MTAT.03.094  
Software Engineering

Lecture 11:  
Agile/Lean Methods

Dietmar Pfahl  
email: dietmar.pfahl@ut.ee

Schedule of Lectures

Week 01: Introduction to SE  
Week 02: Requirements Engineering I  
Week 03: Requirements Engineering II  
Week 04: Analysis  
Week 05: Dev. Infrastructure I  
Week 06: Dev. Infrastructure II  
Week 07: ICS Day / ATI Päev 2014  
Week 08: Architecture and Design  
Week 09: Refactoring  
Week 10: Verification & Validation I  
Week 11: Verification & Validation II  
Week 12: Agile/Lean Methods  
Week 13: Software Quality Management  
Week 14: Measurement & Process Improvement  
Week 15: no lecture  
Week 16: no lecture

Structure of Lecture 11

• Light-weight processes / Evolutionary development  
• Agile Processes/Methods  
  • Extreme Programming (XP)  
  • Scrum (intro)  
• KANBAN  
• Lean Processes/Methods

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.
Structure of Lecture 11

- Light-weight processes / Evolutionary development
- Agile Processes/Methods
  - Extreme Programming (XP)
  - Scrum (intro)
- KANBAN
- Lean Processes/Methods

Extreme Programming (XP)

- Origin: Kent Beck, Ward Cunningham, Ron Jeffries (end of 1990s)
- Idea: "light weight" process model, agile process
- Characteristics:
  - "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

13 XP Practices

Project Cycle
- Planning Poker
- Small Releases
- Wholeness Tests
- 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

Requirements vs. User Stories

Traditional requirement – "shall"
- "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"

Planning Poker

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 (cont’d)

- **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.
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 refactoring can be automated
Catalogue of refactorings:
http://www.refactoring.com/catalog/index.htm

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

Structure of Lecture 11

- Light-weight processes / Evolutionary development
- Agile Processes/Methods
  • Extreme Programming (XP)
  • Scrum (intro)
- KANBAN
- Lean Processes/Methods

Simple Design

Characterisation:
- Four characteristics of simple design, listed in priority order:
  1. The system runs all the tests.
  2. It contains no duplicate code.
  3. The code states the programmers' intent very clearly.
  4. 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. (YAGNI: You Aren't Going to Need It)
- Chances are they won't be (YAGNI: You Aren't Going to Need It). Let the user pay for the generality.
- Chances are they won't be
- Don't add infrastructure or other features that might be needed later. Choices are they won't be (YAGNI: You Aren't Going to Need It). Let the 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, patterns, and larger scale systems are born.
- Remember that in just code. If this 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.htm

Simple Design

Characterisation:
- Four characteristics of simple design, listed in priority order:
  1. The system runs all the tests.
  2. It contains no duplicate code.
  3. The code states the programmers' intent very clearly.
  4. 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. (YAGNI: You Aren't Going to Need It)
- Chances are they won't be (YAGNI: You Aren't Going to Need It). Let the user pay for the generality.
- Don't add infrastructure or other features that might be needed later. Choices are they won't be (YAGNI: You Aren't Going to Need It). Let the 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, patterns, and larger scale systems are born.
- Remember that it is just code. If this 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.
The Term “Scrum”

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

What is Scrum?

• 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 Manifesto of Kent Beck at al.
  • Human-centred
  • Technology and tools have secondary role
  • Close cooperation with customer
  • Empirical learning 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 Elements – Process, Artifacts, Roles

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 (2-4 weeks) 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 end of Sprint: Product Owner accepts/rejects 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: 3 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

---

**Product Owner (cont’d)**
- 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

---

**Scrum Team**
- Small team size:
  - Typically 5 to 9 team members
  - Cross-functional: Design, coding, testing, etc.
  - Members must have a broad range of skills
  - 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
  - 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)
  - Membership 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
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.
  - **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.

Requirements Workshop (Backlog Refinement Meeting)
- Joint workshop with product owner, team, end users, and all other relevant stakeholders (e.g., marketing, sales) - duration: approx. 2 hours
- **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 (so-called sprints)
- Existing high-priority themes / requirements are detailed (-> proper User Stories) and acceptance criteria are defined
- Often using index cards on Meta Planning Boards
- Team estimates the cost/size/complexity of requirements
  - E.g., using points on a Fibonacci series (0, 1, 2, 3, 5, 8, 13...)
  - Done as part of an estimation workshop, perhaps using “planning poker”

Product Backlog
- A list of all desired work on the project (e.g., the requirements)
- Ideally expressed such that each item has value to the user or customers of the product
- Prioritized by the product owner
- Reprioritized at the start of each sprint
- Coarse-grained at the beginning of the project, continuously refined in requirements workshops
- Evolves over time!

