

Softwaretechnik II

Oliver Hummel, IPD

Topic 3

Agile Development

SOFTWARE DESIGN AND QUALITY GROUP
INSTITUTE FOR PROGRAM STRUCTURES AND DATA ORGANIZATION, FACULTY OF INFORMATICS

sdq.ipd.kit.edu



Overview on the Agile Lecture

■ Content

- Motivation
- Very briefly: XP (Values, Principles, and Practices)
- Scrum
 - Process, Roles, Artefacts, and Meetings
- Conclusions

■ Learning Goals

- Get acquainted with agile development ideas and principles
- Understand how to perform a project with a Scrum process
- Be able to critically evaluate agile practices
 - and to chose and apply helpful techniques for a given project context

Herzliche Einladung

- Zur Gastvorlesung von Dr. Joachim Melcher
 - andrena objects

Agile Softwareentwicklung in der Praxis

(Arbeitstitel ☺)



- **Montag, 4. November 2013, 15:45 im Daimler-Hörsaal**
- Dr. Joachim Melcher studierte Informatik an der Universität Karlsruhe (TH). Danach war er als wissenschaftlicher Mitarbeiter am Karlsruher Institut für Technologie u.a. in der Java-Programmier-Ausbildung tätig und promovierte dort im Bereich Geschäftsprozessmanagement. Er ist Lehrbeauftragter an der Hochschule Ludwigshafen und der Dualen Hochschule in Karlsruhe. Seit 2011 arbeitet er bei der andrena objects AG, wo seine Interessen vor allem bei Methoden zur Softwarequalitätssicherung sowie agilen Entwicklungsmethodiken liegen.

Motivation

Traditional Methods

-> Heavyweight: 1825 g

- Environment
 - Complicated software systems
 - High market competition
 - Time and budget pressure
 - No predefined conditions
 - ...
- Problems
 - Too “heavyweight” processes
 - Big design up front
 - Too many artifacts
 - Inflexible
 - ...
- Solutions
 - Extreme Programming (XP)
 - Scrum
 - Crystal
 - Feature-Driven Development ...



Agile Methods
-> Lightweight: 346 g

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
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler

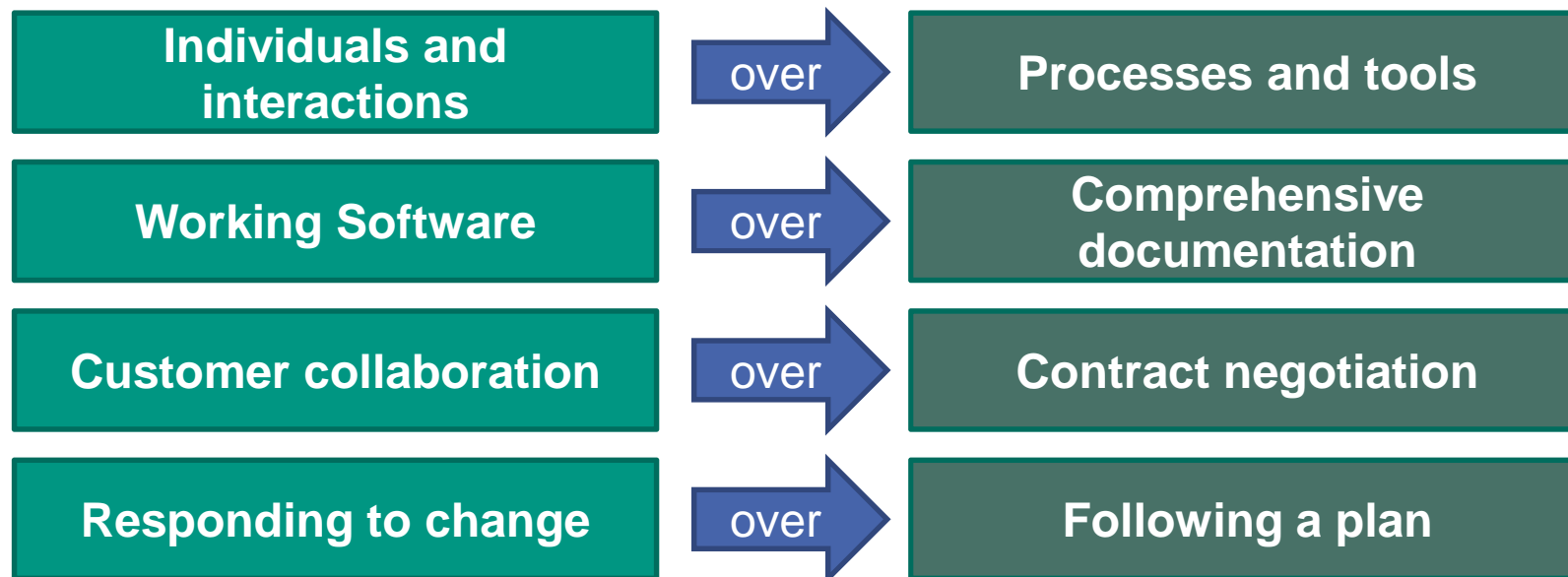
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick

Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

AgileManifesto.org, 2001

Agile Manifesto

- “That is, **while there is value in the items on the right**, **we value the items on the left more**”



- These ideas may also be used in “non agile” methods!

