UNIVERSITÄT
BERN

ESE
Einführung in Software Engineering

3. The Planning Game

Prof. O. Nierstrasz

Roadmap

- XP — coping with change and uncertainty
- Customers and Developers — why do we plan?
- The Planning Game
  - Exploration — User stories
  - Estimation
  - Commitment
  - Steering
- Iteration
- Scrum
- Agile lessons from industry

Based on a presentation by Matthias Rieger.

Sources

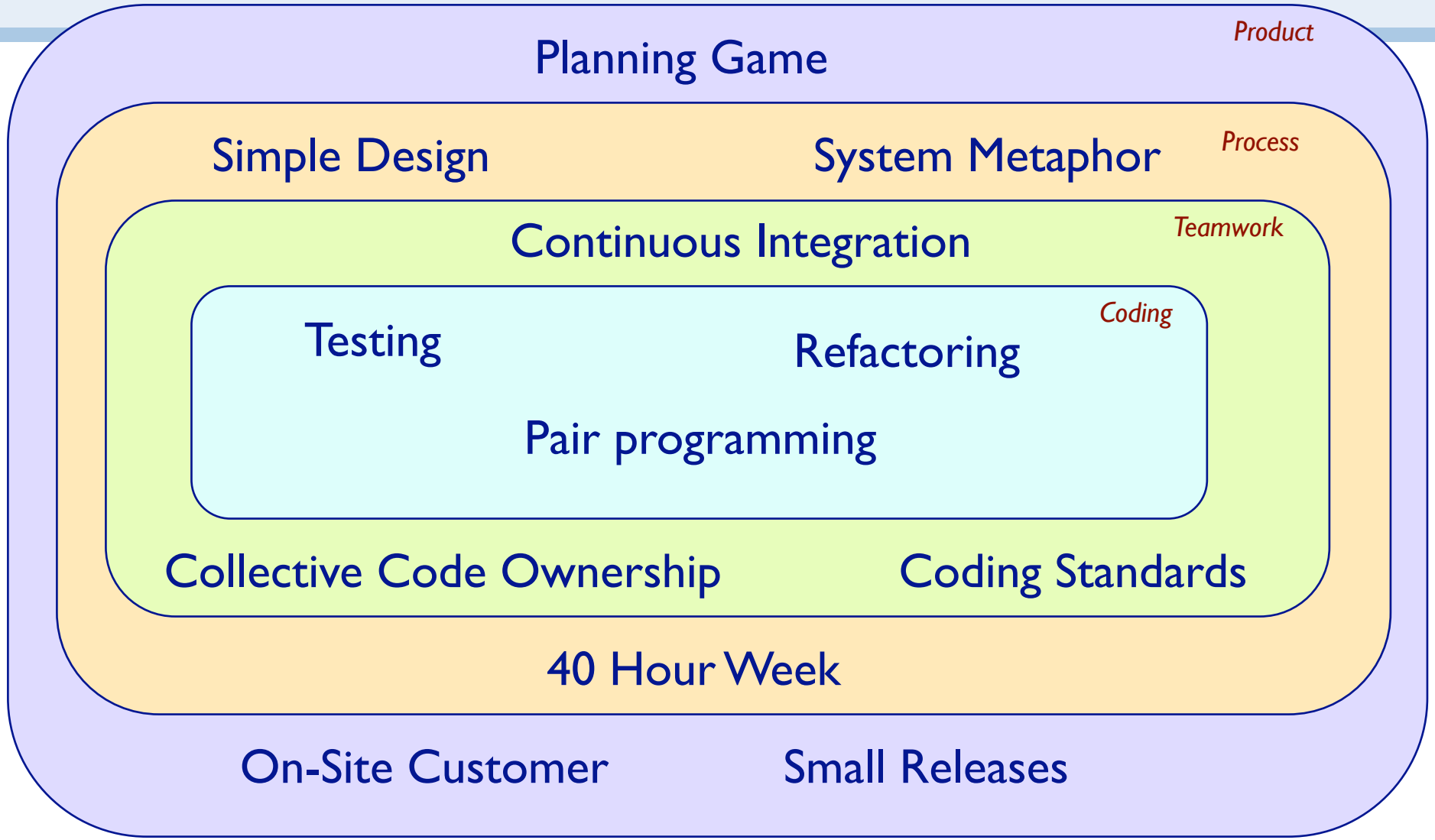
- *eXtreme Programming Explained: Embrace Change*. Kent Beck. Addison-Wesley Pub Co; ISBN: 0201616416; 1st edition (October 5, 1999)

