Course "Softwareprozesse"

**Agile Management Practices:**
Scrum, Lean SD, Kanban, ...

Lutz Prechelt
Freie Universität Berlin, Institut für Informatik

- Why so little XP?
- Scrum
  - Backlogs, planning, sprint, daily scrum, retrospective
- Lean Software Development (Lean SD)
  - 7 principles
- Kanban
  - 6 principles
  - flow systems
  - 6 practices
- Other sources
  - Agile Development in the Large
  - Pragmatic Programmer, ...
Learning objectives

• Understand the basic ideas of management-centric agile approaches:
  • Scrum
  • Lean software development
  • Kanban

• Understand the process structure of Scrum and Kanban

• Become aware of some other work on agile management practices
Why so little XP?

- Why are Scrum, Lean, and Kanban so much more popular and talked-about if only XP has the all-important technical practices?

- Two possible explanations:
  1. The XP technical practices have long become mainstream and a matter-of-course.
  2. Developers shy away from the technical practices because those require so much skill and discipline.

- Both explanations are partially true
  - The second more than the first
    - cf. less than 50% usage of Simple Design, TDD, or Collective Ownership

- Anyway, management practices are helpful, too.
  - So let's talk about them (Scrum and Kanban provide bundles of management practices)
Scrum

  - first formulates the Scrum ideas

- Ken Schwaber, Jeff Sutherland: "The Scrum Guide", Nov. 2017
  - the definitive version of Scrum
    - our reference
  - updated in irregular intervals
  - lots of small and not-so-small differences over time and compared to other sources. Beware!

Many other sources, e.g.:
Scrum? What a strange word!

'scrum' is a standard situation in Rugby

(Scrum's "Daily Scrum" resembles it)
What Scrum is and is not

- Scrum is an approach for managing a development process
  - Any knowledge work, not only software development
  - Scrum thus does not describe technical development activities

- Scrum's goal is *facilitating the self-organization of the Scrum Team* so that it can adapt to
  - the specifics of the project and
  - their changes over time

- Scrum is currently the most-used agile "method"
Roles in a *Scrum Team*

- **Product Owner**
  - Represents all customers, manages the *Product Backlog*
  - Sets priorities, selects stories for a *Sprint*

- **Scrum Master**
  - Responsible for ensuring a smooth execution of the Scrum process (as teacher and coach, not as a manager)
    - This role targets Development Team and Product Owner
  - Responsible for removing organizational obstacles

- **Development Team**
  - The developers (typically 3-9), viewed as a self-organizing group of technical and process experts
    - The role is Dev Team (as opposed to Scrum Team), *not* Developer!

- **Scrum Team**
  - All of the above together
  - The *Scrum Team* as a whole is responsible for product delivery
  - Larger projects can use multiple Scrum Teams
Structure of the Scrum process: Anatomy of an iteration ("Sprint")

(Scrum Master supports all activities)

Product Owner

Development Team

Sprint Planning → (development) → Daily Scrum → Sprint Review → Sprint Retrospective

Product Backlog

Sprint Goal

Sprint Backlog

Increment

Definition of "Done"

(process flow:  ➔  data flow:  →  participation:  — — — — —)

Lutz Prechelt, prechelt@inf.fu-berlin.de
More on the roles

- **Product Owner**
  - Core responsibilities:
    - Discovering and defining valuable functionality
    - Negotiating value production with the Development Team
    - Supporting the Development Team in their value production
  - Has the sole authority over the Product Backlog
    - but Sprint Goals are defined together with the Development Team
  - Has the sole authority for cancelling a Sprint in midflight

- **Scrum Master**
  - Can be a technical or non-technical person:
    - Needs to understand software development, but does not need to be able to perform it.
  - The "removing obstacles" role is particularly important if the surrounding organization is not yet agile:
    - Protect the Development Team from management interference
    - Management should target the Product Owner only
Activities: How long?

