Course "Spezielle Themen der Softwaretechnik"

Agile Methods: Scrum, Crystal, Lean SD, ...

Lutz Prechelt
Freie Universität Berlin, Institut für Informatik
http://www.inf.fu-berlin.de/inst/ag-se/

- Scrum
  - The daily scrum
- The Crystal Light family
  - Crystal Clear
- Feature-Driven Development (FDD)
- Lean Software Development (Lean SD)

- Adaptive Software Development (ASD)
- Rational Unified Process (RUP)
- Dynamic Systems Development (DSDM)
- Pragmatic Programmer
Learning objectives

• Understand the basic ideas, strengths, and application scope of several other agile approaches

• Thereby get an overview of the methods space of agile methods overall
More agile methods

- Scrum
  - Ken Schwaber

- Crystal
  - Alistair Cockburn

- Feature-Driven Development (FDD)
  - Coad, Palmer, Felsing

- Lean Software Development
  - Mary and Tom Poppendieck

- Adaptive Software Development (ASD)
  - Jim Highsmith

- Rational Unified Process (RUP)
  - Philippe Kruchten, Ivar Jacobsen, et al.

- Dynamic Systems Development Method (DSDM)
  - DSDM consortium

- Agile development in the large
  - Jutta Eckstein

- The Pragmatic Programmer
  - Andrew Hunt, David Thomas

(in a rather random order)
Source

- For an overview and comparison of several agile methods, see Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, Juhani Warsta: "Agile Software Development Methods: Review and Analysis", VTT Publications 478

- A 112-page technical report that describes several methods
  - XP, Scrum, Crystal, FDD, RUP, DSDM, ASD, and Open Source development,
- in a somewhat uniform way:
  - Process, Roles and responsibilities, Practices, Adoption and experiences, Scope of use, Current research
  - Much of the following (including several graphics) is taken from this report
- Very useful reading!
Scrum

• Ken Schwaber, Mike Beedle: "Agile Software Development with Scrum", Prentice Hall 2001
• Ken Schwaber: "Agile Project Management with Scrum", Microsoft Press 2004
• http://www.controlchaos.com/
Scrum? What a strange word!

'scrum' is a standard situation in Rugby
Scrum basics

- Scrum is an approach for managing a development process
  - not only for software development
- It does not describe technical development activities

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

- **Product Owner**
  - Represents all customers, manages the Product Backlog
  - Sets priorities, selects requirements 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 both Team and Product Owner
  - Responsible for removing organizational obstacles
  - Master and Team together are responsible for product delivery

- **Team**
  - The developers (typically 5-9), viewed as a self-organizing group of technical and process experts
  - Larger projects can use multiple teams
  - Note the role is team, not developer!

- Sometimes, the Scrum Master will be Product Owner or Team member, too. This produces conflict, but is possible.
Scrum process elements

- **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: Sprint**
  - The unit of iterative development, addressing
  - usually 2-5 customer-chosen requirements (the Sprint Backlog)
  - and taking a fixed time (usually one month)
  - for doing analysis, design, implementation, testing

- **Product: Sprint Backlog List**

- **?: Current Approach**
  - Technology, Architecture, Conventions, Resources
  - Can be modified at any time, typically before a Sprint

- **Activity: Sprint review meeting**
  - A postmortem for process and approach adaptation
Scrum process elements: The Daily Scrum

A (perhaps the) key feature of the Scrum process:

- A Scrum Team holds a daily meeting to say and hear
  - what has been done,
  - what is to be done,
  - what is problematic and who could help,
  - what adjustments might be needed to succeed with the Sprint.
- The meeting is strictly limited to 15 minutes
  - and is performed standing up rather than sitting down
Scrum center of attention: The Sprint

- During a Sprint, requirements are fixed, but the process it not
  - Daily scrum may adapt anything as needed
Scrum engineering techniques

- Scrum itself is a management method, not an engineering method

- However, it is compatible with any engineering approach that can be applied in monthly iterations

- Scrum is often combined with XP practices
  - Scrum replaces/extends the planning game
Scaling Scrum

• Ken Schwaber 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
  • If necessary, a third level could scale up to 1000.
The Crystal Light family

  - Contains a sketch of *Crystal Orange* (in Ch.4)
  - Other books may or may not be forthcoming

- *Crystal Light* is a family of methods for different project sizes and criticalities
  - Each tries to be as concrete as possible to be used as a template
  - Project size is measured by the number of people required
  - Criticality is measured by the loss incurred if requirements or implementation are not correct
Crystal Light criticality levels

- Criticality is measured by possible loss incurred by a failure:
  - C (Comfort):
    A mere nuisance; will not do harm
    - e.g. a failure in a game
  - D (Discretionary money):
    Significant monetary loss, but bearable
    - e.g. a customer is lost due to bad service
  - E (Essential money):
    My enterprise may go bankrupt
    - e.g. huge loss due to an incorrect financial transaction system
  - L (Life):
    Somebody may be injured or may even die
    - e.g. vehicle control systems
- For each Crystal variant, different behaviors are described depending on criticality level
  - Cockburn does not claim Crystal to be suitable for criticality L
Crystal variants

