

INF5181: Process Improvement and Agile Methods in Systems Development

Lecture 03: Agile Principles and Processes



Dr. Dietmar Pfahl

email: dietmarp@ifi.uio.no

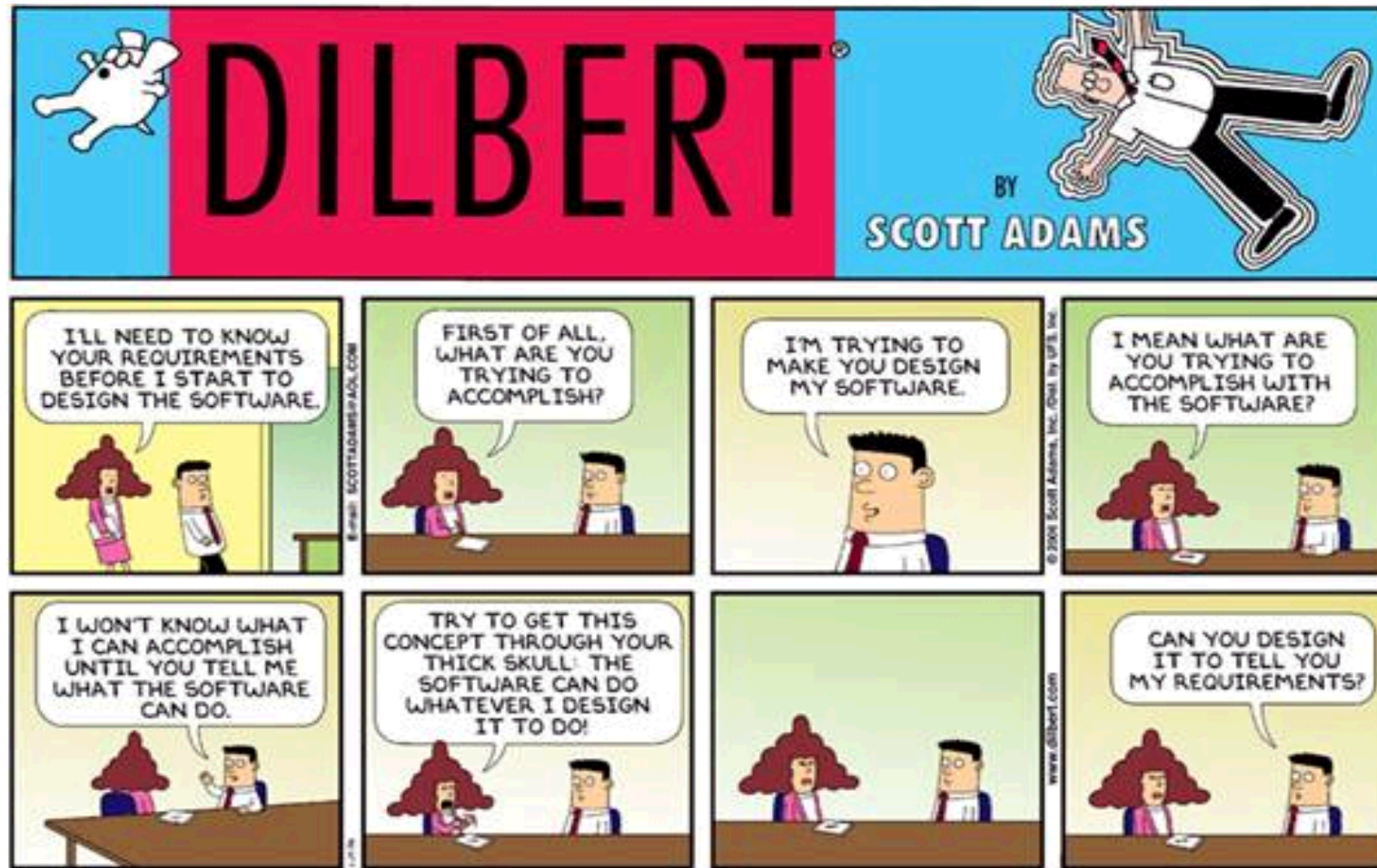
Fall 2012

Structure of Lecture 03

- Hour 1:
 - Light-weight (agile) processes / Evolutionary development ←
 - Extreme Programming (XP)
- Hour 2:
 - Question/answer session about homework 1
 - Info on mandatory short presentation (→ Lecture 5)
 - Question/answer session about project
- Hour 3:
 - Scrum
 - Choosing the right process (model)



Requirements and Customers



© Scott Adams, Inc./Dist. by UFS, Inc.



UNIVERSITETET
I OSLO

The Agile Manifesto

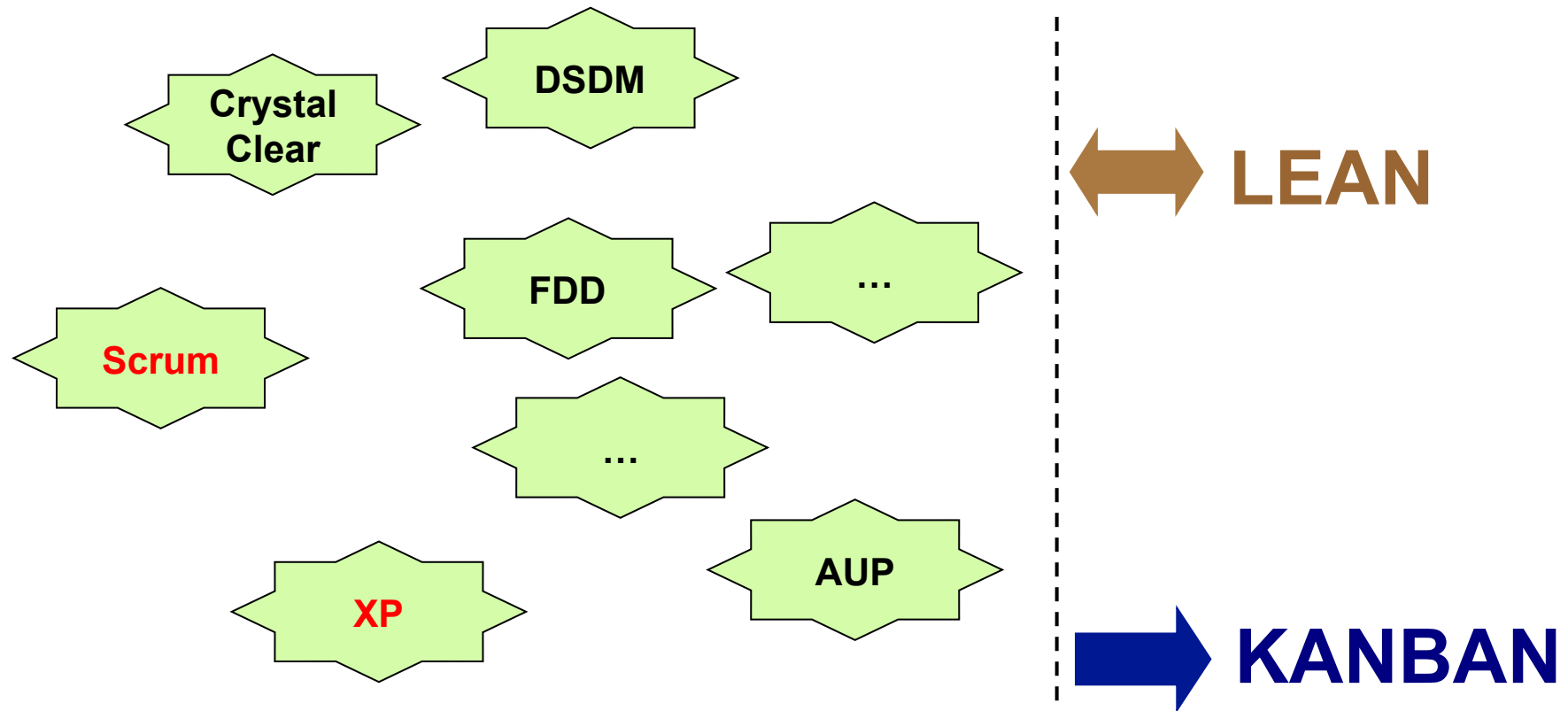
Kent Beck et al. (2001):

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.



There exists more than one Agile Method!



AUP = Agile Unified Process
DSDM = Dynamic Systems Development Method
FDD = Feature-Driven Development
XP = Extreme Programming

Structure of Lecture 03

- Hour 1:
 - Light-weight (agile) processes / Evolutionary development
 - Extreme Programming (XP) ←
- Hour 2:
 - Question/answer session about homework 1
 - Info on mandatory short presentation (→ Lecture 5)
 - Question/answer session about project
- Hour 3:
 - Scrum
 - Choosing the right process (model)



Extreme Programming

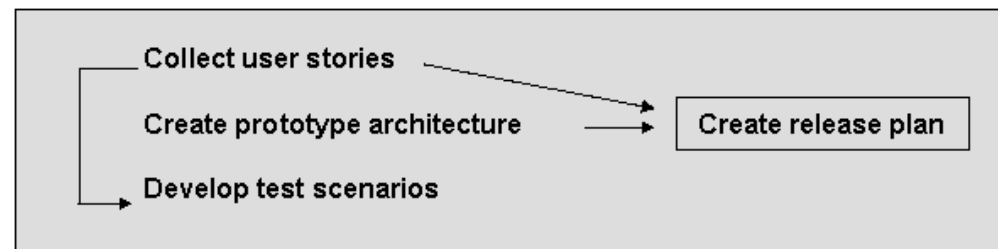
- **Origin:** Kent Beck, Ward Cunningham, Ron Jeffries (end of 1990s)
- **Idea:** “light weight” process model, agile process
- **Characteristic:**
 - “Minimum” of accompanying measures (docs, modeling , ...)
 - Team orientation (e.g., joint responsibility for all dev. artifacts)
 - Small teams (12-14 persons)
 - Involvement of user/client at an early stage
 - Social orientation
- **Scope:**
 - Pilot or small projects with low criticality of the results



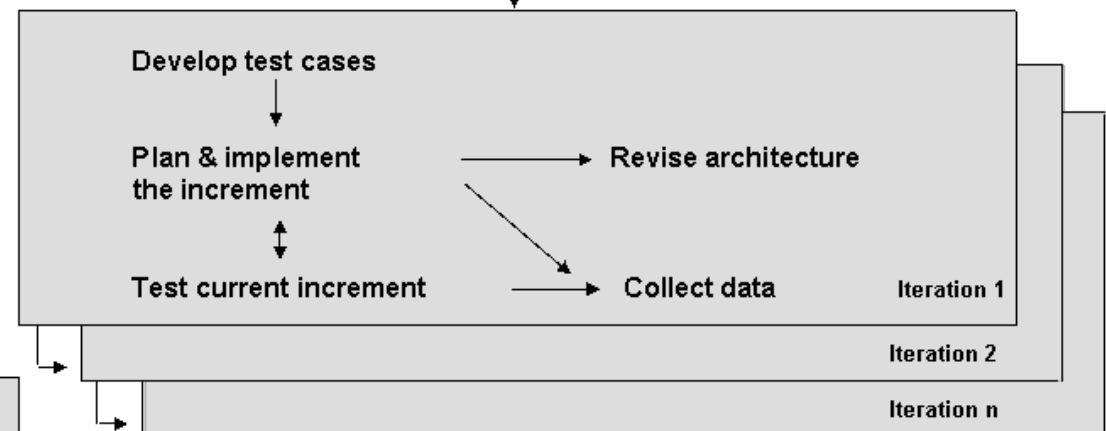
Extreme Programming – Overview

- **User stories** (something like use cases) are written by the **customer**.
- Complex stories are **broken down** into simpler ones (like a WBS).
- Stories are used to **estimate** the required amount of work.
- Stories are used to create **acceptance tests**.
- A **release plan** is devised that determines which stories will be available in which release.
- Don't hesitate to change what doesn't work.

Planning



Iterative Phase



<http://www.extremeprogramming.org/rules.html>

13 XP Practices

Project Cycle

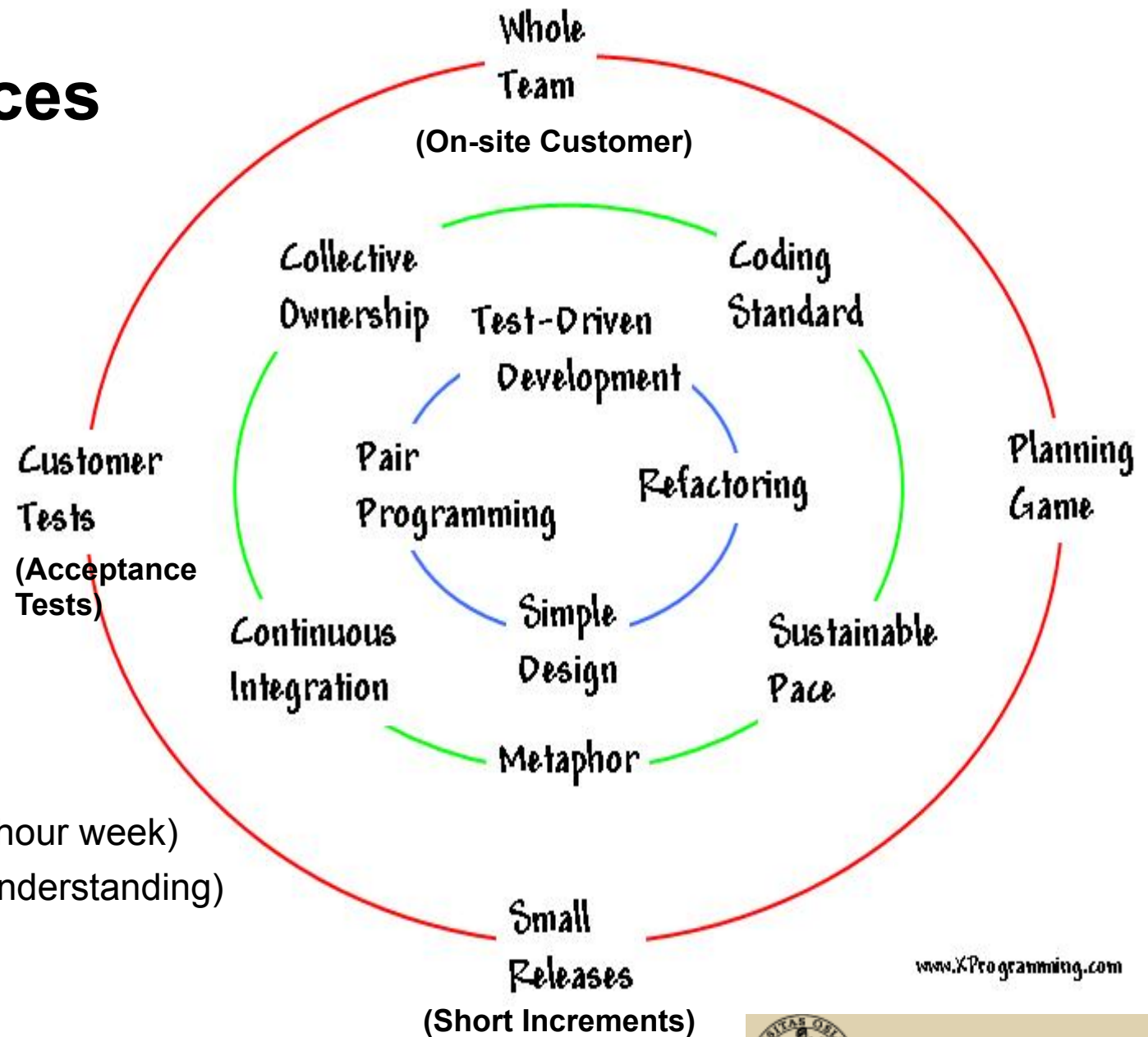
- Planning Game (Poker)
- Small Releases
- Whole Team
- Customer Tests

Development Cycle

- Simple Design
- Pair Programming
- TDD (Unit Test)
- Refactoring

Supporting Practices

- Coding Standard
- Sustainable Pace (40-hour week)
- Metaphor (Common Understanding)
- Continuous Integration
- Collective Ownership



XP – Rules and Practices (Combined)

<http://www.extremeprogramming.org/rules.html>

Planning

User stories are written (by the customer!).
Release planning creates the schedule.
Make frequent small releases.
The Project Velocity is measured.
The project is divided into iterations.
Iteration planning starts each iteration.
Move people around.
A stand-up meeting starts each day.
Fix XP when it breaks.

