

**Speakers notes:**

It begins with Lean as a concept, optimizing the whole value flow

With the Agile concept we focus on cooperation to eliminate waste

Scrum is one typical method that can be used to plan and keep good control of what to do and who is doing what

XP is yet another method, but in this case a specific one for SW development (eXtreme Programming)

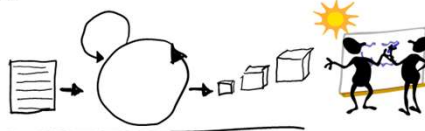
# WATERFALL, AGILE, LEAN

Waterfall



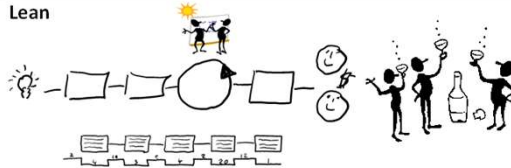
Schedule large work orders and align people by workflow

Agile



Schedule small work orders and align people by schedule

Lean



Schedule small work orders and align people by workflow

# AGILE



[HTTP://AGILEMANIFESTO.ORG/](http://agilemanifesto.org/)

## Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

**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.

Kent Beck	James Grenning	Robert C. Martin
Mike Beedle	Jim Highsmith	Steve Mellor
Arie van Bennekum	Andrew Hunt	Ken Schwaber
Alistair Cockburn	Ron Jeffries	Jeff Sutherland
Ward Cunningham	Jon Kern	Dave Thomas
Martin Fowler	Brian Marick	

[HTTP://AGILEMANIFESTO.ORG/ISO/HU/MANIFESTO.HTML](http://agilemanifesto.org/iso/hu/manifesto.html)

### Kiáltvány az agilis szoftverfejlesztésért

A szoftverfejlesztés hatékonyabb módját tárjuk fel saját tevékenységünk és a másoknak nyújtott segítség útján. E munka eredményeképpen megtanultuk értékelni:

**Az egyéneket és a személyes kommunikációt a módszertanokkal és eszközökkel szemben**

**A működő szoftvert az átfogó dokumentációval szemben**

**A megrendelővel történő együttműködést a szerződéses egyeztetéssel szemben**

**A változás iránti készséget a tervek szolgai követésével szemben**

Azaz, annak ellenére, hogy a jobb oldalon szereplő tételek is értékkel bírnak, mi többre tartjuk a bal oldalon feltüntetetteket.

Kent Beck	James Grenning	Robert C. Martin
Mike Beedle	Jim Highsmith	Steve Mellor
Arie van Bennekum	Andrew Hunt	Ken Schwaber
Alistair Cockburn	Ron Jeffries	Jeff Sutherland
Ward Cunningham	Jon Kern	Dave Thomas
Martin Fowler	Brian Marick	

## PRINCIPLES OF THE AGILE MANIFESTO (1/2)

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

<http://www.agilemanifesto.org/principles.html>

56

The manifesto also includes twelve principles. Here they are.  
12 principles: they are each self-explanatory.

## PRINCIPLES OF THE AGILE MANIFESTO (2/2)

7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity--the art of maximizing the amount of work not done--is essential. (YAGNI – You Aren't Gonna Need It.)

11. The best architectures, requirements, and designs emerge from self-organizing teams.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

<http://www.agilemanifesto.org/principles.html>

# AGILE APPROACHES

Agile methods are not unified, there is diversity

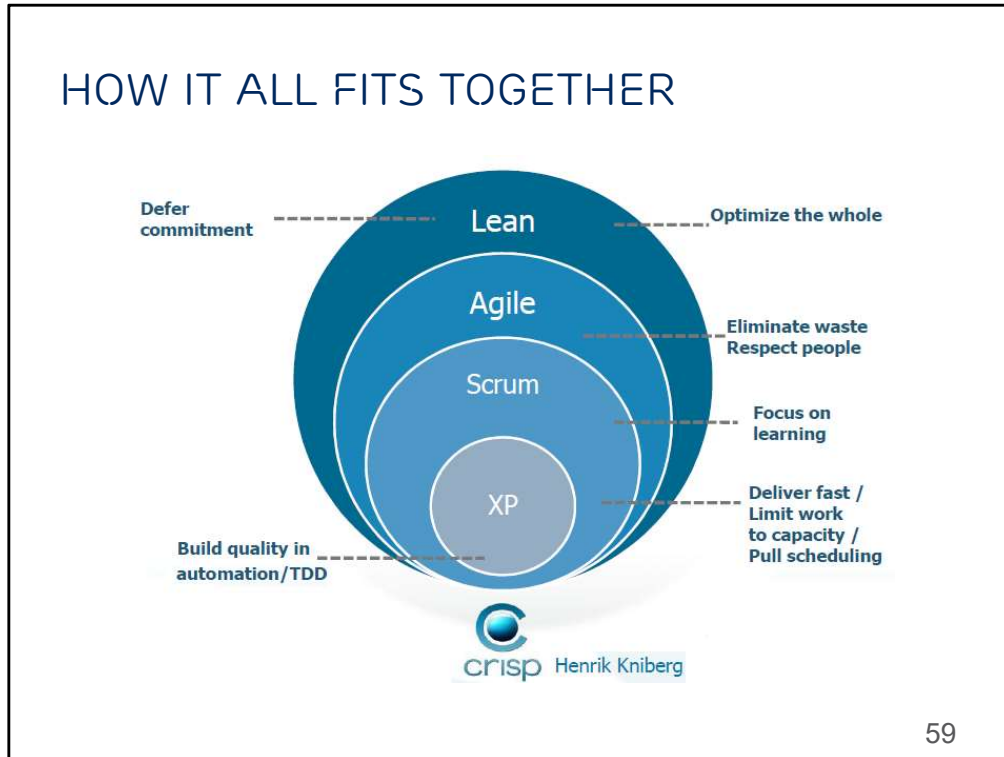
Each method implements the Agile Manifesto differently

We will consider

- Extreme Programming (XP)
- Scrum
- Kanban

There are common practices across these methods, which we'll examine





### Speakers notes:

It begins with Lean as a concept, optimizing the whole value flow

With the Agile concept we focus on cooperation to eliminate waste

Scrum is one typical method that can be used to plan and keep good control of what to do and who is doing what

XP is yet another method, but in this case a specific one for SW development (eXtreme Programming)

# EXTREME PROGRAMMING (XP)

- Formulated in 1999 by Kent Beck, Ward Cunningham and Ron Jeffries
- Agile software development methodology (others: Scrum, DSDM, Kanban)
- Developed in reaction to high ceremony methodologies

