Lean Software Development
Agile Methods
Lectures 9 and 10

Artem Kruglov, Giancarlo Succi

Innopolis University

a.kruglov@innopolis.ru, *g.succi@innopolis.ru*

16th and 17th February 2021

Outline

- **Agile manifesto**
- Keeping the Process Under Control
- Job Enrichment
- To XP or not to XP
- XP
- Let‘s work on differences
- Control model
- Coordination mechanisms

Agile Manifesto

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.

Agile Manifesto – Exercise

Inside the class workbook, please create for every pair of statements on example of when you selected the approach on the left and when you selected the approach on the right, and provide a short explanation why.

Agile Manifesto

The "Agile Manifesto" identifies two sets of values: the values lying on the left of the document and the values lying on the right of the document.

Values on the left	Values on the right
Individual and interactions	Processes and tools
Working software	Comprehensive documentation
Customer collaboration	Contract negotiation
Responding to change	Following a plan

Agile Manifesto I

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software
- Welcome changing requirements, even late in development, Agile processes harness change for the customer's advantage
- Business people and developers must work together daily throughout the project

Agile Manifesto II

- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale
- Build projects around motivated individuals
- Give them the environment and support they need, and trust them to get the job done

Agile Manifesto III

- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation
- Working software is the primary measure of progress

Agile Manifesto IV

- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely
- Continuous attention to technical excellence and good design enhances agility (continued)

Agile Manifesto V

- Simplicity – the art of maximizing the amount of work not done – is essential
- The best architectures, requirements, and designs emerge from self-organizing teams
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

Ideas related to principles in the Agile Manifesto

Year	Principle	Idea
1954	11	Management by objectives [16]
1968	5	Job enrichment [23]
1970s	3	Rapid application development [28]
1971	6	Communication improvement by collocation of workers [35]
1975	4	Involvement of stakeholders [26]
1978	7–12	Toyota production system [30]
1986	12	Plan-Do-Study-Act [15, 34]
1988	3	Prototyping and iterative development [9]

Software best practices are not new

Best practice	Year	Introduced by
Requirements	0s	(Exists since the beginning of time)
Pair programming	1950s	John Von Neumann (IBM)
Project planning	1960s	Mercury project (NASA)
Risk management	1960s	Mercury project (NASA)
Software architecture	1960s	Frederick P. Brooks, Edsger W. Dijkstra, David L. Parnas
Software reuse	1960s	Malcolm D. McIlroy (AT&T)
Test-driven design	1960s	Mercury project (NASA)
Coding standards	1970s	Brian W. Kernighan, P. J. Plauger
Collective ownership	1970s	Unix, open source
Continuous integration	1970s	IBM federal systems division
Data hiding and abstraction	1970s	David L. Parnas
Documentation	1970s	David L. Parnas
Incremental releases	1970s	Victor R. Basili, Albert J. Turner

Software best practices are not new

Best practice	Year	Introduced by
On-site customer	1970s	Harlan D. Mills (IBM federal systems division)
Simple design	1970s	Victor R. Basili, Albert J. Turner
Software measurements	1970s	Tom Gilb, Maurice H. Halstead
Evolutionary design	1980s	Tom Gilb
Patterns	1980s	Tom DeMarco, Timothy Lister, Gang of Four
Peopleware, sustainable pace	1980s	Tom DeMarco, Timothy Lister
Use cases	1980s	Ivar H. Jacobson
Software economics & estimation	1980s	Barry W. Boehm
Metaphor	1990s	Kent Beck, Martin Fowler, Howard G. Cunningham
Refactoring	1990s	William F. Opdyke, Martin Fowler
Retrospectives	1990s	Norman L. Kerth, Linda Rising

Outline

- Agile manifesto
- **Keeping the Process Under Control**
- Job Enrichment
- To XP or not to XP
- XP
- Let‘s work on differences
- Control model
- Coordination mechanisms

Keeping the Process Under Control

Michael L. Harris proposed an effective way to classify different types of control and in which circumstances these types work best.

