LTAT.05.003
Software Engineering

Lecture 12 (continued):
Agile/Lean Methods

Dietmar Pfahl
email: dietmar.pfahl@ut.ee

Fall 2020
Schedule of Lectures

Week 01: Introduction to SE
Week 02: Requirements Engineering I
Week 03: Requirements Engineering II
Week 04: Analysis
Week 05: Development Infrastructure
Week 06: Continuous Development and Integration
Week 07: Project Estimation / Architecture and Design I
Week 08: Architecture and Design II
Week 09: Verification and Validation I
Week 10: Verification and Validation II
Week 11: Refactoring (and TDD)
Week 12: Agile/Lean Methods
Week 13: Industry Guest Lecture – Anton Keks, Codeborne
Week 14: Course wrap-up, review and exam preparation
Week 15: No regular lecture – exam consultation on request
Structure of Lecture 12

• 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.
Kanban (Jap.): literally 'signboard' or 'billboard'

**SCRUNM** vs. **KANBAN**

- Time-Boxing
- Velocity

- Task-Boxing
- Lead-Time
Time-boxing vs. Task-boxing

Scrum has sprints (iterations) of 1 (to 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?
A Scrum team will only commit to items that they think they can complete within one iteration (based on the definition of “Done”). If an item is too big to fit in a sprint, the team and product owner will try to find ways to break it into smaller pieces until it does fit. If items tend to be big, iterations will be longer (although usually no longer than 4 weeks).

Kanban teams try to minimize lead time and level the flow, so that indirectly creates an incentive to break items into relatively small pieces. But there is no explicit rule stating that items must be small enough to fit into a specific time box. On the same board we might have one item that takes 1 month to complete and another item that takes 1 day.
Velocity vs. Lead-time

SCRUM focuses on:
• Avg. 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)
Visualization of Work Flow: Issue Tracker

Example: PivotalTracker (www.pivotaltracker.com)
Visualization of Work Flow: Issue Tracker

Example: PivotalTracker
Visualization of Work Flow: Issue Tracker

Show several projects concurrently
Visualization of Work Flow: Issue Tracker

Analytics
Scrum Board versus Kanban Board

From: Kanban and Scrum - making the most of both by Henrik Kniberg and Mattias Skarin on Dec 21, 2009
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.

From: Kanban and Scrum - making the most of both by Henrik Kniberg and Mattias Skarin on Dec 21, 2009
### User Story

- As a user I can add an image in news wizard
- As a user I want to change my password

### Task

- Partial look
- Link to my page
- Apache config.

### Development

- Business logic

### Staging

- As an admin... verify business logic

### Verification

- As a tester... check business logic

### Scheduled

- Release xx
  - As a user...
  - As a user...
  - As a user...

### Production

- Problem
  - GetFeatureImage does not work
  - Fix server
  - Fix JavaScript

### Suggestions

- As a publisher I can add new article
- Article type to wizard
- Database changes
- Article icon
- Privelege check

### Expedite

- (1)
What is the right WIP limit?

We’re idle & bored! Let’s increase WIP limit to 8!
What is the right WIP limit?

Problem with integration server. Can’t finish D & E! We’ll work on F & G instead!

Oops. WIP limit reached. Now we HAVE to stop and fix the integration server!

Let’s reduce WIP limit to 4, so we react earlier next time!
### Differences between Scrum and Kanban

<table>
<thead>
<tr>
<th>Scrum</th>
<th>Kanban</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cannot add items to ongoing iteration.</td>
<td>Can add new items whenever capacity is available</td>
</tr>
<tr>
<td>A sprint backlog is owned by one specific team</td>
<td>A Kanban board may be shared by multiple teams or individuals</td>
</tr>
<tr>
<td>Prescribes 3 roles (PO/SM/Team)</td>
<td>Doesn’t prescribe any roles</td>
</tr>
<tr>
<td>A Scrum board is reset between each sprint</td>
<td>A Kanban board is persistent</td>
</tr>
<tr>
<td>Prescribes a prioritized product backlog</td>
<td>Prioritization is optional.</td>
</tr>
</tbody>
</table>
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, work flow is continuously optimized based on empirical data (velocity / lead time)

• Both are Lean
Visualize Your Workflow
Limit Your WIP
Use Lead-Time as default metric

RUP has over 30 roles, over 20 activities, and over 70 artifacts.
Structure of Lecture 12

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

- Originates from Toyota Production System (TPS)
  - Also called Just-In-Time system
- Post WWII Japanese automobile industry could not compete with U.S. mass production systems
- Inspiration for TPS found in the 1950’s from U.S. supermarkets
  - Customers could get what they wanted, when they wanted it and shelves were refilled when items were about to run out.
- The concepts transferred to the domain of software engineering by Mary and Tom Poppendieck (2003, 2007).
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 & 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)
Focus on reducing the activities that do not create value

Traditional approach

Focus on the efficiency of the activities that create value for the customers
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

Traditional approach

Focus on the efficiency of the activities that create value for the customers

- 60-70% (waste)
- 20-30% (value adding)
- 10-20% (imposed waste)
Seven Principles of Lean SW Development
(by Mary Poppendieck)

1. Optimize the Whole
2. Eliminate Waste
3. Build Quality In
4. Learn Constantly
5. Deliver Fast
6. Engage Everyone
7. Keep Getting 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.
End of Lecture 12