## EXTREME PROGRAMMING (XP)

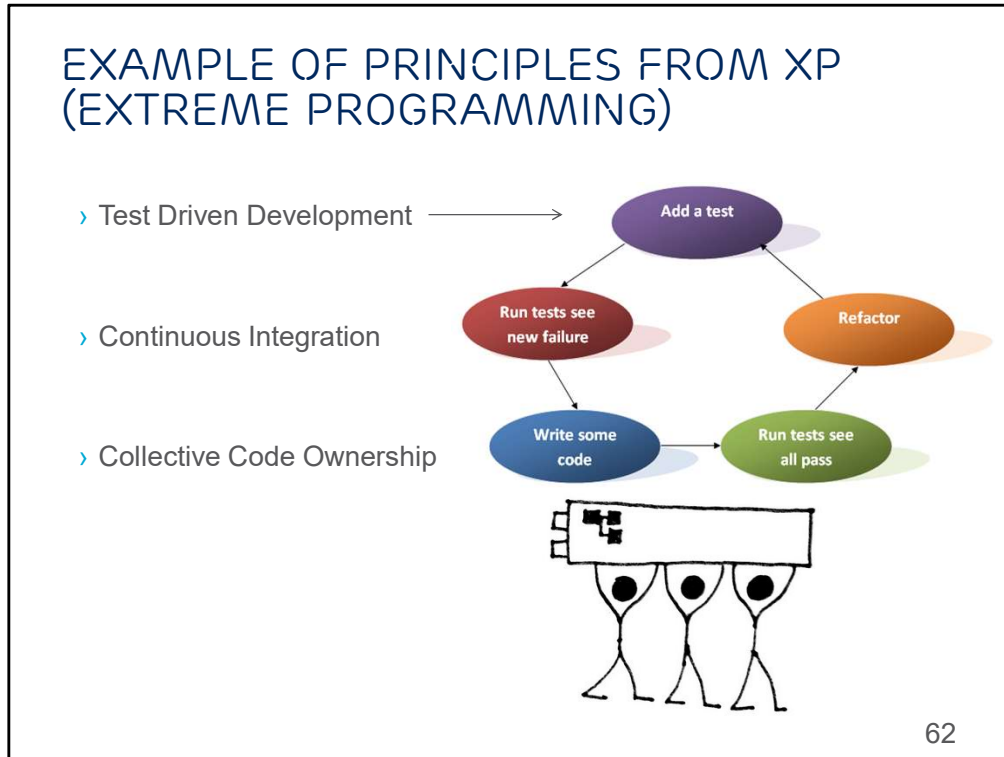
A software development process

Designed to focus on four things:

- Coding, Testing, Listening, Designing

Feedback is built into the development practices, not bolted on.

No phases: you do all four of those things *all the time*.



### Speakers notes:

Collective code ownership doesn't mean that everyone is supposed to do everything. It means that we try learn more from each other to become less vulnerable so e.g Charles can keep on working with a design task even if Edith is on sick leave on a Monday.

## XP: WHY?

### Previously:

- Get all the requirements before starting design
- Freeze the requirements before starting development
- Resist changes: they will lengthen schedule
- Build a change control process to ensure that proposed changes are looked at carefully and no change is made without intense scrutiny
- Deliver a product that is obsolete on release

## XP: EMBRACE CHANGE

### Recognize that:

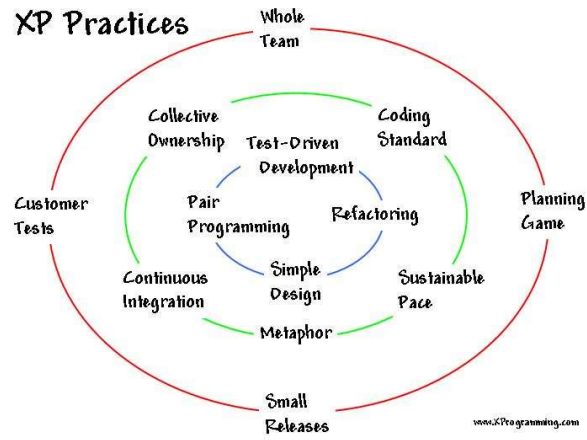
- All requirements will not be known at the beginning
- Requirements will change

Use tools to accommodate change as a natural process

Do the simplest thing that could possibly work and refactor mercilessly

Emphasize values and principles rather than process

# XP PRACTICES



(Source: <http://www.xprogramming.com/xpmag/whatisxp.htm>) 65

## THE CORE XP PRACTICES

### ***Rapid, fine feedback:***

- Test-driven design (via unit and acceptance tests)
- On-site customer
- Pair programming

### ***Continuous process:***

- Continuous integration
- Merciless refactoring
- Small, frequent releases

### ***Shared Understanding:***

- Planning game
- Simple Design
- System Metaphor
- Collective Code Ownership
- Coding Conventions

### ***Developer Welfare:***

- Forty-hour week

66

The idea of a metaphor in Extreme Programming is to develop a common vision of how the program works. At its best, a metaphor is a simple evocative description of how the program works.



# THE XP TEAM

How to design and program the software

- programmers, designers, and architects

Where defects are likely to hide

- testers

Why the software is important

- product manager

The rules the software should follow

- domain experts

How the software should behave

- interaction designers

How the user interface should look

- graphic designers

How to interact with the rest of the company

- project manager

Where to improve work habits

- coach

# XP PRACTICES: WHOLE TEAM

All contributors to an XP project are one team

Must include a business representative: the 'Customer'


- Provides requirements
- Sets priorities
- Steers project

Team members are programmers, testers, analysts, coach, manager

Best XP teams have no specialists

## XP TEAM SIZE

Assume teams with 4 to 10 programmers (5 to 20 total team members).



Applying the staffing guidelines to a team of 6 programmers produces a team that also includes 4 customers, 1 tester, and a project manager, for a total team size of 12 people.

## FULL-TIME TEAM MEMBERS

All the team members should sit with the team full-time and give the project their complete attention.



This particularly applies to customers, who are often surprised by the level of involvement XP requires of them.

70

Some organizations like to assign people to multiple projects simultaneously. This *fractional assignment* is particularly common in *matrix-managed organizations*. (If team members have two managers, one for their project and one for their function, you are probably in a matrixed organization.)

# XP PRACTICES: PLANNING GAME

Two key questions in software development:

- Predict what will be accomplished by the due date
- Determine what to do next

Need is to steer the project

Exact prediction (which is difficult) is not necessary

# XP PRACTICES: PLANNING GAME

## XP Release Planning

- Customer presents required features
- Programmers estimate difficulty
- Imprecise but revised regularly

## XP Iteration Planning

- Two week iterations
- Customer presents features required
- Programmers break features down into tasks
- Team members sign up for tasks
- Running software at end of each iteration

## XP PRACTICES: CUSTOMER TESTS

The Customer defines one or more automated acceptance tests for a feature

Team builds these tests to verify that a feature is implemented correctly

Once the test runs, the team ensures that it keeps running correctly thereafter

System always improves, never backslides

## XP PRACTICES: SMALL RELEASES

Team releases running, tested software every iteration

Releases are small and functional

The Customer can evaluate or in turn, release to end users, and provide feedback

Important thing is that the software is visible and given to the Customer at the end of every iteration



## XP PRACTICES: SIMPLE DESIGN

Build software to a simple design

Through programmer testing and design improvement, keep the software simple and the design suited to current functionality

Design steps in release planning and iteration planning

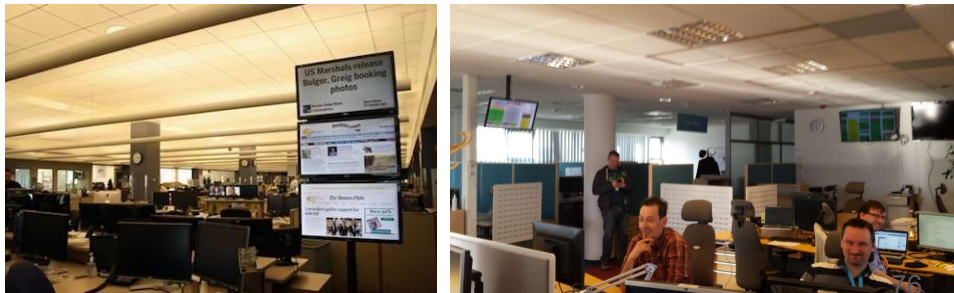
Teams design and revise design through refactoring, through the course of the project

## XP PRACTICES: INFORMATIVE WORKSPACE

Your workspace is the cockpit of your development effort: create an informative workspace

An informative workspace broadcasts information into the room (eg. radiators)

It's improve stakeholder trust



- Your workspace is the cockpit of your development effort. Just as a pilot surrounds himself with information necessary to fly a plane, arrange your workspace with information necessary to steer your project: create an informative workspace.
- An informative workspace broadcasts information into the room. When people take a break, they will sometimes wander over and stare at the information surrounding them. Sometimes, that brief zoneout will result in an aha moment of discovery.
- An informative workspace also allows people to sense the state of the project just by walking into the room. It conveys status information without interrupting team members and helps improve stakeholder trust.