- All activities are *time-boxed*
  - which means: They can be shorter, but not longer

- Sprint length:
  - Scrum guide: "one month or less"
    - (MLS: month-long Sprint)
  - most teams use 1 week or 2 weeks
  - short Sprints require shortening planning, review, and retrospective
  - but have *many advantages*

- Sprint Planning:
  - time-boxed to 8 hrs / MLS

- Sprint Review:
  - time-boxed to 4 hrs / MLS

- Sprint Retrospective:
  - time-boxed to 4 hrs / MLS

- Daily Scrum:
  - time-boxed to 15 minutes
Sprint Planning, Sprint Goal, Backlogs

- **Product Backlog:**
  - contains requirements
  - often as Stories
  - modified only by Product Owner

- **Sprint Planning:**
  - method not prescribed
  - often like the XP Planning Game, but many variations and extensions

- **Sprint Goal:**
  - is set by Product Owner and Development Team together
  - is fixed for the Sprint
  - and defines its "topic" only

- **Sprint Backlog:**
  - initially contains stories
  - which get refined into tasks during Sprint Planning
    - by Product Owner and Development Team
  - and further refined into tasks and subtasks during development
    - by Development Team
    - during and outside the Daily Scrum

Lutz Prechelt, prechelt@inf.fu-berlin.de
The Daily Scrum

What gave Scrum its name

• Intended as catalyst for self-organization
  • coordination of development
  • report on each member's status and day plan, problems, help needs, collaboration needs

• A stand-up meeting
  • same time every day
  • everybody must attend

• Purpose often not understood by teams
  • and then hardly useful

Lutz Prechelt, prechelt@inf.fu-berlin.de
Increment, Sprint Review

Increment:
- "An increment is a body of inspectable, *done* work that supports empiricism at the end of the Sprint.
- The increment is a step toward a vision or goal.
- The increment must be in useable condition regardless of whether the Product Owner decides to release it."

Sprint Review:
- Presents, demos, explains, and inspects the Increment
- Reviews Product Backlog
  - may modify it
- Reviews strategy
  - timeline, budget, potential capabilities, and marketplace for anticipated releases

Lutz Prechelt, prechelt@inf.fu-berlin.de
Definition of "Done"

• The "Definition of Done" defines when a feature or story is considered complete.

• It describes acceptable levels of e.g.(!)
  • automated tests
  • code review
  • known defects and user-visible weaknesses
  • acceptance testing
  • documentation
  • technical debt
  • perhaps even deployment
  • and anything else relevant for this particular team

• Maintaining and strictly obeying a definition of Done is Scrum's approach to process-quality assurance
  • and a key element of product-quality assurance

• https://www.agilealliance.org/glossary/definition-of-done
Sprint Retrospective

• Sprint Review is for learning on the product level
• Retrospective is for learning on the process level
• Topics:
  • What went well? How can we optimize it?
  • What went not well? How should we improve it?
  • What did not work at all? Why? What do we do about that?
  • What is the status of difficult changes decided in earlier retros?

• Many teams suffer from the "Elephant in the room" effect:
  • Initial retros are helpful and produce progress,
  • but after easy problems are solved, nobody talks about the most important problem,
    • it is just too hard.
• Result: Little process improvement
  • and self-organization at a tactical level only
• Retrospectives practices may help
In practice, different sets of technical practices are used by different Scrum teams

- often too few
Scaling Scrum

• Ken Schwaber claims he has coached a project using Scrum that took 2.5 years and had 3500 participants overall.

• The technique to do this is the "Scrum of Scrums":
  • One participant of each daily Scrum is sent of the daily Scrum-of-Scrums on a second project-level.
  • This scales Scrum from 10 up to 100 participants. A third level could scale up to 1000.
  • It is difficult to get this to work.
    • see PasLasHei12
Lean Software Development

• Mary and Tom Poppendieck: "Lean Software Development: An Agile Toolkit", Addison-Wesley 2003

• Mary and Tom Poppendieck: "The Lean Mindset: Ask the Right Questions", Addison-Wesley 2013