[AgileManifesto.org]

Exercise: Agile vs. „Before“ Agile



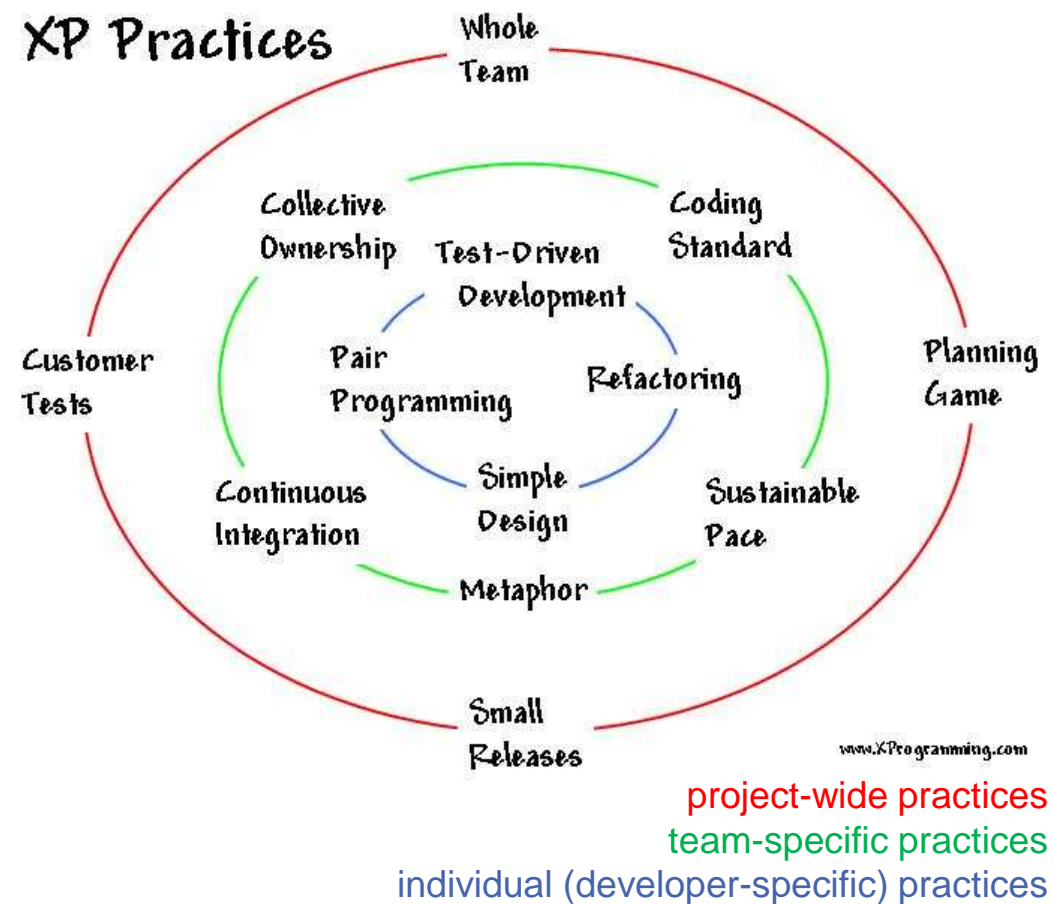
■ *Create Conceptual Pairs*

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.
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 is essential.
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 behaviour accordingly.

[AgileManifesto.org]