## XP PRACTICES: PAIR PROGRAMMING

All production software is built by two programmers, sitting side by side, at the same machine

All production code is therefore reviewed by at least one other programmer

Research into pair programming shows that pairing produces better code in the same time as programmers working singly

Pairing also communicates knowledge throughout the team

## XP PRACTICES: TEST-DRIVEN DEVELOPMENT

Teams practice TDD by working in short cycles of adding a test, and then making it work

Easy to produce code with 100 percent test coverage

These programmer tests or unit tests are all collected together

Each time a pair releases code to the repository, every test must run correctly

## XP PRACTICES: DESIGN IMPROVEMENT

Continuous design improvement process called 'refactoring':

- Removal of duplication
- Increase cohesion
- Reduce coupling

Refactoring is supported by comprehensive testing - customer tests and programmer tests

## XP PRACTICES: CONTINUOUS INTEGRATION

Teams keep the system fully integrated at all times

Daily, or multiple times a day builds

Avoid 'integration hell'

Avoid code freezes

10 minutes build

80

'integration hell', e.g., integrating a big chunk of code changes at the last minute which results in conflicts, and can take more time to resolve as compared to the time required to make original changes.

## XP PRACTICES: COLLECTIVE CODE OWNERSHIP

Any pair of programmers can improve any code at any time

All code gets the benefit of many people's attention

Avoid duplication

Programmer tests catch mistakes


Pair with expert when working on unfamiliar code

## XP PRACTICES: CODING STANDARD

Use common coding standard



All code in the system must look as though written by an individual



Code must look familiar, to support collective code ownership



## XP PRACTICES: METAPHOR

XP Teams develop a common vision of the system

With or without imagery, define common system of names

Ensure everyone understands how the system works,  
where to look for functionality, or where to add functionality

## XP PRACTICES: SUSTAINABLE PACE

Team will produce high quality product when not overly exerted

Avoid overtime, maintain 40 hour weeks

'Death march' projects are unproductive and do not produce quality software

Work at a pace that can be sustained indefinitely

84

In [project management](#), a **death march** is a project where the members feel it is destined to fail, or requires a stretch of unsustainable overwork. The general feel of the project reflects that of an actual [death march](#) because the members of the project are forced to continue the project by their superiors against their better judgment.

## CHARACTERISTICS OF SUCCESSFUL XP PROJECTS

- Very rapid development
- Exceptional responsiveness to user and customer change requests
- High customer satisfaction
- Amazingly low error rates
- System begins returning value to customers very early in the process

# XP VALUES

Communication

Simplicity

Feedback

Courage

## XP VALUES: COMMUNICATION

Poor communication in software teams is one of the root causes of failure of a project

Stress on good communication between all stakeholders-- customers, team members, project managers

Customer representative always on site

Paired programming

## XP VALUES: SIMPLICITY

### 'Do the Simplest Thing That Could Possibly Work'

- Implement a new capability in the simplest possible way
- Refactor the system to be the simplest possible code with the current feature set

### 'You Aren't Going to Need It' (YAGNI)

- Never implement a feature you don't need now

## YOU AREN'T GONNA NEED IT (YAGNI)

Important aspect of simple design: avoid speculative coding.

- Whenever you're tempted to add something to your design, ask yourself if it supports the stories and features you're currently delivering. If not, well... **you aren't gonna need it**. Your design could change. Your customers' minds could change.

Similarly, remove code that's no longer in use.

- You'll make the design smaller, simpler, and easier to understand. If you need it again in the future, you can always get it out of version control. For now, it's a maintenance burden you don't need.

89

Important aspect of simple design: avoid speculative coding. Whenever you're tempted to add something to your design, ask yourself if it supports the stories and features you're currently delivering. If not, well... you aren't gonna need it. Your design could change. Your customers' minds could change.

Similarly, remove code that's no longer in use. You'll make the design smaller, simpler, and easier to understand. If you need it again in the future, you can always get it out of version control. For now, it's a maintenance burden you don't need.

We do this because excess code makes change difficult. Speculative design, added to make specific changes easy, often turns out to be wrong in some way, which actually makes changes more difficult. It's usually easier to add to a design than to fix a design that's wrong. The incorrect design has code that depends on it, sometimes locking bad decisions in place.

## XP VALUES: FEEDBACK

- Always a running system that delivers information about itself in a reliable way
- The system and the code provides feedback on the state of development
- Catalyst for change and an indicator of progress



## XP VALUES: COURAGE

Projects are  
people-centric

Ingenuity of people  
and not any  
process that  
causes a project to  
succeed

# XP CRITICISM

Unrealistic--  
programmer  
centric, not  
business focused

Detailed  
specifications are  
not written

Design after  
testing

Constant  
refactoring

Customer  
availability

12 practices are  
too  
interdependent

# XP THOUGHTS

The best design is the code.

Testing is good. Write tests before code. Code is complete when it passes tests.

Simple code is better. Write only code that is needed. Reduce complexity and duplication.

Keep code simple. Refactor.

Keep iterations short. Constant feedback.

## COMMON XP MISCONCEPTIONS

### No written design documentation

- *Truth: no formal standards for how much or what kind of docs are needed.*

### No design

- *Truth: minimal explicit, up-front design; design is an explicit part of every activity through every day.*

### XP is easy

- *Truth: although XP does try to work with the natural tendencies of developers, it requires great discipline and consistency.*

## MORE MISCONCEPTIONS

### XP is just legitimized hacking

- *Truth: XP has extremely high quality standards throughout the process*
- *Unfortunate truth: XP is sometimes **used as an excuse** for sloppy development*

### XP is the one, true way to build software

- *Truth: it seems to be a sweet spot for certain kinds of projects*

## XP SUMMARY (BY ISTQB)

### Values:

- communication, simplicity, feedback, courage, respect

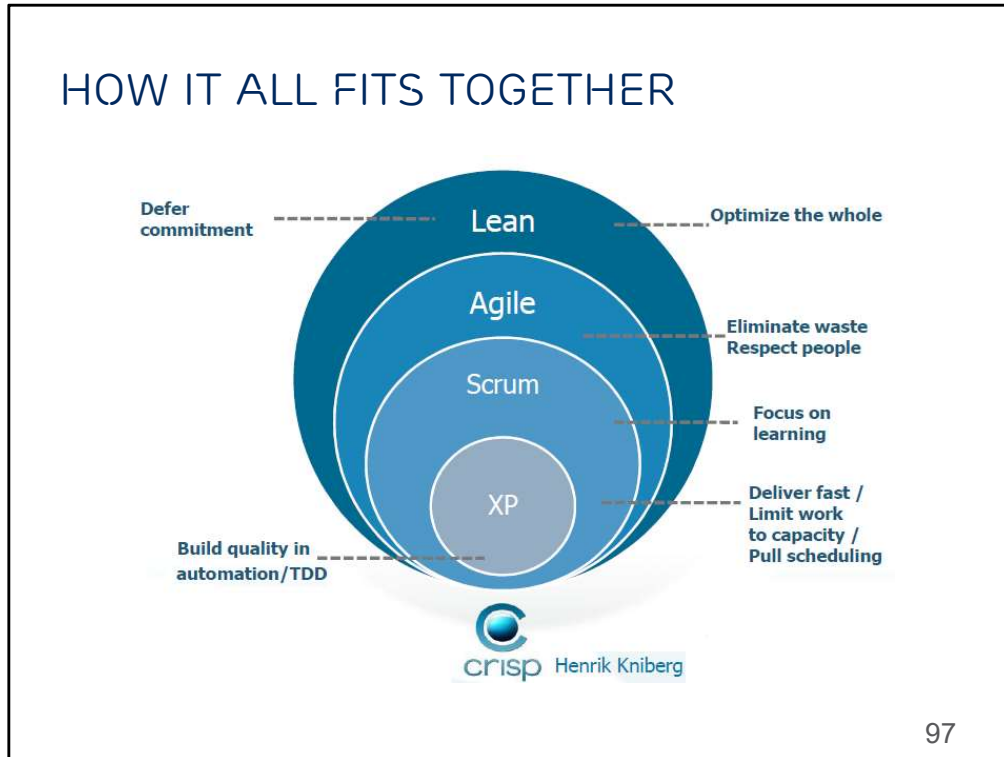
### Principles:

