quality as a team: how well they cooperate and communicate during game.
- What are the moves of the game:
  - There is nothing in the game but people’s ideas and the communication of those ideas to their colleagues (including the sponsor) and to the computer.

Emotions, wishes and thoughts

- The task facing the developers:
  - They are working on a problem they don’t fully understand and that lives in emotions, wishes and thoughts and that changes as they proceed.
  - They need to understand.
    - Problem space.
    - Imagine some mechanism in a viable technology space.
    - Express in an executable language which lacks many features of expression to a system that is unforgiving of mistakes.

What is software development?

- Software Development is a resource-limited) cooperative game of invention and communication.
  - The primary goal of the game is to deliver useful, working software.
  - The secondary goal of the game is to set up for the next game. The next game may be to alter or replace the system or to create a neighboring system.

Not many people have articulated this before

Software and Engineering

- Considering software development as a game with moves is profitable.
  - Gives us a way to make meaningful decisions on a project.
- In contrast: speaking of software development as engineering or model building does not help.

Engineering

- People mostly use engineering to create a sense of guilt for not having done enough of something, without being clear of what that something is.
- Dictionary: The application of science and mathematics by which the properties of matter and the sources of energy in nature are made useful to man (Webster's Dic.).

What is “doing engineering”

- In my experience: involves creating a trade-off solution in the face of conflicting demands.
- Also applies to software development.

Confusing act and outcome

- Outcome: The factory, which is run while specific people watch carefully for variations in quantity and quality of the items being manufactured.
- Act: ill-defined creative process the industrial engineer goes through to invent the manufacturing plant.

More like Engineering?

- When people say: “Make software development more like engineering” they often mean, “Make it more like running a plant, with statistical quality control”.
- But: running the plant is not the act of doing engineering.

Look up previous solutions

- The other part of “doing engineering”
- Civil engineers are not supposed to invent new structures.
  - Take soil samples and use the code books to look for the simplest structure that handles the required load over the given distance building on the soil at hand.
  - Centuries of tabulation of known solutions

Fits marginally

- This only fits marginally the current state of software development
- We are still in the stage where there is competition between designs.
- Technologies are changing fast that few code books exist
- Today there are more variations between systems than there are commonalities.

Return

- Return to consider engineering as thinking and making trade-offs.

Software and Model Building

- Ivar Jacobson: “software development is model building”
- Leads to inappropriate project decisions

Interesting part not in models

- If software development were model building, then the valid measure of the quality of the software or of the development process would be the quality of the models (fidelity, completeness)

But successful project teams say

- The interesting part of what we want to express doesn’t get captured in those models. The interesting part is what we say to each other while drawing on the board.
- We don’t have time to create fancy or complete models
- Paying attention to the models interfered with developing the software

Sufficiency

- The work products of the team should be measured for sufficiency with respect to communicating with the target group.
- It does not matter if incomplete, incorrect syntax, … if they communicate sufficiently to the recipients.

Modeling as team communication

- Can be too much or too little.
- How much modeling to do? Subject of this book.

Programmers as Communications Specialists

- Game of communication: different light on programmers …
- Stereotyped as noncommunicative individuals who like to sit in darkened rooms
- High acceptance of programming in pairs … Programmers thought they would not like it but they like it! (Extreme Programming)

Game of invention

- So far not as a game of communication
- Interest of programmers to discuss programming matters gets in the way of them discussing business matters with sponsors, users and business experts.

Universities

- Can reverse the general characteristics by creating software development curricula that contain more communication-intensive courses
- Attracts different students (University of Aalborg, Denmark).

Gaming Faster

- We should not expect orders of magnitude improvement in program production.
- As much as programming languages may improve, programminvg will still be limited by our ability to think through the problem and the solution.

Analogy

- Two other fields of thought expression
  - Writing novels
  - Writing laws: Lawyers won’t get exponentially faster at creating contracts and laws!

Diminishing Returns

- Because a software development project is resource limited, spending extra to make an intermediate work product better than it needs to be for its purpose is wasteful.

What is software development?

- Software Development is a resource-limited) cooperative game of invention and communication.
  - The primary goal of the game is to deliver useful, working software.
  - The secondary goal of the game is to set up for the next game. The next game may be to alter or replace the system or to create a neighboring system.

Not many people have articulated this before

Peter Naur

Programming as Theory Building

From Computing: A Human Activity

(1992, ACM Press)

What goes on in software development? Intro.

- Most accurate account
- Quality is related to the match between the theory of the problem and the theory of the solution.
- The designer’s job is not to pass along the design but the theories that drive the design.

What is programming

- Should be regarded as an activity by which the programmers form or achieve a certain kind of insight, a theory, of the matters at hand.
- Not as a production of a program and other texts.

Programming and the Programmer’s Knowledge

- Programming = the whole activity of design and implementation
- Programming = building up knowledge
- What kind of knowledge?
- A theory: a person who has or posses a theory knows how to do certain things and can support the actual doing with explanations, justifications and answers to queries.

Theory transcends documentation in at least 3 essential ways

- How are affairs of the world mapped into the program text? For any aspect of the world the programmer can state its manner of mapping into the program text. [AOSD]
- Can support the program text with some justification.
- Is able to respond constructively to any demand for a modification. Similarity of new demand to similarities already in the system.

Problems and Costs of Program Modifications

- Cost savings by modifying existing program rather starting from scratch.
- Cheaper? Not supported by other complicated man-made constructions: bridges, buildings, etc. Often demolish and rebuild is most economical.
- Program modification is just text editing?

Program flexibility

- Build into the program operation facilities that are not immediately demanded.
- May be expensive.
  - AOSD:
    - extend program by addition not modification.
    - Works best if program is very systematically organized (easier to write pointcuts).

Similarity

- Similarity:
  - Requirements for existing solution
  - Requirements for new demands
- To see the similarities we need to understand the "theory" behind the existing solution.
- Person having the theory must already be prepared to respond to questions that give rise to program modifications (theory stays the same).

Decay

- Decay of program text if people are making modifications without understanding theory behind the program.
- We want theory conforming modifications to the program text. Otherwise we get unintegrated patches.

Life cycle of a program

- Birth: building of theory.
- Life: programmer team possessing theory remains in active control of the program.
- Death: programmer team is dissolved.
- Revival: rebuilding of its theory by a new programmer team.
- New programmers need to work in close contact with programmers who have theory.
- Start from scratch?

Method and theory building

- Method
  - Set of work rules
  - Which notations/languages
  - Documents to produce
- Theory cannot be expressed
  - No right method
- Contradiction?

Software Development

- Should be based on scientific manners?
  - Are scientific methods helpful to scientists? Debatable.
  - Not contradicted by such works as Polya’s on problem solving (How to solve it and Patterns of Plausible Inference).
    - Does not present a method on how to proceed.
    - A collection of suggestions aiming at stimulating the mental activity of the problem solver.
    - Highly relevant to programming.

Dismissal of method

- Have methods been successful?
- Controlled experiments would be very expensive.
- AOSD
  - Is it a method? Yes, e.g. combined with Extreme Programming.
  - Do we need a controlled experiment? No!