Designing

Simplicity.
Choose a system metaphor.
Use CRC* cards for design sessions.
Create spike solutions to reduce risk.
No functionality is added early.
Refactor whenever and wherever possible.

* CRC = Class Responsibility Collaborator

Coding

The customer is always available.
Code must be written to agreed standards.
Code the unit test first.
All production code is pair programmed.
Only one pair integrates code at a time.
Integrate often.
Use collective code ownership.
Leave optimization till last.
No overtime.

Testing

All code must have unit tests.
All code must pass all unit tests before it can be released.
When a bug is found (acceptance) tests are created.
Acceptance tests are run often and the score is published.



Requirements vs. User Stories

Traditional requirement – “shall” statements:

- “The system shall provide a user configurable interface for all user and system manager functions”
- “The user interface shall be configurable in the areas of:
 - Screen layout
 - Font
 - Background and text color

Corresponding “User Story”:

- “As a system user or system manager, ...
- ... I want be able to configure the user interface for screen layout, font, background color, and text color, ...
- ... So that I can use the system in the most efficient manner”

1

2

3

who - what - why

From Requirement to User Story – Functional Requirements

Requirement:

- The system shall provide the capability for making hotel reservations.

User Story 1:

- As a premiere member, I want to search for available discounted rooms.

User Story 2:

- As a vacationer, I want to search for available rooms.

User Story 3:

- As a vacationer, I want to save my selections.



From Requirement to User Story – Non-Functional Requirements

Requirement:

- The system shall ...

User Story 4:

- As a vacationer and user of the hotel website, I want the system to be available 99.99% of the time.

User Story 5:

- As a vacationer, I want web-pages to download in <4 seconds.

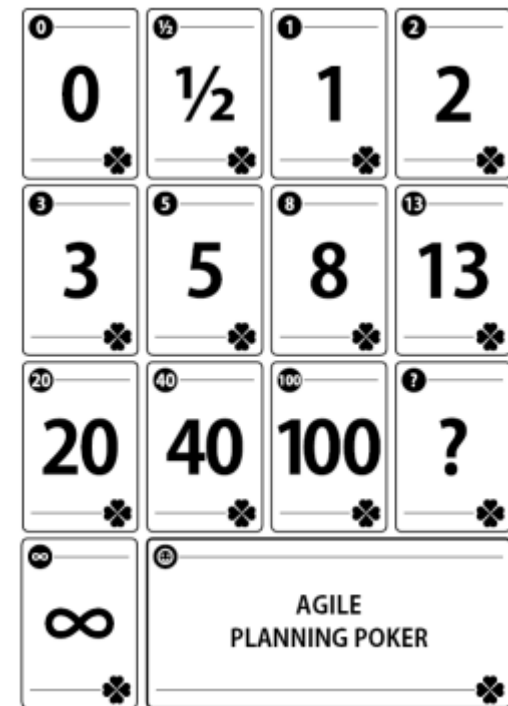
User Story 6:

- As the hotel website owner, I want 10,000 concurrent users to be able to access the site at the same time with no impact to performance.



Planning Poker /1

- Participants in planning poker include all of the developers on the team
- Step 1: Give each estimator a deck of cards
- Step 2: Moderator reads description of User Story to be estimated.
- Step 3: Product owner answers any question the estimators may have about the User Story.
- Step 4: Each estimator privately selects a card representing his or her estimate. Cards are not shown until each estimator has made a selection.
- ...

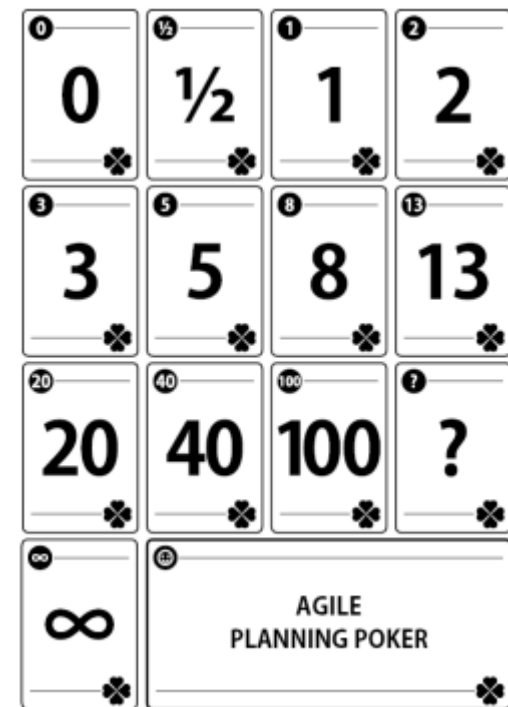


Planning Poker /2

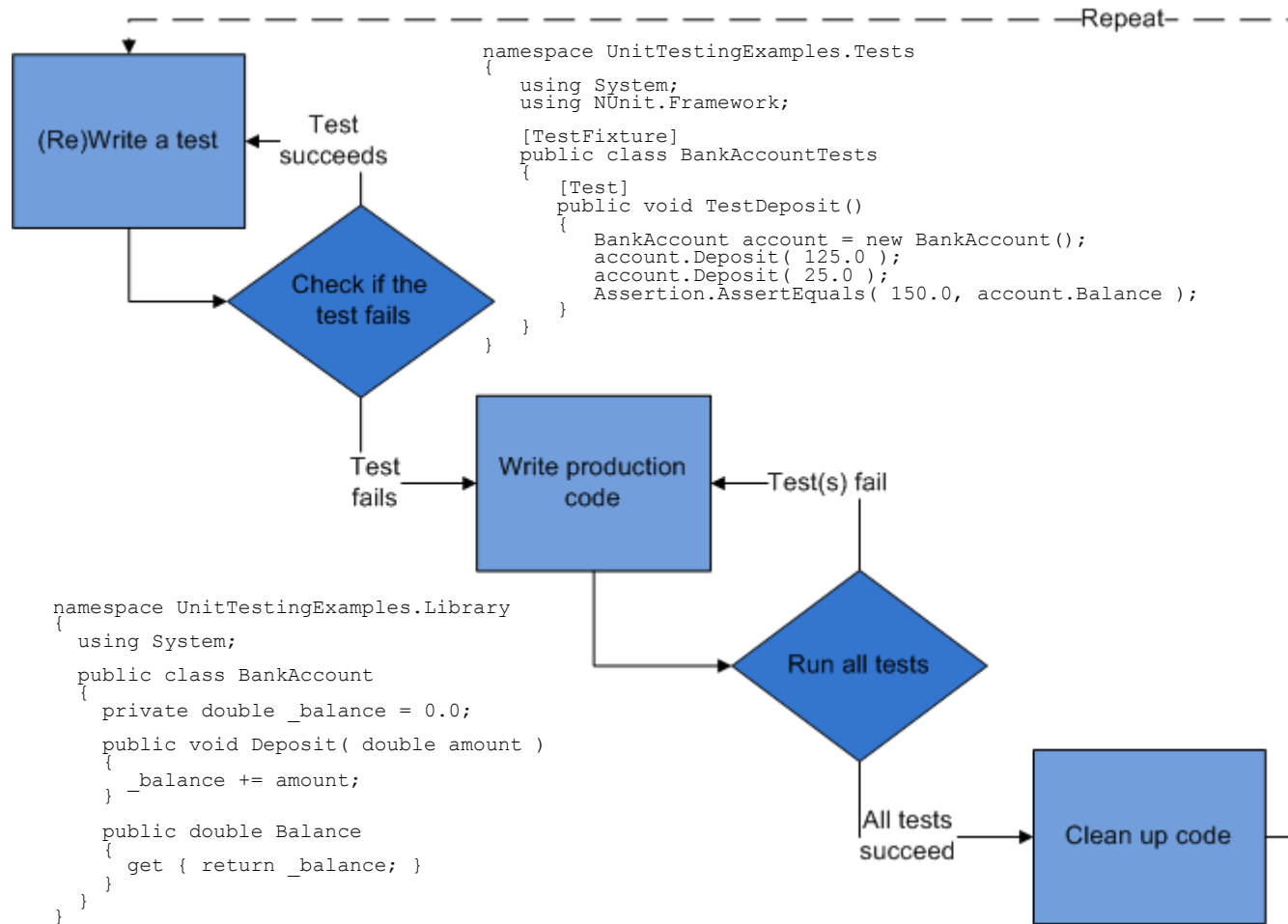
- Step 5: When everyone has made an estimate, the cards are simultaneously turned over.
- Step 6: If estimates differ, the highest and lowest estimates are explained by the estimators - otherwise the estimation is completed for this User Story.
- Step 7: The group can discuss the story and their estimates for a few more minutes. The moderator can take any notes he/she thinks will be helpful when this story is being programmed and tested. After the discussion, each estimator re-estimates by selecting a card.

→ Go to Step 5.

Note: In many cases, the estimates will already converge by the second round. But if they have not, continue to repeat the process. The goal is for the estimators to converge on a single estimate that can be used for the story. It rarely takes more than three rounds, but continue the process as long as estimates are moving closer together.



Test-Driven Development



- Unit Test
- Functionality-oriented
- Regression-testing can be automated

(Source: Wikipedia)



Simple Design



Characterisation:

- Four characteristics of simple design, listed in priority order:
 - The system runs all the tests.
 - It contains no duplicate code.
 - The code states the programmers' intent very clearly.
 - It contains the fewest possible number of classes and methods.
- The practice of TDD describes how the system is created in many small steps, driven by tests that programmers write. Each of these tests is a probe into the design of the system, allowing the developers to explore the system as it is being created. Thus, in XP, design interleaves with coding, i.e., design quite literally happens all the time.

Guidelines to help in arriving at a simple design:

- Look for a simple – but not stupid – way to solve a problem. Pay attention to good design principles when forming a system incrementally. (→ design patterns)
- Don't add infrastructure or other features that *might* be needed later. Chances are they won't be (YAGNI: You Aren't Going to Need It). Let the user stories force you to change the design.
- Don't generalize a solution until it is needed in at least two places. Follow the first rule above and keep implementation simple. Let the second user pay for the generality.
- Seek out and destroy duplication and other 'code smells' (or: 'design smells'). The practice of refactoring is the most powerful tool in the arsenal. It is through removing duplication that new classes, methods, and larger scale systems are born.
- Remember that it is just code. If it is getting overly complex and painful, delete it. It can always be recreated again in less time and better than the first time by leveraging what was learned the first time.



Refactoring

- Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior. (Invented by Martin Fowler)

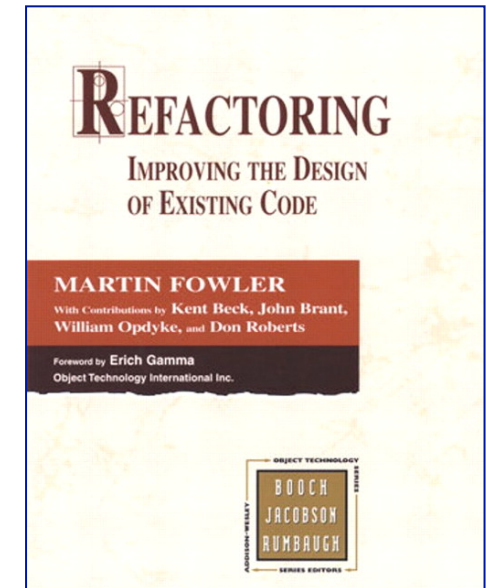
- Many refactorings can be automated

- Catalogue of refactorings:

<http://www.refactoring.com/catalog/index.html>

- Note: It is not always clear (a) how to detect refactoring opportunities and (b) what refactoring(s) are most appropriate (→ 'code smells':

http://en.wikipedia.org/wiki/Code_smell)



Pair Programming

Characterisation:

- *Two programmers* work together at one computer.
- One, the *driver*, writes code ...
- ... while the other, the *observer* (or navigator [1]), reviews each line of code as it is typed in.
- The two programmers *switch roles* frequently.

Benefits:


- Studies found that programmers working in pairs produce
 - shorter programs,
 - with better designs
 - and fewer bugsfaster

Challenges:

- Total amount of effort (person-hours) increases.
- Management needs to balance faster completion of the work and reduced testing and debugging time against the higher cost of coding.
- The benefit of pairing is greatest on tasks that the programmers do not fully understand before they begin: that is, challenging tasks that call for creativity and sophistication. On simple tasks, which the pair already fully understands, pairing results in a net drop in productivity.
- Productivity can also drop when novice-novice pairing is used without coaching.



Structure of Lecture 03

- Hour 1:
 - Light-weight (agile) processes / Evolutionary development
 - Extreme Programming (XP)
- Hour 2:
 - Question/answer session about homework 1 ← 
 - Info on mandatory short presentation (→ Lecture 5)
 - Question/answer session about project
- Hour 3:
 - Scrum
 - Choosing the right process (model)



Course Evaluation, Marking, and Grades

Part 1: Project / 80% of grade [32 marks]

Homework 1: Process Modeling [6 marks]

Homework 2: Measurement [6 marks]

Short Presentation [0 marks]

Project Report [20 marks]

Evaluation criteria for Project Report:

- Content [14 marks]
- Clarity/Conciseness [4 marks]
- Formality (cover page, captions, referencing, etc.) [2 marks]

Note:

- A mandatory short presentation is required
- Failing to do the oral presentation will result in a penalty of 2 marks!

Part 2: Oral Exam / 20% of grade [8 marks]

Duration: approximately 15-20 minutes

Subject:

Questions about the course (lecture, materials)

Evaluation criteria:

- Correctness and completeness [6 marks]
- Clarity and conciseness [1 mark]
- Relevance (→ is the answer to the point?) [1 mark]

Total Marks in Course: 40 (100%)



Project & Exam Schedule

- 10-Sep-2012: **Homework 1** – Process Modeling
 - Deliver PDF by email to dietmarp@ifi.uio.no at 15:30
- 20-Sep-2012: **Student Presentation** (5 min)
 - Send slides at the latest 24 hours before presentation by email to dietmarp@ifi.uio.no
- 15-Oct-2012: **Homework 2** – Measurement
 - Deliver PDF by email to dietmarp@ifi.uio.no at 15:30
- 15-Nov-2012: **Project Report**
 - Deliver PDF by email to dietmarp@ifi.uio.no before 20:00
 - Evaluation results ready one week before oral exam
- 13/14-Dec-2012: **Oral Exam** (15-20 min)



Homework 1: Process Modeling

- Maximum number of marks: 6
- Submission deadline: Monday, September 10, 2012, no later than 15:30 via email to dietmarp@ifi.uio.no
- Format: PDF
- Note: Before submitting, please make sure that the cover page of your homework report carries
 - your name,
 - the course id (INF5181),
 - the submission date, and
 - a reference to the type of submission (e.g., ‘Homework 1: Process Modeling’).