- humanity, economics, mutual benefit, self-similarity, improvement, diversity, reflection, flow, opportunity, redundancy, failure, quality, baby steps, accepted responsibility

### Primary practices:

- sit together, whole team, informative workspace (radiators), energized work, pair programming, stories, weekly cycle, quarterly cycle, slack (do not use 100% allocation), 10 minute build, continuous integration, test first programming, incremental design

Many other agile practices use some aspects of XP



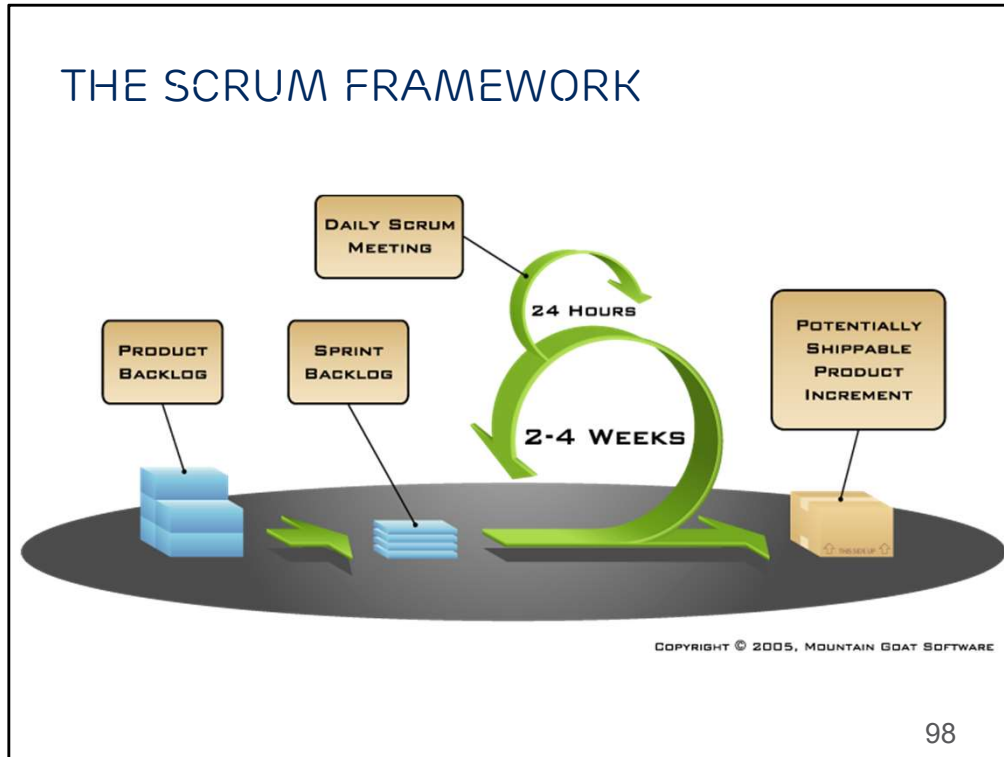
### Speakers notes:

It begins with Lean as a concept, optimizing the whole value flow

With the Agile concept we focus on cooperation to eliminate waste

Scrum is one typical method that can be used to plan and keep good control of what to do and who is doing what

XP is yet another method, but in this case a specific one for SW development (eXtreme Programming)

**Speakers notes:**

Process description of Scrum as one example of a method that can be used within Lean and Agile product development





### Speakers notes:

#### Product owner

- Represents the interests of all the stakeholders
- ROI objectives
- Prioritizes the product backlog

#### Team

- Cross-functional
- Self-managing
- Self-organizing

#### Coach

- Coaches the team in the Agile and Lean process
- Challenges the team for continuous improvement
- Teaching the way we do Agile & Lean
- Ensures the following of Agile & Lean rules and practices

## USER STORIES AND ESTIMATION (1)

Describe requirements in product backlog

Syntax: As <role> I want to <requirement>  
because <business reason>

Example:

- As a customer I want to reserve movie tickets with my mobile
- Because I want to be sure that I have a seat when I arrive to the theater

100

### **Speakers notes:**

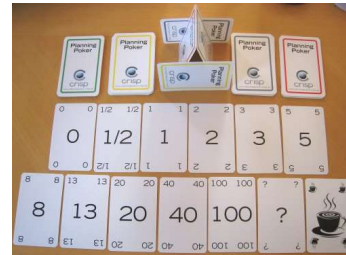
User stories are a way of describing customer requirements without having to create formalized requirement documents and without performing administrative tasks related to maintaining them.

A user story could describe a small feature but normally a feature is divided into several user stories.

## USER STORIES AND ESTIMATION (2)

### Planning poker method

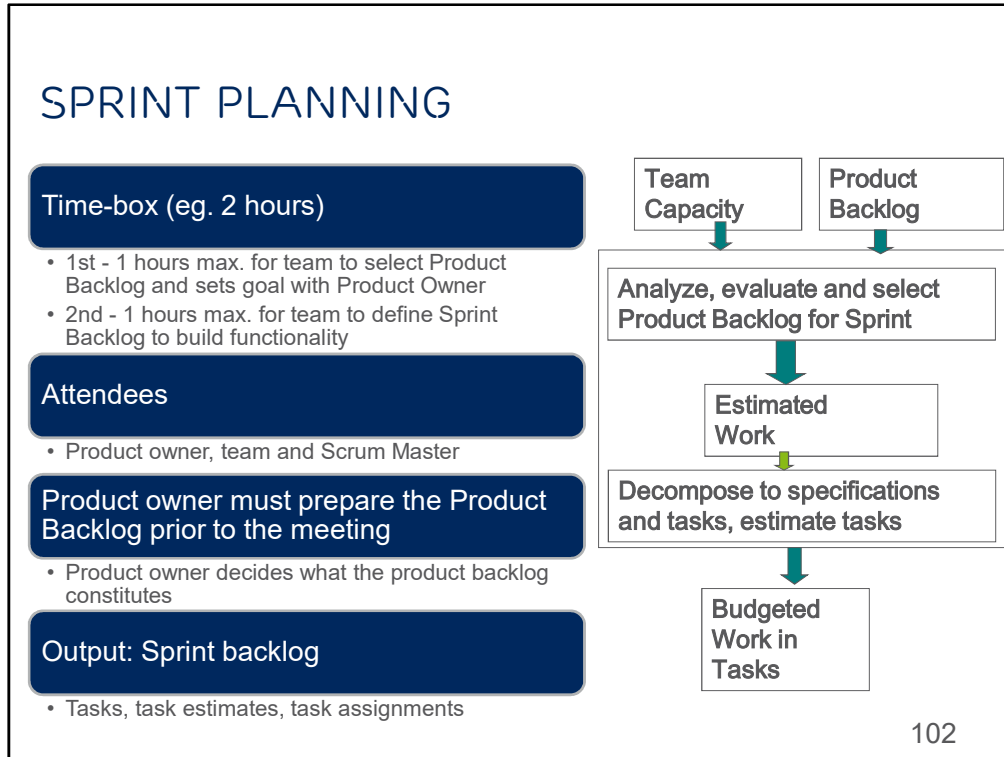
- Product owner (or a stakeholder with the best knowledge) explains the story
- Team members estimate the story independently and select a card
- They show the cards simultaneously
- Explain why estimates differ
- End or go back to step 2



101

#### Speakers notes:

This is an exercise which will focus on the ability to cooperate in a Team



### Speakers notes:

The very first time a Team work like this is set up it might take an hour or two.