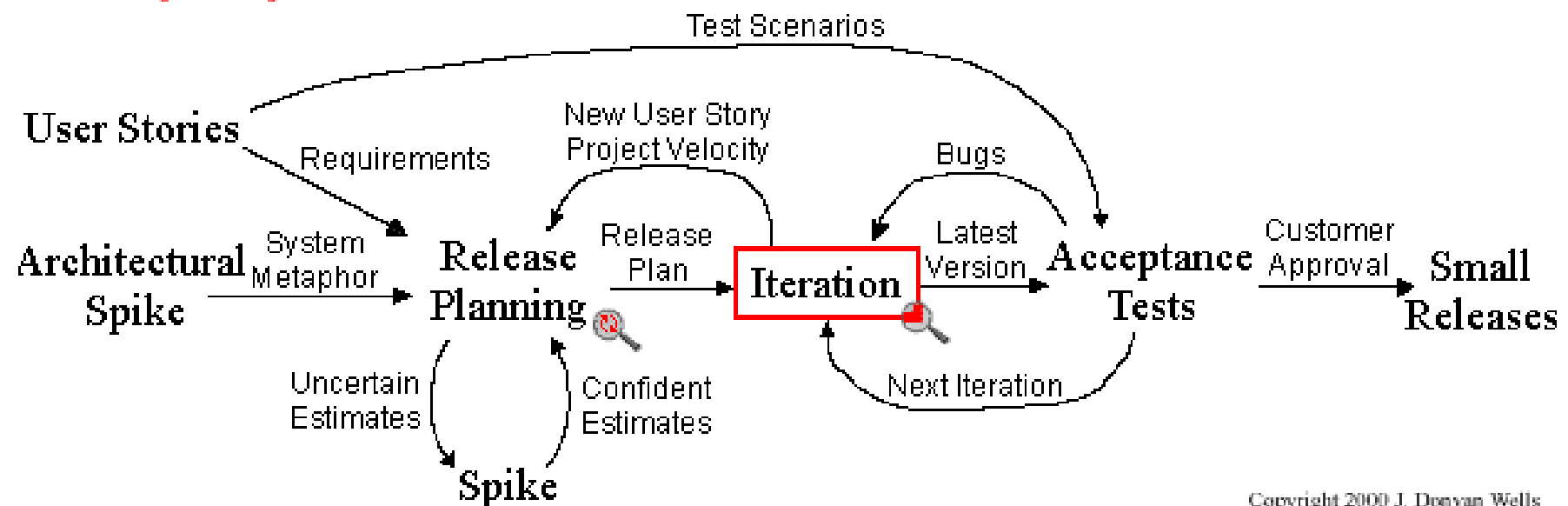
Extreme Programming

- XP = a set of *values, principles and practices*
 - practices partially existed before XP, but were consolidated by it
- Main XP values
 - Communication
 - Simplicity
 - Feedback
 - Courage
- Some XP Principles
 - quick delivery
 - rapid feedback
 - keep it simple
 - incremental change
 - embrace change





Extreme Programming Project



Copyright 2000 J. Donovan Wells

[extremeprogramming.org]

Common XP (Agile) Criticisms

- An ad-hoc process that cannot be replicated
 - ➔ not usable for large projects
- Missing documentation
- Clients need to cooperate in a project
- Some XP practices are not yet fully validated
 - E.g. pair programming
 - current understanding: more expensive, but produces better quality
 - high learning curve for TDD due to new way of thinking
 - unclear if it produces a better quality software

[dilbert.com]

■ ...



The „First“ XP Project

- The Chrysler Comprehensive Compensation project was started in 1993 by Tom Hadfield, Director of Payroll Systems. The end goal was to build a new system to support all the payroll processing for 87,000 employees by 1999. In 1996 Kent Beck was hired to get the thing working; at this point the system had not printed a single paycheck.
- In 1997 the development team adopted a way of working which is now formalized as Extreme Programming. The one-year delivery target was nearly achieved, with the actual delivery being a couple of months late; the small delay being primarily due to lack of clarity regarding some business requirements.
- (...)

http://en.wikipedia.org/wiki/Chrysler_Comprehensive_Compensation_System

Further Reading

1. Inform yourself about the C3 project on the Web.
 1. Briefly summarize and discuss your findings.
 2. How well comes XP out of this affair in your opinion?
 3. What might have went wrong?

- Scrum is an **agile management framework** supporting the development of software
 - **having three roles, four ceremonies, and three artefacts**
 - delivering working software in Sprints
 - usually 30-day iterations
 - As an agile framework Scrum incorporates the **values of the agile manifesto**
 - create value and satisfy the customer
 - without over-exploiting your most precious resource: the people
- ➔ **it's the people** that develop the software

Scrum (rugby):
a rugby restart after an interruption
[businessanalystmentor.com]



Scrum in 6 Bullet Points [adapted from Scrum Center]

- The Product Owner is responsible for the creation and maintenance of a prioritised list of required features (usually collected as user stories), called the Product Backlog, a complete and dynamic ToDo list of the project.
- Before each sprint a Sprint Planning Meeting takes place where the Team decides how many of the features having the highest priority can be delivered during the sprint and records them in a Sprint Backlog.
- The Team synchronises its actions during the sprint at a Daily Scrum Meeting, the sprint progress is recorded in a Burn Down Chart.
- The Scrum Master trains the Team, removes the impediments and secures effective work of the Team.
- Valuable functionality is being developed during the sprint – a potentially deliverable product increment.
- The Team presents each product increment at a Sprint Review Meeting, after which it discusses potential improvements at a Retrospective Meeting

➔ *draw this information as a domain (meta) model...*



Scrum in 6 Bullet Points [adapted from Scrum Center]

- The **Product Owner** is responsible for the creation and maintenance of a prioritised **list of required features** (usually collected as **user stories**), called the **Product Backlog**, a complete and dynamic **ToDo list** of the project.
- Before each **sprint** a **Sprint Planning Meeting** takes place where the **Team** decides how many of the **features** having the highest **priority** can be delivered during the sprint and records them in a **Sprint Backlog**.
- The Team synchronises its actions during the sprint at a **Daily Scrum Meeting**, the sprint **progress** is recorded in a **Burn Down Chart**.
- The **Scrum Master** trains the Team, removes the **impediments** and secures effective **work** of the Team.
- Valuable **functionality** is being developed during the **sprint** – a potentially deliverable product **increment**.
- The Team presents each product increment at a **Sprint Review Meeting**, after which it discusses potential **improvements** at a **Retrospective Meeting**