- So far, only Crystal Clear has been spelled out in detail
- and Crystal Orange in short form
- (Cockburn also talks of "Magenta, Blue, and so on")
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 for level-3 practitioners:
  - "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"
Crystal Clear
Project Safety "Properties"

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

- 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
Crystal Clear vs. XP

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

• If a team can increase its discipline and consistency of action, they can lighten their methodology even more
  • 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.
Feature-Driven Development (FDD)

- http://www.featuredrivendevelopment.com/

Stephen Palmer
Feature-Driven Development (FDD)

- FDD is a classical incremental development process
  - In each iteration (about 2-10 days), one or several features are built,
  - each by a feature team, headed by a Chief Programmer.
- FDD is not particularly lightweight
  - it targets larger or large projects
FDD Roles

FDD defines a multitude of different roles:

- **Project Manager**
  - Administrative and financial leader
- **Chief Architect**
  - Rules in all design issues
  - Sometimes separate technical and domain architects
- **Chief Programmer**
  - Leads a feature team
  - Coordinates with other CPs
- **Class Owner**
  - Developer in a feature team: designs, codes, tests, documents
  - Individual code responsibility

- **Development Manager**
  - Solves conflicts, manages resources
- **Domain Expert**
- **Domain Manager**
  - Resolves conflicts among Domain Experts
- **Release Manager**
- **Technology/Language Guru**
- **Build Engineer**
- **Toolsmith**
- **System Administrator**
- **Tester**
  - For customer-level validation
- **Deployer**
  - e.g. for data conversion
- **Technical Writer**
Lean Software Development

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

- 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 in your verification process, your 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, Defer Commitment

- **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.

- **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.
Lean SD: Deliver Fast

- **Defer Commitment (cont'd)**
  - **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.

- **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, deliver superior quality, and are more attuned to their customers' needs.
  - **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.
Lean SD:  
Respect People

- **Deliver Fast (cont'd)**
  - **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.

- **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.
Adaptive Software Development (ASD)

- http://www.adaptivesd.com
Adaptive Software Development (ASD)

- Targeted at large projects
  - Goal: "Balancing on the edge of chaos"
  - concentrates on culture, not on techniques

- Component-centered, not task-centered
Rational Unified Process (RUP)

- Philippe Kruchten, Ivar Jacobson, et al.
- There is a large number of books about RUP
- A number of RUP variants exist
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
  • Full RUP has more than 100 different product types
  • Tailoring is left to the user (but supported by tools)
  • RUP is inherently iterative in any case
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
Rational Unified Process: Agile variants

Agile variants of RUP:

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

• dX
  - A minimal version of RUP very closely resembling XP
      - http://www.objectmentor.com/resources/articles/RUPvsXP.pdf

• 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
Dynamic Systems Development Method (DSDM)

- www.dsdm.org
  - Public and free, but registration is required
  - Outflow of Rapid Application Development (RAD) in the 1990s

- A more heavyweight process, centered around time-boxing
  - Focussing on project-level steps, not on SW development
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.agilebuch.de/

Jutta Eckstein
Agile development in the large (2)

- The book does not claim to present a 'method'
  - This is a German author after all...

- Has a discussion of scaling agile development to large projects (30-200 people)

- Discusses a number of aspects or techniques ignored by many of the other publications, such as:
  - Using explicit "communication teams"
  - Coping with virtual and distributed teams
  - Handling the surrounding organization (see next slide)
Agile development in the large (3)

• 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, outsorcing, part-time employees
• The book ends with a case-story of a complex project
  • Most useful part of the book!
The Pragmatic Programmer

• Andrew Hunt, David Thomas: "The Pragmatic Programmer: From Journeyman to Master", Addison-Wesley 1999
• http://www.pragmaticprogrammer.com
The Pragmatic Programmer (2)

- Not really a method as such, but rather a book of good advice
  - and a highly acclaimed one

- Framed in the form of 70 "tips", based on a few 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
The Pragmatic Programmer (3)

Fills in some details missing in other methods, such as:

- Some hints about HOW to keep a design simple
- Some hints about HOW to write sensible automated tests (e.g. assertions)
- Some hints about WHEN and HOW to use refactoring

Will be a useful companion no matter which method you are using, agile or other.
Summary

• There is a broad range of methods that could be considered agile methods
• They range from the super-light (Crystal Light) to the very complex (Rational Unified Process, RUP)
• They focus on different strengths, e.g.:
  • Communication and management (Scrum)
  • Simplicity (Crystal)
  • Comprehensiveness and scalability (RUP)
  • Holistic approach (Lean SD)
Thank you!