The selection of the most effective control mechanism depends on two factors:

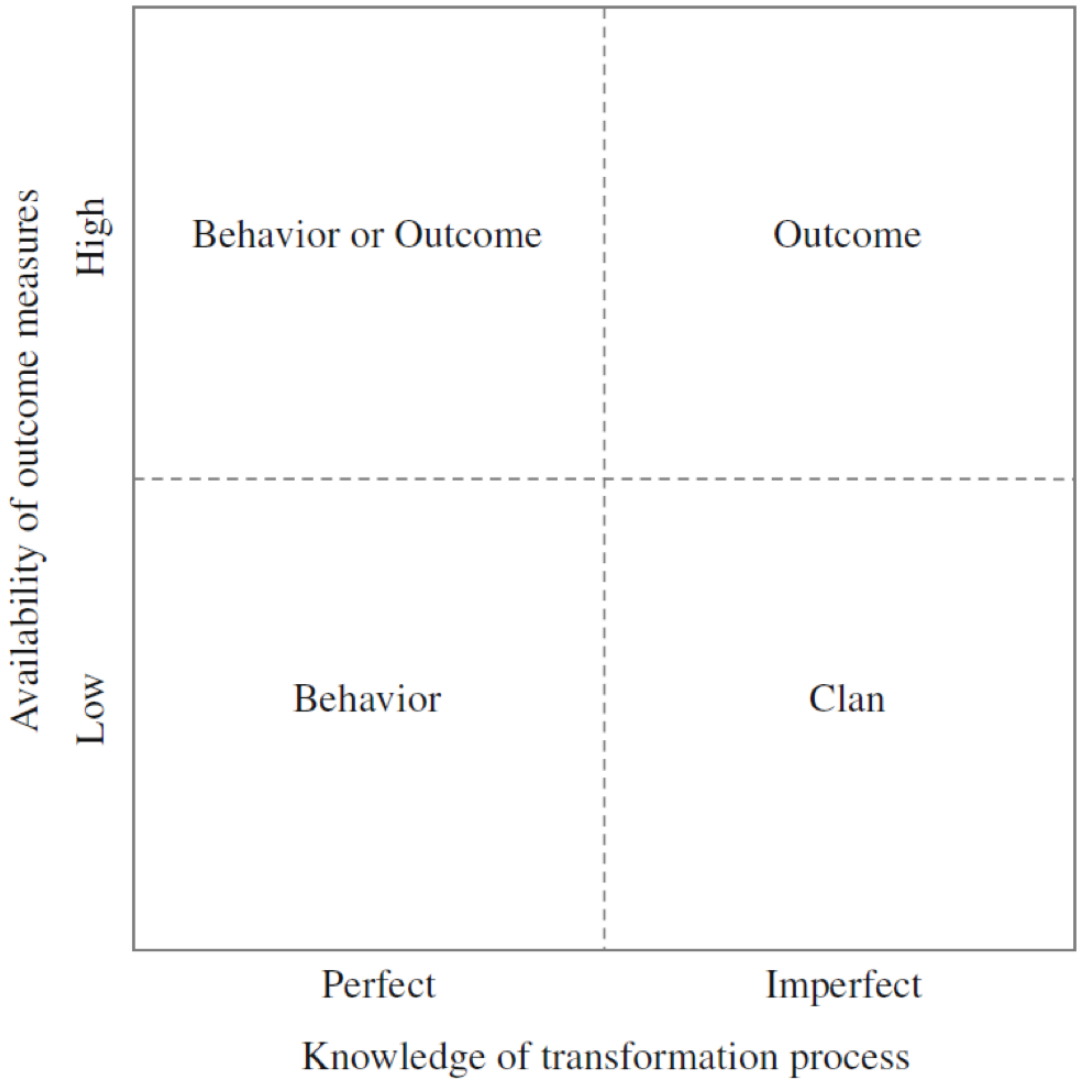
- the ability to measure the output, the "measurability," and
- the ability to specify in details the steps required to accomplish a given task, the "specifiability."

Keeping the Process Under Control

So if we put these two factors in a two-dimensional diagram, we obtain four areas:

- an area with high level of measurability and of specifiability;
- an area where we have a high level of measurability of the output but little specifiability;
- an area where we have a high level of specifiability but little measurability; and
- an area where we can neither measure directly the output nor specify the steps of the work.

Keeping the Process Under Control



Outline

- Agile manifesto
- Keeping the Process Under Control
- **Job Enrichment**
- To XP or not to XP
- XP
- Let‘s work on differences
- Control model
- Coordination mechanisms