This example could be a SW Team with a “normal size” of 6-8 members, (depending on the product, its maturity and complexity) that after implementation of Agile and Lean now can be done within a few

minutes, or significantly shorter planning time.

## DEFINITION OF DONE (DOD) 10 POINT CHECKLIST

Code produced (all 'to do' items in code completed)

Code commented, checked in and run against current version in source control

Peer reviewed (or produced with pair programming) and meeting development standards

Builds without errors

Unit tests written and passing

Deployed to system test environment and passed system tests

Passed UAT (User Acceptance Testing) and signed off as meeting requirements

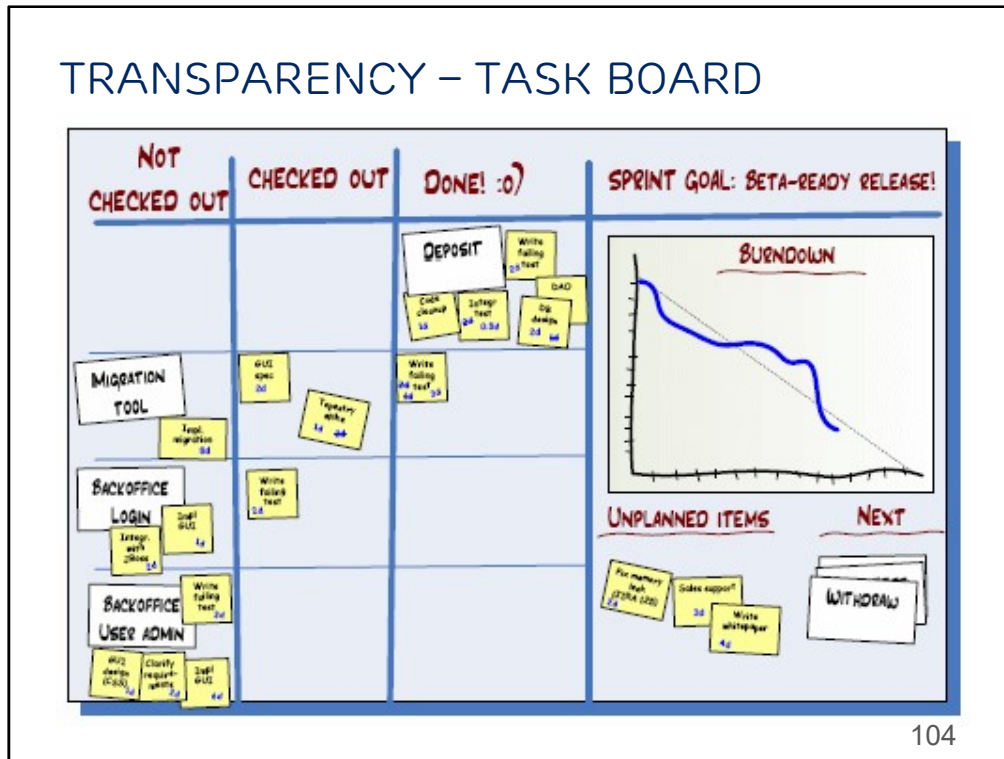
Any build/deployment/configuration changes implemented/documented/communicated

Relevant documentation/diagrams produced and/or updated

Remaining hours for task set to zero and task closed

103

See more at: <http://www.allaboutagile.com/definition-of-done-10-point-checklist/#sthash.8rcJSONz.dpuf>



Picture of task board: Kniberg, Henrik 2006. Scrum and XP from the Trenches. <http://www.crisp.se/henrik.kniberg/ScrumAndXpFromTheTrenches.pdf>

### Speakers notes:

Normally the team has their Daily Scrum standing at this task board. A Daily Scrum is a:

- Daily 15 minute work meeting;
- Same place and time every day;
- Where everyone answers three questions;
  - What have you done since last meeting?
  - What will you do before next meeting?
  - What is in your way?
- In order to find Impediments and make Decisions

**The definition of Done is very important to agree upon, settle this within the Team**

## RETROSPECTIVES

### Set the stage

- Focus for this retrospective

### Gather data

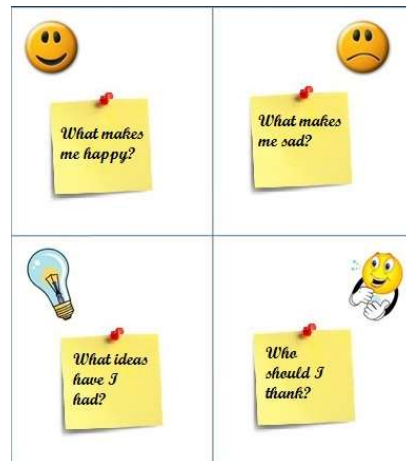
- Ground it in facts, not opinions

### Generate insights

- Observe patterns

### Decide what to do

- Move from discussion to action



105

### Speakers notes:

Point out that retrospectives are for the team and should thereby be run by the team, not a manager (the team should even decide if the manager is allowed to participate).

The goal is to find impediments for better ways of working. Earlier, before Agile ways of working, this was normally done once or twice a day. Now, we want to do this at the end of every sprint.



## SCRUM, SUMMARY (BY ISTQB)

### Practises

- Sprint (Iteration)
- Product increment
- Products backlog
- Definition of Done (DoD) – exit criteria
- Timeboxing – fix duration for iteration, fix daily meetings
- Transparency

No specific software development techniques

### Roles

- Scrum Master (SM) ensures practices and rules are implemented, followed – process focused scrum theory
- Product Owner (PO) represents the customer and owns product backlog – he/she can change product backlog any time
- Development Team (3-9, self-organized) develops and tests product

Scrum does not prescribe testing approach

## 看板 – KANBAN CARDS LIMIT EXCESS WORK IN PROGRESS

看板 – kanban literally means “visual card,” “signboard,” or “billboard.”

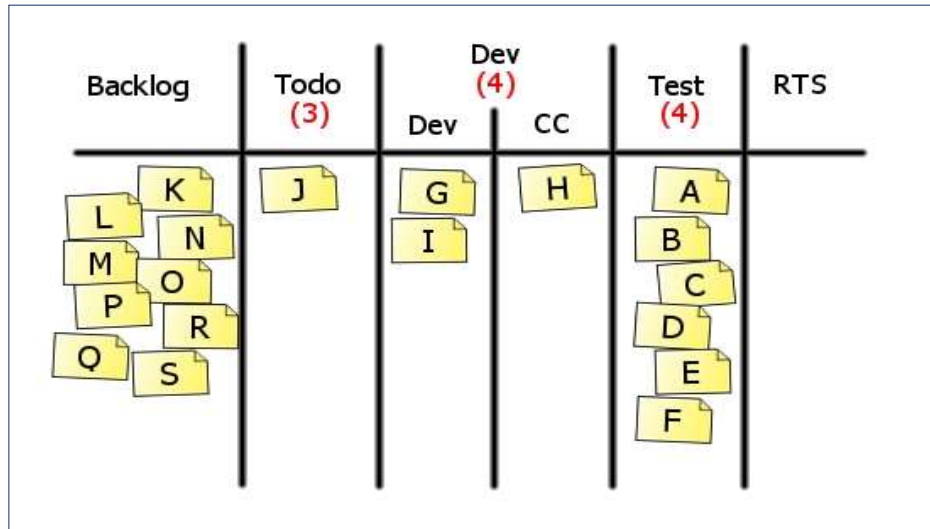
Toyota originally used Kanban cards to limit the amount of inventory tied up in “work in progress” on a manufacturing floor

kanban cards act as a form of “currency” representing how WIP (Work In Progress) is allowed in a system.