- www.extremeprogramming.org

Roadmap

- **XP — coping with change and uncertainty**
- Customers and Developers — why do we plan?
- The Planning Game
  - Exploration — User stories
  - Estimation
  - Commitment
  - Steering
- Iteration
- Scrum
- Agile lessons from industry

Extreme Programming



See also: www.extremeprogramming.org

Driving Metaphor

- Driving a car is not about pointing the car in one direction and holding to it; driving is about *making lots of little course corrections.*

"Do the simplest thing that could possibly work"

Roadmap

- XP — coping with change and uncertainty
- **Customers and Developers — why do we plan?**
- The Planning Game
  - Exploration — User stories
  - Estimation
  - Commitment
  - Steering
- Iteration
- Scrum
- Agile lessons from industry

Why we plan

We want to ensure that

- we are always working on the most *important* things
- we are *coordinated* with other people
- when *unexpected* events occur, we *understand the consequences* on priorities and coordination

Plans must be

- easy to *make and update*
- *understandable* by everyone that uses them

The Planning Trap

Plans project a *likely* course of events

— Plans must try to create *visibility*: where is the project

But: A plan does not mean you are in control of things

— Events happen

— Plans become invalid

Having a plan isn't everything, planning is.

— Keep plans honest and expect them to always change

Customer-Developer Relationships

A well-known experience in Software Development:
The customer and the developer sit in a small boat in the ocean and *are afraid of each other.*

Customer fears	Developer fears
They won't get what they asked for	They won't be given clear definitions of what needs to be done
They must surrender the control of their careers to techies who don't care	They will be given responsibility without authority
They'll pay too much for too little	They will be told to do things that don't make sense
They won't know what is going on (the plans they see will be fairy tales)	They'll have to sacrifice quality for deadlines

Result: A lot of energy goes into protective measures and politics instead of success

The Customer Bill of Rights

You have the right to an overall plan	To steer a project, you need to know what can be accomplished within time and budget
You have the right to get the most possible value out of every programming week	The most valuable things are worked on first.
You have the right to see progress in a running system.	Only a running system can give exact information about project state
You have the right to change your mind, to substitute functionality and to change priorities without exorbitant costs.	Market and business requirements change. We have to allow change.
You have the right to be informed about schedule changes, in time to choose how to reduce the scope to restore the original date.	XP works to be sure everyone knows just what is really happening.

The Developer Bill of Rights

You have the right to know what is needed, with clear declarations of priority.	Tight communication with the customer. Customer directs by value.
You have the right to produce quality work all the time.	Unit Tests and Refactoring help to keep the code clean
You have the right to ask for and receive help from peers, managers, and customers	No one can ever refuse help to a team member
You have the right to make and update your own estimates.	Programmers know best how long it is going to take them
You have the right to accept your responsibilities instead having them assigned to you	We work most effectively when we have accepted our responsibilities instead of having them thrust upon us

Separation of Roles

Customer makes *business* decisions
Developers make *technical* decisions

Business Decisions	***Technical Decisions***
Scope	Estimates
Dates of the releases	Dates within an iteration
Priority	Team velocity
	Warnings about technical risks

The Customer owns "what you get" while the Developers own "what it costs".