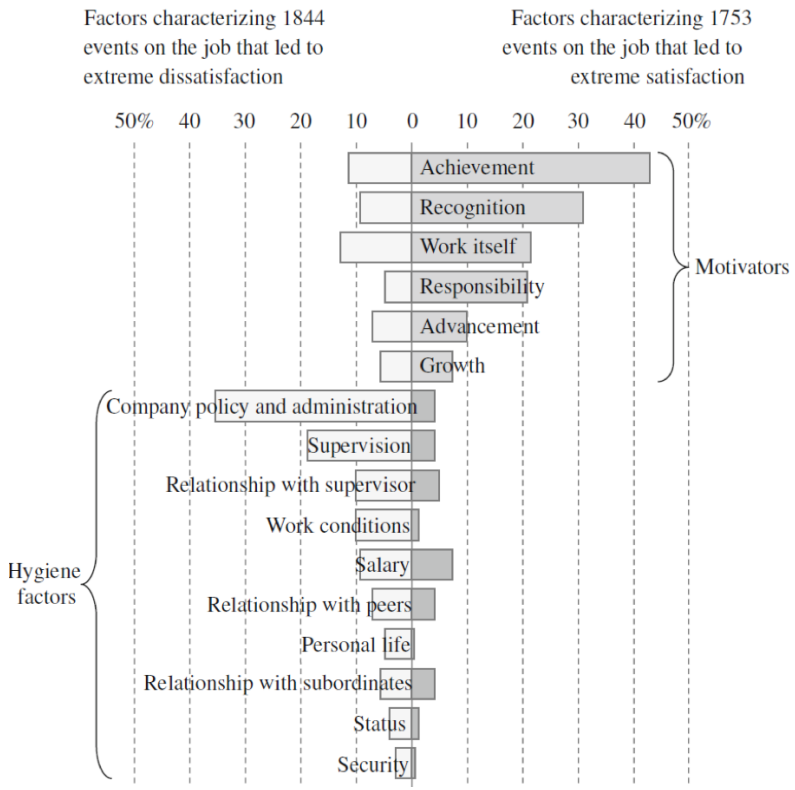
Job Enrichment

Job Enrichment can be an effective mechanism to promote an effective process control, especially if an approach based on Clan Control is adopted.

Techniques	Motivators involved
Removing some controls while retaining accountability	Responsibility and personal achievement
Increasing the accountability of individuals for own work	Responsibility and recognition
Giving a person a complete natural unit of work (module, division, area, and so on)	Responsibility, achievement, and recognition
Granting additional authority to employees in their activity; job freedom	Responsibility, achievement, and recognition
Making periodic reports directly available to the workers themselves rather than to supervisors	Internal recognition
Introducing new and more difficult tasks not previously handled	Growth and learning
Assigning individuals specific or specialized tasks, enabling them to become experts	Responsibility, growth, and advancement

Job Enrichment

Factors affecting job attitudes



Job Enrichment

Herzberg claims that it is of paramount importance to increase the accountability and responsibility of employees, so that they themselves become motivated and interested that their work has a high-quality outcome.
Herzberg's recommendations:

- to remove some controls while retaining accountability reduces a hygiene factor (supervision) and increases motivator factors (responsibility and personal achievement);
- to increase the accountability of individuals for own work reduces hygiene factors (supervision, security; more accountability is connected to a higher risk to fail) and increases a motivator factor (responsibility and recognition);

Job Enrichment

- to give a person a complete natural unit of work reduces hygiene factors (company policy and administration, supervision) and increases motivator factors (responsibility, achievement, and recognition);
- to grant additional authority to employees in their activity reduces hygiene factors (company policies, supervision) and increases motivator factors (responsibility, achievement, and recognition);
- to make periodic reports directly available to the works themselves rather than to supervisors reduces a hygiene factor (supervision) and increases a motivator factor (internal recognition);

Job Enrichment

- to introduce new and more difficult tasks not previously handled reduces a hygiene factor (company policies and administration) and increases motivator factors (growth and learning); and
- to assign individuals specific or specialized tasks, enabling them to become experts, increases motivator factors (responsibility, growth, and advancement).

Outline

- Agile manifesto
- Keeping the Process Under Control
- Job Enrichment
- **To XP or not to XP**
- XP
- Let‘s work on differences
- Control model
- Coordination mechanisms

Polls

- Who likes to spend days in discussing analysis and design choices and writing analysis and design documents?
- Who likes Z?
- Who likes to write extensive and carefully checked documentation?
- Who likes to attend long and detailed planned strategic meetings?
- Who likes to jump onto the code, just after hearing the first line of requirements?
- Who likes Java or Smalltalk or C++? (Inclusive OR _)
- Who likes to let a grad student do the documentation?
- Who likes to have short, effective meetings?

What if ...

- We could jump onto code without writing analysis and design documents? *Aren't they useless, anyway?*
- We could do without need of documentation? *Who reads documentation?*
- We could reduce to the minimum the time devoted to meetings? *Why get many involved when one can do a better job, even with a higher consensus?*

Wouldn‘t it be a nice world?

“This is XP!!”

What is XP?