➔ *draw this information as a domain (meta) model...*



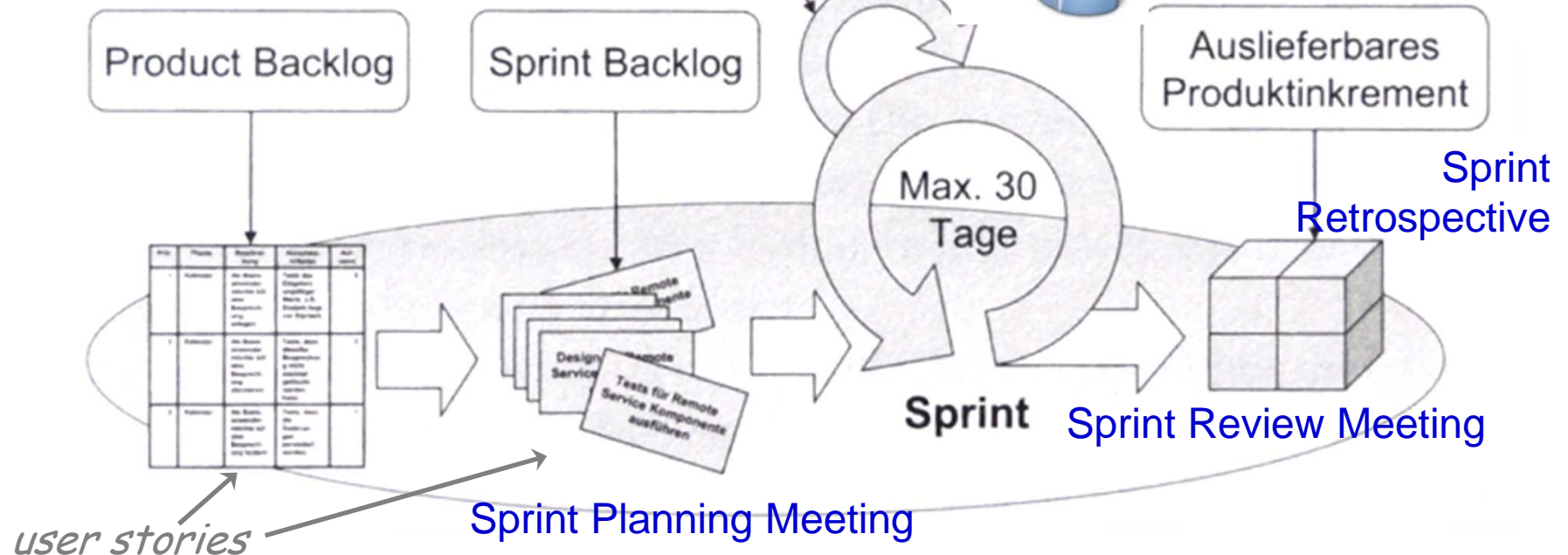
Scrum at a Glance



- three roles
- four meetings
- three artefacts



Daily Scrum



adapted from [Pichler]

Types of Scrum Meetings

- *Sprint Planning*

- is used to develop a detailed plan for the iteration



- *Sprint Review*

- demonstrate potentially shippable code, developed during the sprint

- *Daily Scrum*

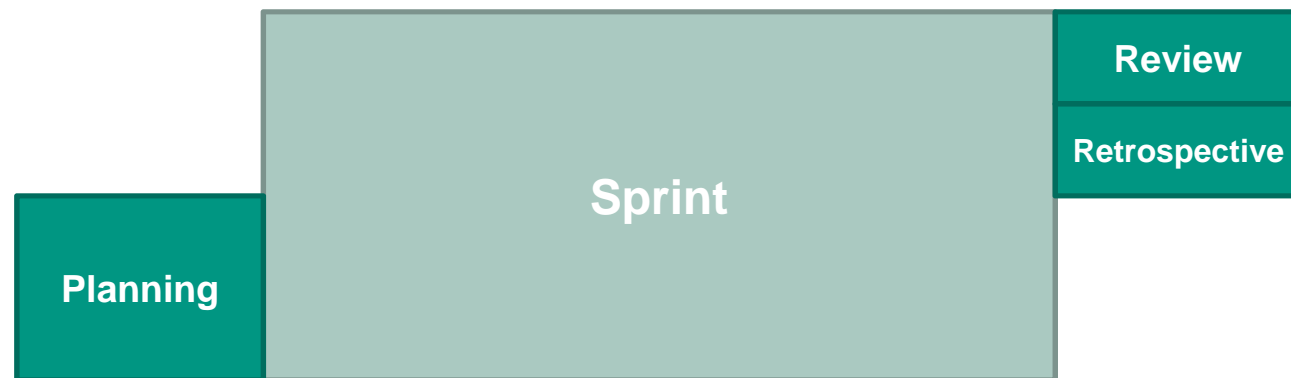
- share information (through 3 questions!)
- discover dependencies
- adjust the work plan
- address individual needs and identified problems

