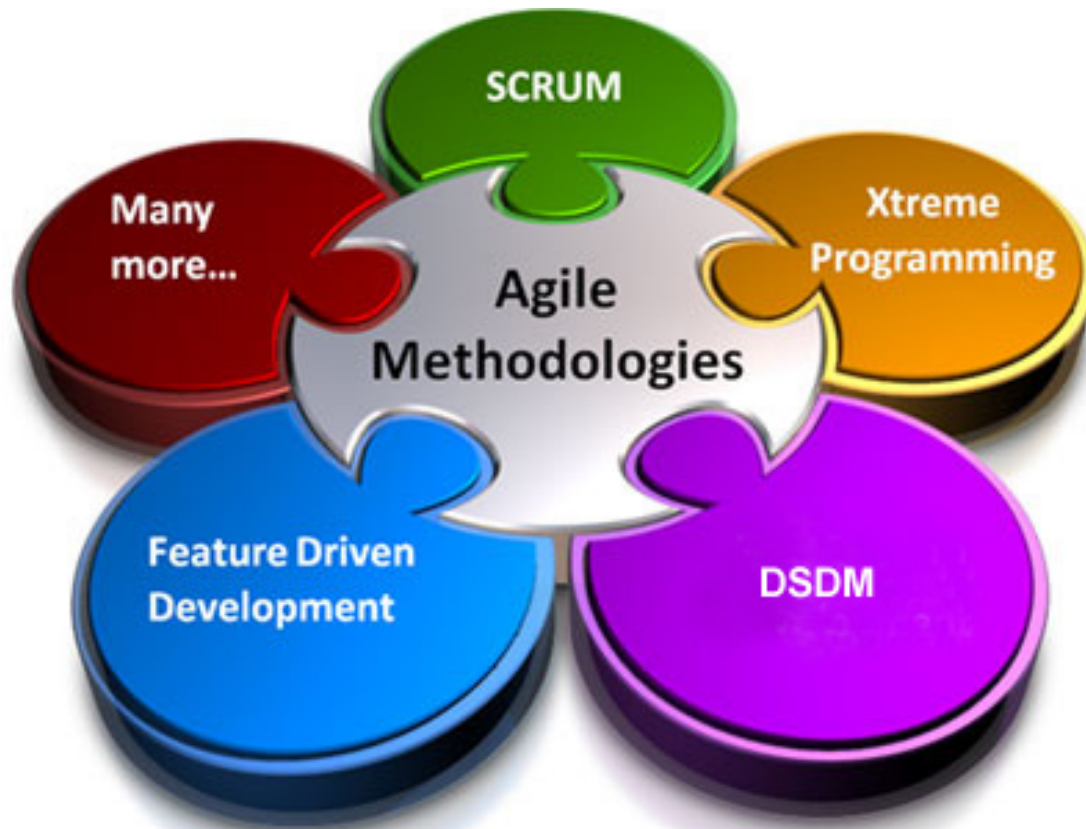


LEAN & AGILE DEVELOPMENT

Tibor Csöndes, Honorary Associate Professor

csondes@tmit.bme.hu

AGILE



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

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)

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

PRINCIPLES OF THE AGILE MANIFESTO (2/2)

Working software is the primary measure of progress.

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

Continuous attention to technical excellence and good design enhances agility.

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

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

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

AGILE APPROACHES

Agile methods are not unified, there is diversity

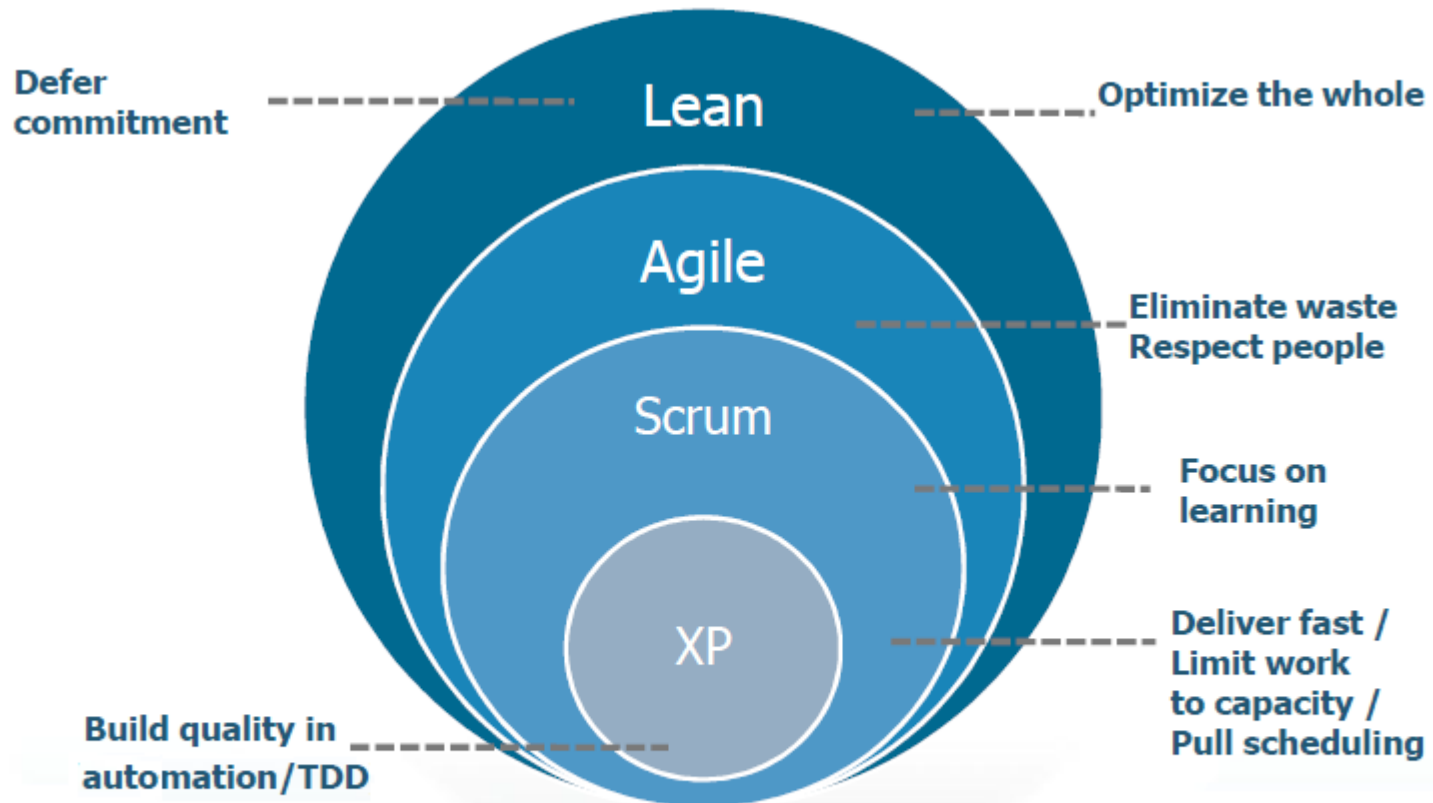
Each method implements the Agile Manifesto differently

We will consider

- Extreme Programming (XP)
- Scrum
- Kanban

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

HOW IT ALL FITS TOGETHER



EXTREME PROGRAMMING (XP)



Formulated in 1999 by Kent Beck, Ward Cunningham and Ron Jeffries

Agile software development methodology (others: Scrum, DSDM, Kanban)

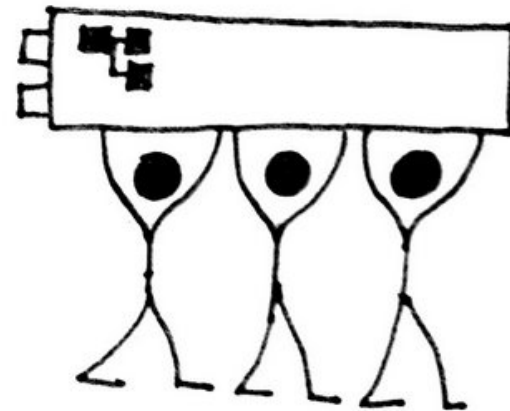
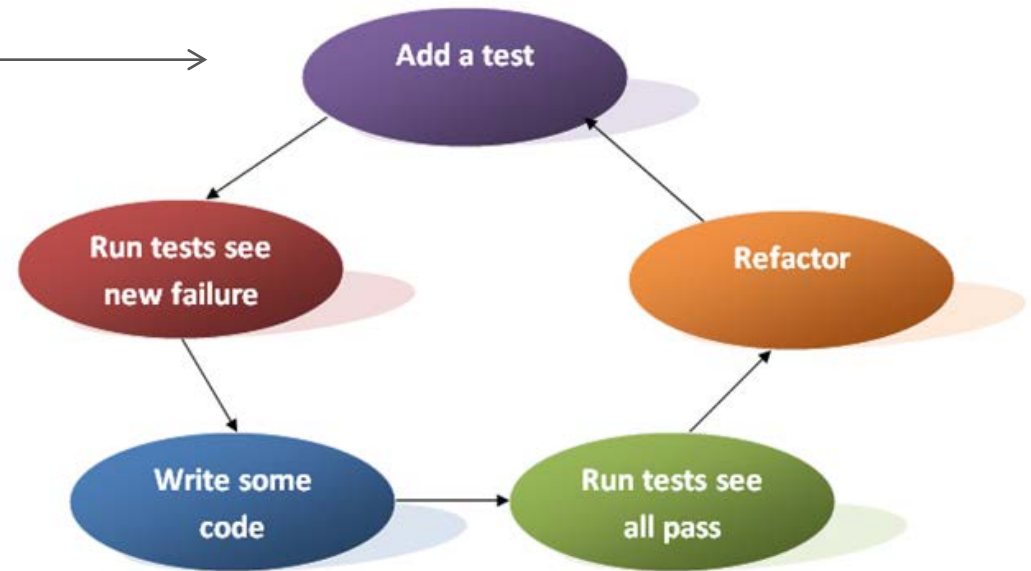
Developed in reaction to high ceremony methodologies