Structure of Lecture 03

- Hour 1:
 - Light-weight (agile) processes / Evolutionary development
 - Extreme Programming (XP)
- Hour 2:
 - Question/answer session about homework 1
 - Info on mandatory short presentation (→ Lecture 5) ←
 - Question/answer session about project
- Hour 3:
 - Scrum
 - Choosing the right process (model)



Short Presentation



- Duration: max. 4 min (incl. 1 min for question/answer)
- Send slides (max. 3 slides + cover) at the very latest until 15.30 of Tuesday, Sept 18, by email to dietmarp@ifi.uio.no (preferred format: .ppt or .pdf)
- Content:
 - **Improvement context (Type of company/project, product, process)**
 - **Improvement Goal(s) (What? How much? When?)**
 - optional: Suggested changes of SW development process (What will be changed how? / Scope of change, Old process → New process)

Short Presentation – Schedule



Lecture 5 (September 20):

Hour 1

- NN01
- NN02
- NN03
- NN04
- NN05
- NN06
- NN07
- NN08
- NN09
- NN10
- NN11

Hour 2

- NN12
- NN13
- NN14
- NN15
- NN16
- NN17
- NN18
- NN19
- NN20
- NN21
- NN22

Hour 3

- NN23
- NN24
- NN25
- NN26
- NN27
- NN28
- NN29
- NN30
- NN31
- NN32
- NN33

The exact schedule will be posted on the course web shortly after the submission deadline of homework 1.

You only need to attend the session (hour) in which you are presenting.

But you are welcome to attend longer, of course.



Structure of Lecture 03

- Hour 1:
 - Light-weight (agile) processes / Evolutionary development
 - Extreme Programming (XP)
- Hour 2:
 - Question/answer session about homework 1
 - Info on mandatory short presentation (→ Lecture 5)
 - Question/answer session about project ←
- Hour 3:
 - Scrum
 - Choosing the right process (model)



Project Report

Task:

- Prepare a (realistic) software process improvement plan for a software/systems development organization
- A suggested report structure is available
- The scope of the SPI plan could be (examples):
 - complete process
 - a sub-process of the complete process
 - an activity of a sub-process
 - a method/technique used in an activity
 - ...

INF 5181 – Process Improvement and Agile Methods in Systems Development

2012 – Project Requirements / Tasks

1. Importance of Project

A total of 32 project marks can be achieved. 32 marks are equivalent to 80% of the maximum of all course marks (40 marks).

2. Elements of the Project

The project consists of four elements which have to be prepared and presented/submitted individually (i.e., no group-work – unless otherwise communicated during the course lectures):

- Homework 1: Process Modeling (6 marks)
- Homework 2: Measurement (6 marks)
- Short Presentation (0 marks)
- Project Report (20 marks)

Notes:

- The purpose of the homework assignments is to prepare students for two important elements of their process improvement project, i.e., process modeling/analysis and measurement. Details about the homework assignments will be presented during the course lectures. The homework assignments will be marked separately from the project report.
- The short presentation is mandatory. Its purpose is to give students a chance to present their ideas about the project they plan to perform. Failing to give the oral presentation will result in a penalty of 2 marks. Details about the presentation (content, duration) will be presented during the course lectures.

3. Details about the Project Report

The task is to describe a (realistic) software process improvement (SPI) plan for a software/systems development organization in the form of a report.

3.1. Project Scope

The scope of the SPI plan could be (examples):

- a complete development process (from requirements elicitation to product shipment)
- a sub-process of the complete process
- an activity of a sub-process
- a method/technique used in an activity
- a maintenance process (or elements of it)
- ...

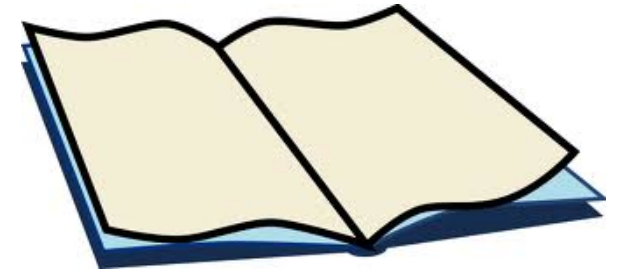
3.2. Examples of Project Topics

Examples of problems and related improvement goals that could be addressed in the project could be:

- Customers find too many defects – Improve software quality
- Inaccurate planning / estimates – Improve planning methods/models




Project Report – Structure



- Cover page (Title, Author, Date, Email, Course ID, ...)
- Table of content
- Content:
 - Improvement context (Type of company, product, process)
 - Improvement Goal(s) (What? How much? When?)
 - Suggested changes of SW development process (What? / Scope, Old process → New process)
 - Implementation of process changes (When? Who is responsible?)
 - Monitoring/Control (How to measure success? By whom?)
 - Discussion (Why? → Improvement methods applied, rationale for changes, risks)
- References

Structure of Lecture 03

- Hour 1:
 - Light-weight (agile) processes / Evolutionary development
 - Extreme Programming (XP)
- Hour 2:
 - Question/answer session about homework 1
 - Info on mandatory short presentation (→ Lecture 5)
 - Question/answer session about project
- Hour 3:
 - Scrum ← 
 - Choosing the right process (model)



The Term “Scrum”

- Originates from Rugby
- Meaning “crowded”
- Complex move that requires team work



What is Scrum? (1/2)

- Agile Management Framework for SW development projects
- With a few clear rules:
 - Roles: Product Owner, Team, Scrum Master
 - Product Backlog, Sprint Backlog, few compact reports
 - Short work cycles (→ "Sprints") for incremental development
- Based on the Agile Manifest of Kent Beck at al.
 - Human-centred
 - Technology and tools have secondary role
 - Close cooperation with customer



What is Scrum? (2/2)

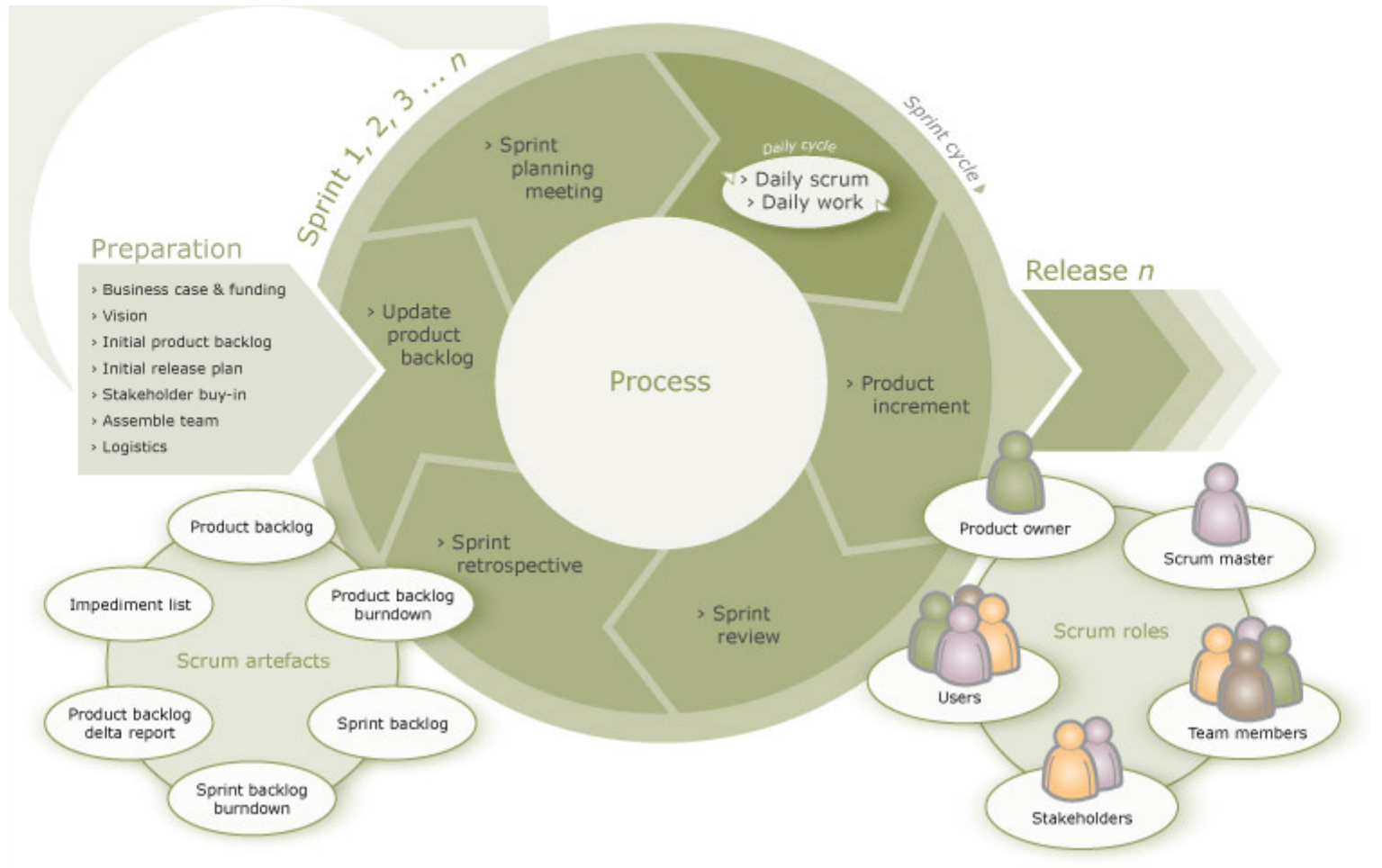
- Empirical learning process
- Learning in each iteration (Sprint): „inspect and adapt“
 - Development speed/productivity
 - Obtained results
 - Team work
 - Usage of Scrum process

Scrum does not define a development methodology, QA strategy, or risk management approach, but asks the team to take care of these issues appropriately.

Scrum may be difficult to use in environments strongly influenced by external factors.

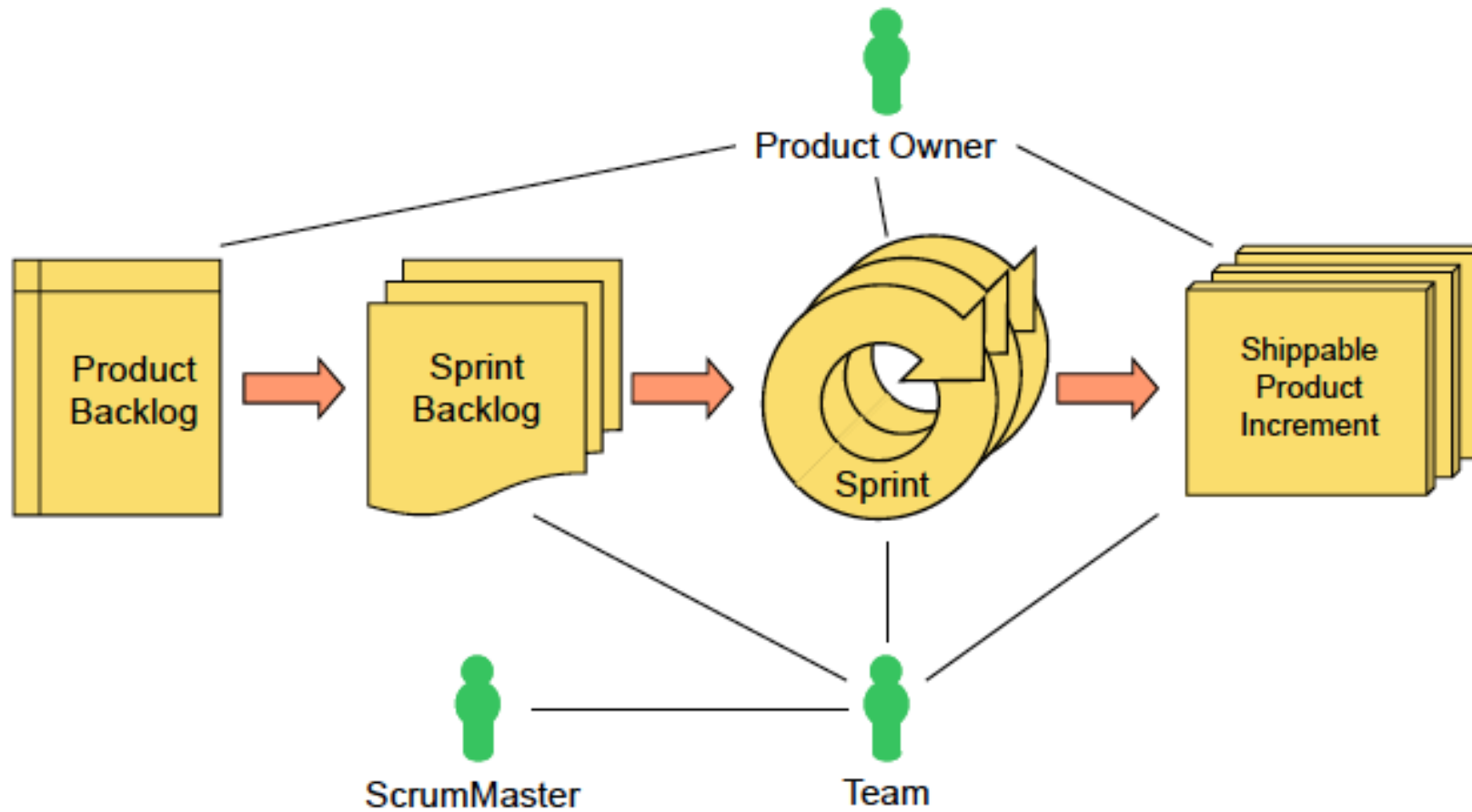


Scrum Elements – Overview



<http://www.scrumforteamssystem.com/processguidance/v1/Scrum/Scrum.html>

Scrum Process – Simplified Overview



Scrum: Backlogs

- **Product Backlog**

- Collection of requirements (user stories) for the product – at project start: a few, little detailed user stories; collection evolves over time and requirements will be refined over time
- Managed by the Product Owner

- **Sprint Backlog**

- Collection of requirements (user stories) that are selected for implementation during next sprint
- Managed by the Team



Scrum: Sprint

- Sprint

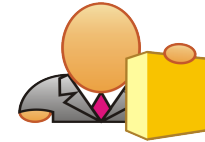
- Period (max. 30 calendar days) in which a shippable product increment (executable, tested, and documented) is created by the Team
- Time-boxed, i.e., ends exactly at the scheduled time
- At the end of the Sprint, the Product Owner has to accept the final results (i.e., the software)
- Partially completed or incorrect results will not be shipped (no compromise on quality) and go back to the Product Backlog for inclusion in the next Sprint (Backlog)