• http://www.poppendieck.com

Mary Poppendieck  Tom Poppendieck
Lean SD principles

- Based on Toyota's principles of **Lean Production**
  - a holistic approach to optimizing cost and quality:
    - a philosophy and set of principles
    - a set of more-or-less (typically less) concrete techniques
    - but not a complete, prescriptive method

- Principles of Lean Software Development:
  1. Eliminate waste
  2. Build quality in
  3. Create knowledge
  4. Defer commitment
  5. Deliver fast
  6. Respect people
  7. Optimize the whole
Lean SD: Eliminate Waste, Build Quality In

• **Eliminate Waste.** The three biggest wastes in SW dev. are:
  • **Extra Features:** We need a process which allows us to develop just those 20% of the features that give 80% of the value.
  • **Churn:** If you have requirements churn, you are specifying too early. If you have test and fix cycles, you are testing too late.
  • **Crossing Boundaries:** Organizational boundaries typically increase cost by over 25%; they interfere with communication.

• **Build Quality In.** If you routinely find defects during verification, your development process is defective.
  • **Mistake-Proof Code with Test-Driven Development:** Write executable specifications instead of requirements.
  • **Stop Building Legacy Code:** Legacy code is code that lacks automated unit and acceptance tests.
  • **The Big Bang is Obsolete:** Use continuous integration and nested synchronization.
Lean SD: Create Knowledge

- **Create Knowledge.** Planning is useful. Learning is essential.
  - **Use the Scientific Method:** Teach teams to establish hypotheses, conduct many rapid experiments, create concise documentation, and implement the best alternative.
  - **Standards Exist to be Challenged and Improved:** Embody the current best known practice in standards that everyone follows. Encourage everyone to challenge the standards.
  - **Predictable Performance is Driven by Feedback:** A predictable organization does not guess about the future and call it a plan; it develops the capacity to rapidly respond to the future as it unfolds.
Lean SD: Defer Commitment

- **Defer Commitment:**
  Abolish the idea that it is a good idea to start development with a complete specification.
  
  - **Break Dependencies:**
    System architecture should support the addition of any feature at any time.
  
  - **Maintain Options:**
    Think of code as an experiment – make it change-tolerant.
  
  - **Schedule Irreversible Decisions at the Last Responsible Moment:**
    Learn as much as possible before making irreversible decisions.
Lean SD: Deliver Fast

- **Deliver Fast.** Lists and queues are buffers between organizations that simply slow things down.
  - **Rapid Delivery, High Quality, and Low Cost are Fully Compatible:** Companies that compete on the basis of speed have a big cost advantage, are more attuned to their customers' needs, and deliver superior quality.
  - **Queuing Theory Applies to Development, not Just Servers:** Focusing on utilization creates a traffic jam that actually reduces utilization. Drive down cycle time with small batches and fewer things-in-process.
  - **Limit Work to Capacity:** Establish a reliable, repeatable velocity with iterative development. Aggressively limit the size of lists and queues to your capacity to deliver.
Lean SD: Respect People

- **Respect People.** Engaged, thinking people provide the most sustainable competitive advantage.
  - **Teams Thrive on Pride, Commitment, Trust, and Applause:** What makes a team? Members mutually committed to achieve a common goal.
  - **Provide Effective Leadership:** Effective teams have effective leaders who bring out the best in the team.
  - **Respect Partners:** Allegiance to the joint venture must never create a conflict of interest.
Lean SD: Optimize the Whole

- **Optimize the Whole.**
  Brilliant products emerge from a unique combination of opportunity and technology.
  
  - **Focus on the Entire Value Stream:**
    from concept to cash,
    from customer request to deployed software.
  
  - **Deliver a Complete Product:**
    Develop a complete product, not just software. Complete products are built by complete teams.
  