EXAMPLE OF PRINCIPLES FROM XP (EXTREME PROGRAMMING)

› Test Driven Development

› Continuous Integration

› Collective Code Ownership



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:

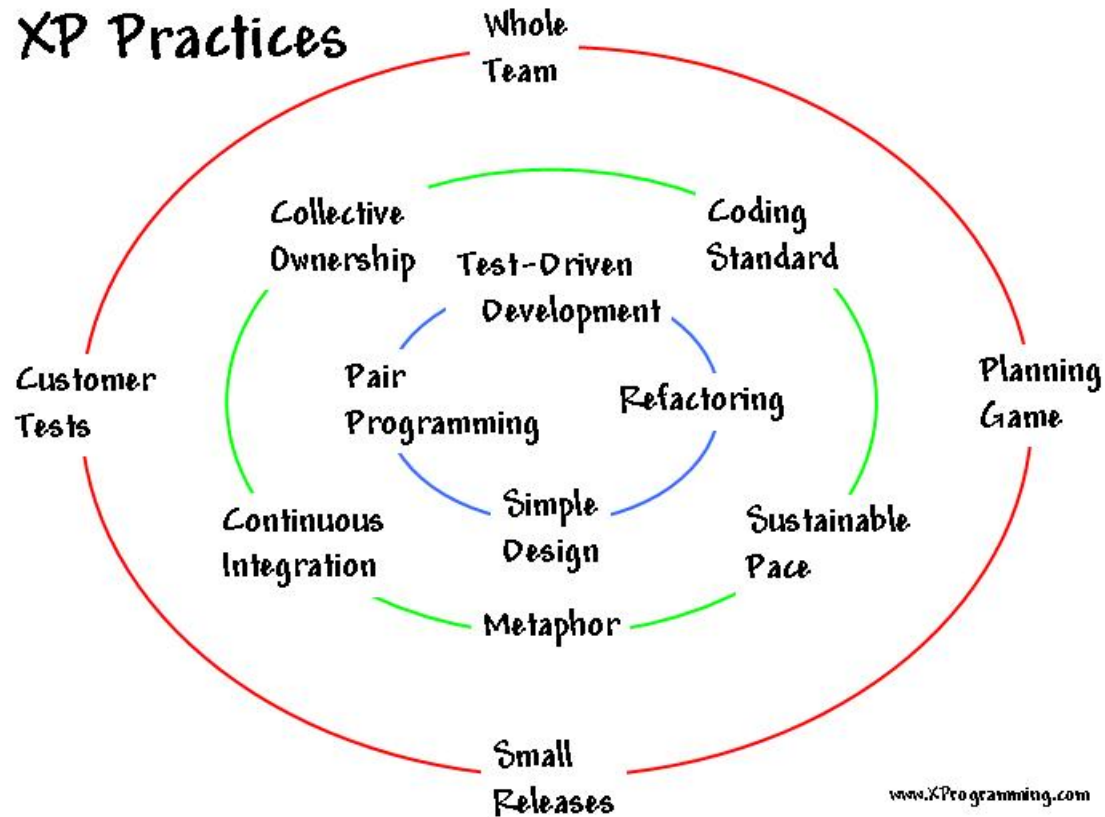
- All requirements will not be known at the beginning
- Requirements will change

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>)

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.

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

Not a one-time thing nor an up-front thing

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



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

XP PRACTICES: COLLECTIVE CODE OWNERSHIP

Any pair of programmers can improve any code at any time

No 'secure workspaces'

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: 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

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)

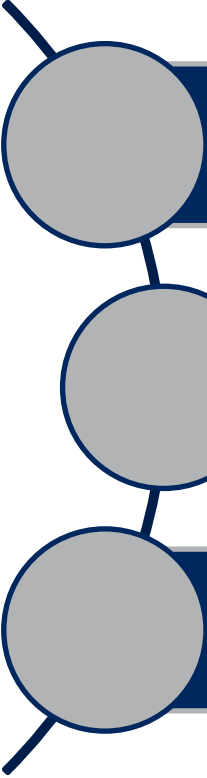
Important aspect of simple design: avoid speculative coding.

- 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.

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:

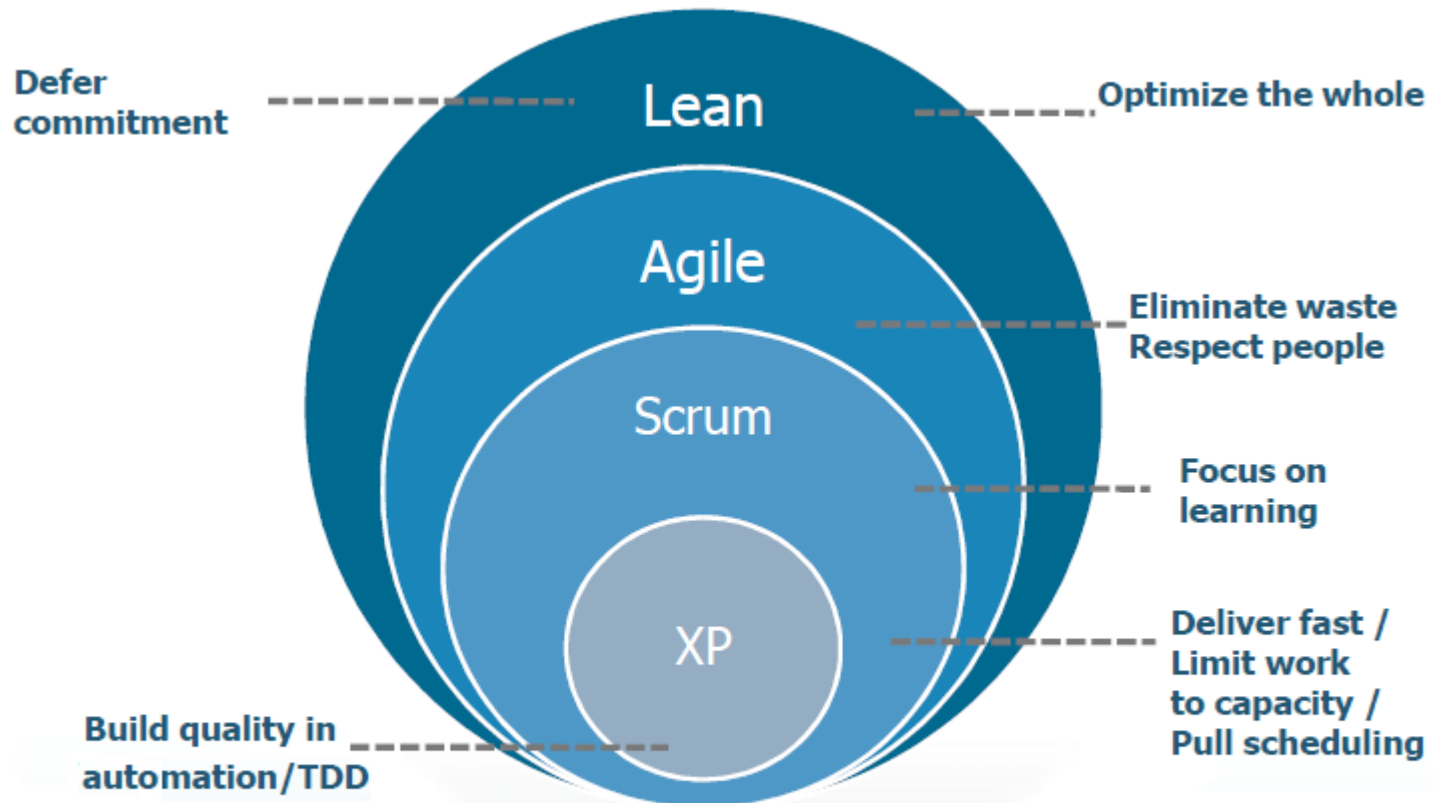
- humanity, economics, mutual benefit, self-similarity, improvement, diversity, reflection, flow, opportunity, redundancy, failure, quality, baby steps, accepted responsibility

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

HOW IT ALL FITS TOGETHER



THE SCRUM FRAMEWORK





ROLES

Product Owner



Team



Coach





USER STORIES AND ESTIMATION (1)