- *Sprint Retrospective*

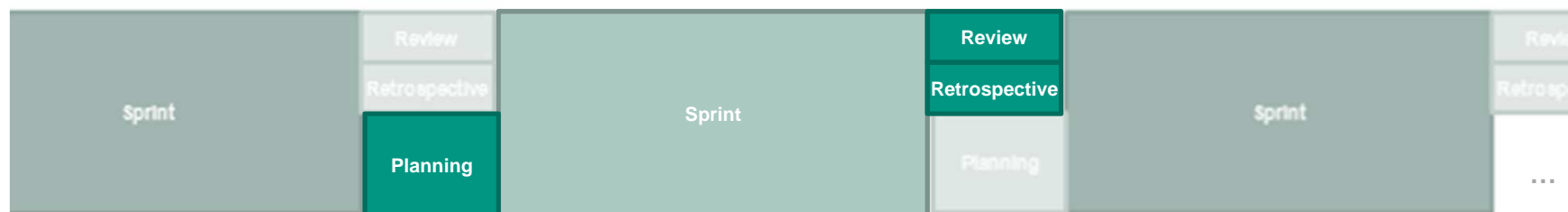
- assesses the work in sprint, identifies good and bad practices

Scrum: Sprint

- *Sprint* – a development iteration of a fixed length (typically 2–4 weeks)
 - in which a set of product backlog items that the Team has committed itself to is developed [Scrum Center]



- Development process is a set of sprints:



Roles in Scrum



- Scrum basically gets along with only 3 *“pig” roles*
 - **Product Owner**
 - responsible for the system
 - decides which system increments make up a release
 - **Team(member)**
 - develops the software
 - decides how many requirements can be implemented in a sprint
 - **ScrumMaster**
 - supports in applying Scrum correctly
 - assists in the continuous improvement of productivity



- Other roles are considered being *“chicken” roles*
 - such as customer stakeholders, managers etc.
- Pigs are committed, chickens are only involved
 - *Chickens must not tell pigs how to do their work*
- **Remember:** individuals and interactions over processes and tools!



Scrum: Product Owner

- Defines the product
 - *decides what will be delivered and he is responsible for the result*
 - comparable with product manager, project lead, and chief architect in one person
- Tasks
 - represents the customer's interest
 - helps team to clarify questions (and quickly)
 - is responsible for the Team working on the right features
 - changes features and priorities before each sprint (if required)
 - accepts / declines the results
 - release management
 - return on investment (ROI) management
- Common mistakes
 - often unavailable
 - has not enough “power” to manage the stakeholders (in the organisation)
 - has not enough technical knowledge
 - more than one Product Owner in a project (for smaller projects)



Scrum: Scrum Master

- Solves problems, responsible for the process (and team spirit)
 - *No staff responsibility!*
- Tasks
 - removes impediments, buffers conflicts
 - secures that the Team can work productively
 - supports the communication between all stakeholders
 - protects the Team from outside world
 - secures the commitment to Scrum principles and values
- **The ideal Scrum Master is a moderator, coach and experienced software developer**
- Signs for a good Scrum Master:
 - working progress is transparent and continuous
 - team is effective
 - working pace is constant (so called “velocity”)
 - team supports the Product Owner at analysis of new features
 - team reports on problems and impediments