Kanban is an emerging process framework that is growing in popularity since it was first discussed at Agile 2007 in Washington D.C.



## TRANSPARENCY – KANBAN APPROACH



108

### Speakers notes:

Working with Kanban is all about optimizing flow. The Kanban board could be used on all levels. Leadership Teams as well as personally.

The numbers show maximum amount of Work in progress in every step of the process. In order to enhance collaboration, the amount shown be a lot smaller than the team-size.

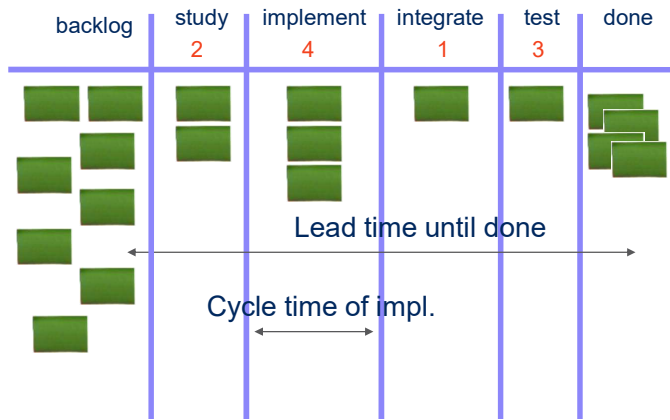
The board could be used either by showing impediments or for using regular job.

# KANBAN BASIC RULES

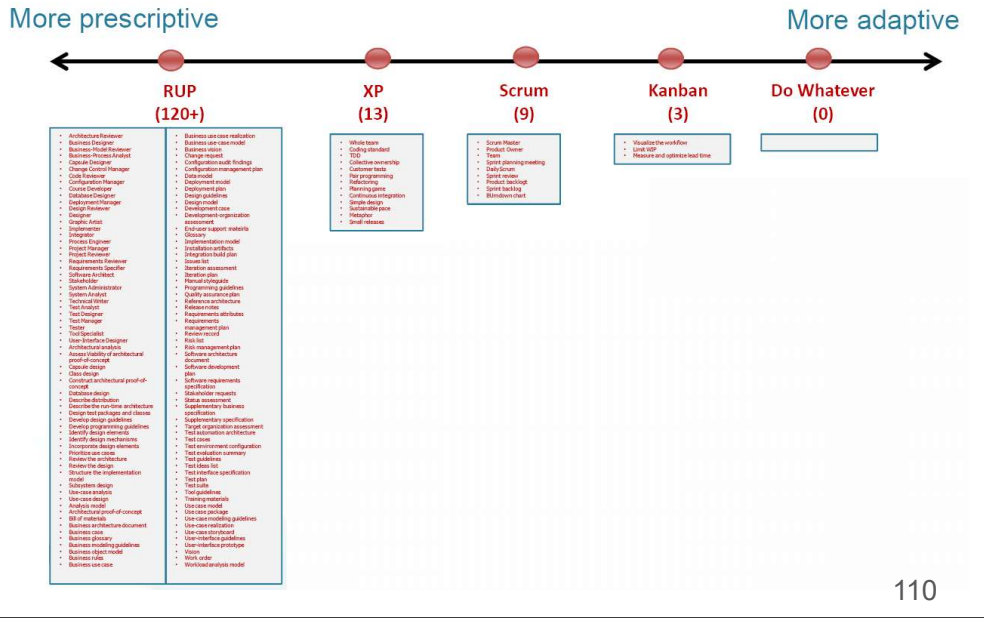
Visualize the workflow

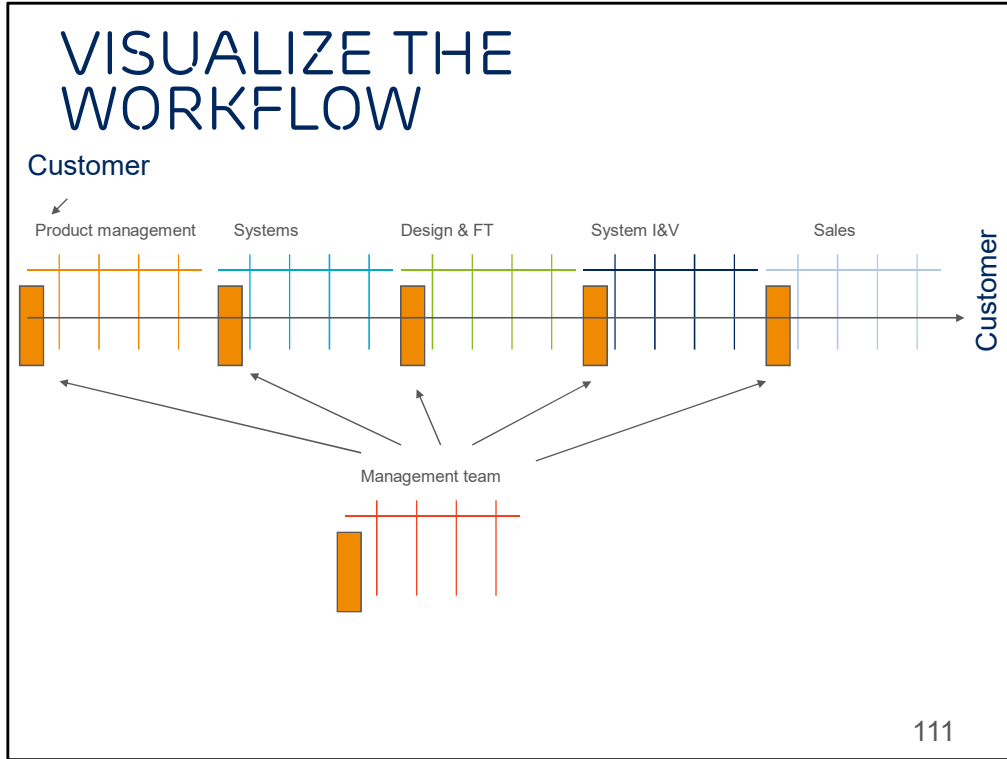
Limit Work In Progress (WIP)

Measure and optimize lead time










# PROCESSES





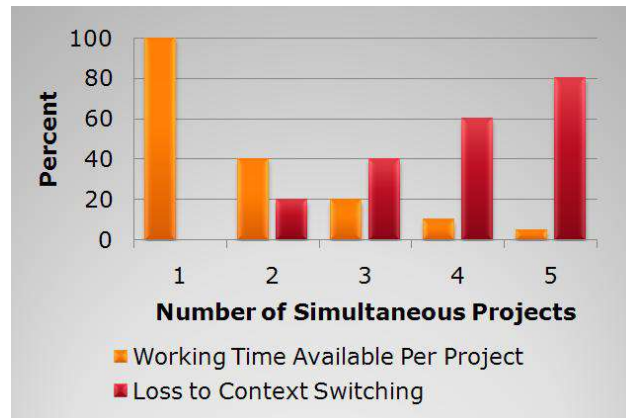
# VISUALIZE THE WORKFLOW

	Planned (x)	Design (2)	Test (6)	Integrate (3)	Done
US x 					
Improvement					
TR, CSR					
US y 					