Describe requirements in product backlog

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

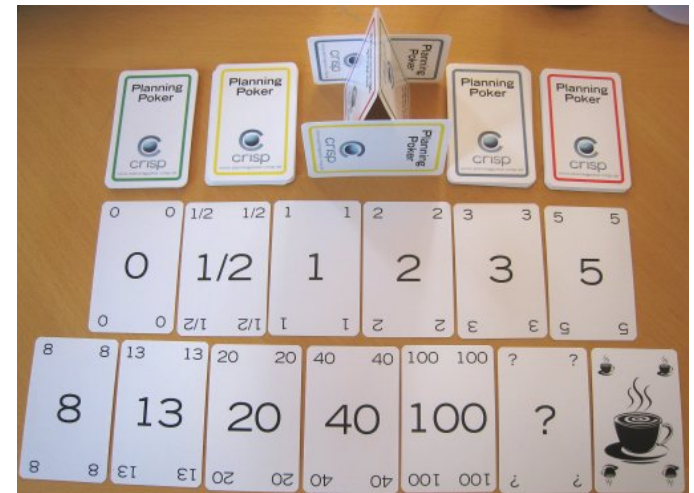
Example:

- 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

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



SPRINT PLANNING

Time-box (eg. 2 hours)

- 1st - 1 hours max. for team to select Product Backlog and sets goal with Product Owner
- 2nd - 1 hours max. for team to define Sprint Backlog to build functionality

Attendees

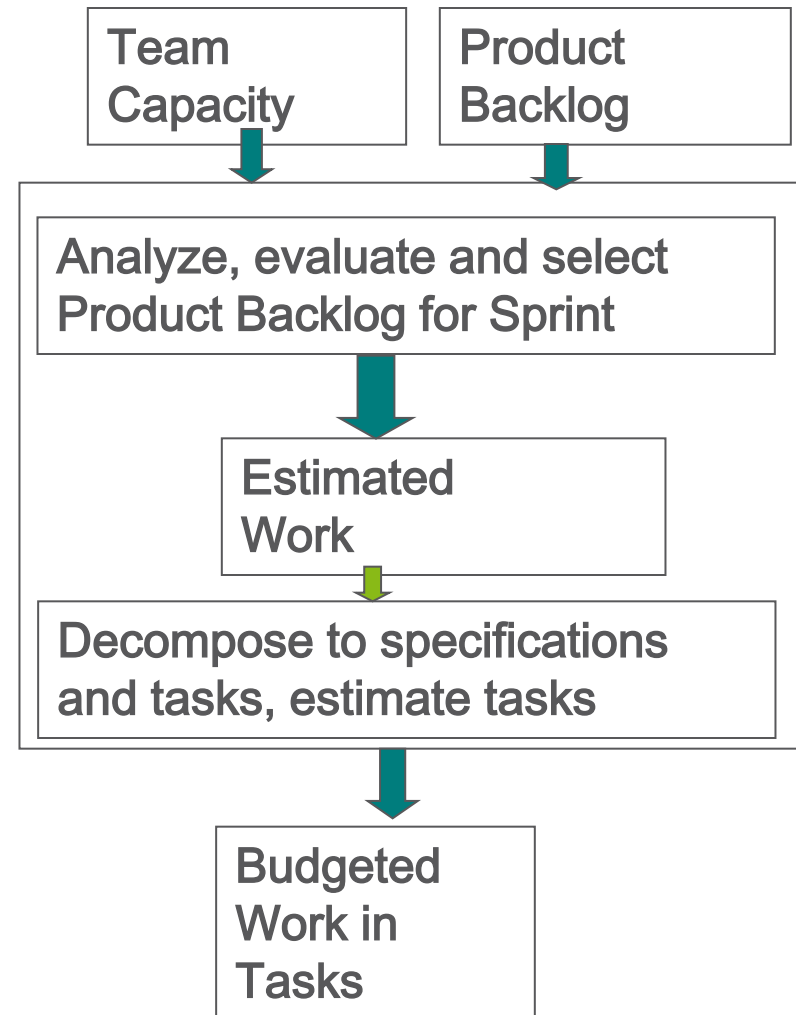
- Product owner, team and Scrum Master

Product owner must prepare the Product Backlog prior to the meeting

- Product owner decides what the product backlog constitutes

Output: Sprint backlog

- Tasks, task estimates, task assignments



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

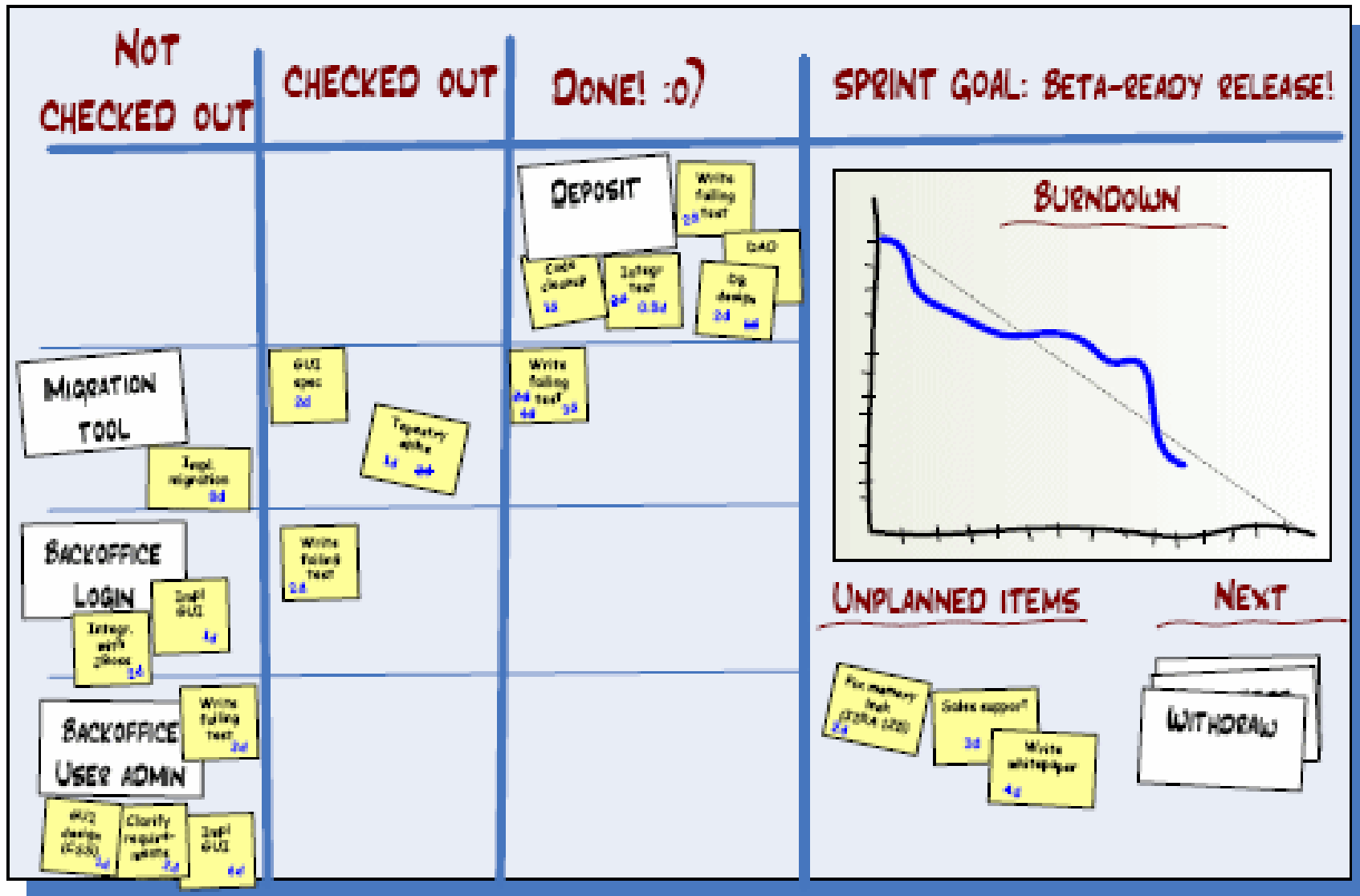
Any build/deployment/configuration changes implemented/documented/communicated