Scrum: The Three Roles

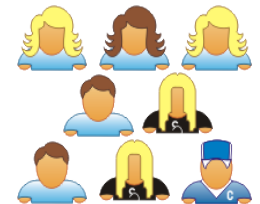
- Product Owner

- Decides which requirements are implemented for a product version
- Decides about when product increments will be shipped



- Team

- Implements requirements
- Decides how many requirements are implemented in a Sprint
- Organizes its activities (→ tasks) independently



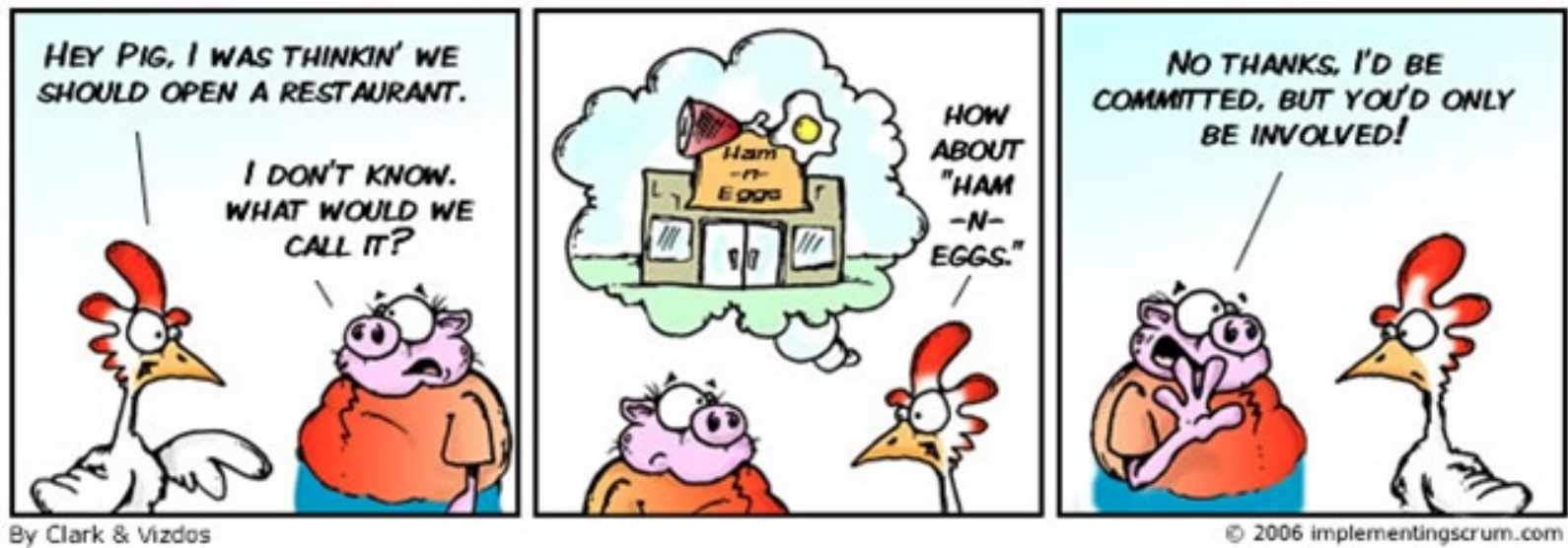
- Scrum Master

- Takes care of the proper implementation of Scrum
- Supports the team in process-related issues



Scrum: Pigs and Chickens

- **Pigs** are “committed”, i.e. they are responsible for results
 - Product Owner, Team, Scrum Master
- **Chickens** are “involved”, i.e. they are influenced by the results, but not directly responsible
 - All other stakeholders (management, sales, marketing, customer, ...)



Scrum Roles: "Pigs" and "Chickens"

"Pig" roles

- Pigs are the ones committed to the project in the Scrum process; they are the ones with "their bacon on the line".

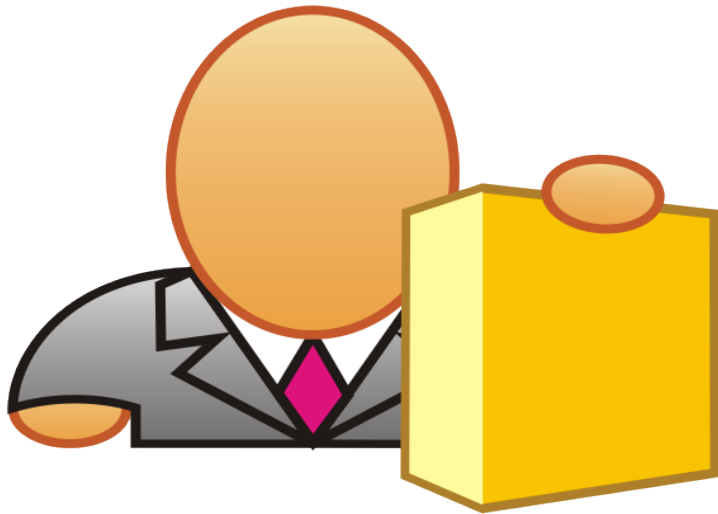
- Product Owner
- Scrum Master (or Facilitator)
- Team



"Chicken" roles

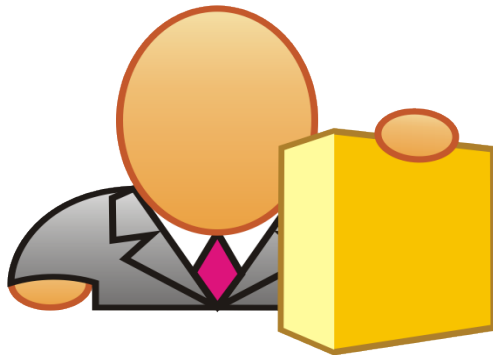
- Chicken roles are not part of the actual Scrum process, but must be taken into account.
 - Users
 - Stakeholders (customers, vendors, senior managers)
- Note: An important aspect of an [Agile](#) approach is the practice of involving users, business and stakeholders into part of the process. It is important for these people to be engaged and provide feedback into the outputs for review and planning of each sprint.

Scrum: Product Owner (1/2)



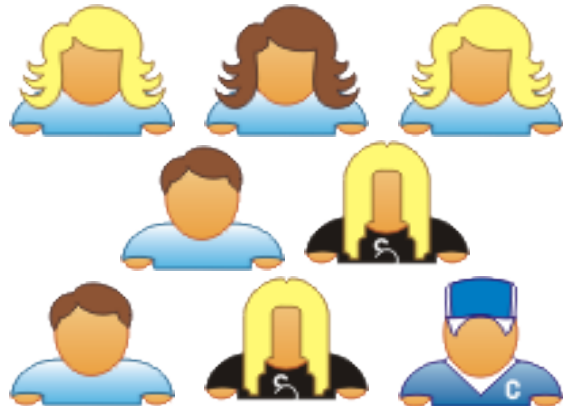
- Elicits and collects customer needs
- Describes requirements
- Decides on release dates and contents
- Is responsible for project success and the profitability of the product
- Prioritizes requirements according to market value
- Adjusts requirements and priority, as needed
- Accepts or rejects work results

Scrum: Product Owner (2/2)



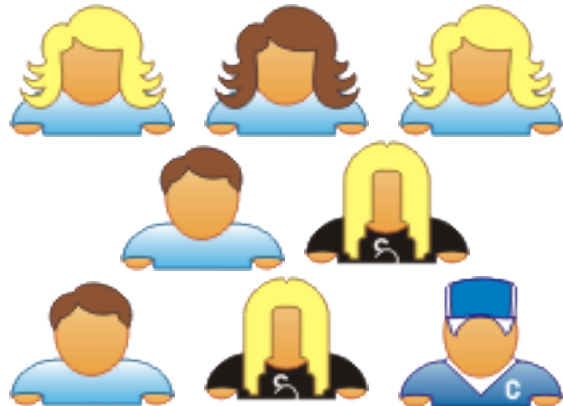
- Works closely with the team
 - Helps to understand customer needs and requirements
 - Details requirements
 - Checks resulting work products and approves them
- Integrates all stakeholders in the development and regularly elicits their needs
 - Besides customer also marketing and sales
 - Product owner combines and filters stakeholder requirements
- Has a sound technical understanding
 - Makes general overall design decisions
 - Combines classical product manager, project manager, and chief architect

Team (1/2)



- Small team size:
 - Typically 5 to 9 team members
- Cross-functional:
 - Design, coding, testing, etc.
 - Members must have a broad range of competencies
 - Every team member is an expert in his/her field but can also take over responsibilities of other team members
- Teams are independent/empowered
 - Decides which requirements to include in next Sprint (i.e., team has power to reject too many requirements)
 - Decides independently which tasks to perform to implement the requirements

Team (2/2)



- Teams are self-organizing
 - Joint, consensual decisions on tasks to perform for obtaining the goal of the Sprint, and on work distribution
 - Work is coordinated via Sprint Backlog, Burn-down Chart, and Daily Scrum
- Members should work in close distance (ideally in the same room)
- Members should be full-time
- Membership should change only between sprints

Scrum Master



- Represents management to the project
- Responsible for enacting Scrum values and practices
- Removes impediments
- Ensures that the team is fully functional and productive
- Enables close cooperation across all roles and functions
- Shields the team from external interferences

Scrum Roles – Summary

"Pig" roles:

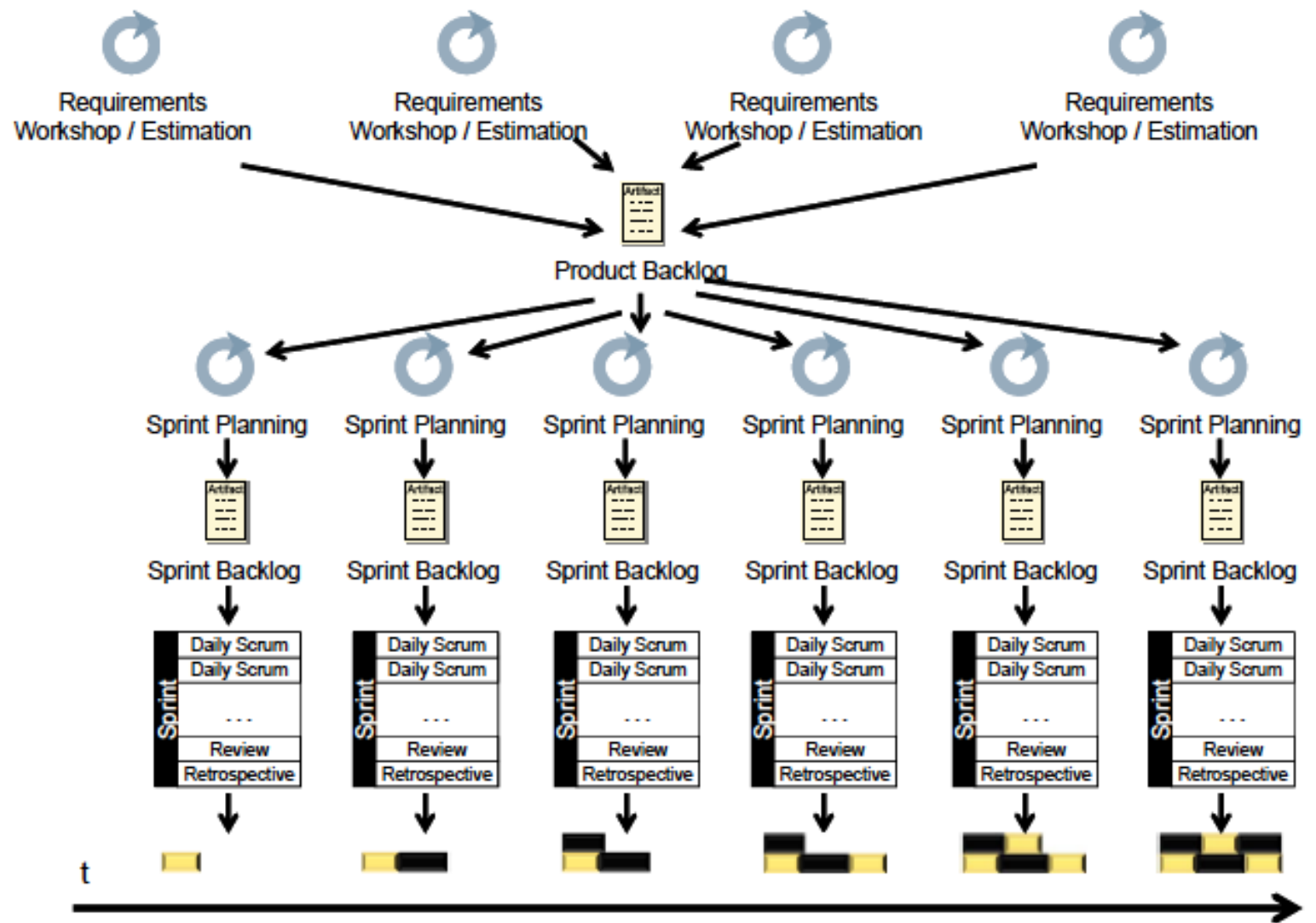
- Product Owner
 - The Product Owner represents the voice of the customer ensuring that the Team works on the right things from a business perspective.
 - The Product Owner writes user stories, prioritizes them, then places them in the product backlog.
- Scrum Master (or Facilitator)
 - Scrum is facilitated by a ScrumMaster, whose primary job is to remove impediments to the ability of the team to deliver the sprint goal.
 - The ScrumMaster is not the leader of the team (as they are self-organizing) but acts as a buffer between the team and any distracting influences.
 - The ScrumMaster ensures that the Scrum process is used as intended. The ScrumMaster is the enforcer of rules.
- Team
 - The team has the responsibility to deliver the product.
 - A team is typically made up of 5–9 people with cross-functional skills to do the actual work (designer, developer, tester, etc.).

"Chicken" roles:

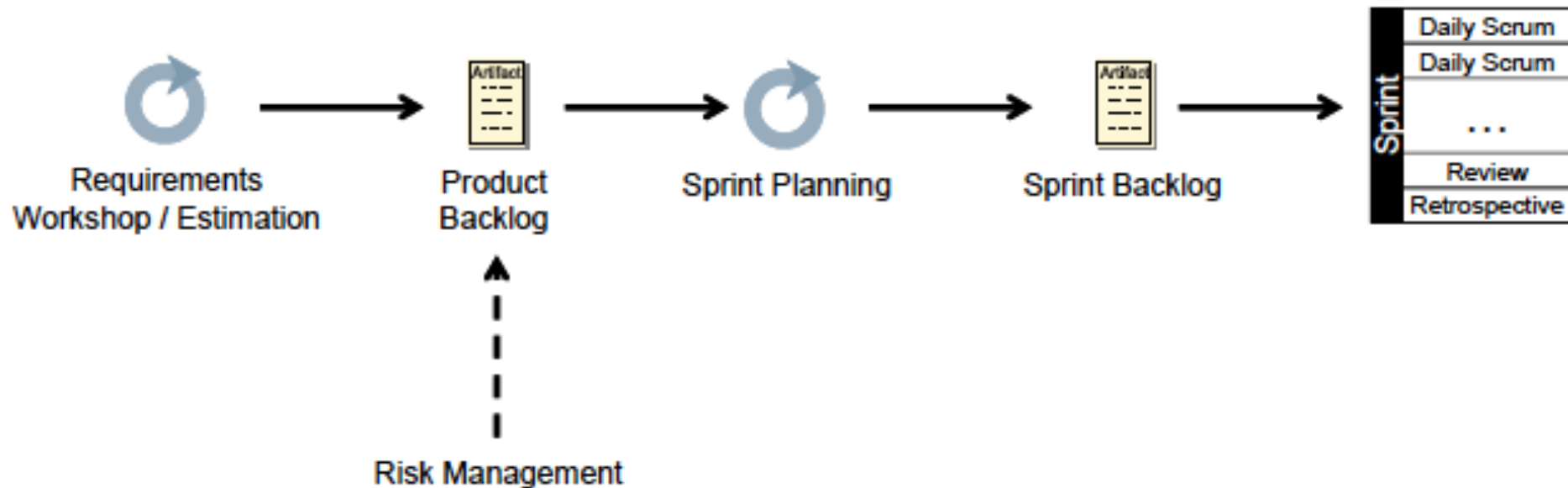
- Users
 - The software is being built for someone.
- Stakeholders (customers, vendors)
 - The people that will enable the project, and for whom the project will produce the agreed-upon benefit(s) which justify it. They are only directly involved in the process at sprint reviews.
- Managers
 - People that will set up the environment for the product development organizations.