  - **Measure Up:**
    Measure process capability with cycle time.
    Measure team performance with delivered business value.
    Measure customer satisfaction with a net promoter score.
Kanban

- Based on ideas from the **Toyota Production System**
  - and later **Lean Production**
- **Many variants**, plenty of chic terminology/jargon
  - for any type of knowledge work, not only SW dev.
  - Therefore, no technical practices; only management practices
- In line with Lean SD,
  - less specific in most places,
  - and more specific in a few (flow system, cadences).
- Compatible with Scrum
  - see **Scrumban**

- Two key ideas:
  - Start with what you are already doing, then improve incrementally
  - Manage the value-creation flow
- Thus, Kanban is more a set of ideas than a fixed-shape method with specific practices
  - although it offers many ideas for concrete practices
- We will use **D. Anderson, A. Carmichael: "Essential Kanban Condensed", 2016** as our reference source
Classification, values, "agendas"

- Kanban is
  - a method for delivering and improving knowledge work via a flow system;
  - a catalyst for rapid and focused change;
  - based on making intangible aspects visible

- 9 values:
  - transparency; balance; collaboration; flow; customer focus; leadership; understanding; agreement; respect

- 3 agendas (purpose of change):
  - inward: sustainability
  - outward: service orientation
  - futureward: survivability
6 Kanban principles

- Change management:
  1. Start with what you do now
  2. Improve through evolutionary change
  3. Encourage leadership at every level

- Service delivery:
  4. Focus on customer needs
  5. Manage work (not people); let people self-organize around it
  6. Evolve policies
The flow system

- The heart of Kanban is "managing flow"
  - flow: How a valuable function progresses from idea to delivered SW
  - lead time: How long this takes
- Low lead times mean high agility
  - can react to a customer need fast
- Lead times get lower if not too many items are being worked on at any time
  - also reduces half-products, hence avoids waste

Flow system ideas:
- Partition flow into N successive steps
- Limit the capacity for each step appropriately
  - "work in progress" (WiP) limit
- Visualize current status on a "Kanban board"
- As a result...
  - ...work is pulled by consumers, not pushed by producers
  - ...blockages are easily detected (and removed)
  - so work flows smoothly
Example Kanban board

Step    WiP limit    sub-state    work item

- Step
- WiP limit
- sub-state
- work item

**POOL OF IDEAS**

- Ongoing
- Ready

**PROPOSALS**

- 4

**SELECTED**

- 4

**DEVELOPMENT**

- Ongoing
- Ready

**ACCEPTANCE**

- 4

**COMPLETE**

**WORK IN PROGRESS**

**LEAD TIME**

**DELIVERY RATE**

**COMMITMENT**

**DELIVERY**

**DISCARDED**

Lutz Prechelt, prechelt@inf.fu-berlin.de
Work-in-progress limitation in action

We’ll do D! No, wait… that would break the Kanban limit of 2!

C is done! Great, hope you start K soon!

Sure, as soon as we sort out the problem with A.

I want F & G as well. But the Kanban limit stops me. Hmmmm…

We don’t need any more hands right now. But this is a recurring problem, so write a test for it to avoid the problem in the future!
Visualize the workflow: A real Kanban board

WiP limits??

Lutz Prechelt, prechelt@inf.fu-berlin.de
6 Kanban practices

• limit WiP

• manage flow:
  • goals: delivery rates high and smooth, lead time minimal
  • methods: identify bottlenecks; apply cost-of-delay archetypes

• visualize:
  • e.g. WiP limits and step policies on kanban board
  • e.g. history by data plots

• make policies explicit:
  • sparse, simple, visible, well-defined, always applied, and (importantly) readily changeable;
  • topics: WiP limits, capacity allocation, definition of Done, ...

• implement feedback loops:
  • e.g. by cadences (cyclical meetings)

• improve collaboratively, evolve by experiments:
  • "it can be useful to employ models and the scientific method"

Lutz Prechelt, prechelt@inf.fu-berlin.de
"Cadences": meetings that establish feedback loops

similar to:
Sprint Review
Sprint Planning
Daily Scrum
## Kanban benefits found in empirical research