Roadmap

- XP — coping with change and uncertainty
- Customers and Developers — why do we plan?
- **The Planning Game**
  - Exploration — User stories
  - Estimation
  - Commitment
  - Steering
- Iteration
- Scrum
- Agile lessons from industry

The Planning Game

A game with a set of rules that ensures that Customer and Developers don't become mortal enemies

Goal:

— *Maximize the value of the software produced by Developers.*

Overview:

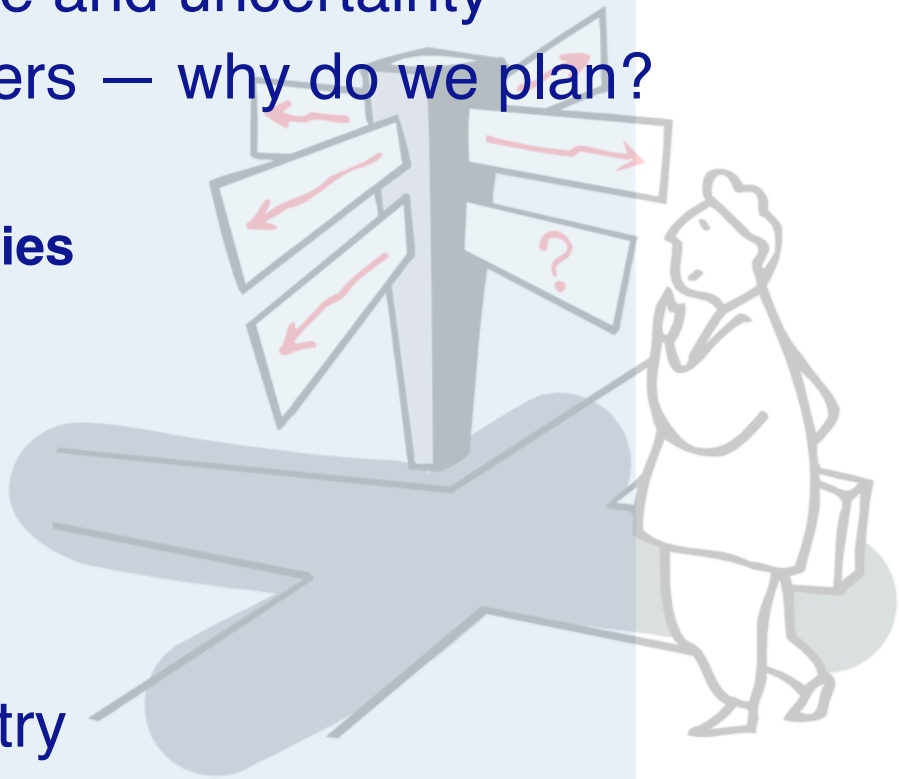
1. ***Release Planning:*** ***Customer*** selects the *scope* of the next release
2. ***Iteration Planning:*** ***Developers*** decide on *what to do* and *in which order*

The Release Planning Game

	Customer	***Developers***
Exploration Phase	Write Story	
		Estimate Story
	Split Story	
Commitment Phase	Sort Stories by Value	
		Sort Stories by Risk
		Set Velocity
	Choose Scope	
Steering Phase	Iteration	
		Recovery
	New Story	Reestimate

Roadmap

- XP — coping with change and uncertainty
- Customers and Developers — why do we plan?
- The Planning Game
  - **Exploration — User stories**
  - Estimation
  - Commitment
  - Steering
- Iteration
- Scrum
- Agile lessons from industry



Planning Game: Exploration Phase

Purpose:

Get an appreciation for what the system should eventually do.

The Moves:

- ***Customer:*** *Write a story*. Discuss it until everybody understands it.
- ***Developers:*** *Estimate a story* in terms of effort.
- ***Customer:*** *Split a story*, if Developers don't understand or can't estimate it.
- ***Developers:*** Do a *spike solution* to enable estimation.
- ***Customer:*** *Toss stories* that are no longer wanted or are covered by a split story.

User Stories

Principles of good stories:

- ***Customers*** write stories.
  - — ***Developers*** do *not* write stories.
- Stories must be *understandable* to the customer
- The *shorter* the better. No detailed specification!
  - — Write stories on *index cards*
- Each story must provide *something of value* to the customer
- A story must be *testable*
  - — then we can know *when it is done*

Writing stories is an iterative process, requiring interaction between Customer and Developers.