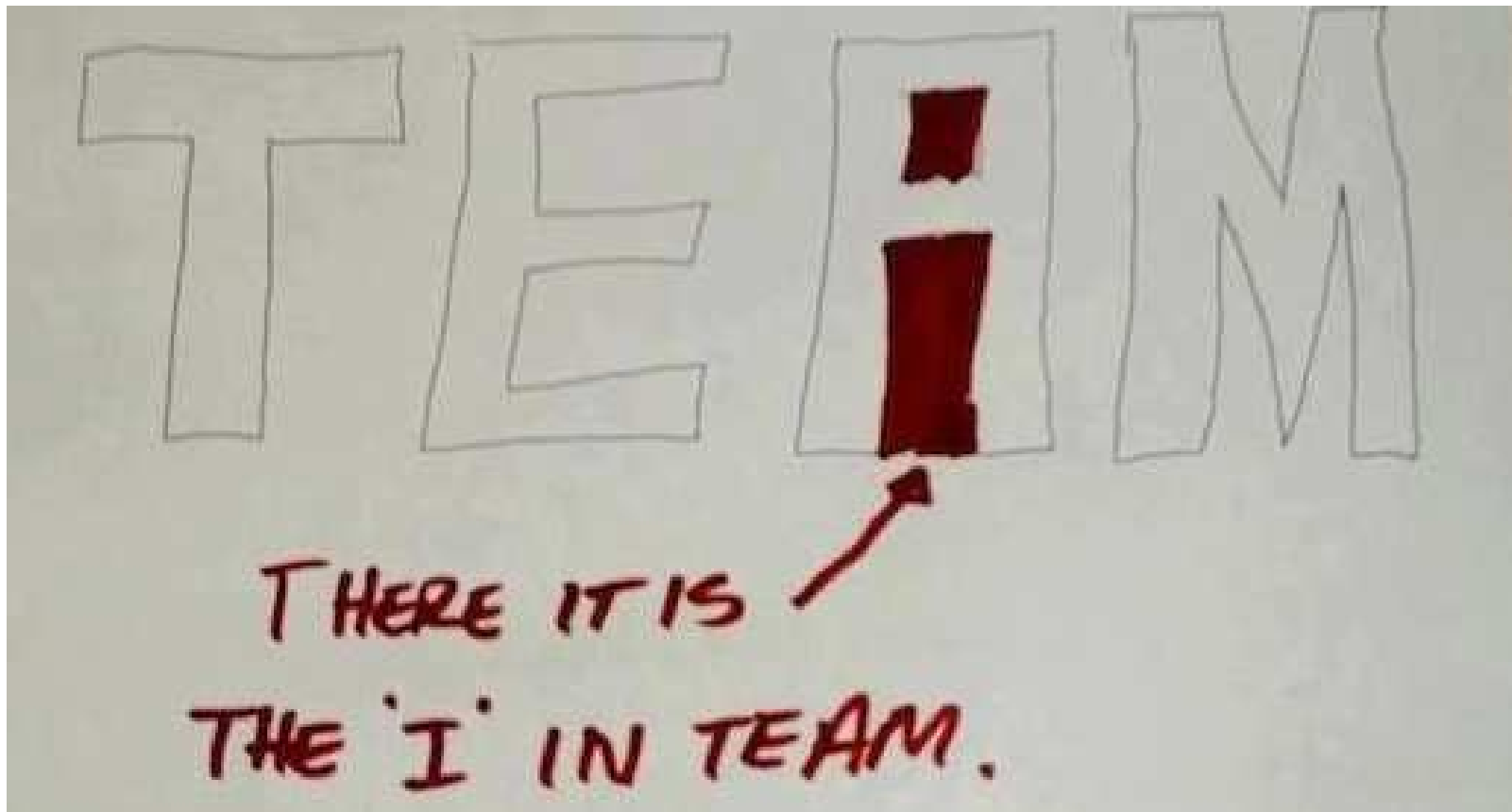
Scrum: The Team

- *Self organised, implements the product increment*
- Tasks:
 - estimates and commits itself to deliver the features
 - clarifies the sprint goal with the Product Owner and commits itself to it
 - supports the product owner in coarse-grained estimation of Product Backlog items
- Properties (an ideal case):
 - multifunctional, no pre-defined roles (← hard in praxis)
 - all needed knowledge shall be represented in team
 - GUI Designer, Developer, Tester ...
 - small (7 +/- people)
 - full time team members
 - self organised
 - working places near each other if possible
 - team members have common goal, respect and support each other



There is no “I” in Team

- Really? 😊



And Who is the Project Manager?

- His responsibilities are taken over from the three Scrum roles as follows

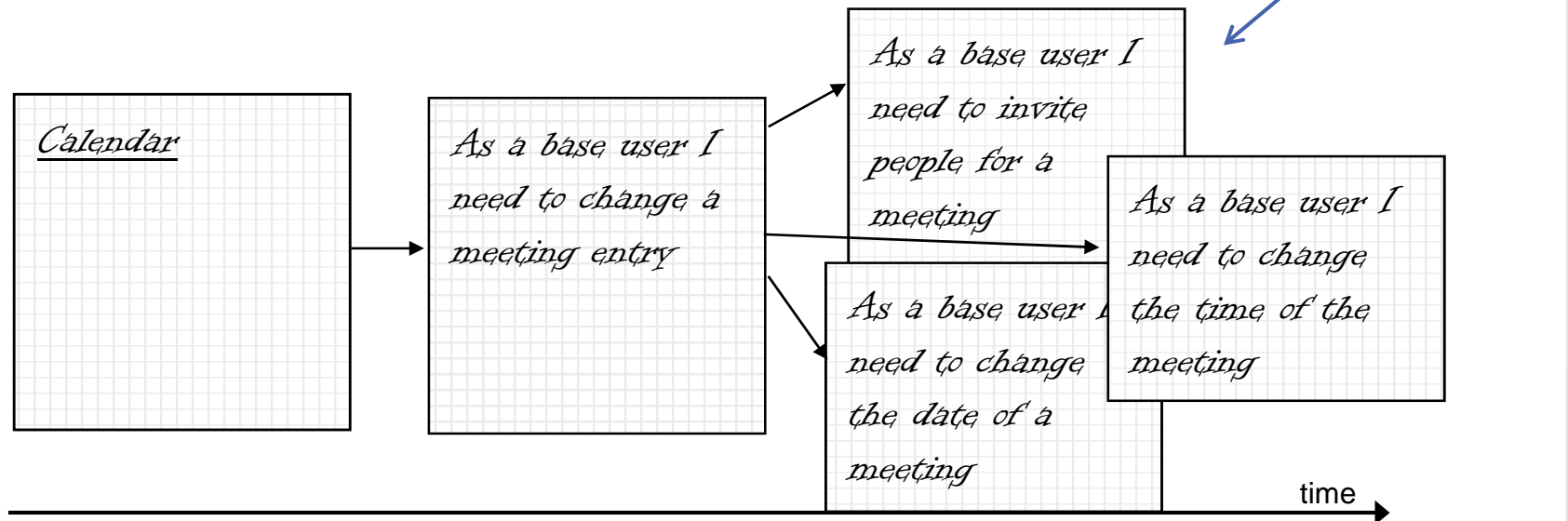


Responsibility	Scrum Role(s)
Scope Management	
Time Management	
Cost Management	
Communication Management	
Risk Management	
Quality Management	
Supplier Management	
Staff Management	