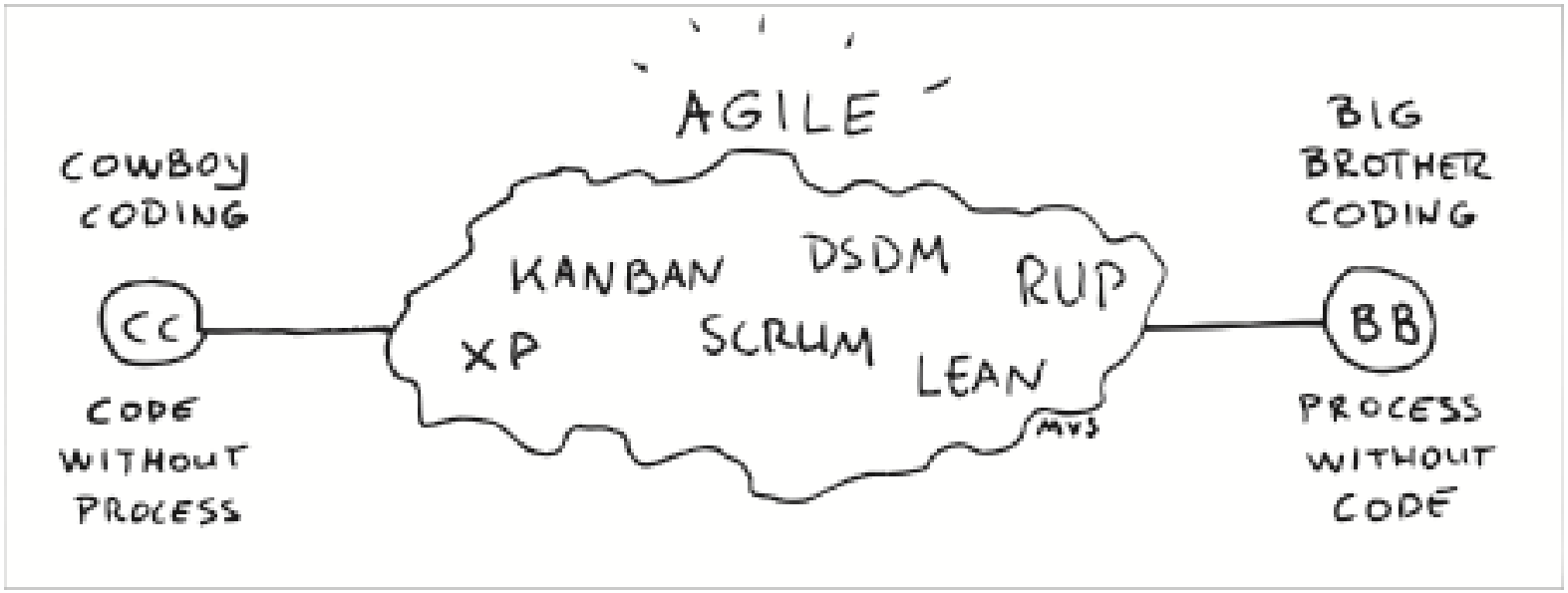
You tell me what you think it is ... I write

- 1: Agile methodology
- 2: Pair programming
- 3: No code ownership
- 4: Customer involvement
- 5: Continuous integration
- 6: Test first

The sad story is...

This is not XP, it is Cowboy Coding

- XP – and Agile Methodologies, are outrageously difficult to do, and even more difficult to manage
- Today, I am going to talk to you on why they are so difficult and why they have a chance to succeed anyway



Outline

- Agile manifesto
- Keeping the Process Under Control
- Job Enrichment
- To XP or not to XP
- **XP**
- Let‘s work on differences
- Control model
- Coordination mechanisms

XP – eXtreme Programming

- Probably, the most "well-heard-of" agile methodology is eXtreme Programming, a.k.a. XP
- XP defines in details a sequence of development principles and of development practices

Values and drivers of XP (1)