Relevant documentation/diagrams produced and/or updated

Remaining hours for task set to zero and task closed

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

TRANSPARENCY – TASK BOARD



RETROSPECTIVES

Set the stage

- Focus for this retrospective

Gather data

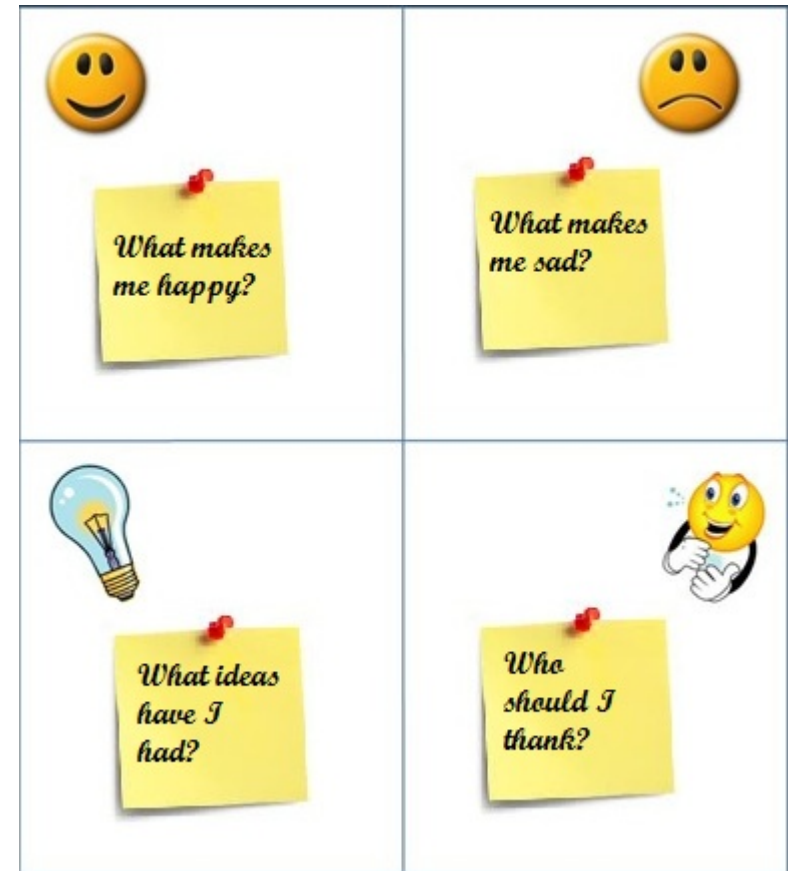
- Ground it in facts, not opinions

Generate insights

- Observe patterns

Decide what to do

- Move from discussion to action



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 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.

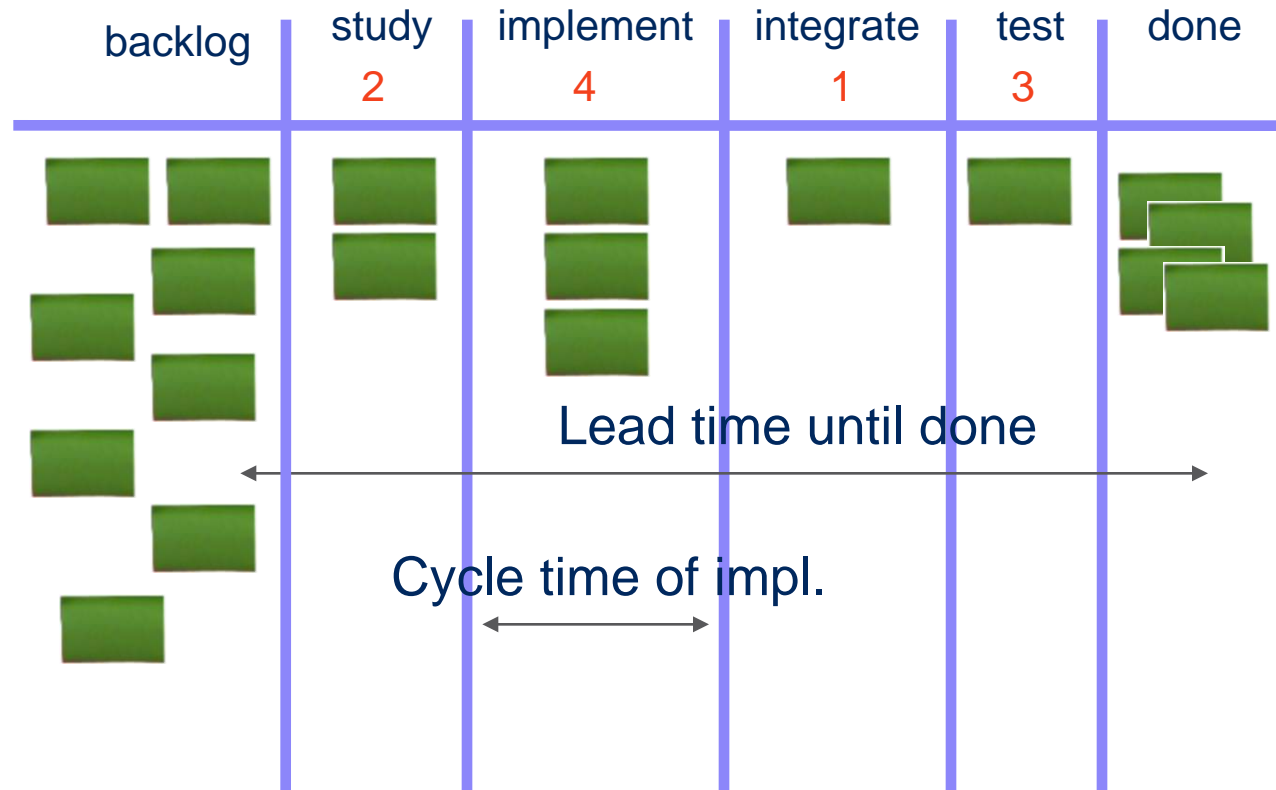


KANBAN BASIC RULES

Visualize the workflow

Limit Work In Progress (WIP)