Stories

A story contains:

- a name
- the story itself
- an estimate

Example:

— When the GPS has contact with two or fewer satellites for more than 60 seconds, it should display the message “Poor satellite contact”, and wait for confirmation from the user. If contact improves before confirmation, clear the message automatically.

Splitting Stories

Developers ask the Customer to split a story if

- They cannot estimate a story because of its complexity
- Their estimate is longer than two or three weeks of effort

Why?

- Estimates get fuzzy for bigger stories
- The smaller the story, the better the control (tight feedback loop)

Roadmap

- XP — coping with change and uncertainty
- Customers and Developers — why do we plan?
- The Planning Game
  - Exploration — User stories
  - **Estimation**
  - Commitment
  - Steering
- Iteration
- Scrum
- Agile lessons from industry

Initial Estimation of Stories

With no history, the first plan is the hardest and least accurate (fortunately, you only have to do it once)

How to start estimating:

— Begin with the stories that you feel the most comfortable estimating.
— Intuitively imagine how long it will take you.
— Base other estimates on the comparison with those first stories.

Spike Solutions:

— Do a quick implementation of the whole story.
— Do not look for the perfect solution!
— Just try to find out how long something takes

Estimating Stories

Keys to effective story estimation:

- Keep it simple
- Use what happened in the past ("Yesterday's weather")
- Learn from experience

Comparative story estimation:

- One story is often an *elaboration* of a closely related one
- Look for stories that have *already* been implemented
- Compare *difficulties*, not implementation time
  - "twice as difficult", "half as difficult"
- *Discuss* estimates in the team. Try to find an agreement.
- *"Optimism wins":* Choose the more optimistic of two disagreeing estimates.

Roadmap

- XP — coping with change and uncertainty
- Customers and Developers — why do we plan?
- The Planning Game
  - Exploration — User stories
  - Estimation
  - **Commitment**
  - Steering
- Iteration
- Scrum
- Agile lessons from industry

Planning Game: Commitment Phase

Purpose:

- ***Customer:*** *to choose scope and date of next delivery*
- ***Developers:*** *to confidently commit to deliver the next release*

The Moves:

- ***Customer:*** *Sort* by stories by *value*
  1. Stories without which the system will not function
  2. Less essential stories, but still providing significant business value
  3. Nice-to-have stories
  - Customer wants the release to be as *valuable* as possible

Commitment Phase …

- ***Developers:*** *Sort* stories by *risk*
  1. Stories that can be estimated precisely *(low risk)*
  2. Stories that can be estimated reasonably well
  3. Stories that cannot be estimated *(high risk)*
  - Developers want to tackle *high-risk first*, or at least make risk visible

- ***Developers:*** Set team *velocity*

  How much ideal engineering time per calendar month/week can the team offer?
  - this is the *budget* that is available to Customer

- ***Customer:*** Choose *scope* of the release, by either
  - fixing the date and choosing stories based on estimates and velocity
  - fixing the stories and calculating the delivery date

Roadmap

- XP — coping with change and uncertainty
- Customers and Developers — why do we plan?
- The Planning Game
  - Exploration — User stories
  - Estimation
  - Commitment
  - **Steering**
- Iteration
- Scrum
- Agile lessons from industry

Planning Game: Steering Phase

Purpose: *Update the plan based on what is learned.*

The Moves:

- *Iteration:* ***Customer*** picks one iteration worth of the most valuable stories.
  - see Iteration Planning
- *Get stories done:* ***Customer*** should only accept stories that are 100% done.
- *Recovery:* ***Developers*** realize velocity is wrong
  - ***Developers*** re-estimate velocity.
  - ***Customer*** can defer (or split) stories to maintain release date.