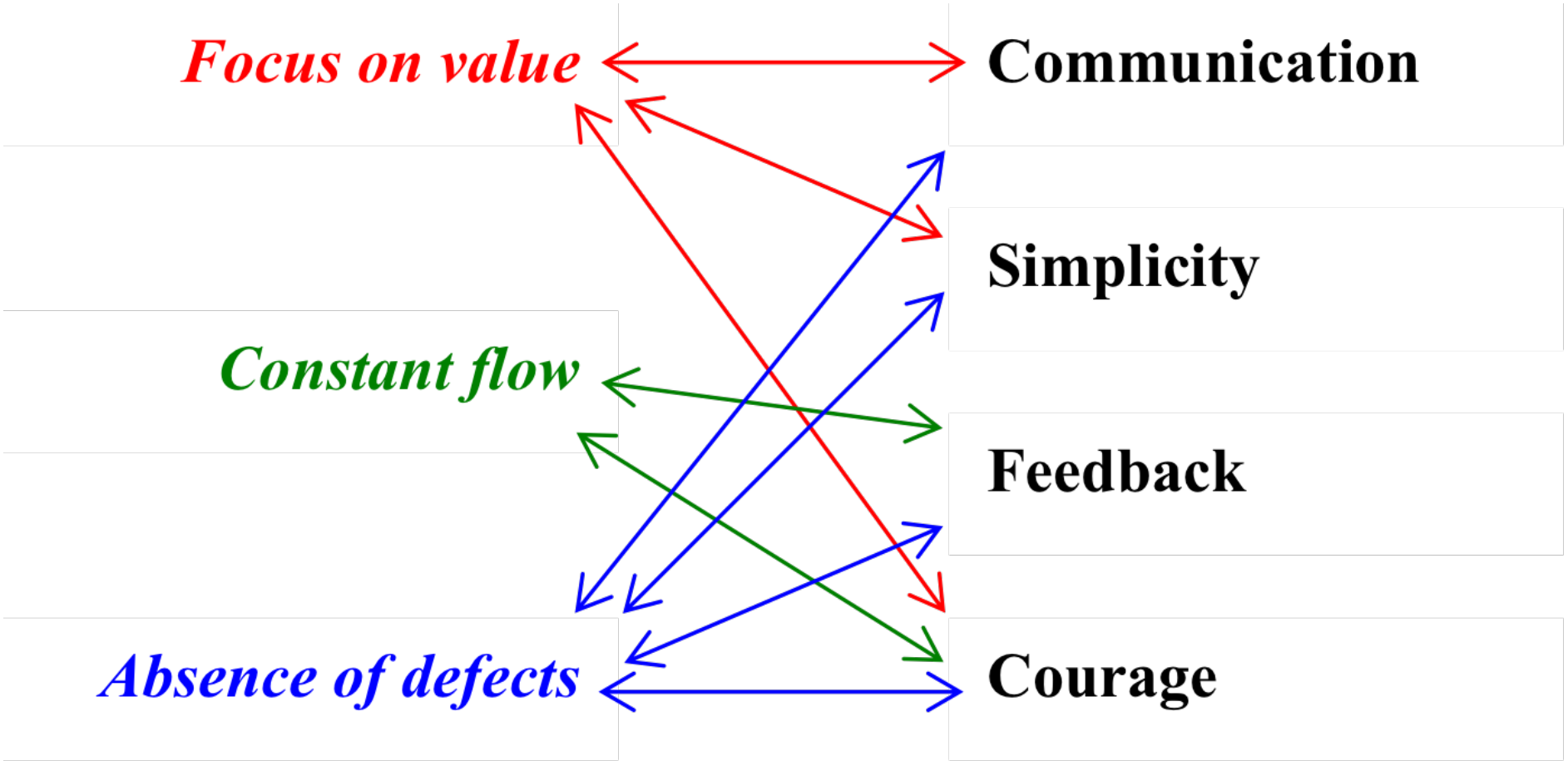
4 Values:

- Simplicity
- Communication
- Feedback
- Courage

3 Drivers:

- Focus on value
- Constant flow of activities
- No defects

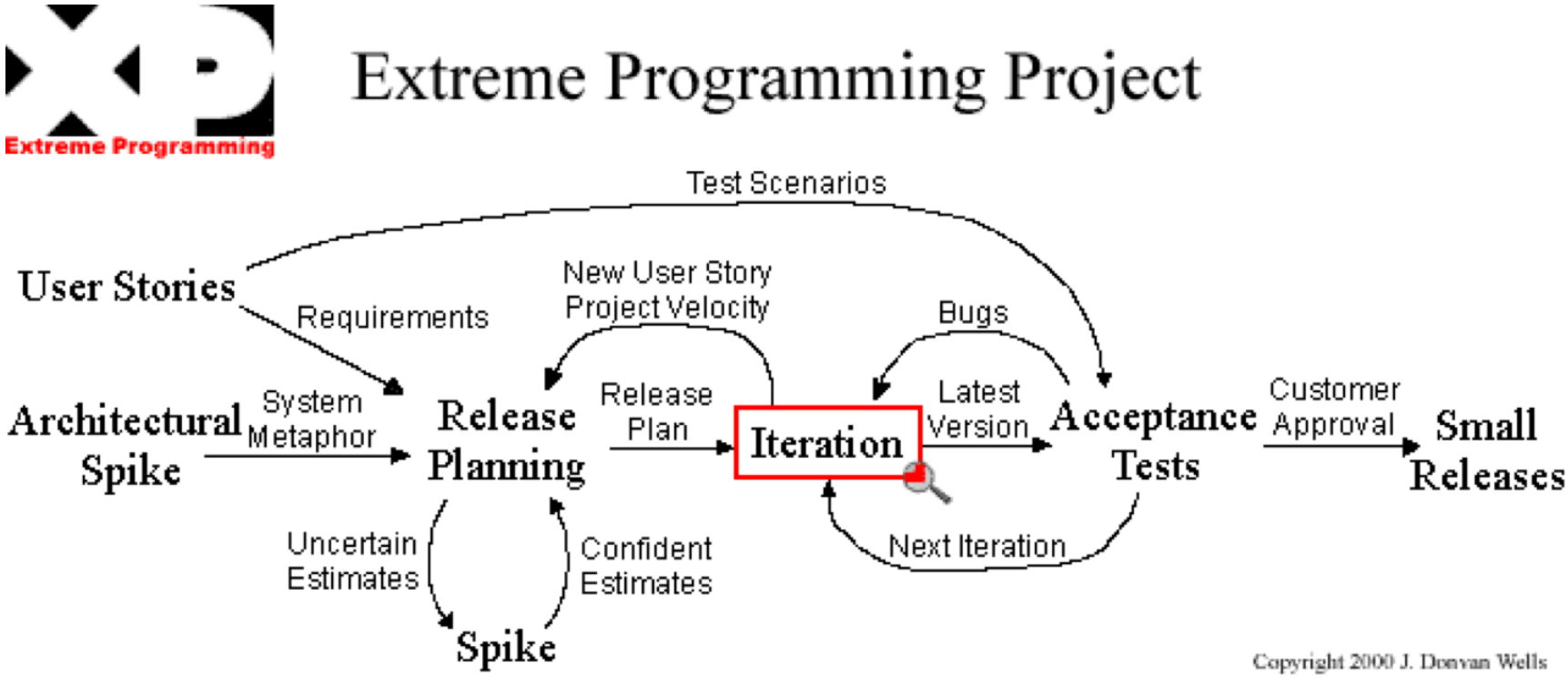
Values and drivers of XP (2)



XP practices

- Planning game
- Short releases
- Metaphor
- Simple design
- Test-driven development
- Refactoring
- Pair programming
- Collective code ownership
- Continuous integration
- 40 hours working week
- On-site customer
- Coding standards

Structure of an XP project



From Don Wells' site http://www.extremeprogramming.org/

Outline

- Agile manifesto
- Keeping the Process Under Control
- Job Enrichment
- To XP or not to XP
- XP
- **Let‘s work on differences**
- Control model
- Coordination mechanisms

Let's get a closer look (1)

- The traditional software engineer (TSE) says: "I need to have completed analysis and design before proceeding to code"
- The cowboy coder (CBC) says: "I do not need any analysis and design"
- The XPer (XP) says: "I do not need to have completed analysis and design before proceeding to code"

Let‘s get a closer look (2)

- TSE says: “I need to write all the documentation in a complete and pervasive way so that people in the future will be able to understand what is in here.”
- CBC says: “I do not need any documentation.”
- XP says: “I need to write the code so that people in the future will be able to understand what is in here. I need to write only the documentation that is needed by people.”

Let's get a closer look (3)

- TSE says: "Especially close to the deadline, I need to work like crazy to get the project done. Programming is hard."
- CBC says: "Only close to the deadline, I need to work like crazy to get the project done. Programming is fun."
- XP says: "Especially close to the deadline, I need to work no more than 40 hrs a week to get the project done, keeping a constant pace and a fresh mind. Programming is fun."

Let's get a closer look (4)

- TSE says: "The code is mine and none is allowed to touch it!"
- CBC says: "The code is mine and none is allowed to touch it!"
- XP says: "The code is of the team and everyone is allowed to modify it also extensively, provided the tests keep running!"

Let's get a closer look (5)

- TSE says: "At the end we will do integration. It is going to be hard, so we need to define precise interaction protocols, and to document them with maximal details."
- CBC says: "At the end we will do integration! No problem, it's easy: it will take 5'."
- XP says: "We need to integrate our system at least daily, so that at the end we will not have any problem."

Let's get a closer look (6)

- TSE says: "The customer should only see working and cleaned-up versions of the product. It is important to balance the contact with the customer so time is not wasted."
- CBC says: "If possible, the customer should only see the final versions of the product. It is important to minimize the contact with the customer so time is not wasted."
- XP says: "The customer should:
  - (a) be constantly exposed to the product being build and to the development team, and, whenever possible,
  - (b) have a representative on site."

Let's get a closer look (7)

- TSE says: "If it is not broken, do not touch it!"
- CBC says: "Even if it is broken, do not touch it! Try to hide it!"
- XP says: "Even if it is not broken, constantly refactor it! Use the test cases to ensure you do not introduce an undesired bug."