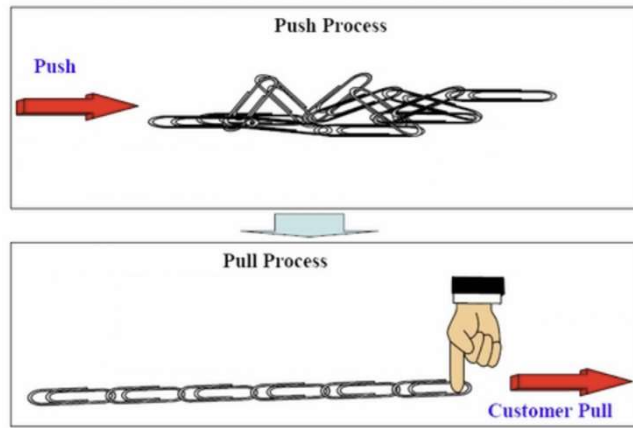


# LIMITING WORK IN PROGRESS

20% time is lost to context switching per task, so fewer tasks means less time lost (from Gerald Weinberg, *Quality Software Management: Systems Thinking*)

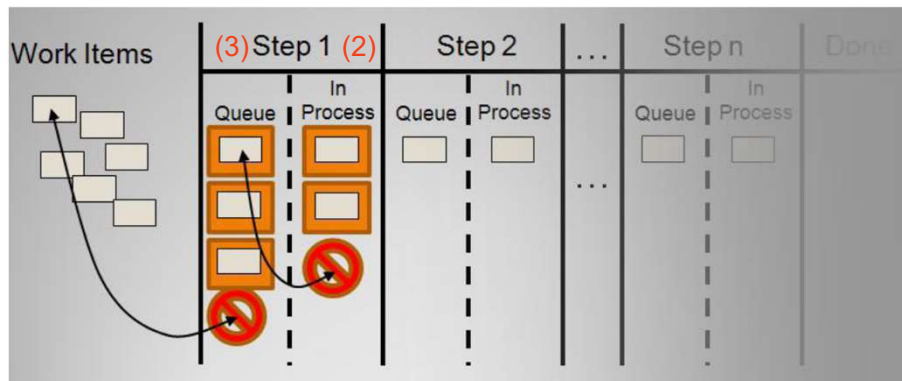


# LIMITING WORK IN PROGRESS

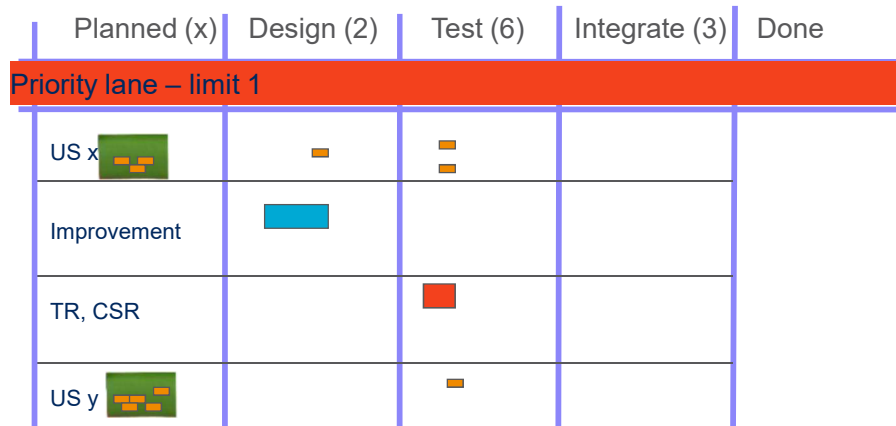


# LIMITING WORK IN PROGRESS


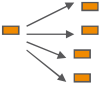





New work items can only be pulled into a state if there is capacity under the WIP limit.



# LIMITING WIP - EMERGENCY



# LIMITING WIP – EXPLODING ITEMS

	Planned (x)	Design (2)	Test (6)	Integrate (3)	Done
US x					
Improvement					
TR, CSR					
US y					

# METRICS

Metrics are a tool for **everybody**

The **team** is **responsible** for its metrics

Metrics allow for **continuous improvement**

Manage **quantitatively** and **objectively** using only a few simple metrics

- Quality
- Work in Process
- Lead / Cycle time
- Waste / Efficiency
- Throughput

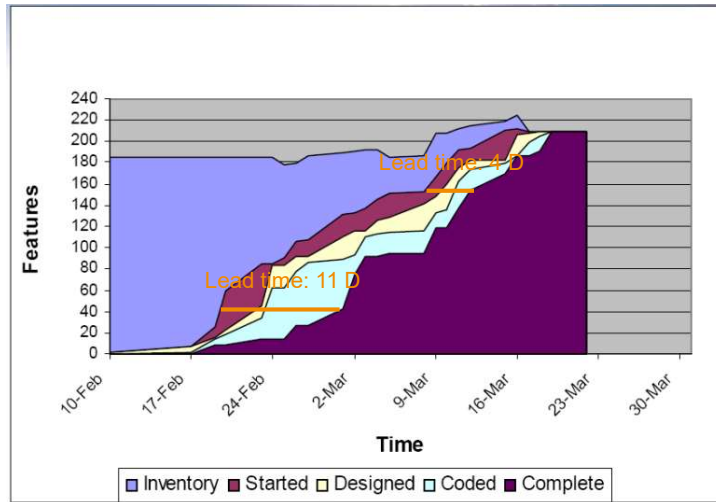
# LITTLE'S LAW FOR QUEUEING THEORY

*Total Cycle Time = Number of  
Things in Progress / Average  
Completion Rate*

The only way to reduce cycle time is by either reducing the WIP, or improving the average completion rate.

- Achieving both is desirable.
- Limiting WIP is easier to implement by comparison.

### USE CUMULATIVE FLOW DIAGRAMS TO VISUALIZE WORK IN PROGRESS

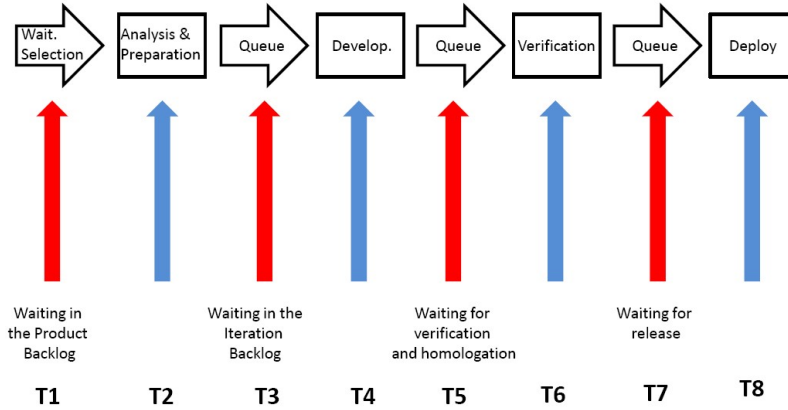


[www.agilemanagement.net/Articles/Papers/BorConManagingwithCumulat.html](http://www.agilemanagement.net/Articles/Papers/BorConManagingwithCumulat.html)



# VALUE STREAM MAPPING




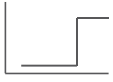




Value Stream for Product XYZ



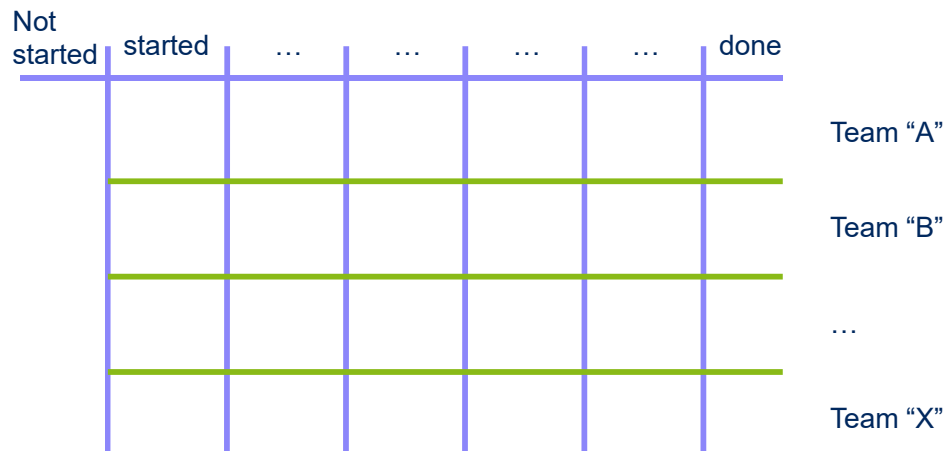
$$\text{Efficiency (\%)} = \frac{\text{TValue} * 100}{\text{TValue} + \text{TWaste}}$$