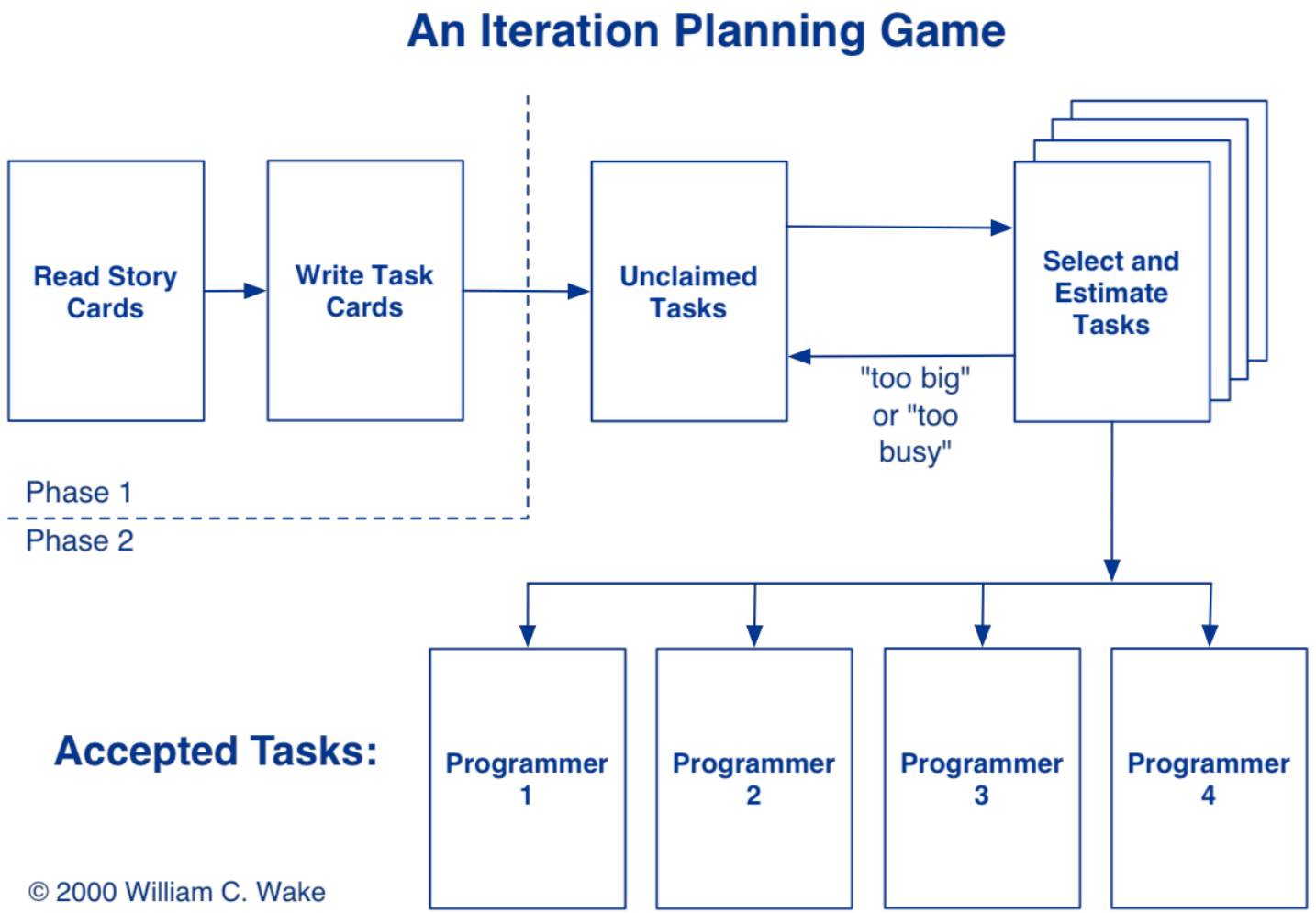
Planning Game: Steering Phase...

- *New Story:* ***Customer*** identifies new, more valuable stories
  - ***Developers*** estimate story
  - ***Customer*** removes estimated points from incomplete part of existing plan, and inserts the new story.
- *Reestimate:* ***Developers*** feel that plan is no longer accurate
  - ***Developers*** re-estimate velocity and all stories.
  - ***Customer*** sets new scope plan.