- A high-level collection of all features, etc. prioritized by business value
 - contains all known requirements and work results, which have to be implemented in order to achieve the project goal
 - these Product Backlog Items are usually expressed as user stories
 - *no activities → these go into the Sprint Backlog*
- A living document that can be changed any time
 - elicitation of new requirements and adaption of old ones
 - a completed version of Product Backlog is achieved at the end of the project
- Items in the Product Backlog –
 - are prioritised
 - according to business value, risk and architectural importance
 - and become estimated over time

Product Backlog Items

- Requirements are captured as **user stories**
- Initially, it is important **not to overload** the **product backlog**
 - but to have enough (fine-grained) requirements for about 2 or 3 sprints
 - it is important to get an overview of the **essential features**
 - required for operation and distribution of the system
- unclear requirements are only coarsely described
 - to be refined later



details will be discussed in the requirements lecture

- Scrum does not utilize a release or project plan in the traditional sense
 - experience tells us that it is **unrealistic** to plan for all contingencies
 - months or even years in advance



Investigate

Design

Implement

→ however, **estimates** of cost and duration of the project are **necessary**

- Scrum thus applies planning on 3 levels –
 - **release planning** which is a coarse forecast based on
 - the product backlog
 - the team's development velocity
 - **sprint planning**
 - **planning of a workday**

- A **coarse** release plan can be created after the product backlog has been populated
 - and a first release of meaningful functionality can be projected
 - mainly based on experience
- *BUT HOW can we come to more meaningful estimates?*
- Let's try something simple
 - What would you do in order to find out how long it takes to **paint a wall**?
 1. find out the size of the wall
 2. find out how much one person can paint within a given time period
 3. calculate how long it takes to paint the wall



- *How can this work in software development?*

- it requires **2 metrics**, namely –

→ *But how can these 2 metrics be measured?*

- *How can this work in software development?*
- it requires **2 metrics**, namely –
 - the **effort** (i.e. the “distance”) ***d*** required to work off the product backlog
 - in meters?
 - in LOC?
 - in function points?
 - in ... ?
 - the development speed (**velocity**) ***v*** of the team
 - i.e. the effort that can be implemented per sprint
- the **time** ***t*** required to implement the whole product backlog can then be calculated with the following equation
 - which is known from physics: **$t = d / v$** or **$v = d / t$**

→ *But how can these 2 metrics be measured?*

Estimating “Development Distance” (“d”)

- Estimating effort in software development is traditionally difficult
 - cf. Function Points and Boehm’s COCOMO (in a later lecture)
- Scrum approaches this challenge pragmatically
 - for sizing the requirements it uses so-called “**story points**”
 - team-specific and based on an interval scale
 - therefore, they are **not equivalent to person hours or days**

Value	Semantics
0	no effort
1	very small effort
2	small effort, equals a small effort plus a small effort
3	medium effort, equals a very small effort added to a small effort
5	large effort, equals a medium effort added to a small effort
8	very large effort, equals medium plus large effort
13	huge effort, equals larger plus very large effort

values are based on _____ series



Poker Cards



[it-zynergy.com]

Rules of the Game



- **Planning Poker** is “played” as follows –
 - each team member receives a deck of cards with each value of the Fibonacci series
 - 1. the product owner explains an item from the product backlog
 - 2. the team clarifies uncertainties with the product owner
 - and discusses the steps required to implement it → *input for sprint backlog*
 - 3. once enough understanding is reached for the estimate
 - each team member privately selects a card
 - representing his/her estimate
 - 4. after all have made their estimates the cards are shown
 - if all estimates match the item is estimated
 - 5. if no consensus has been reached, go back to step 2
- Furthermore –
 - abstention from voting should be the exception
 - do **not estimate buffers**
 - re-estimating in a later sprint is allowed

Velocity (“v”)



- The velocity for a sprint is the sum of all **approved efforts** divided by the time for this sprint
 - **counted fully or not at all**
 - i.e. even requirements finished to 95% are not counted, for example