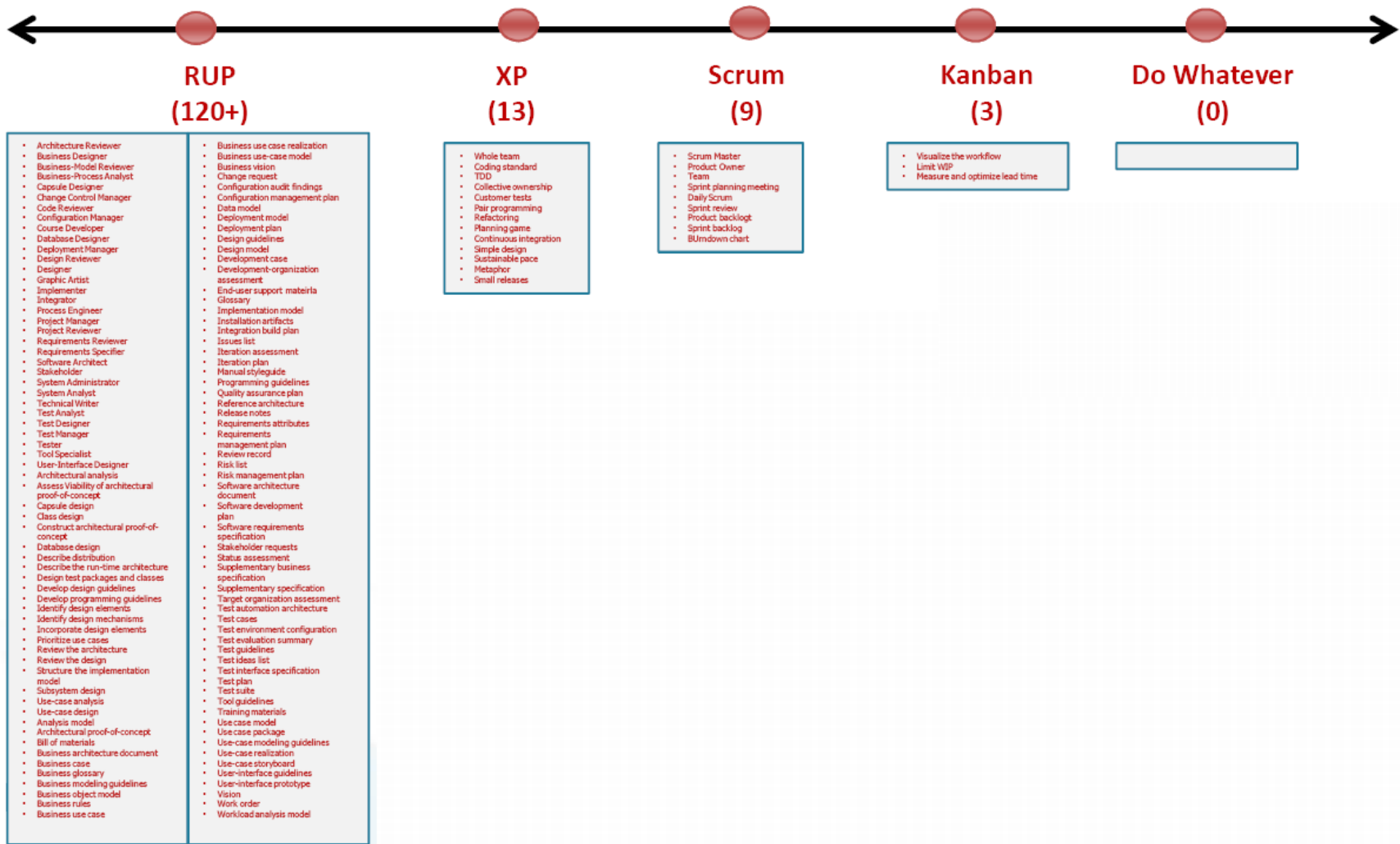
Measure and optimize lead time



PROCESSES

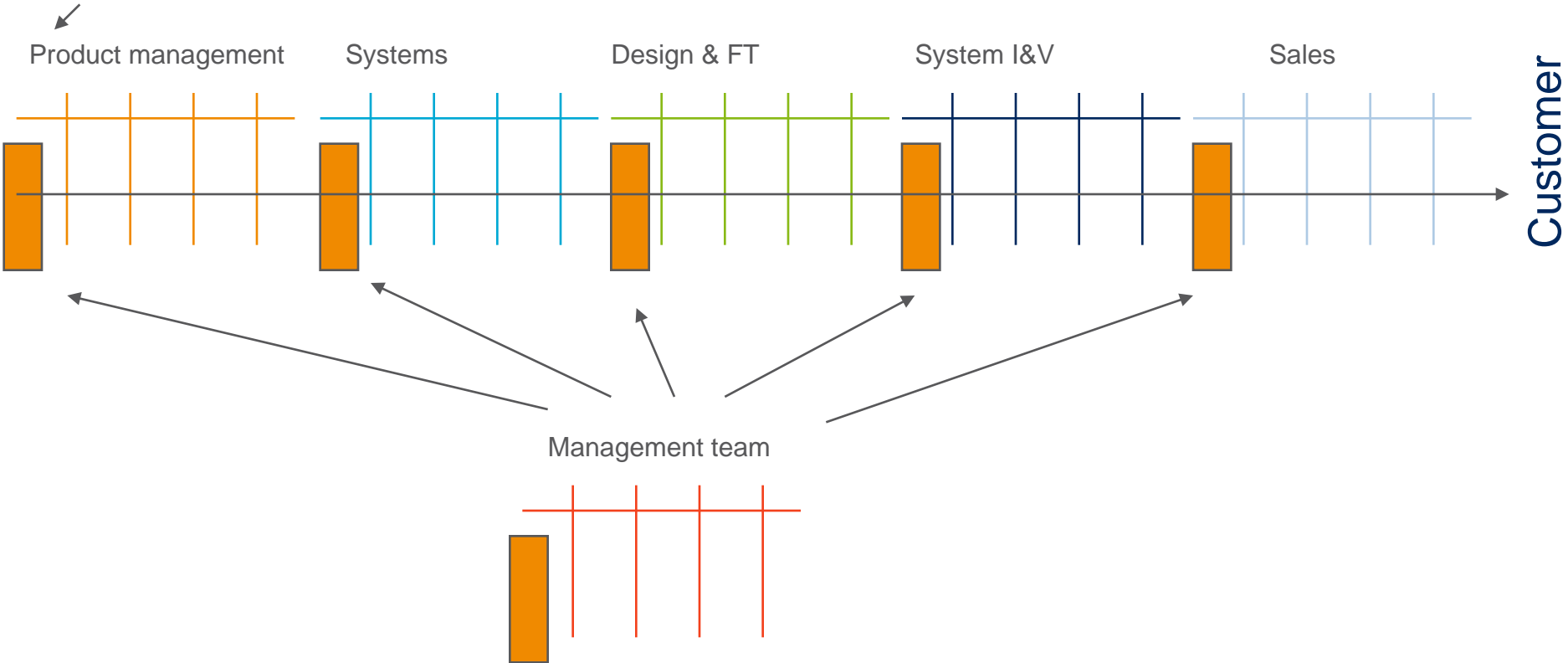
More prescriptive

More adaptive

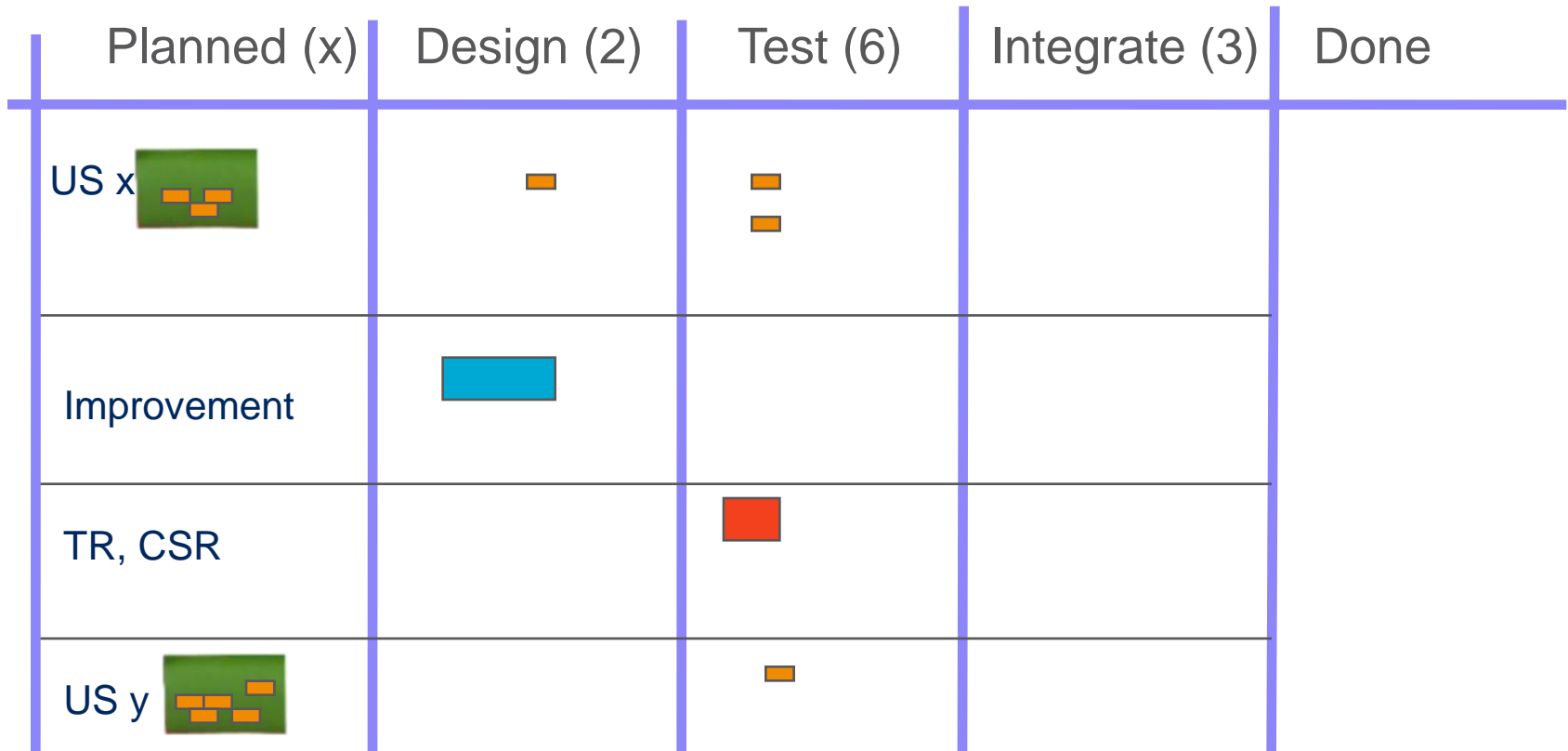


VISUALIZE THE WORKFLOW

Customer

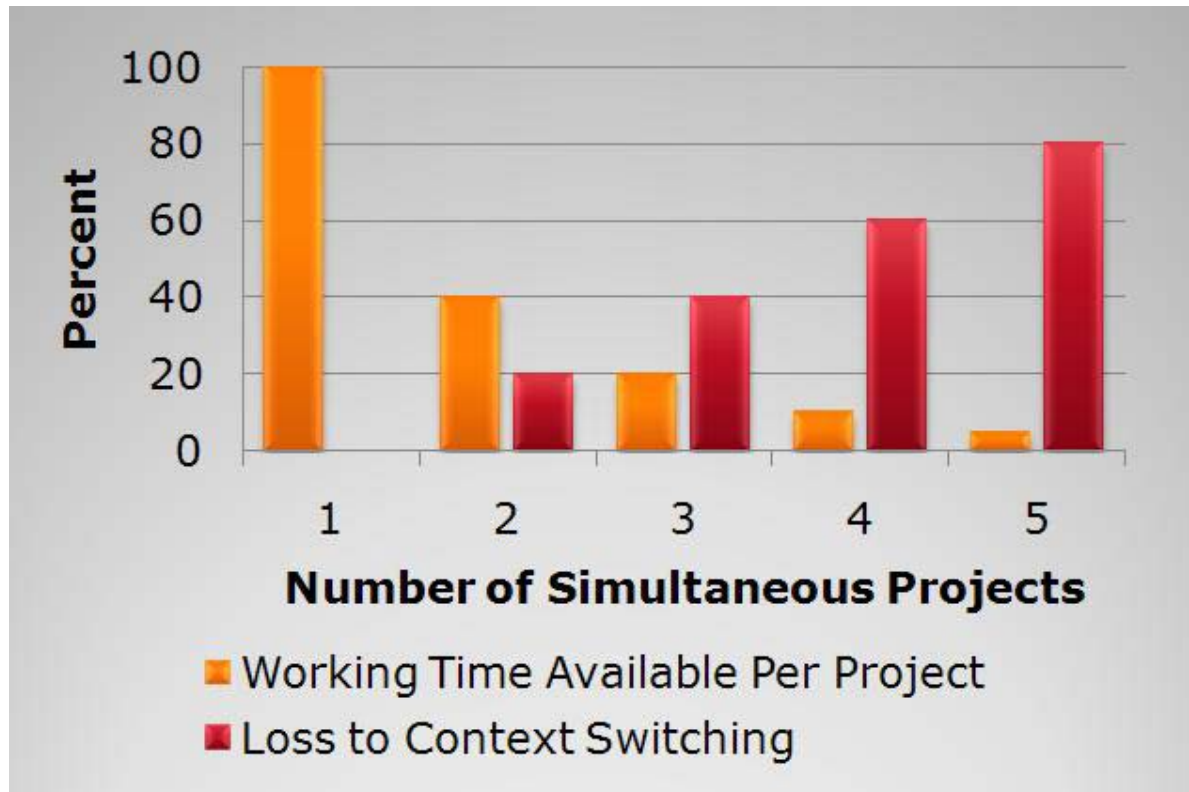


VISUALIZE THE WORKFLOW

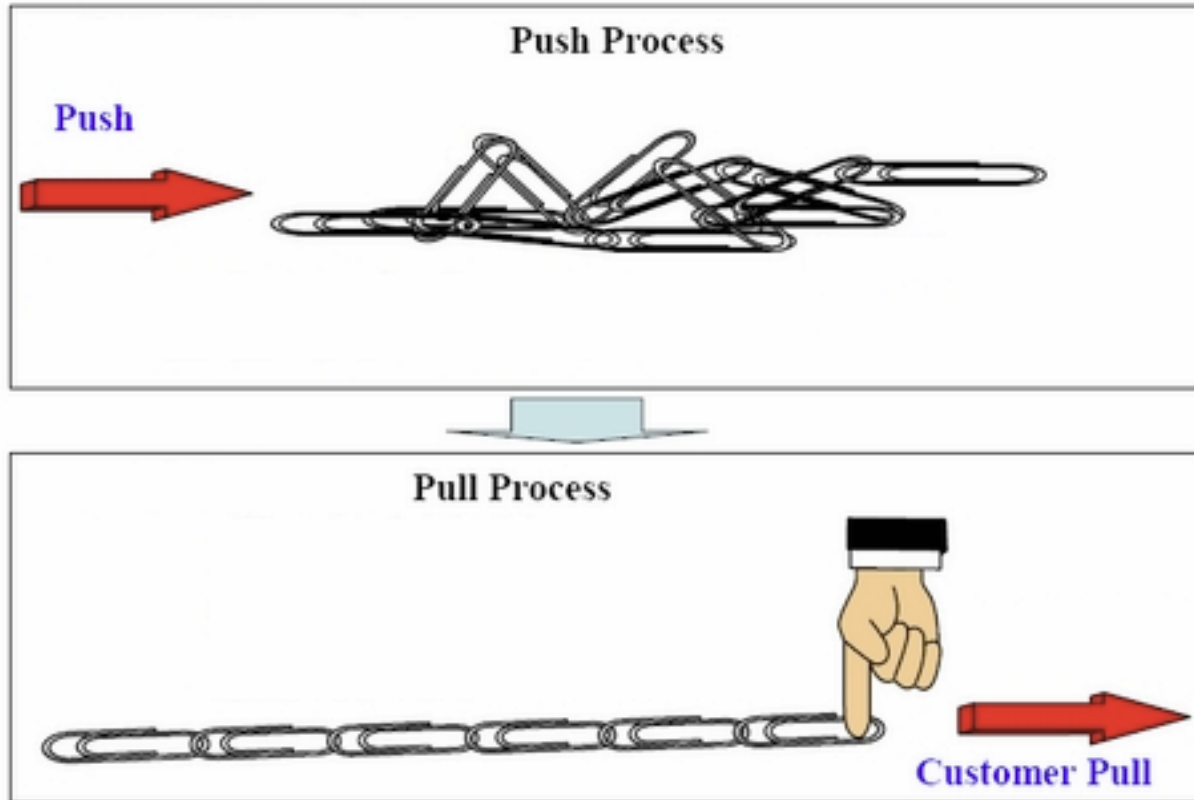


LIMITING WORK IN PROGRESS