Roadmap

- XP — coping with change and uncertainty
- Customers and Developers — why do we plan?
- The Planning Game
  - Exploration — User stories
  - Estimation
  - Commitment
  - Steering
- **Iteration**
- Scrum
- Agile lessons from industry

Iteration Planning



Iteration Planning

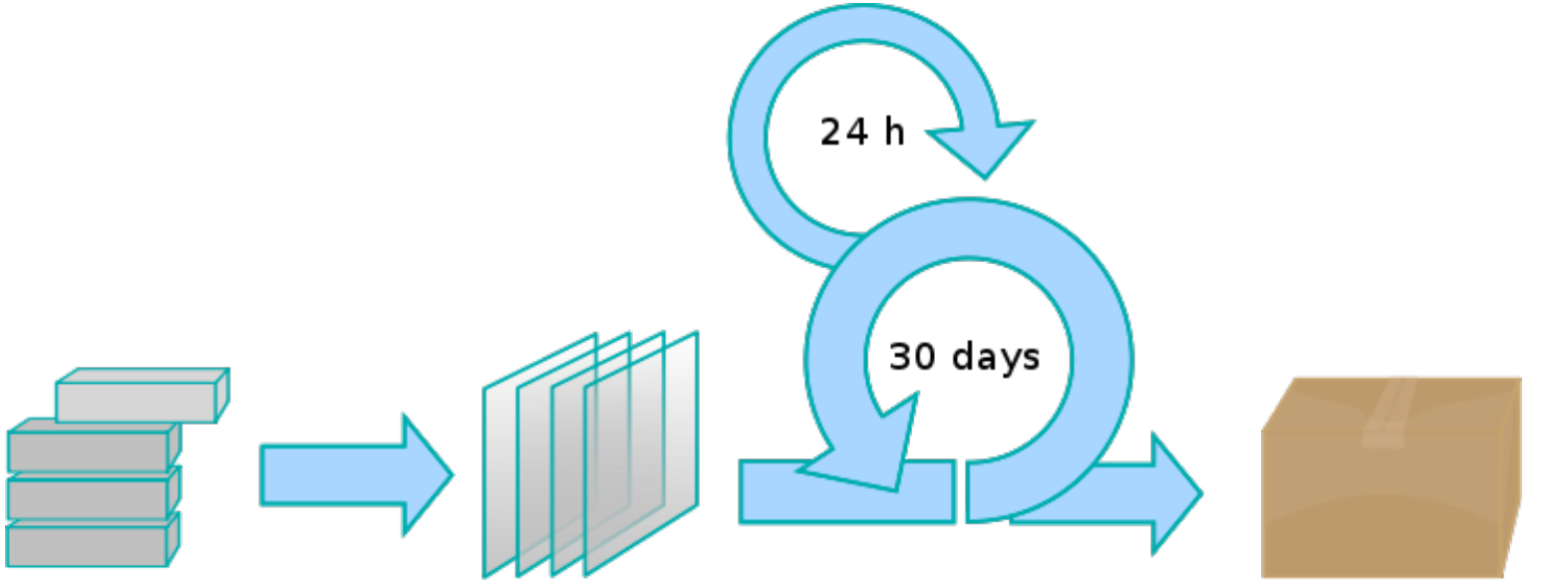
- ***Customer*** *selects* stories to be implemented in this iteration.
- ***Customer*** *explains* the stories in detail to the Developers
  - Resolve ambiguities and unclear parts in discussion
- ***Developers*** brainstorm *engineering tasks*
  - A task is small enough that everybody fully understands it and can estimate it.
  - Use short CRC or UML sessions to determine how a story is accomplished.
  - Observing the design process builds common knowledge and confidence throughout the team
- ***Developers*** /pairs *sign up* for work and estimates
  - Assignments are not forced upon anybody (Principle of Accepted Responsibility)
  - The person responsible for a task gets to do the estimate

Roadmap

- XP — coping with change and uncertainty
- Customers and Developers — why do we plan?
- The Planning Game
  - Exploration — User stories
  - Estimation
  - Commitment
  - Steering
- Iteration
- **Scrum**
- Agile lessons from industry