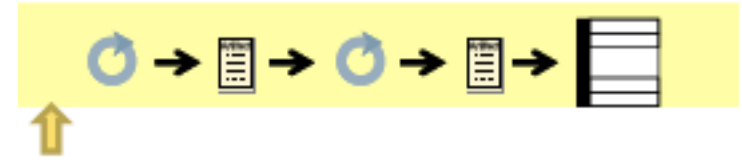


Typical Scrum Project



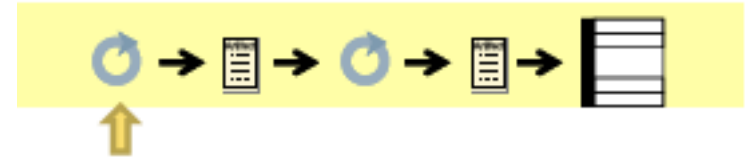
Excerpt of a Typical Scrum Project





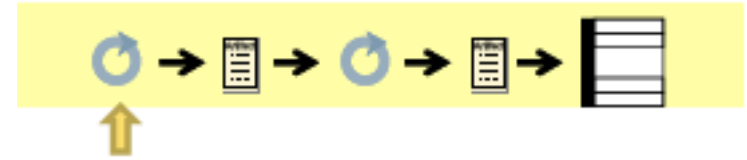
Initial Product Backlog

- Product Owner
 - fills the initial product backlog with the product properties from the product concept and other requirements from focus groups, interviews, user observation, etc.
 - Coarse-grained!
 - **Goal:** All known functional and non-functional requirements should briefly be described
- Product Owner
 - groups similar requirements into themes
 - prioritizes themes and individual requirements (if necessary) according to usefulness, risk, and cost
- Further refinement of high-priority requirements via requirements workshops
- Initial requirements are recorded for 2-3 Sprints



Requirements Workshop

- Joint workshop with product owner, team, end users, and all other relevant stakeholders (e.g., marketing, sales)
- **Goal:** Common understanding of requirements
- Fills and refines the Product Backlog
- New themes / requirements are first described only at the level of coarse-grained stories
- Existing high-priority themes / requirements are detailed and acceptance criteria are defined
- Often with the help of index cards on Meta Planning Boards
- Team estimates the cost of requirements
 - E.g. using points on a Fibonacci series (0, 1, 2, 3, 5, 8, 13, ...)
 - Done as part of an estimation workshop or using “planning poker”

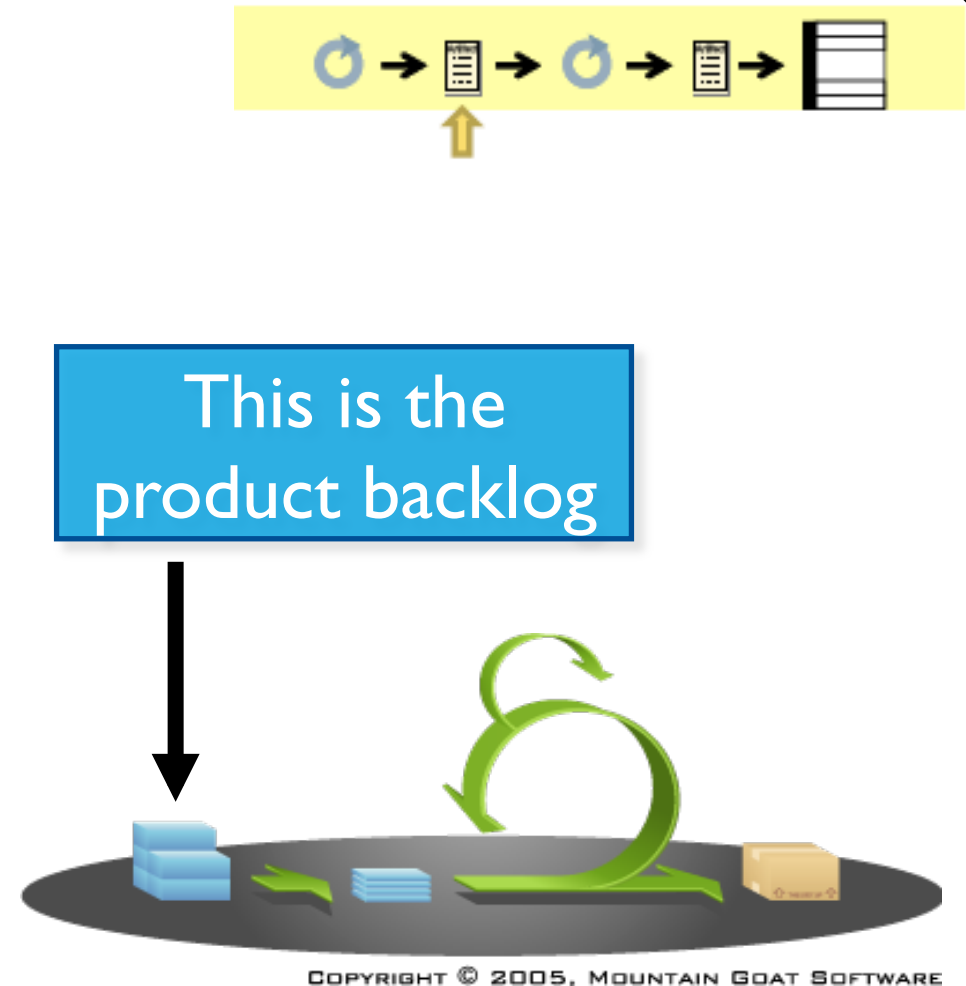


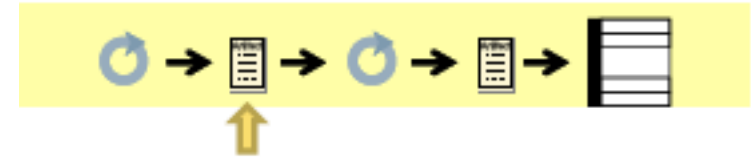
Estimation Workshop

- Product Owner explains requirements to the team
- Team estimates effort (no other persons outside the team)
 - Starting with small requirements and incrementally adding larger ones (relative to the smaller ones)
 - The estimates are only stable within a team; it is not possible to transfer these estimates to other teams without further considerations!
 - All activities are taken into account (development, test, integration, documentation, ...)
 - Single estimates are done using, e.g., planning poker
- If requirements are too vague to estimate, they have to be clarified first (e.g., making use of a so-called exploration Sprint)
- If a requirement is estimated as being “huge” (e.g., much larger than all others), there may be a lack of understanding
- The Scrum Master moderates the estimation workshop

Product Backlog

- The requirements
- A list of all desired work on the project
- Ideally expressed such that each item has value to the users or customers of the product
- Prioritized by the product owner
- Reprioritized at the start of each sprint

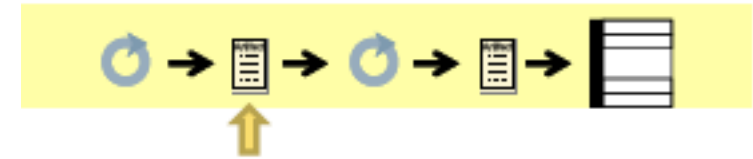




Product Backlog (1/2)

- Includes product requirements, possibly grouped by themes, and acceptance criteria as well as estimated effort
 - Functional and non-functional
 - Estimates cover all required work results, e.g. test environment
- Provides counter-measures for identified risks (e.g., creation of prototypes)
- Content is usually very coarse-grained at the beginning of the project
- Only at the end of the project all implemented requirements are described in detail!

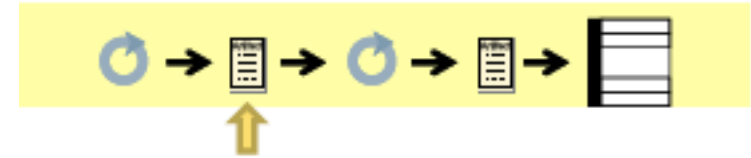
→ Product Backlog changes during entire project period



Product Backlog (2/2)

- Requirements (User Stories) are usually represented on index cards or in a table, e.g.:

Priority	Theme	Description	Acceptance Criteria	Effort
1	Washing	As a user I want to start a quick wash program	Check start conditions, for example, open door	5
2	Washing	As a user I want to be able to interrupt the short washing program	Check that the washing program is actually running	3
3	Washing



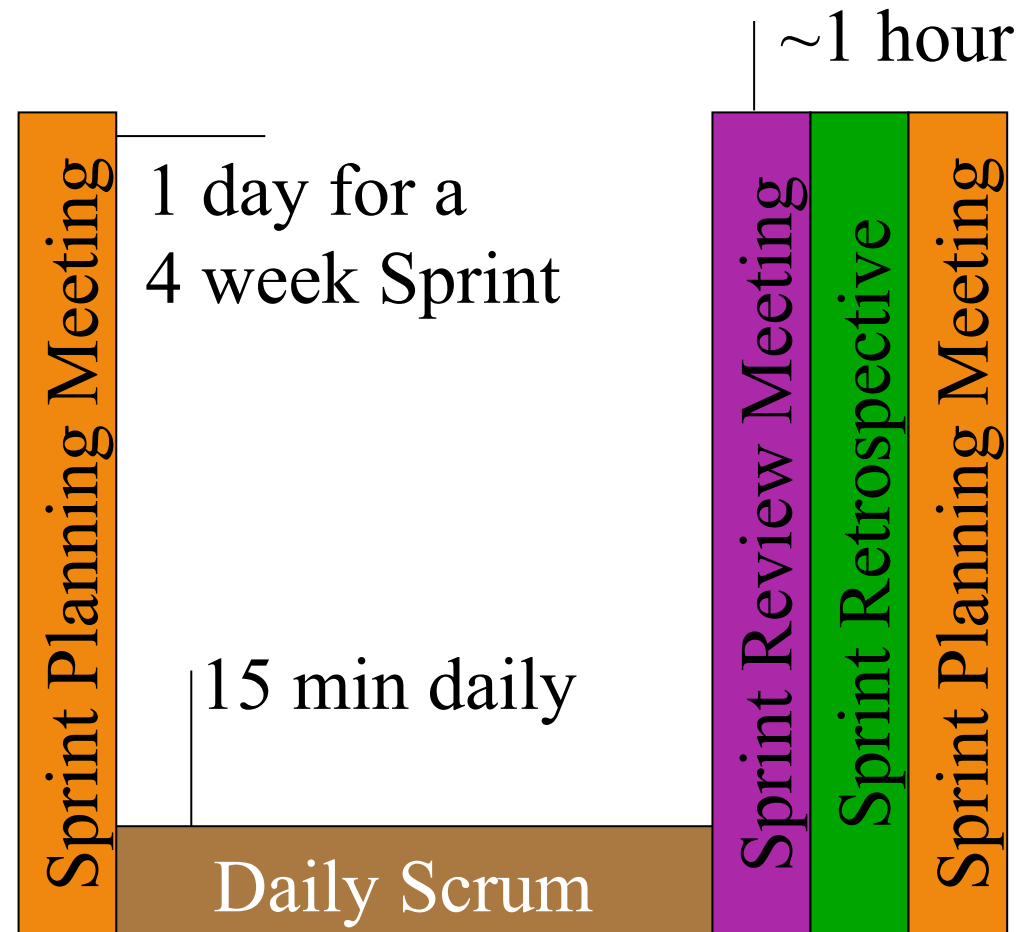
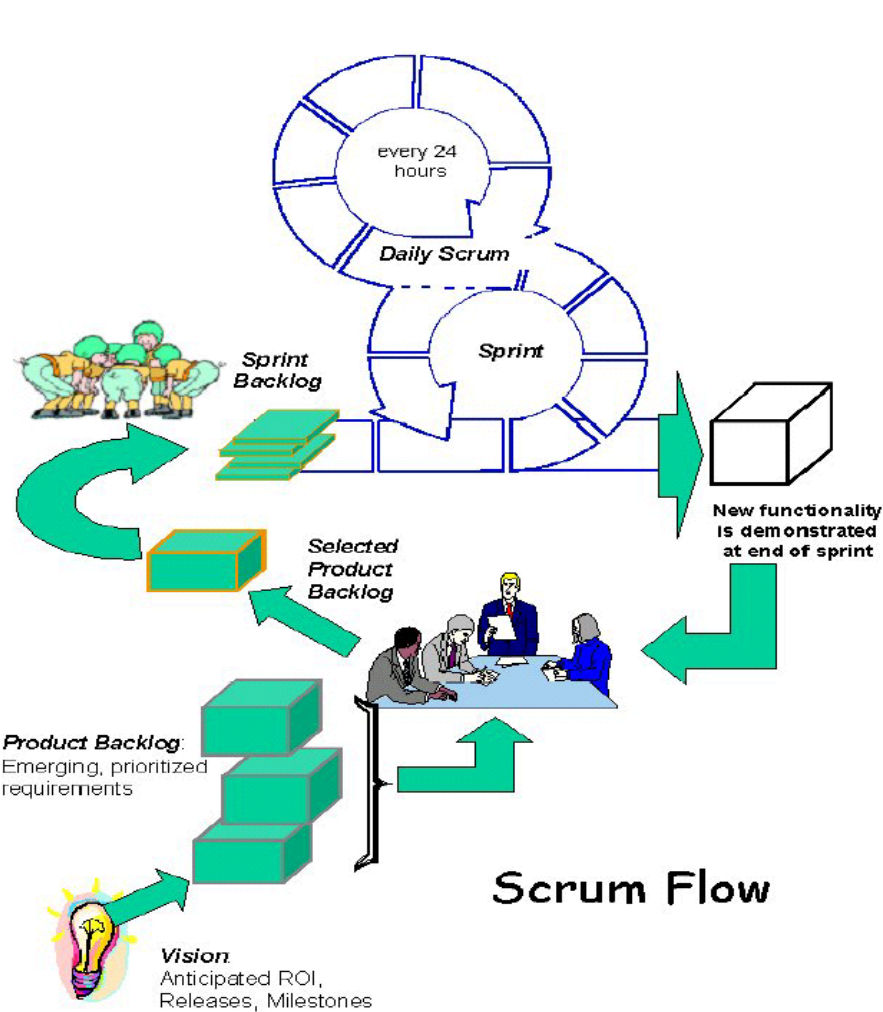
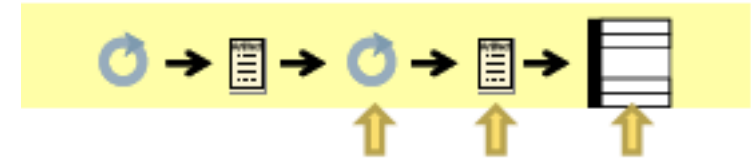
Risk Management