<table>
<thead>
<tr>
<th>#</th>
<th>Reported benefit</th>
<th>Primary study</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Improve visibility and transparency</td>
<td>P1, P2, P3, P4, P5, P6, P7, P11, P13, P14, P15, P17, P19, P20, P22, P23</td>
</tr>
<tr>
<td>2</td>
<td>Better control of project activities and tasks</td>
<td>P1, P2, P5, P9, P10, P11, P13, P15, P19, P20, P22, P23</td>
</tr>
<tr>
<td>3</td>
<td>Identify impediments to flow</td>
<td>P1, P2, P3, P5, P9, P15, P17, P20, P22, P23</td>
</tr>
<tr>
<td>4</td>
<td>Improve workflow</td>
<td>P2, P4, P6, P11, P16, P19, P20</td>
</tr>
<tr>
<td>5</td>
<td>Faster time-to-market</td>
<td>P6, P7, P10, P16, P23</td>
</tr>
<tr>
<td>6</td>
<td>Improve prioritisation of products and tasks</td>
<td>P1, P3, P15, P17</td>
</tr>
<tr>
<td>7</td>
<td>Decrease defects and bugs</td>
<td>P2, P7, P14, P21</td>
</tr>
<tr>
<td>8</td>
<td>Improve quality</td>
<td>P6, P7, P16, P17</td>
</tr>
<tr>
<td>9</td>
<td>A lightweight intuitive method</td>
<td>P14, P15, P16, P17</td>
</tr>
<tr>
<td>10</td>
<td>Improve communication and collaboration</td>
<td>P1, P4, P6, P7, P9, P14, P17</td>
</tr>
<tr>
<td>11</td>
<td>Improve team motivation</td>
<td>P4, P6, P11, P16, P17, P19</td>
</tr>
<tr>
<td>12</td>
<td>Team building and cohesion</td>
<td>P5, P7, P17, P20, P23</td>
</tr>
<tr>
<td>13</td>
<td>Increase customer satisfaction</td>
<td>P6, P7, P14, P15, P17, P20</td>
</tr>
<tr>
<td>14</td>
<td>Promoting a culture of continuous learning</td>
<td>P7, P10, P16, P20, P23</td>
</tr>
<tr>
<td>15</td>
<td>Strategic alignment</td>
<td>P3, P5, P7</td>
</tr>
</tbody>
</table>

Beware of varying frames of reference!

M.O. Ahmad, D. Dennehy, K. Conboy, M. Oivo: Kanban in software engineering: A systematic mapping study, Journal of Systems and Software 2018
Other method-oids

Agile development in the large

  - "Agile Software Development in the Large: Diving into the Deep", Dorset House B&T 2004
- [http://www.jeckstein.de/](http://www.jeckstein.de/)
- [http://www.agilebuch.de/](http://www.agilebuch.de/)

- How to scale agile devmt. to 30-200 people:
  - Using explicit "communication teams"
  - Coping with virtual and distributed teams
  - Handling the surrounding organization
    - see next slide

Jutta Eckstein
Agile development in the large (2)

- Handling the surrounding organization:
  - Talk early to people unfamiliar with Agile Development, such as
    - project planning and control departments,
    - the Method Police (process quality assurance group),
    - the Tool Support group
    - if relevant: Human Resources, Legal, Marketing
  - Integrate the QA department (if any) into the project
  - Integrate the Operations department into the project
  - Larger organizations tend to have higher fractions of below-average developers
    - To compensate for that, work towards a Learning Organization
  - Make learning materials part of the project deliverables
    - always to be kept consistent, part of acceptance testing
  - Handle insourcing, outsourcing, part-time employees
- The book ends with a case-story of a complex project
  - Perhaps the most useful part of the book!
Agile variants of **Rational Unified Process (RUP)**:

- include technical and management practices

- Project-specific variants
  - formed by leaving out many RUP process elements and executing the rest with an agile mindset

- dX
  - RUP in XP mode: A minimal version of RUP resembling XP

- Agile modeling
  - Not a full process, just an approach to modeling
  - Based on 11 practices in four categories: Iterative and Incremental Modeling, Teamwork, Simplicity, Validation