Scrum

The Scrum Software Development Process for Small Teams. In IEEE Software, July 2000



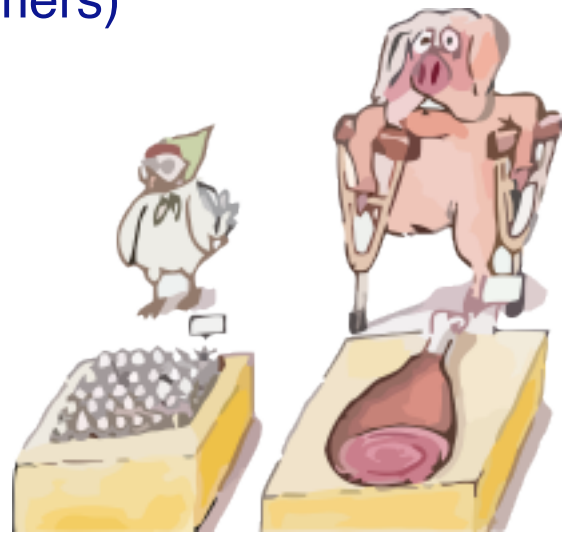
Scrum is a framework of an agile process to manage software projects

Scrum roles

1. **Scrum Master** — manages the process
   - removes impediments to the team
2. **Product owner** — stakeholder
   - prioritizes product backlog items
3. **Team** — ~7 developers (analysis, design etc)
   - decides which backlog items go into a sprint

Chickens and Pigs

- Pigs are *committed* to the project
  - Scrum Master, Product Owner, Team
  - Part of the Scrum process
- Chickens are only *involved*
  - Managers, Stakeholders (vendors, customers)
  - Should be taken into account
  - Not part of the process!



Daily Scrum (standup meeting)

- Same time, same place, every day
- Start *on time*
- Max 15”
- Only pigs may speak
- Answer 3 questions
  1. What have I done since yesterday?
  2. What am I planning to do today?
  3. What problems are preventing me from reaching my goal?

Other Scrum meetings

- **Scrum of scrums**
  - meeting of teams, after the daily scrum
- **Sprint planning**
  - prepare sprint backlog at start of sprint
- **Sprint review**
  - review work completed at end of sprint
- **Sprint retrospective**
  - review sprint process

Roadmap

- XP — coping with change and uncertainty
- Customers and Developers — why do we plan?
- The Planning Game
  - Exploration — User stories
  - Estimation
  - Commitment
  - Steering
- Iteration
- Scrum
- **Agile lessons from industry**

Slides by Tamar Richner

Agile Lessons from Industry

- Need dedicated team
- Willingness to prioritize requirements
- Engaged product manager
- Clear project governance
- Project team should shape the process to its needs
- Transparency and openness
- Project success measured by delivered value

Ingredients for Success

For the collaboration:

- Stakeholders must understand the process
- Need efficient decision making process
- Someone must translate technical ←→ business
- Team must agree on ground rules
- Plan time to assess and improve collaboration

Ingredients for Success

For planning reliability:

- Aim for reliable sprint planning
  - shorter is easier to estimate
- Define clearly what “complete” means
- Put project management and planning on the sprint backlog

Agile methods still need classical project management

- Leadership is needed to facilitate “self-organization”
- Frequent planning is required
- Structure and discipline are needed
- Continuous dialog with product owner takes time
- Not every phase of project can be “agiled”
  - e.g. product launch and move to operation

What you should know!

- Why is planning more important than having a plan?
- Why shouldn't Customers make technical decisions?
  Why shouldn't Developers make business decisions?
- Why should stories be written on index cards?
- Why should the Customer sort stories by value?
- Why should the Developer sort stories by risk?
- How do you assign stories to Developers?

Can you answer the following questions?

- What is “extreme” about XP?
- What is the differences between a User Story and a Use Case?
- Are Developers allowed to write stories?
- What is the ideal time period for one iteration?
- How can you improve your skill at estimating stories?

License

> http://creativecommons.org/licenses/by-sa/3.0/