- Each time the release plan is updated (i.e., at least once per Sprint), risks should be identified and addressed accordingly
- Together with Product Owner and Team
- Identified risks are analyzed and appropriate counter-measures are added as entries in the Product Backlog
- If a requirement has a difficult to assess risk, it gets a high priority and is implemented in the next Sprint
 - Early clarification of actual risk
 - Fail-early approach: risks occur as early as possible

→ Contrast to many classical approaches (often ignoring risks early on)

→ Requires appropriate culture

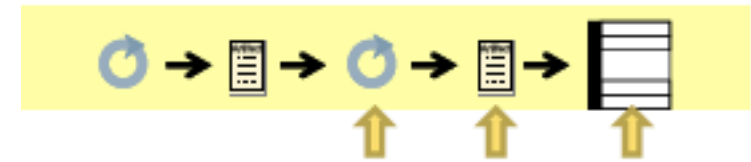
Scrum – Sprint



Sprint – Meetings

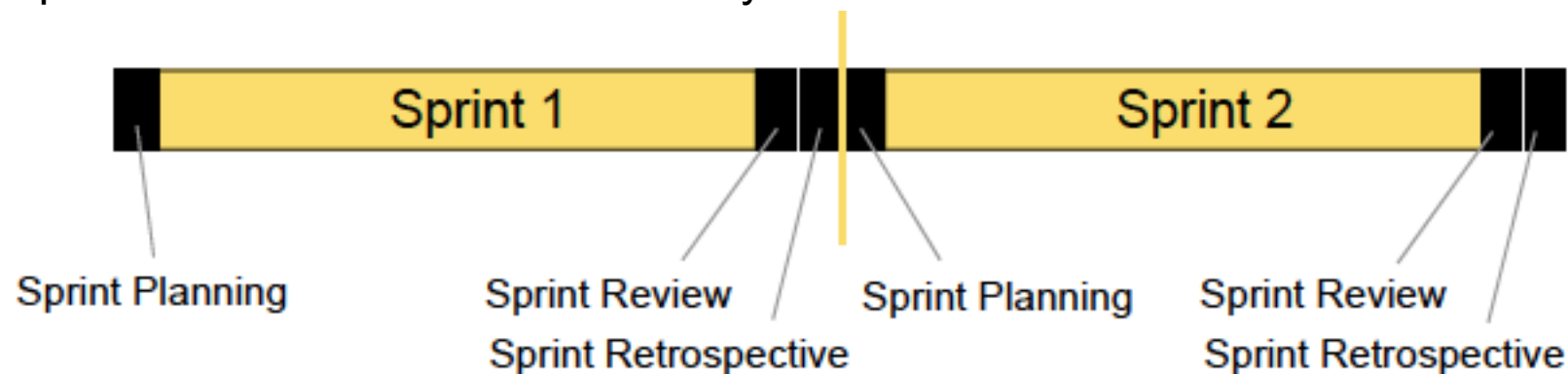
- **Daily Scrum**
 - Each day during the sprint, a project status meeting occurs. This is called a "scrum", or "the daily standup". Daily scrum guidelines:
 - The meeting starts precisely on time. Often there are team-decided punishments for tardiness (e.g. money, push-ups, hanging a rubber chicken around your neck)
 - All are welcome, but only "pigs" may speak
 - The meeting is time-boxed (15 minutes) regardless of the team's size
 - All attendees should stand (it helps to keep meeting short)
 - The meeting should happen at the same location and same time every day
 - During the meeting, each team member answers three questions:
 - What have you done since yesterday?
 - What are you planning to do by today?
 - Do you have any problems preventing you from accomplishing your goal?
 - It is the task of the ScrumMaster to remind the team of these questions.
- **Sprint Planning Meeting**
 - Select what work is to be done
 - Prepare the Sprint Backlog that details the time it will take to do that work
 - 8 hour limit
- **Sprint Review Meeting**
 - Review the work that was completed and not completed
 - Present the completed work to the stakeholders (a.k.a. "the demo")
 - Incomplete work cannot be demonstrated
 - 4 hour time limit
- **Sprint Retrospective**
 - All team members reflect on the past sprint.
 - Make continuous process improvement.
 - Two main questions are asked in the sprint retrospective: What went well during the sprint? What could be improved in the next sprint?
 - 3 hour time limit

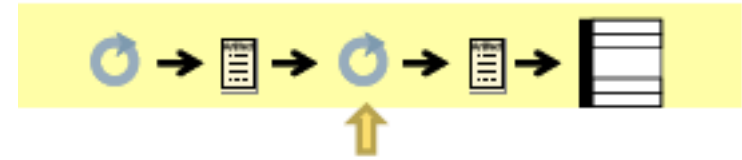




Sprint – Overview

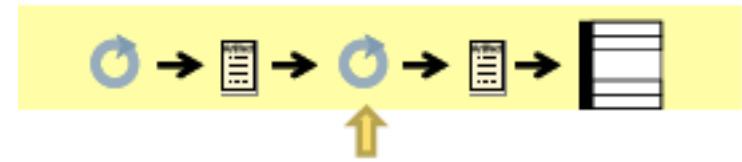
- Fixed period during which all activities specified in the Sprint Backlog are carried out
- Max Sprint length: 30 days, but can also be shorter
- No extension possible (time-boxing approach) – if not all activities can be completed by the end of the Sprint, they fall back into the Product Backlog
- Sprint is preceded by Sprint Planning and succeeded by Sprint Review and Sprint Retrospective meetings
- Sprints follow each other seamlessly





Sprint Planning

- Planning of all activities to be carried out in the next Sprint
- Product Owner defines goal for the Sprint and selects requirements to be implemented
- Team identifies necessary activities for their implementation and estimates costs
- Team decides which and how many requirements are implemented in the Sprint, based on
 - the priorities defined by the Product Owner
 - the estimated effort
 - the team's speed of development (→ productivity)
- Typically, planning is done with about 70% of the available capacity
- Results are documentation in the Sprint Backlog

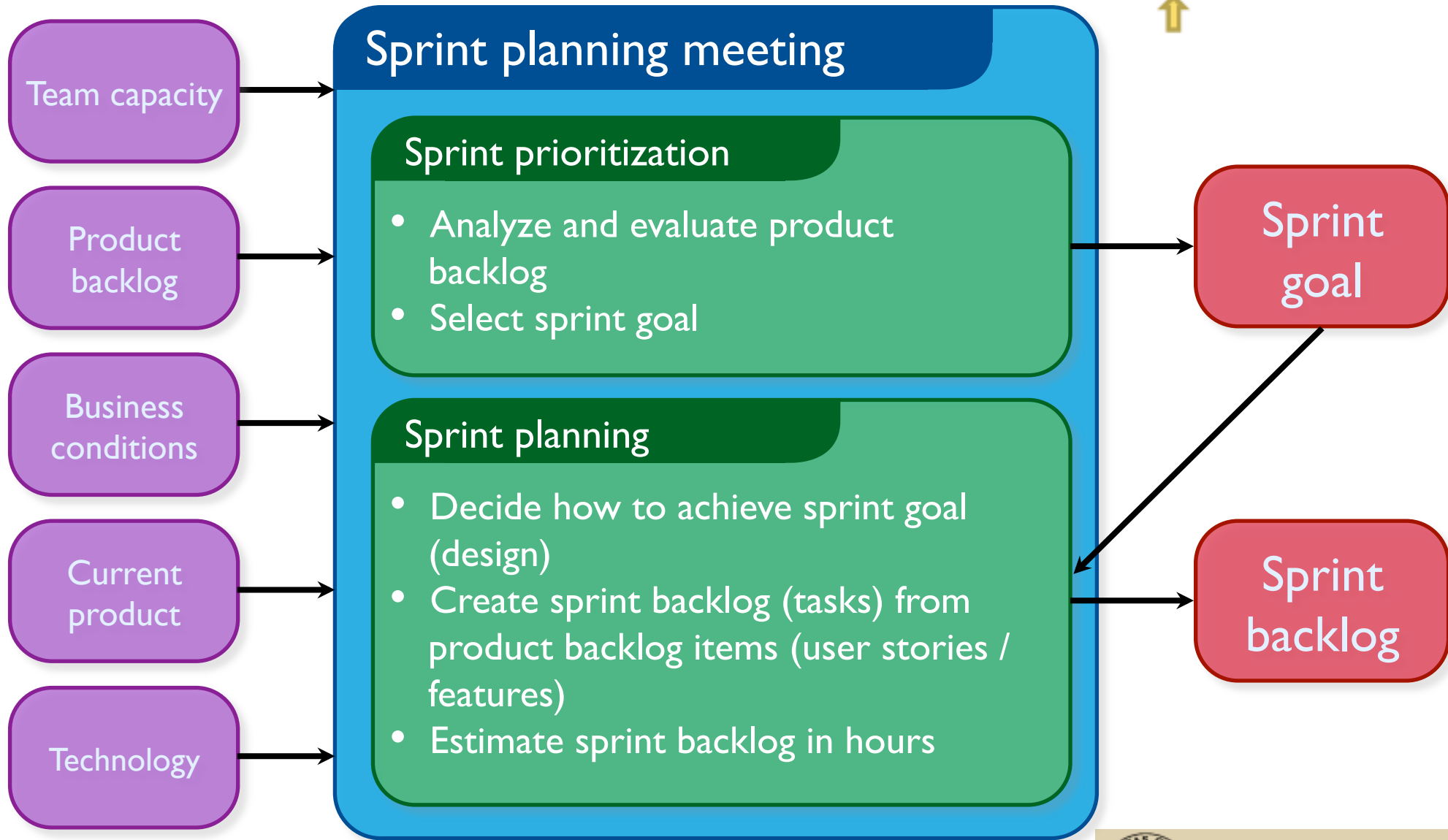


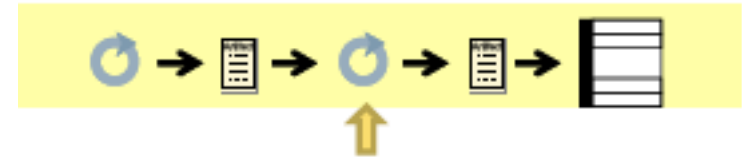
Sprint Planning Meeting

- Team selects items from the product backlog they can commit to complete
- Sprint backlog is created
 - Activities/Tasks are identified and each is estimated (1-16 hours)
 - Collaboratively, not done alone by the Scrum Master
- High-level design is considered

As a vacation planner, I want to see photos of the hotels.

Code the middle tier (8 hours)
Code the user interface (4)
Write test fixtures (4)
Code the foo class (6)
Update performance tests (4)





Sprint Goal – Examples

- A short statement of what the work will be focused on during the sprint

Database Application

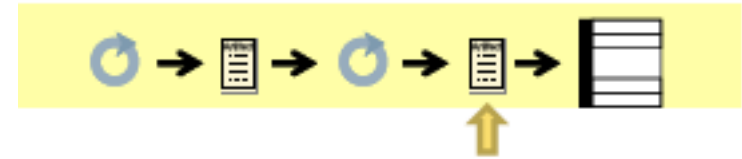
Make the application run on SQL Server in addition to Oracle.

Life Sciences

Support features necessary for population genetics studies.

Financial services

Support more technical indicators than company ABC with real-time, streaming data.



Sprint Backlog

- Created during the Sprint Planning
- Updated at least at the end of every day
- Includes all activities that have to be carried out in the Sprint
- Allows the team to organize all activities
- Usually documented as index cards on a Meta Planning Board

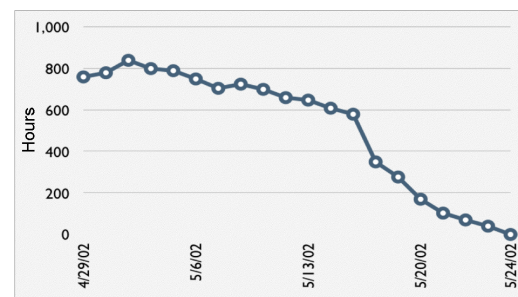
Priority	Requirement	To be Done	In Progress	Done
1				
2				
...				



Managing the Sprint Backlog

- Individuals sign up for work items (activities/tasks) of their own choosing
 - Work is never assigned!
- Estimated work remaining is updated daily
- Team can add, delete or change work items in the sprint backlog
- Work for the sprint emerges
- If work is unclear, define a sprint backlog item with a larger amount of time and break it down later
- Update work remaining as more becomes known

- Visualisation → Burn-down chart



Daily Scrum

- Daily meetings of all team members
- Up to 15 minutes, usually in the mornings
- Always same time, same place
- Daily planning of activities
- Every team member reports
 - Activities completed since the last Daily Scrum
 - Planned activities until the next Daily Scrum
 - Perceived obstacles to the implementation of the activities
- Problems in the Daily Scrum are only presented - not solved
- Right to speak for Team and Scrum Master
- Other stakeholders (including Product Owner) may attend but should only listen

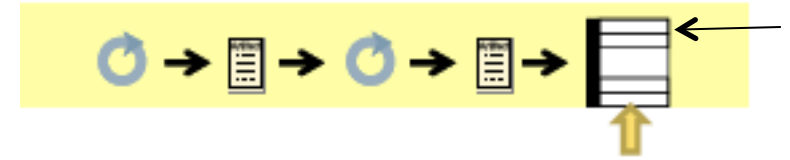
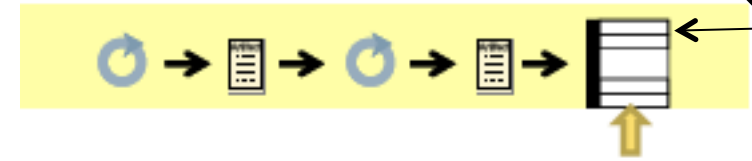
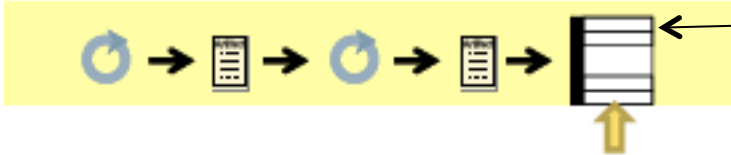


Photo: <http://wiki.ipponsoft.de/ewiki/AgileWiki.php?page=ScrumAtTOI>
Meta Planning Boards: Daily tasks of team, Sprint Backlog, and Product Backlog

Daily Scrum

- Parameters
 - Daily
 - 15-minutes
 - Stand-up
- Not for problem solving
 - Whole world is invited
 - Only team members and Scrum Master can talk
 - Other roles may attend and listen
 - Helps avoid other unnecessary meetings





Daily Scrum – 3 Questions

NB:

- These questions are not status reports for the Scrum Master
- They are commitments in front of peers

- 1 What did you do yesterday?
- 2 What will you do today?
- 3 Is anything in your way?