- ...
The Pragmatic Programmer

- Andrew Hunt, David Thomas: "The Pragmatic Programmer: From Journeyman to Master", Addison-Wesley 1999
- 70 "tips" of advice and practices for individuals, not teams
- Management advice has to do a lot with discipline, attitude, and practical responsibility
The Pragmatic Programmer (2)

- **Principles:**
  - Take responsibility for what you do.
    - Think in terms of solutions, not of excuses.
  - Don't just accept bad design or coding – improve them
  - Actively introduce process changes where necessary
  - Create software that delights your customer – and then stop
  - Automate
  - Broaden your knowledge. Learn. Improve yourself.
  - Improve your self and your communication skills
Still other methods

For large projects:
- **SAFe**: Scaled Agile Framework
  - orchestrates many teams
  - somewhat popular
- **DSDM**: Dynamic Systems Development Method
  - strict time-boxing
- **FDD**: Feature-driven development
  - modeling, parallel feature teams, individual code ownership, ...

For small projects:
- **Crystal Clear**
  - minimal framework for easy introduction and maximum freedom for the team

Lutz Prechelt, prechelt@inf.fu-berlin.de
Summary

- Scrum and Kanban provide agile management practices
- Both aim at delivering valuable functionality frequently
- Scrum centers on self-organization
- Kanban centers on continuous process improvement
  - and emphasizes visualization
- Lean Software Development provides a strong formulation of many ideas from which to build agile methods

- Understanding the commonalities of these three is a good foundation for implementing agile ideas truthfully

- There are many other agile methods besides these
  - a few (such as SAFe) even somewhat well-known
Thank you!
Old slides
Crystal Clear, The Crystal Light family

  - Contains a sketch of Crystal Orange (in Ch.4)

- **Crystal Light** is meant to be a family of methods for different project sizes and criticalities
  - Clear, Yellow, Orange, Red, Blue, "and so on"
  - Only Crystal Clear has been spelled out
  - and can be taken seriously
  - Other books may or may not be forthcoming
    - probably not
Crystal Clear
Goals and Practices

http://alistair.cockburn.us/index.php/Crystal_Clear_distilled

• "Crystal Clear is a highly optimized way to use a small, colocated team,
  • prioritizing for safety in delivering a satisfactory outcome,
  • efficiency in development, and
  • habitability of the working conventions."

• Brief description of Crystal Clear:
  • "The lead designer and two to seven other developers
  • ... in a large room or adjacent rooms,
  • ... using information radiators such as whiteboards or flip charts,
  • ... having easy access to expert users,
  • ... distractions kept away,
  • ... deliver running, tested, usable code to the users
  • ... every month or two (quarterly at worst),
  • ... reflecting and adjusting their working conventions periodically"
"The people set in place the safety properties below using the techniques they feel appropriate.

- The first three properties are required in Crystal Clear;
- the next four get the team further into the safety zone.

1. Frequent Delivery
2. Reflective Improvement
3. Osmotic Communication
4. Personal Safety
5. Focus
6. Easy Access to Expert Users
7. A Technical Environment with Automated Tests, Configuration Management, and Frequent Integration"

sort of a bare-bones summary of Agile
Crystal process improvement technique: Reflection workshop

- Hang a flipchart
- Fill in the chart
  - 30 minutes
- Hang the chart in a public, visible, frequently seen place!

- Try the ideas
- Repeat each month or after each iteration

<table>
<thead>
<tr>
<th>Keep these</th>
<th>Try these</th>
</tr>
</thead>
<tbody>
<tr>
<td>test lock-down</td>
<td>pair testing</td>
</tr>
<tr>
<td>quiet time</td>
<td>fines for interruptions</td>
</tr>
<tr>
<td>daily meetings</td>
<td>programmers help testers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problems</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>too many interruptions</td>
<td></td>
</tr>
<tr>
<td>shipping buggy code</td>
<td></td>
</tr>
</tbody>
</table>

(Headings are part of the chart. Entries are examples only.)
Crystal Clear vs. XP

http://alistair.cockburn.us/index.php/Crystal_light_methods

- Crystal is based on developers' maximum individual preference
- XP is based on having everyone follow disciplined practices