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 extensive 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 sequentially

'Slice' of a Typical Scrum Project
Sprint Meetings

- 1 day for a 4 week Sprint
- 15 min daily

Sprint Backlog

- Created during the Sprint Planning Meeting
- Updated at least at the end of every day
- Includes all activities (tasks) 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

Development Sprint

- Normal Sprint in a Scrum project
- Implementation of all activities that are described in the Sprint Backlog
- Design
- Coding
- Integration
- Test
- Documentation
- ...
- Documented in the Sprint Backlog (activity started / completed)

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

- 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?

Sprint Review

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

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

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)

Other Plans and Reports

• Release plan
  • Documents the functionality (planned 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

Sprint Burndown Chart

Exercise:
1. Only 70% of the full capacity are planned/allocated on day 0
2. A work day has 8 work hours
Scalability of Scrum

- 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 11

- Light-weight processes / Evolutionary development
- Agile Processes/Methods
  - Extreme Programming (XP)
  - Scrum (intro)
- KANBAN
- Lean Processes/Methods
Kanban (Jap.): literally 'signboard' or 'billboard'.

**Velocity vs. Lead-time**

**Scrum**
- Focuses on: Flow of work items (throughput/velocity) = the number of features (user stories, tasks, etc.) implemented per unit of time (with given workforce).

**Kanban**
- Focuses on: Lead-time (cycle time) = the average time it takes to finish a work item (from start to end).

**Time-boxing vs. Task-boxing**

Scrum has sprints (iterations) of 2-4 weeks (= time box).
But: it is not always easy to divide the tasks or features of the systems to fit into such time intervals.

What about instead limiting the amount of tasks or features (= task box) that can be worked on concurrently and deliver when finished?

**Velocity vs. Lead**

**Scrum** focuses on:
- Flow of work items (throughput/velocity) = the number of features (user stories, tasks, etc.) implemented per unit of time (with given workforce).

**Kanban** focuses on:
- Lead-time (cycle time) = the average time it takes to finish a work item (from start to end).

**Kanban Board**

A Work Item represents a unit of work to be carried out by the development team.
Describe a Work item on a post-it sheet and put it on a board in one of the categories: "To do", "Ongoing" or more detailed states. "Done" shows the Work Items that are finished.

**Scrum Board versus Kanban Board**

Max WIP

**What is the right WIP limit?**

We're slide 6/10 just added: Let's increase WIP limit to 3!
What is the right WIP limit?

Differences between Scrum and Kanban

**Time-boxed iterations**
- **Prescribed**:
  - Scrum
  - Kanban

**Team commits**
- **Specific amount of work for this iteration**
- **Default metric for planning and process improvement**
  - **Velocity**

**Commitment**
- **Optional**
- **Default metric for planning and process improvement**
  - **Lead time**

**Similarities between Scrum and Kanban**

- Both use pull scheduling
- Both limit WIP (but in different ways)
- Both use transparency to drive process improvement
- Both focus on delivering releasable software early and often
- Both are based on self-organizing teams
- Both require breaking the work into pieces
- In both, workflow is continuously optimized based on empirical data (velocity / lead time)
- Both are Lean
Structure of Lecture 11

- Light-weight processes / Evolutionary development
- Agile Processes/Methods
  - Extreme Programming (XP)
  - Scrum (intro)
- KANBAN
- Lean Processes/Methods

Main Goals of LEAN

1. All processes shall give value
   - Remove everything that does not create value
2. Ensure good flow in the processes to avoid bottlenecks and queues (-> work not piling up and waiting)
3. All activity shall be based on need (-> Pull)
   - If there is no demand for a product or service, the related task is unnecessary
4. Become a learning organization with focus on continuous stepwise improvement
   - Kaizen (= small change for the better)

Seven Wastes of Software Development

- Handoffs. Passing the information/work to someone else, getting information/work from someone else.
- Partially done work. Something that is not done. E.g. untested code, undocumented or not maintained code.
- Task switching. How many other tasks people need to do. E.g. the amount of projects done simultaneously.
- Delays. Waiting for something.
- Extra features. Something that is not really needed.
- Defects. Something that does not meet the targets, or is not what it is supposed to be. E.g. software bugs, incorrectly implemented business requirements.
- Relearning (waste of knowledge). E.g. forgetting decisions, re-trying solutions already tried, the inability to utilize the knowledge of other people.

Focus on reducing the activities that do not create value

The approach to continuous improvement

Focus on removing/reducing the activities that do not create value for our customers

20-30% (waste)
60-70% (value adding)
Traditional approach
Focus on the efficiency of the activities that create value for the customers

Next Lecture

- Date/Time:
  - Friday, 28-Nov, 14:15-16:00
- Topic:
  - Software Quality Management by Kerli Rungi (QA Manager at Playtech)
- For you to do:
  - Finish and submit Lab Task 6 on time!