Let's get a closer look (8)

- TSE says: "Plan everything well in advance so that there will be no need to change! A change is a clear symptom of bad and/or not sufficient planning."
- CBC says: "Do not plan anything and try not to change! A change is a clear symptom of an annoying customer or manager."
- XP says: "Plan everything that you can reasonably foresee and get ready to change! Changes occur naturally in software projects."

Let's get a closer look (9)

- TSE says: "Change is evil! Fight it!"
- CBC says: "Change is evil! Fight it!"
- XP says: "Change is human! Get ready to cope with it!"

Think!

- Control “in the process” vs. control “of the process”
- Endogenous control vs. exogenous control

Outline

- Agile manifesto
- Keeping the Process Under Control
- Job Enrichment
- To XP or not to XP
- XP
- Let‘s work on differences
- **Control model**
- Coordination mechanisms

Exogenous or endogenous? (1)

- Put a sign saying “cars higher than 2 meters cannot proceed”
- Tell the students always to bring their student-id to school
- Ask the employees to meet the secretary everyday for news
- Put a bar on the way at the height of 2.05 meters so that higher cars must stop
- Lock the gate of the school with an electronic locker that can be opened only with the student-id
- Locate the desk of the secretary in the hall, right in front of the main entrance so that she can see all the employees

Exogenous or endogenous? (2)

- Write on the board: "Remember to switch on the lights when anyone is in and to switch it off when the room is empty."
- Remind constantly people of the correct layout of the tools in the bags
- Set up a written company policy that requires people to check in their modules at least daily
- Put an infrared sensor-operated light switch to keep the light on only if there is anyone in.
- Organize the bags in pockets each fitting well only one kind of tools
- Define a system-wide log-off procedure that executes a check in automatically, and set an automatic log-off for idle time over 1hr after 6PM

Endogenous Control in the Agile Manifesto I

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software
- Welcome changing requirements, even late in development, Agile processes harness change for the customer‘s advantage
- Business people and developers must work together daily throughout the project

Endogenous Control in the Agile Manifesto II

- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale
- Build projects around motivated individuals
- Give them the environment and support they need, and trust them to get the job done

Endogenous Control in the Agile Manifesto III

- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation
- Working software is the primary measure of progress

Endogenous Control in the Agile Manifesto IV

- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely
- Continuous attention to technical excellence and good design enhances agility

Endogenous Control in the Agile Manifesto V

- Simplicity – the art of maximizing the amount of work not done – is essential
- The best architectures, requirements, and designs emerge from self-organizing teams
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

Altogether...

- Endogenous control is way more effective than exogenous control
- But it may be hard to achieve!
- We try to obtain an adaptive development process whose control is built-in the process rather than added on the process
- This is one of the tenets of Lean Management
- Visual control (and more)

Outline

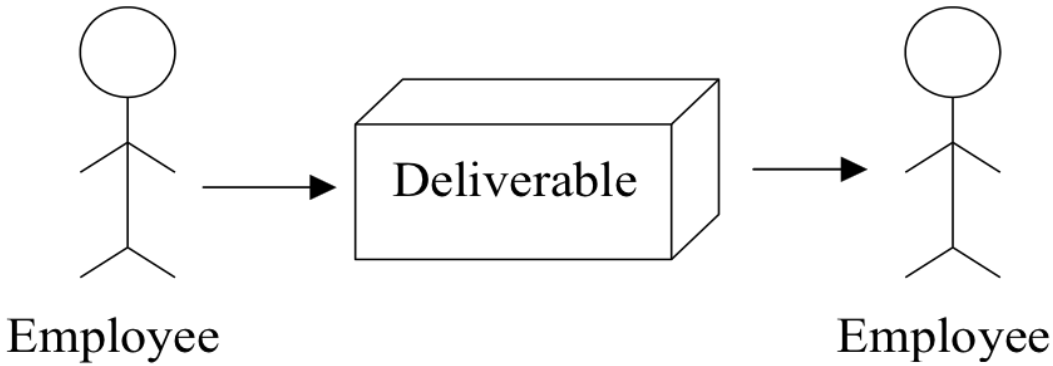
- Agile manifesto
- Keeping the Process Under Control
- Job Enrichment
- To XP or not to XP
- XP
- Let‘s work on differences
- Control model
- **Coordination mechanisms**

Coordination mechanisms

According to Malone and Crowston (1994) there are three ways people coordinate:

- Sequentially or individual
- Via shared resource
- Via common output

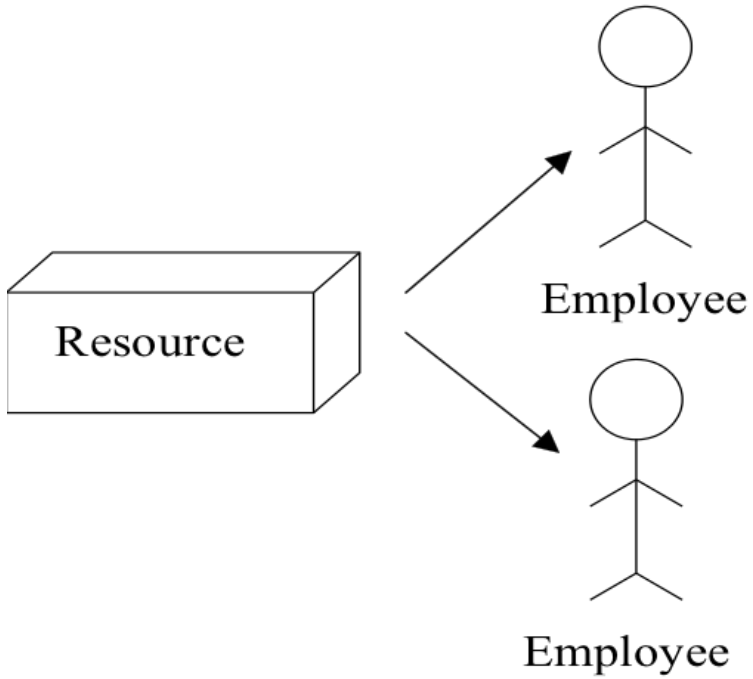
Sequential



- First write the analysis document
- On the basis of the analysis document write the design document

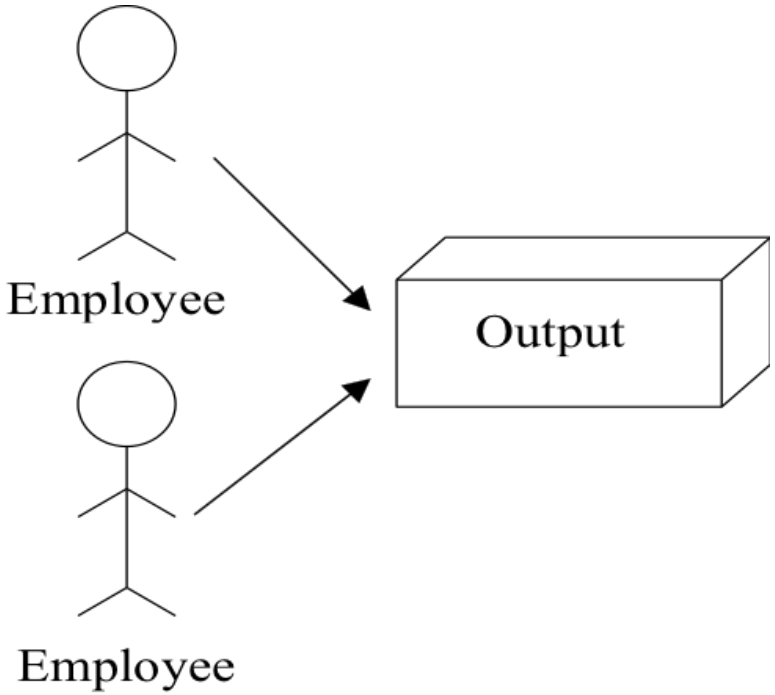
(Is this endogenous control?)

Shared resource



- The development may start only after the customer has committed to the project
- The classes to develop are assigned to developers on a First Come – First Served basis

Common output



- The team leader proposes a schedule for the week to come; the team has to agree unanimously on it
- The team sits together to produce the final release that requires everyone approval

What is most difficult?

- Easiest: Sequential
- Then: Shared resource – there is the need to respect priorities
- Most difficult: Common output
  - people need to work together, hand in hand, producing the same good
- Most of traditional software development is based on exogenous control and sequential coordination

Revising XP practices (1)

Practice	Control	Coordination
Planning game	Endogenous	Common output
Short releases	Endogenous	Sequential
Metaphor	Exogenous	Common output
Simple design	Exogenous	Shared resource
Test-driven development	Endogenous	Shared resource
Refactoring	Endogenous	Common output

Revising XP practices (2)

Practice	Control	Coordination
Pair Programming	Endogenous	Common output
Collective code ownership	Endogenous	Common output
Continuous integration	Endogenous (e.g., via Cruisecontrol)	Common output
40 hours week	Exogenous	Sequential
On-site customer	Exogenous	Shared resource
Coding standards	Exogenous (unless with a checker)	Sequential