- XP pursues greater productivity through increased discipline, but is harder for a team to follow:
  - Crystal Clear permits greater individuality within the team, and more relaxed work habits, for some loss in productivity.
  - Crystal Clear should be easier for a team to adopt, but XP produces better results if the team can follow it.
  - A team can start with Crystal Clear and move up to XP later.
  - A team that falls off XP can back up to Crystal Clear.
Scrum center of attention: The Sprint

- During a Sprint, requirements are fixed, but the process is not.
  - Team may adapt anything as needed
Scrum artifacts

- **Product:** *Sprint Backlog List* (fine-grained task list)
- **Product:** *Product Backlog List*
  - Collects all requirements that are currently known
    - Including priorities and effort estimates
  - Can be updated at any time (by any stakeholder)
- **Activity:** *Retrospective*
  - A postmortem after each sprint: optimize process and approach
- **Activity:** *Sprint* (followed by the *Sprint Review*)
  - The unit of iterative development, addressing
  - usually 2-5 customer-chosen requirements (→ Product Backlog)
  - and taking a fixed time (usually one month)
  - for doing analysis, design, implementation, testing
- **Process definition:** *Current Approach*
  - Technology, Architecture, Conventions, Resources
  - Can be modified at any time

Scrum artifacts

- **Product:** *Sprint Backlog List* (fine-grained task list)
  - Tasks for the current sprint
- **Product:** *Product Backlog List*
  - Collects all requirements that are currently known
    - Including priorities and effort estimates
  - Can be updated at any time (by any stakeholder)
- **Activity:** *Retrospective*
  - A postmortem after each sprint: optimize process and approach
- **Activity:** *Sprint* (followed by the *Sprint Review*)
  - The unit of iterative development, addressing
  - usually 2-5 customer-chosen requirements (→ Product Backlog)
  - and taking a fixed time (usually one month)
  - for doing analysis, design, implementation, testing
- **Process definition:** *Current Approach*
  - Technology, Architecture, Conventions, Resources
  - Can be modified at any time

Kanban

• Kanban: Japanese for "signboard" (i.e. a kanban is a card)
• Originates from Toyota Production System ca. 1950
  • is an application of Lean principles

• The core principle is evolutionary improvement in small steps
  • valid for both process and product
• The core metaphor is the work-flow
  • from upstream to downstream

• War cry:
  • Waterfall: "Never change a running system"
  • Kanban: "Always run a changing system"

• http://www.infoq.com/articles/hiranabe-lean-agile-kanban
Kanban principles

1. **Visualize the workflow**
   - because good overview is needed for efficient improvements

2. **Limit work-in-progress**
   - to limit complexity, minimize waste, reduce cycle time, and establish a predictable development speed (velocity)
   - buzzword: "pull, not push" (the crucial point is a limited buffer)

3. **Manage flow**: monitor, measure, report
   - to evaluate process improvements

and also:

- Spell out process rules
  - a corollary of "visualize the workflow": agreeing on changes requires a common process view

- Improve the process by using the scientific method
  - theorize, predict, experiment, validate
Visualize the workflow (perhaps on multiple levels)

http://www.infoq.com/articles/agile-kanban-boards
Rational Unified Process (RUP)

- Philippe Kruchten, Ivar Jacobson, et al.
- There is a substantial number of books about RUP
- A number of RUP variants exist

Philippe Kruchten

Ivar Jacobson
Rational Unified Process (RUP)

- RUP is a huge process
  - targeted mainly at large projects

- It is built around modeling (using UML) and tool-centric, object-oriented, component-based software construction
  - and other "best practices"

- It is normally considered a rather heavyweight process, but can be instantiated as an agile one
  - (appropriate when substantial upfront design is needed)
  - RUP is inherently iterative in any case
  - Full RUP has more than 100 different product types
  - Tailoring is left to the user (but supported by tools)
Rational Unified Process: Dimensions

RUP has three dimensions:

1. A set of best practices
2. 4 lifecycle phases

3. A number of process areas (called 'disciplines') and corresponding workflows