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

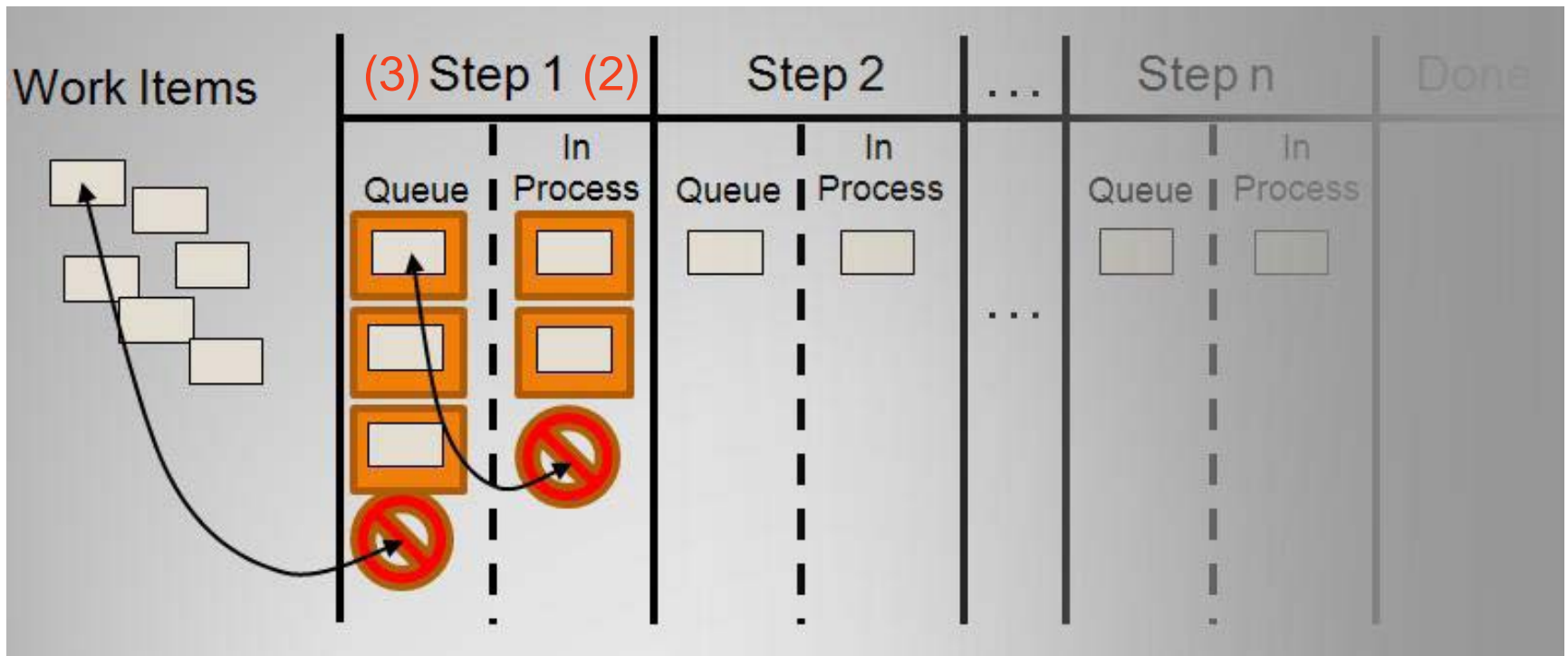


LIMITING WORK IN PROGRESS



LIMITING WORK IN PROGRESS

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



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

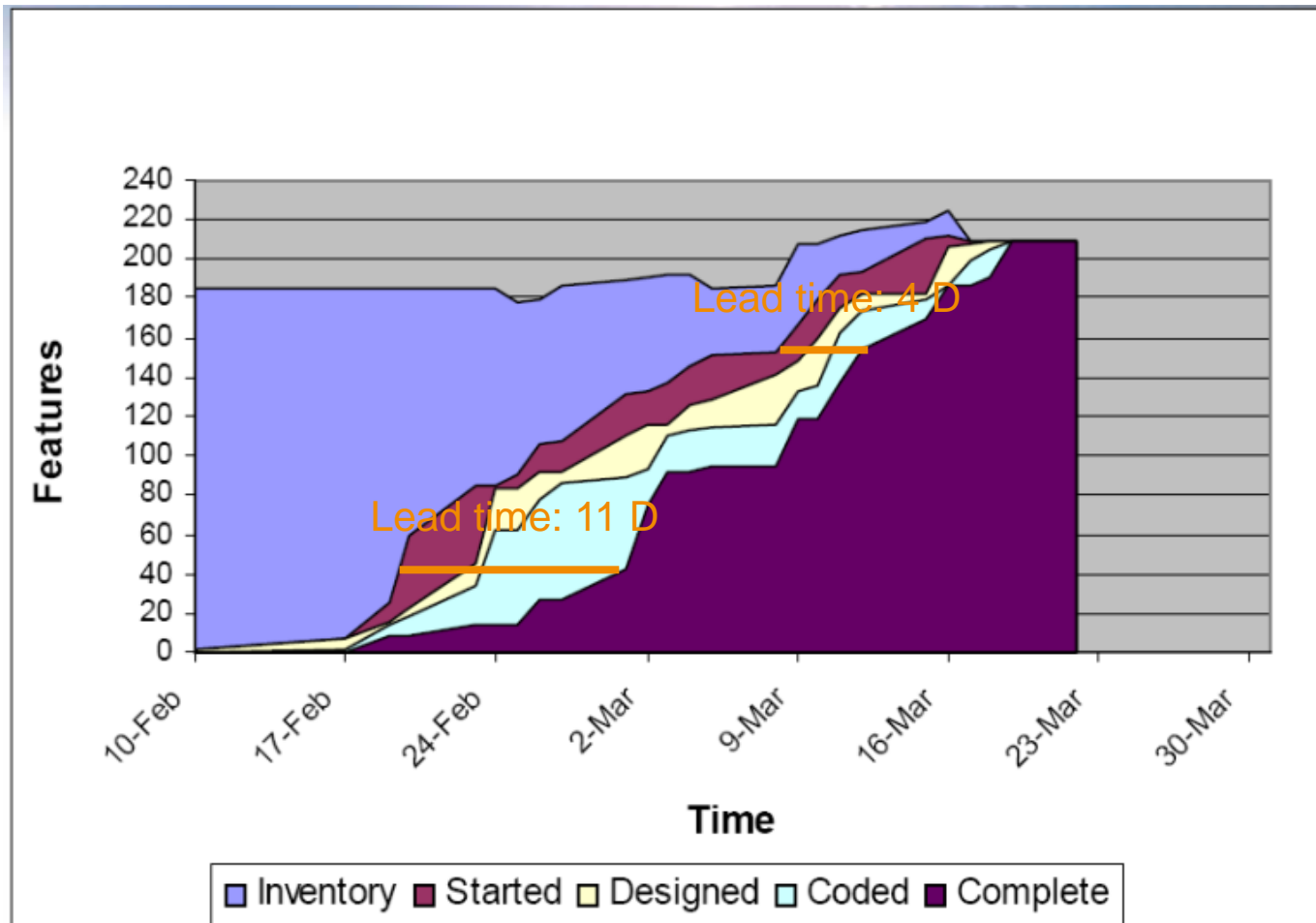
LITTLE'S LAW FOR QUEUEING THEORY

*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.