Development Sprint

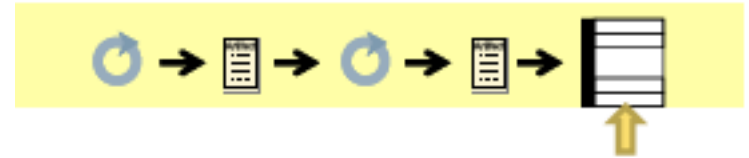
- Normal Sprint in a Scrum project
- Implementation of all activities that are described in the Sprint Backlog

- Design
- Coding
- Integration
- Test
- Documentation
- ...

Priority	Requirement	To be Done	In Progress	Done
1				
2				
...				

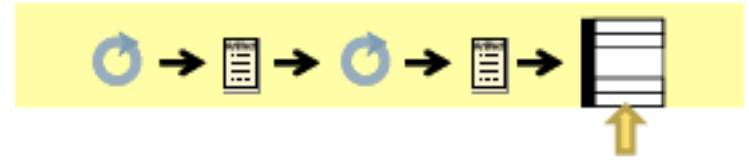


- Documented in the Sprint Backlog (activity started / completed)



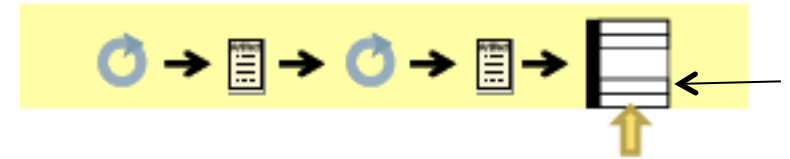
Exploration Sprint

- For developing knowledge required
- For addressing risks, for example by creating a prototype
- For determining customer needs, e.g., by creating GUI mockups
- **Important:** Clear separation between exploration result and production code
 - The exploration result is discarded, no shippable product increment is created!
- Exploration Sprints are usually shorter than normal development Sprints
- Rule of thumb: If teams spend more than 1/3 of its effort on exploration activities, an exploration Sprint should be inserted



Release Sprint

- Sprint at the end of a release cycle or the overall project
- Can combine tasks that would lead to large portions of non-productive effort in a normal Development Sprint
 - e.g., complex customer-specific configurations of the product
- Create no added value from a customer perspective (since no functionality is added)
 - use seldom, keep short



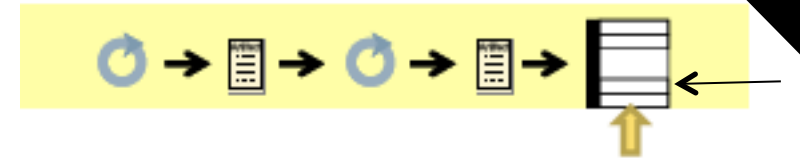
Sprint Review

- Typical duration: 1-2 hours
- Team presents all implemented requirements
 - At last the official build
 - On a test environment that is as similar as possible to the final target environment
- Only fully and accurately implemented requirements are approved (→ shippable product increment!)
 - That means, 99% implemented counts as non implemented

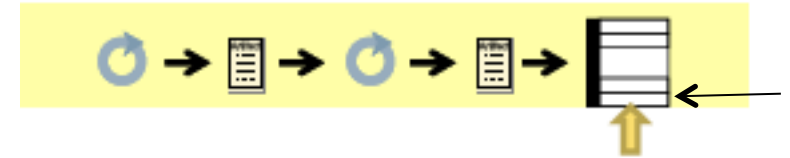
- **Goal:** Assessment of the resulting work results and approval by the Product Owner
- **Participants:**
 - Team,
 - Product Owner,
 - Scrum Master,
 - and possibly other stakeholders



Sprint Review



- Team presents what it accomplished during the sprint
- Duration: max. 2 hours
- Typically takes the form of a demo of new features or underlying architecture
- Informal
 - 2-hour prep time rule
 - No slides
- Participants: Team, Product Owner, Scrum Master
- Invite the world



Sprint Retrospective

- Directly after the Sprint Review
- Concludes the Sprint
- Typically slightly longer than the Sprint Review
- Reflection
 - What went well?
 - What has gone wrong?
 - What could be improved and how?
- **Goal:** Improve team collaboration and the application of Scrum
- **Participants:**
 - Team,
 - Product Owner,
 - Scrum Master,
 - possibly other stakeholders or managers (for the removal of obstacles in the future)



Sprint Retrospective

- Periodically take a look at what is and is not working
- Typically 15–30 minutes
- Done after every sprint
- Whole team participates
 - Scrum Master
 - Product Owner
 - Team
 - Possibly customers and others



Other Plans and Reports

- **Release plan**
 - Documents the functionality (to be) shipped in product releases
- **Speed of development report**
 - Documents development speed over Sprints
- **Sprint burn-down charts**
 - Documents Sprint progress on a daily basis
- **Obstacle Report**
 - Documents obstacles encountered
- **Theme Park**
 - Provides thematic overview on completion status
- **Final Sprint report**
 - Documents Sprint results

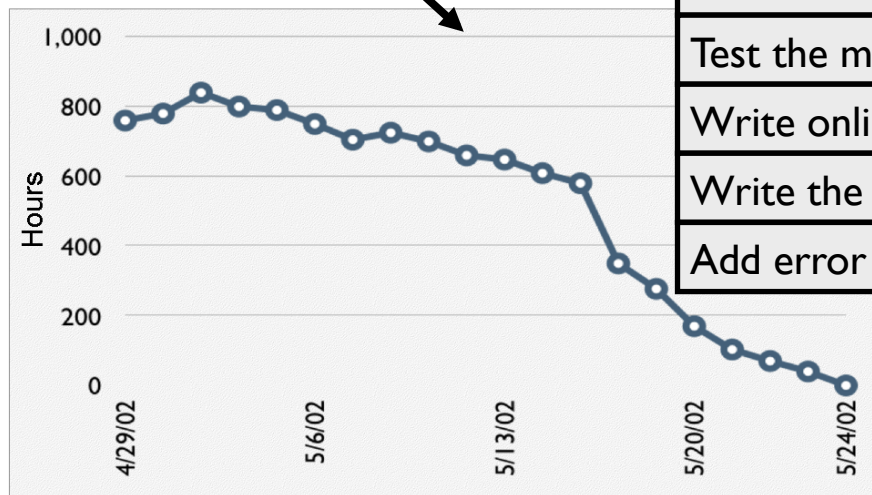


Scrum – Main Artifacts

- Product backlog
- Sprint backlog
- Burn down chart

Backlog item	Estimate
Allow a guest to make a reservation	3
As a guest, I want to cancel a reservation.	5
As a guest, I want to change the dates of a reservation.	3
As a hotel employee, I can run RevPAR reports (revenue-per-available-room)	8
Improve exception handling	8
	30

Tasks	Mon	Tues	Wed	Thur	Fri
Code the user interface	8	4	8		
Code the middle tier	16	12	10	4	
Test the middle tier	8	16	16	11	8
Write online help	12				
Write the foo class	8	8	8	8	8
Add error logging			8	4	



Scrum – Artifacts

Product backlog

- The product backlog is a high-level document for the entire project. It contains backlog items: broad descriptions of all required features, wish-list items, etc. It is the "What" that will be built. It is open and editable by anyone and contains rough estimates of both business value and development effort. Those estimates help the Product Owner to gauge the timeline and, to a limited extent, priority.
 - For example, if the "add spellcheck" and "add table support" features have the same business value, the one with the smallest development effort will probably have higher priority, because the return-on-investment is higher.
- The product backlog is property of the Product Owner. Business value is set by the Product Owner. Development effort is set by the Team.

Sprint backlog

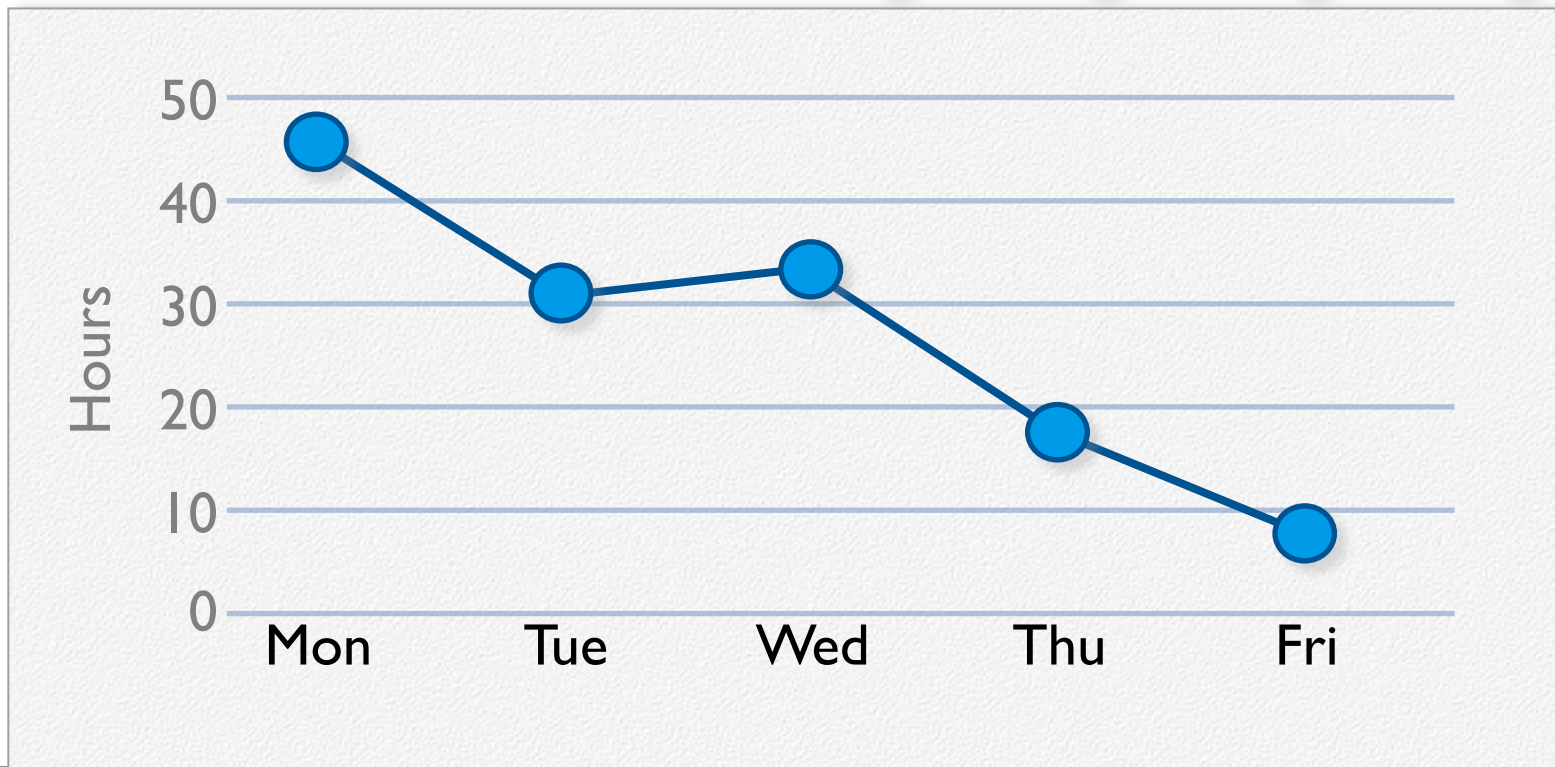
- The sprint backlog is a greatly detailed document containing information about how the team is going to implement the requirements for the upcoming sprint. Tasks are broken down into hours, with no task being more than 16 hours. If a task is greater than 16 hours, it should be broken down further. Tasks on the sprint backlog are never assigned; rather, tasks are signed up for by the team members as they like.
- The sprint backlog is property of the Team. Estimations are set by the Team.

Burn down chart

- The [burn down chart](#) is a publicly displayed chart showing remaining work in the sprint backlog. Updated every day, it gives a simple view of the sprint progress.



Tasks	Mon	Tues	Wed	Thur	Fri
Code the user interface	8	4	8		
Code the middle tier	16	12	10	7	
Test the middle tier	8	16	16	11	8
Write online help	12				



Scrum: Prerequisites and Risks/Challenges



Scrum: Prerequisites and Risks (1/2)

- Scrum has a different perspective on employees, management, distribution of power as compared to traditional project management approaches
 - In particular, higher and top-level management must understand and actively support Scrum - not a passive “Commitment”
- Organizational culture
 - Scrum requires openness, honesty, respect for each other
 - In organizations with many in-fighting this is typically missing
- Customer also must re-think their role
 - Close involvement through many iterations is often unfamiliar
 - Creates additional work on the client side
 - Not every customer wants to see the creation of the product



Scrum: Prerequisites and Risks (2/2)

- Partitioning of the product
 - Product must be partition-able so that it can actually be developed incrementally
 - For example, this is not possible in certain regulated industries that have certify full requirements specifications very early
 - Not all requirements can be partitioned equally well
 - In particular, non-functional requirements, such as performance, safety, security can hardly be partitioned – and must therefore be re-examined in each iteration and ensured
 - But: Safe + Safe \neq Safe!
 - There exists no modeling technique that can guarantee non-functional requirements iteratively (in contrast to conventional waterfall development techniques)



Scrum: Experience from Industrial Practice

- Nowadays, predominant approach in small and medium-sized companies
- Examples of large companies: Microsoft and SAP
- SAP (since 2005)
 - 6 local and 11 distributed pilot projects from October 2005 to March 2006
 - Observed value
 - Great transparency regarding project status
 - Improved communication
 - Improved: Team feeling, team creativity, team motivation
 - Improved productivity
 - Early clarification of many issues and problems
 - Nowadays more widely used within SAP
 - But, adjusted to the specific needs of SAP!



Scalability of Scrum



Scrum of Scrums of ...