# PRIORITIZATION METHOD

- › Business value?
- › Cost of delay classification

		<b>Expedite:</b> critical, and immediate cost of delay; can exceed other kanban limit (bumps other work) 1 <sup>st</sup> priority
		<b>Fixed date:</b> cost of delay goes up significantly after deadline; 2 <sup>nd</sup> priority
		<b>Accelerating:</b> cost of delay goes up increasingly over time; 3 <sup>rd</sup> priority
		<b>Normal:</b> Cost of delay linear over time; 4 <sup>th</sup> priority

# SCALING – SWIM LANES



# R&D AGILE TARGET – OTHER PART

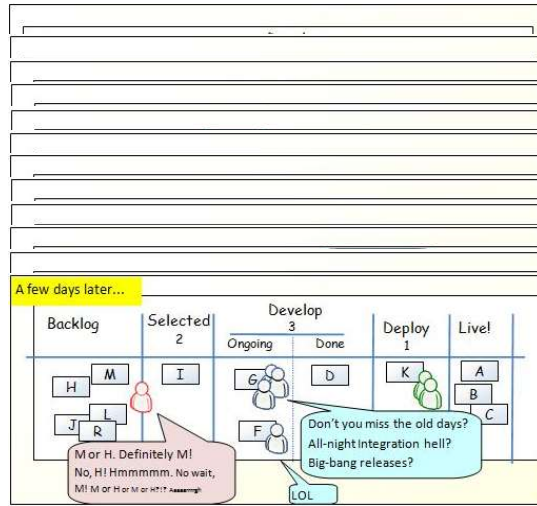
Requirement	Point
Tasks related to team are collected, prioritized and updated continuously in a shared excel sheet / team's whiteboard	
At least the next task states are available on the team's board: "Not Started", "In progress", "Blocked", "Done".	5
There are limits set for each of the "active" states. "Keep focus."	
Daily scrum meeting (What did you do yesterday? What do you plan to do today? Are there any impediments?)	5
Self organized team – team members select the tasks based on priorities.	5
Retrospective meetings in every 2-4 weeks (what went well, what should be improved)	5
Team tracks the lead-time of each task. (Average lead-time.)	
Co-located team	5

Visualize the workflow

Limit WIP

Measure and Optimize lead time

# ONE DAY IN KANBAN LAND



## AFTER A KANBAN IMPLEMENTATION...

“Nothing else in their world should have changed. Job descriptions are the same. Activities are the same. Handoffs are the same. Artifacts are the same. Their process hasn't changed other than you are asking them to accept an WIP limit and to pull work rather than receive it in a push fashion”

**David Anderson.**

## SOURCES

- › <http://www.limitedwipsociety.org/>
- › <http://www.crisp.se/henrik.kniberg/kanban-vs-scrum.pdf>

## KANBAN SUMMARY (BY ISTQB)

Optimize flow of work in value-added chain

### Instruments:

- Kanban board
- Work-in-progress limit
- Lead time

Both Kanban and Scrum provide status transparency and backlogs, but:

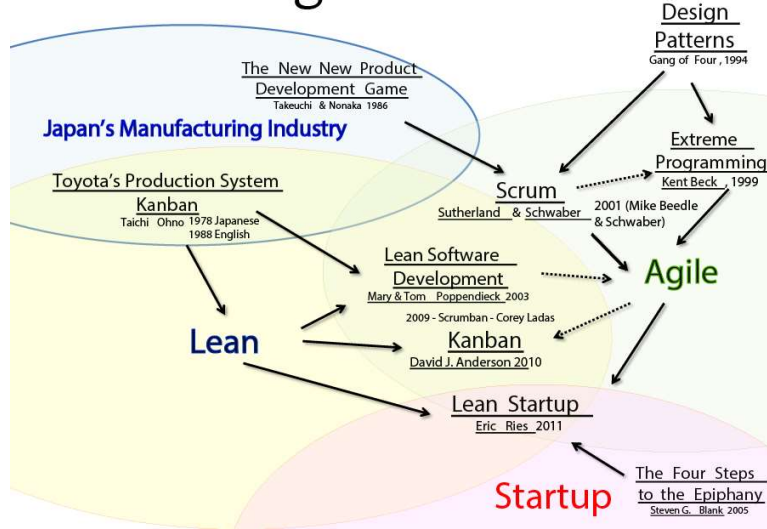
- Iteration is optional in Kanban
- Items can be delivered one at a time or in a release
- Timeboxing is optional



# LEAN & AGILE

## Agile & Lean

2013 Yasunobu Kawaguchi  
Mashup @agustinvillena - @josephhurtado



# DOCUMENTATION SYSTEMS

Helps in generating on-line documentation or offline reference manual from documented source files.

Combine source code with documentation and other reference materials

Make it easier to keep the documentation and code in sync

We will see:

- Doxygen
- Javadoc
- T3Doc

# DOXYGEN

Source code documentation generator tool, Doxygen is a documentation system for C++, C, Java, Objective-C, Python, IDL (Corba and Microsoft flavors), Fortran, VHDL, PHP, C#, and to some extent D.

## Most useful tags:

- `\file`
- `\author`
- `\brief`
- `\param`
- `\returns`
- `\todo` (not used in assignments)

# JAVADOC

Attach special comments, called *documentation comment* (or *doc comment*) to classes, fields, and methods. `/** ... */`

Use a tool, called *javadoc*, to automatically generate HTML pages from source code.

Javadoc Tags: Special keyword recognized by javadoc tool. Common Tags:

- @author Author of the feature
- @version Current version number
- @since Since when
- @param Meaning of parameter
- @return Meaning of return value
- @throws Meaning of exception
- @see Link to other feature

# T3DOC

TTCN-3 source code tagging

Standard: ETSI ES 201 873-10

Example

```
• /*****  
  ** @desc XYZ                **  
  ** Initialize to pre-trial defaults. **  
  **                          **  
  123  
  *****/
```

# TTCN-3 DOCUMENTATION TAGS

	Simple Data Types	Structured Data Types	Component Types	Port Types	Modulepars	Constants	Templates	Signatures	Functions (TTCN-3 and external)	Altsteps	Test Cases	Modules	Groups	Control Parts	Component local definitions	Used in implicit form (see clause 7)	Embedded in other tags
<a href="#">@author</a>	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
<a href="#">@config</a>											X						
<a href="#">@desc</a>	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
<a href="#">@exception</a>								X								X	
<a href="#">@member</a>		X <sup>1</sup>	X	X	X <sup>1</sup>	X <sup>1</sup>	X <sup>1</sup>									X	
<a href="#">@param</a>							X	X	X	X	X					X	
<a href="#">@priority</a>											X						
<a href="#">@purpose</a>											X	X					
<a href="#">@remark</a>	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
<a href="#">@reference</a>	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
<a href="#">@requirement</a>									X	X	X	X					
<a href="#">@return</a>								X	X							X	
<a href="#">@see</a>	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
<a href="#">@since</a>	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
<a href="#">@status</a>	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
<a href="#">@url</a>	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X
<a href="#">@verdict</a>									X	X	X	X					
<a href="#">@version</a>	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		

NOTE: <sup>1</sup> Preceding language elements of record, set, union or enumerated types only.