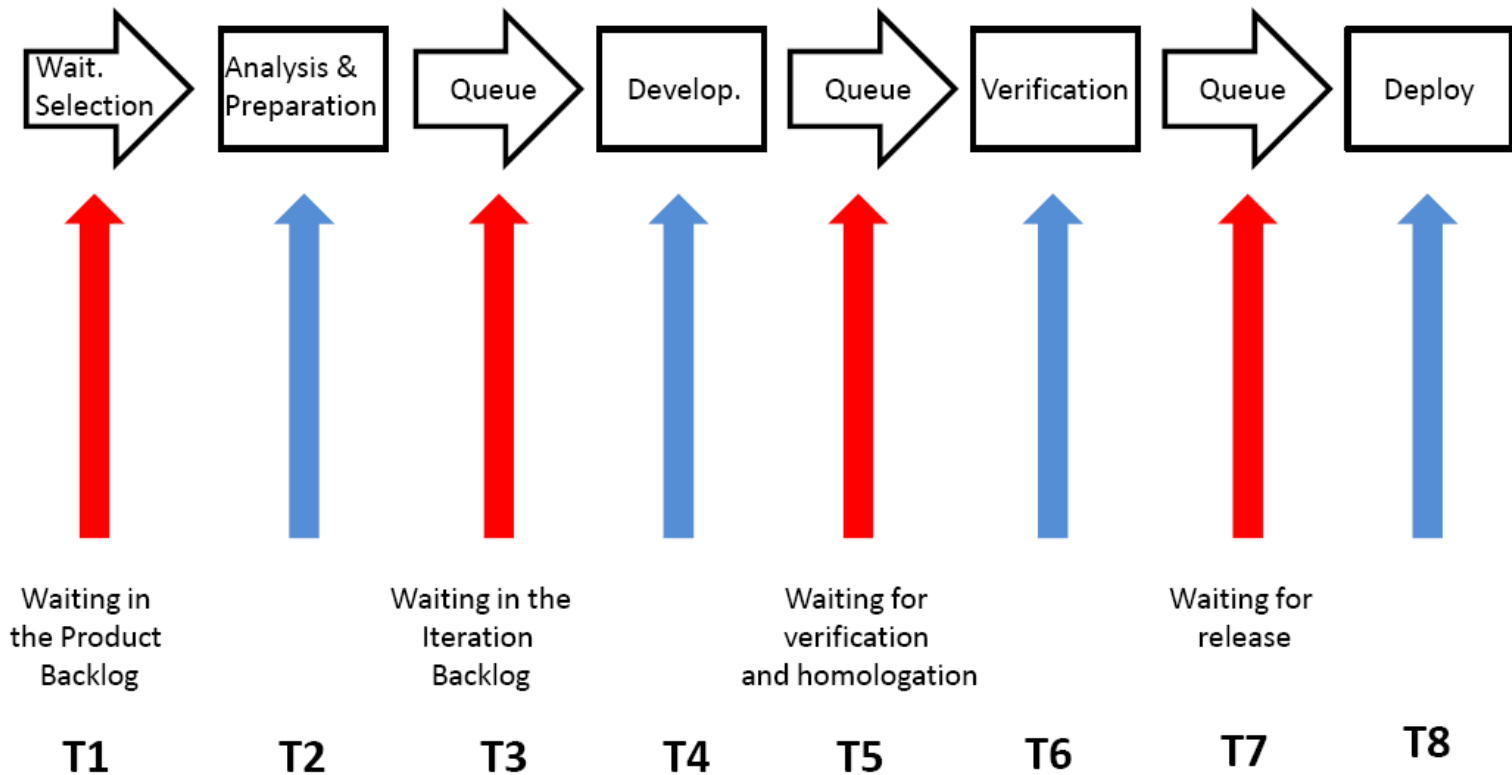
USE CUMULATIVE FLOW DIAGRAMS TO VISUALIZE WORK IN PROGRESS



www.agilemanagement.net/Articles/Papers/BorConManagingwithCumulat.html

VALUE STREAM MAPPING

Value Stream for Product XYZ



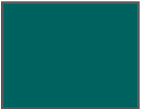







$$\text{Efficiency (\%)} = \frac{\text{TValue} * 100}{\text{TValue} + \text{TWaste}}$$