- Typical individual team is 7 ± 2 people
 - Scalability comes from teams of teams
- Factors in scaling
 - Type of application
 - Team size
 - Team dispersion
 - Project duration
- Scrum has been used on multiple 500+ person projects (e.g., SAP)

Structure of Lecture 03

- Hour 1:
 - Light-weight (agile) processes / Evolutionary development
 - Extreme Programming (XP)
- Hour 2:
 - Question/answer session about homework 1
 - Info on mandatory short presentation (→ Lecture 5)
 - Question/answer session about project
- Hour 3:
 - Scrum
 - Choosing the right process (model)

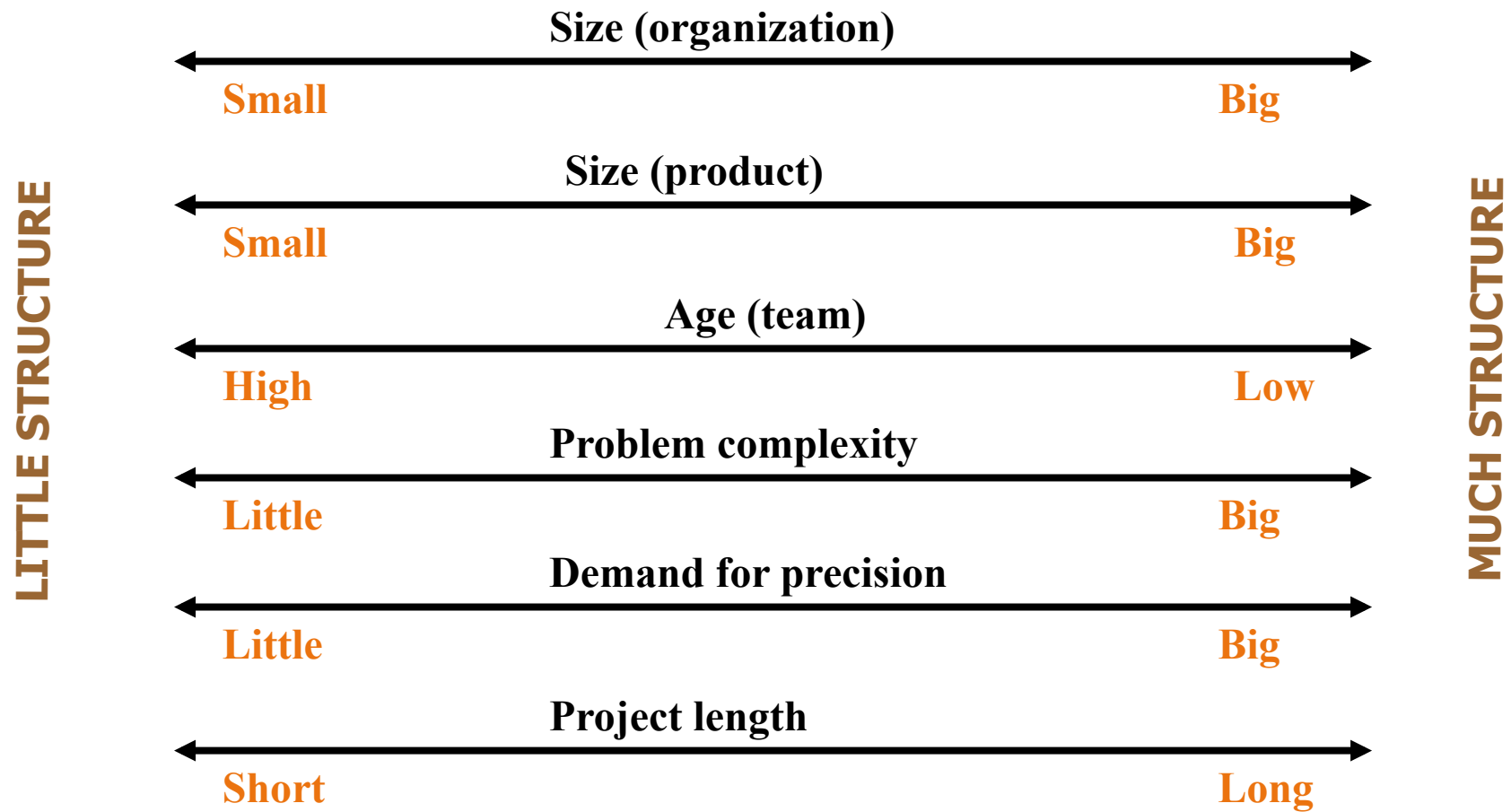


Choosing a Process Model is Difficult !

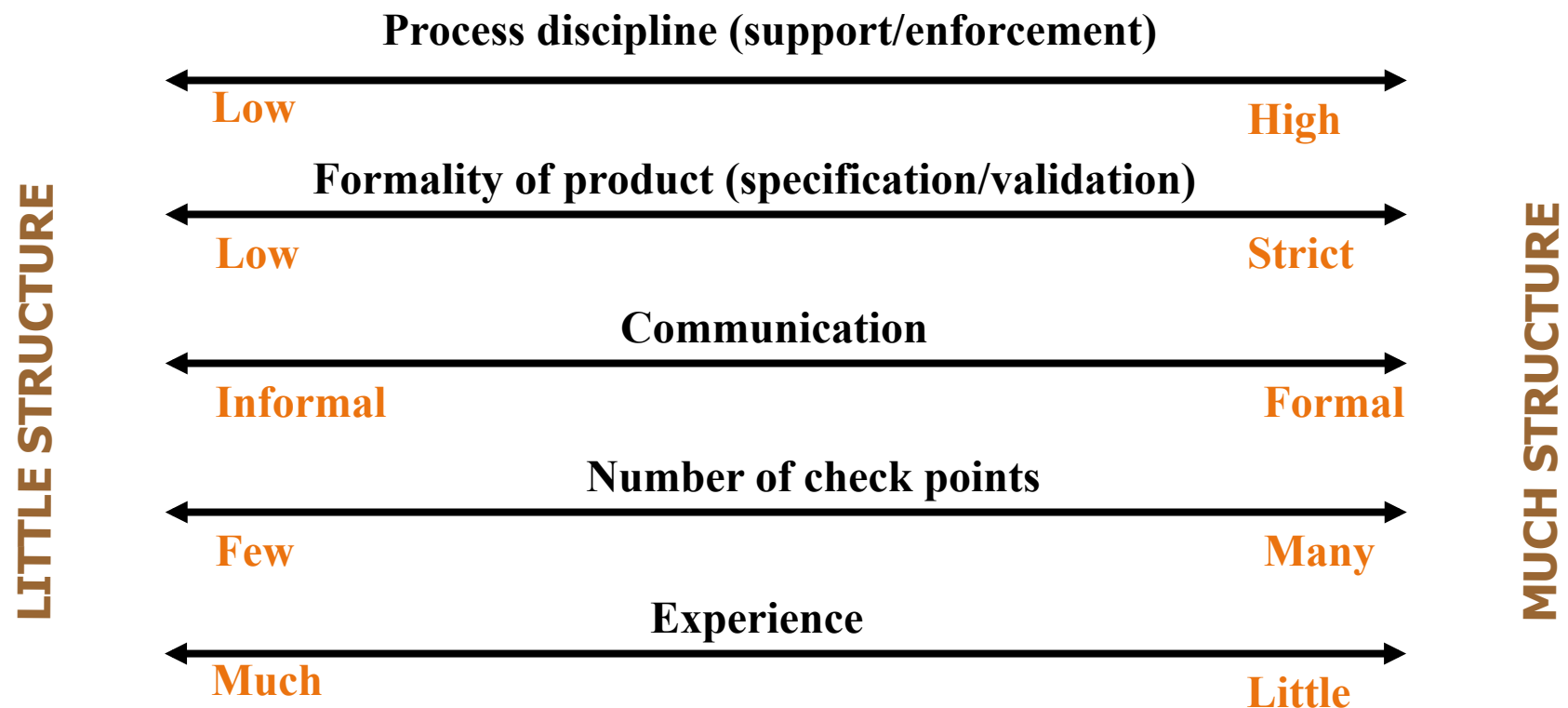
- What you should first decide is whether you actually need a prescriptive process model.
- To make the choice it is important to know your organization/
project.
 - What characteristics does the project have?
 - What characteristics affect the choice of the process model?
 - Can we use the same model everywhere, or do we need variants (a repertoire of different models)?



How Much Structure is needed?



How Much Structure is adequate?



How much Agility is Recommended?

- Source: Boehm, B.; Turner, R.; Observations on balancing discipline and agility, Proceedings of the Agile Development Conference, 2003. ADC 2003. Page(s):32-39

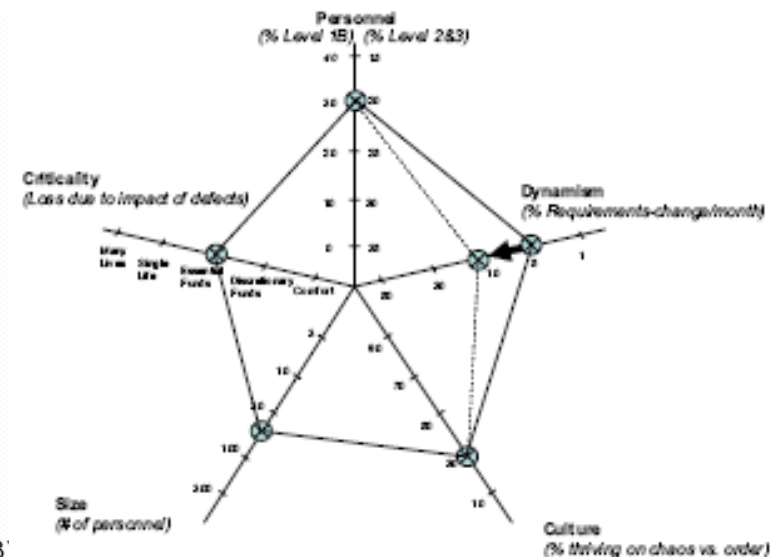
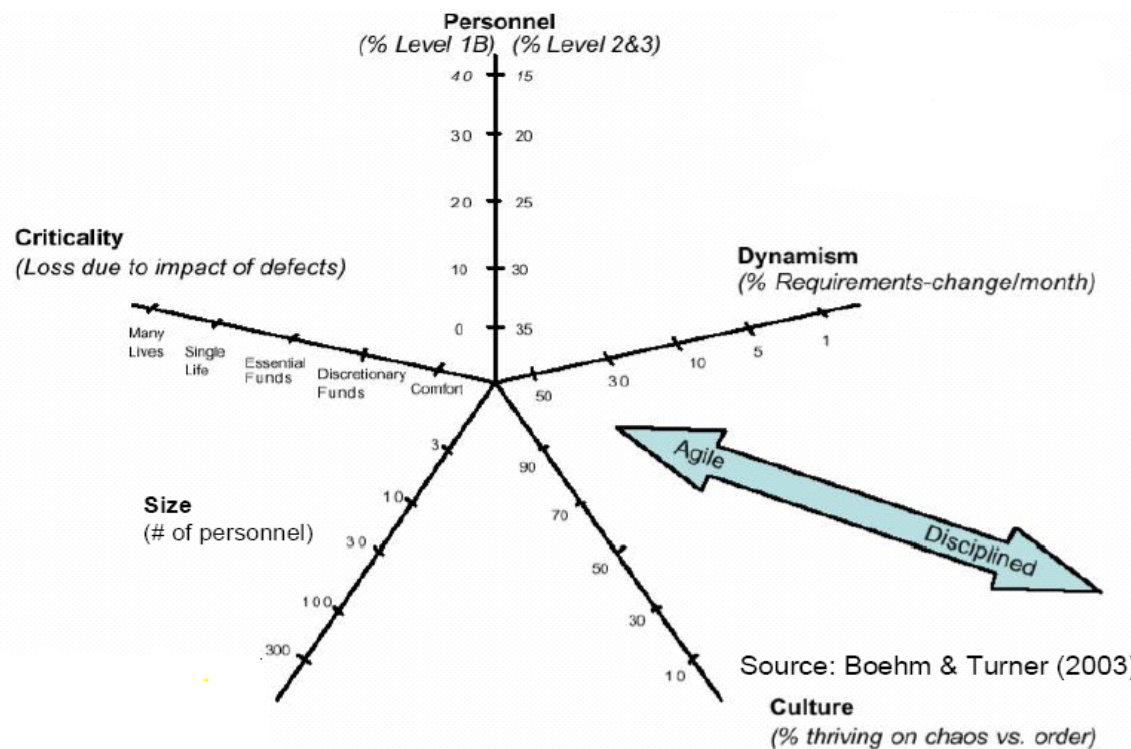


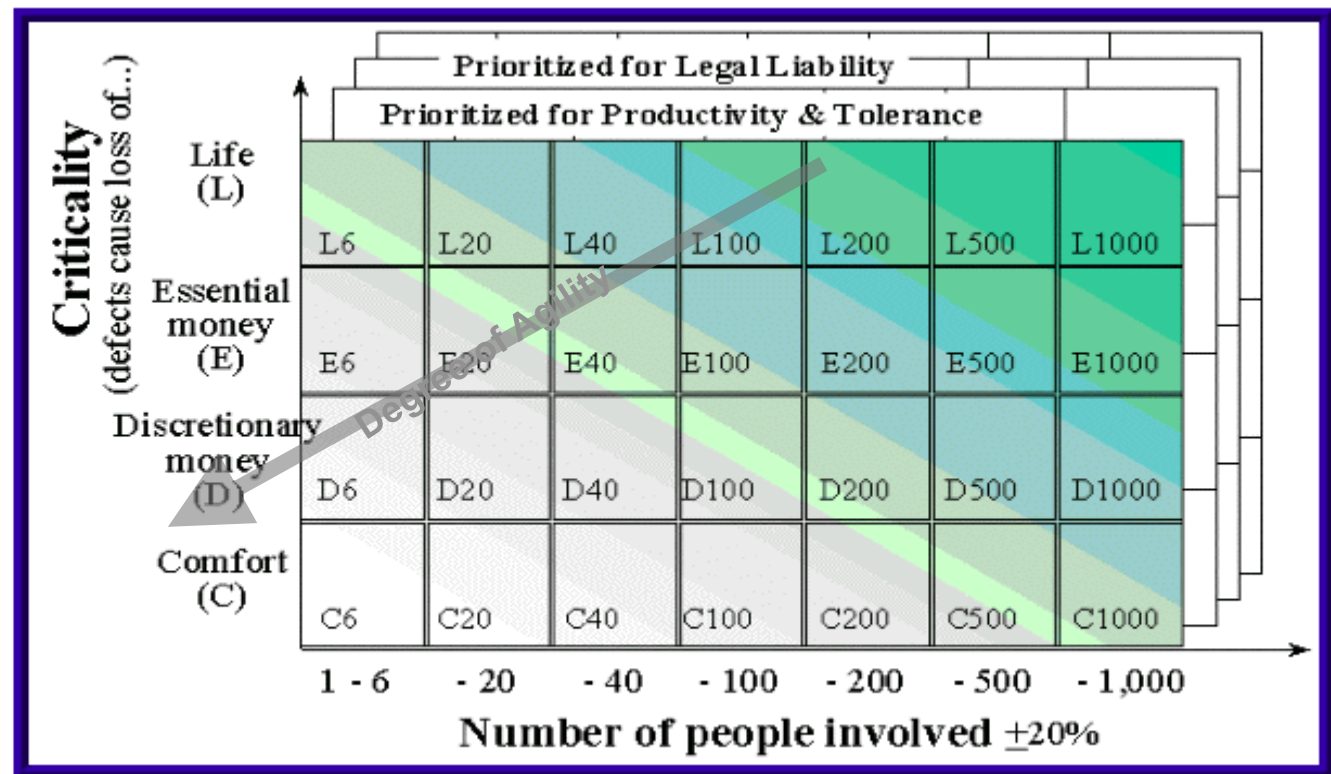
Figure 2. Sample highly-mixed profile



Alistair Cockburn – Project Categorizing

“Any one methodology is likely to be appropriate for only one of the boxes on one of the planes. Thus, at least 150 or so methodologies are needed!”

[Alistair Cockburn: Selecting a Project 's Methodology. IEEE Software 17(4): (2000)]



Next Lectures

- Topic: Lean Principles and Processes
- **Date: 13 Sep 2011**

- Topic: Student Presentations
- **Date: 20 Sep 2011**
- Important: Check course web for presenter list

- Topic: Flow-based Agile Development (KANBAN)
- **Date: 27 Sep 2011**



Bente Anda



Dag Sjøberg