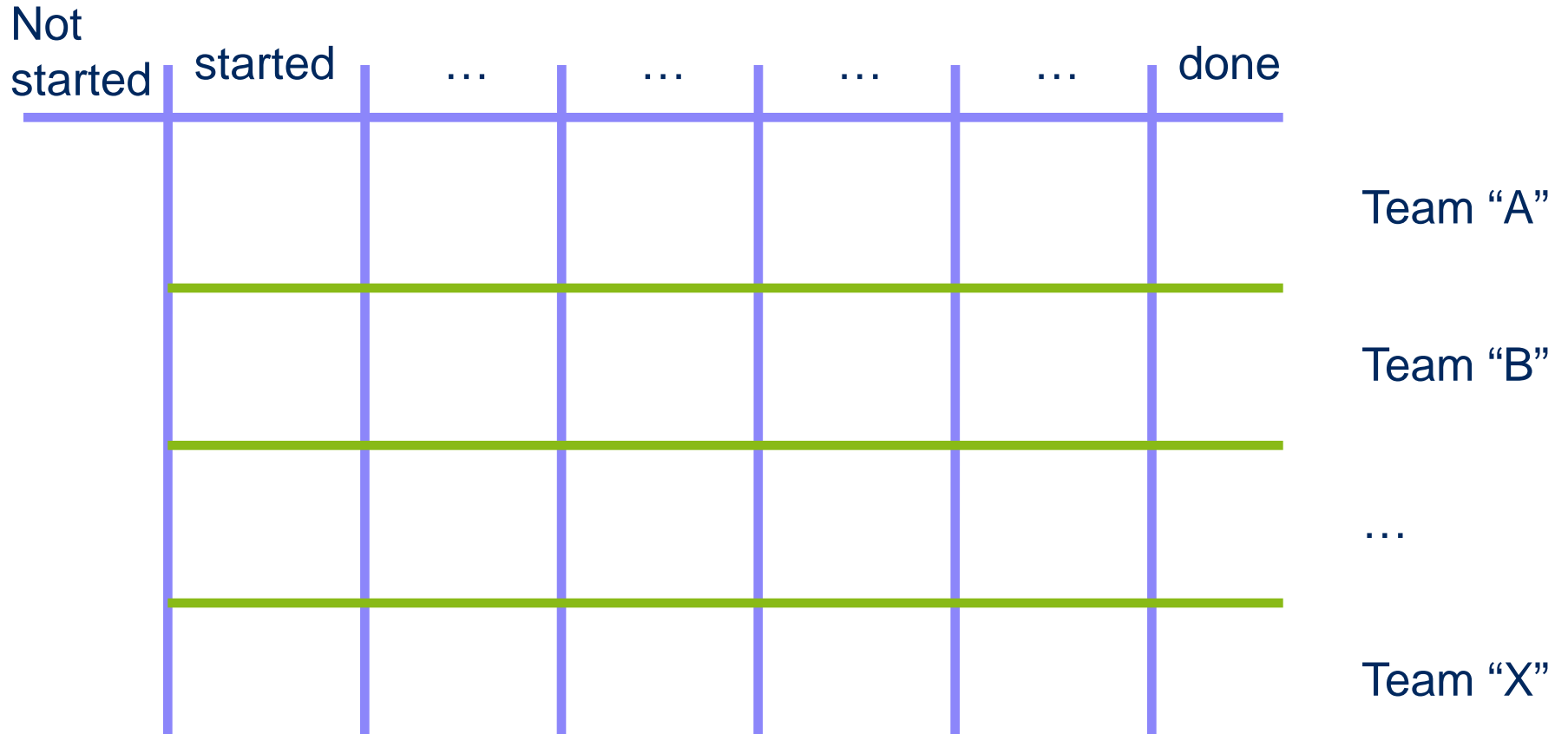
PRIORITIZATION METHOD



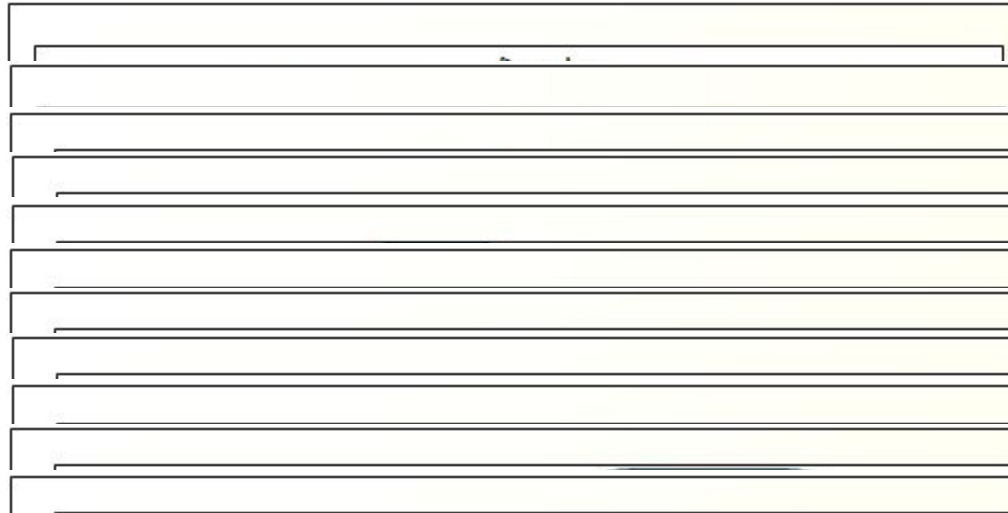
- › Business value?
- › Cost of delay classification

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

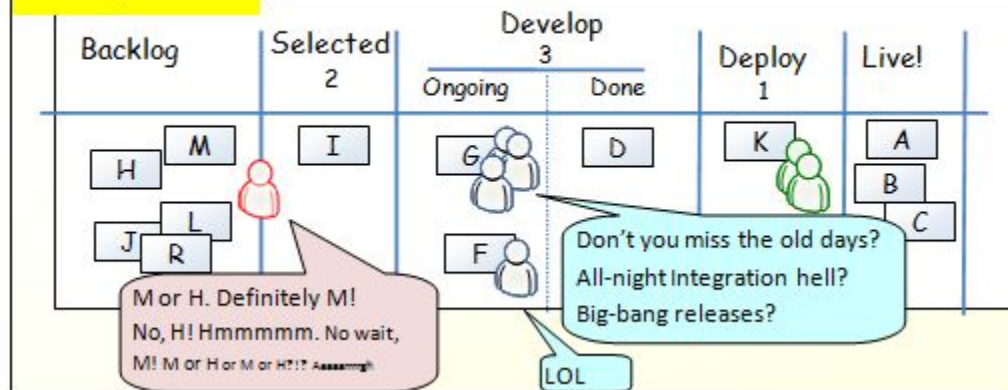
SCALING – SWIM LANES



ONE DAY IN KANBAN LAND



A few days later...



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

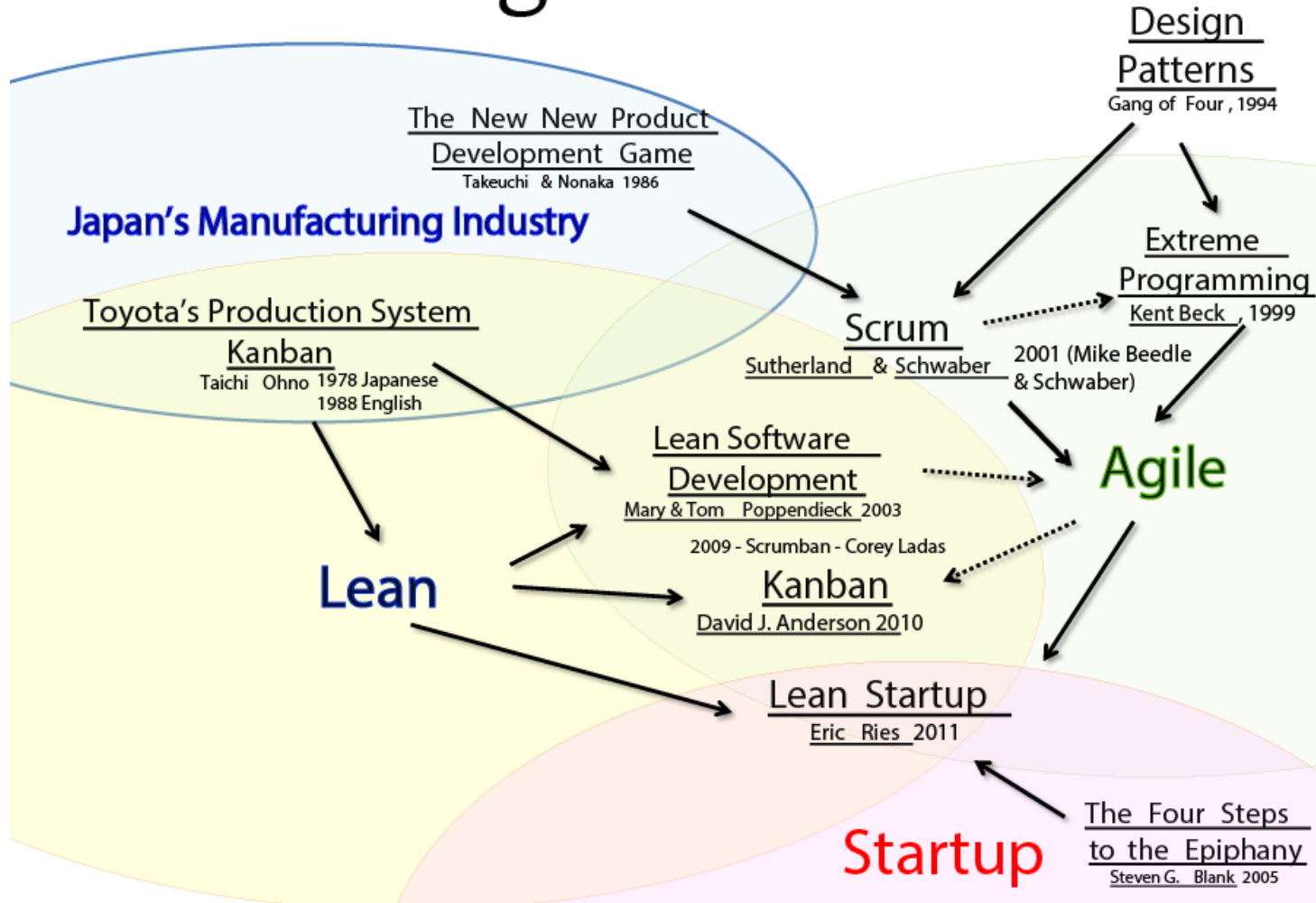
Both Kanban and Scrum provide status transparency and backlogs, but:

- 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 @agustinvilena - @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

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
@author	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
@config											X						
@desc	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
@exception								X								X	
@member		X ¹	X	X	X ¹	X ¹	X ¹									X	
@param							X	X	X	X	X					X	
@priority											X						
@purpose											X	X					
@remark	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
@reference	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
@requirement									X	X	X	X					
@return								X	X							X	
@see	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X
@since	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
@status	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
@url	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X
@verdict									X	X	X	X					
@version	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		

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