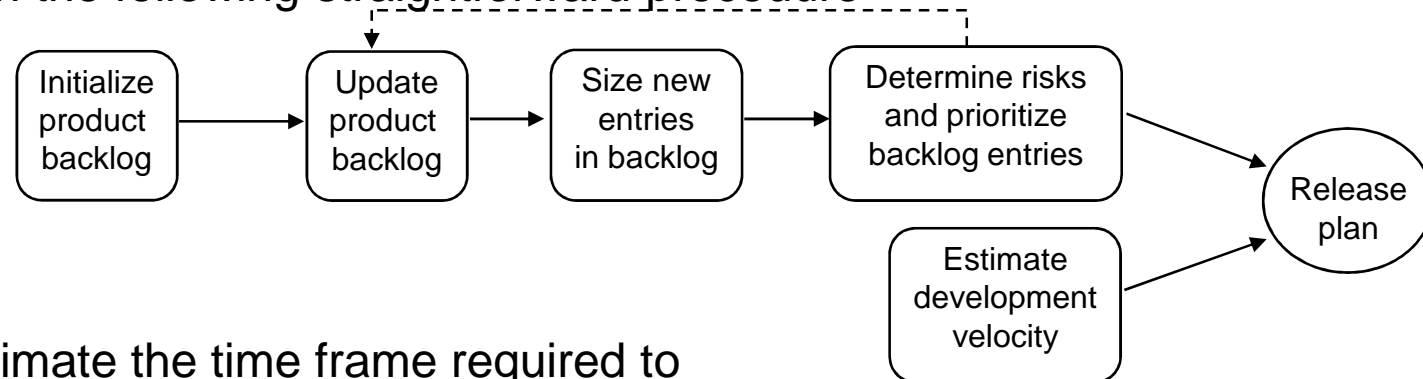
Selected requirements	Feature approved?	Estimated story points	Scored story points
Story A	yes	2	2
Story B	yes	2	2
Story C	yes	1	1
Story D	yes	3	3
Story E	yes	2	2
Story F	no	2	0
			$v = \frac{10 \text{ SPs}}{2 \text{ weeks}}$

- It is usual that the velocity fluctuates slightly
 - due to holidays, sicknesses, etc.
 - it is typically also slightly rising within the first few sprints

- Definition of Done: When is a user story really finished?
 - Please come up with criteria that can be used in order to determine whether something is finally done and briefly explain/discuss them
 - Check your solution against some Definition of Done that you google on the Web

Creating the Release Plan

- The product owner can create the release plan
 - once the previous activities have been completed
 - with the following straightforward procedure –



1. estimate the time frame required to
2. implement all requirements in the product backlog
 - ➔ this also allows to predict the cost to be expected
3. prioritize user stories
4. distribute the effort on particular sprints as evenly as possible
5. assign release dates for software increments

➔ A **honest revision** after each sprint is recommended

- “plan the work, work the plan”

Exemplary Sprint- and Release Plan

	Sprint	3		4		5	
begin/end date	Start- und Endtermin	07. März 2007	20. März 2007	21. März 2007	03. April 2007	04. April 2007	17. April 2007
goal	Sprint-Ziel	UI-Risiken adressieren		Thema A fertigstellen		Thema B fertigstellen	
backlog entries	Anforderungskatalogeinträge	17, 18, 19, 20, 21, 22, 23		25, 28, 30, 31, 32, 33, 34		26, 27, 29, 35	
velocity	Geschwindigkeit	18		20		10	
comments/assumptions	Kommentare und Annahmen			Maximale Geschwindigkeit		Osterferien	

	Sprint	6		7		8	
	Start- und Endtermin	18. April 2007	27. April 2007	02. Mai 2007	15. Mai 2007	16. Mai 2007	28. Mai 2007
	Sprint-Ziel	Thema C fertigstellen, Funktionalität A, B, C als Beta freigeben		Thema D		Thema E	
	Anforderungskatalogeinträge	TBD		TBD		TBD	
	Geschwindigkeit	20		20		18	
	Kommentare und Annahmen	Erste Freigabe der Software als Beta		Der erste Mai ist Feiertag		Pfingstferienbeginn am 29. Mai, Freigabe V1.0	

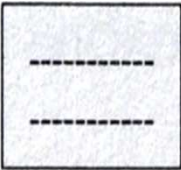


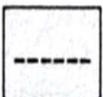
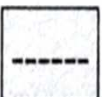
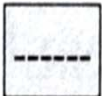


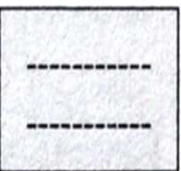
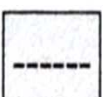


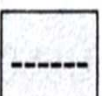
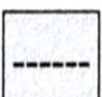
[Pichler]

The Sprint Backlog...

- ... contains all requirements (user stories) the team has committed to deal with in the sprint planning meeting
 - taken from the product backlog
 - **they need to be decomposed into developer tasks**
- ... is updated on a daily basis
- ... contains three categories
 - **to do**
 - **in progress**
 - **finished**
- potentially newly arising tasks are estimated and added to the sprint (or even the product) backlog
 - it is **not allowed** to add new requirements (user stories) in the middle of a sprint to the sprint backlog
 - those have to be added to the product backlog and assigned to a new sprint

Sprint Backlog Example

- ... as typically hanged on a pinboard

		Activities (Tasks)		
Prio	Requirements	To do	Ongoing	Finished
1		   	 	
2		   		
...

[Pichler]

Real Life Backlog Example

- A white board example [Boris Gloger, borisgloger.com]:



- common way of maintaining artefacts in Scrum
- only possible in non-distributed Projects
- knowledge might get completely lost after the end of the project
- otherwise consider tools: ScrumWorks, Agilefant, Excel, etc.

Filling the Sprint Backlog

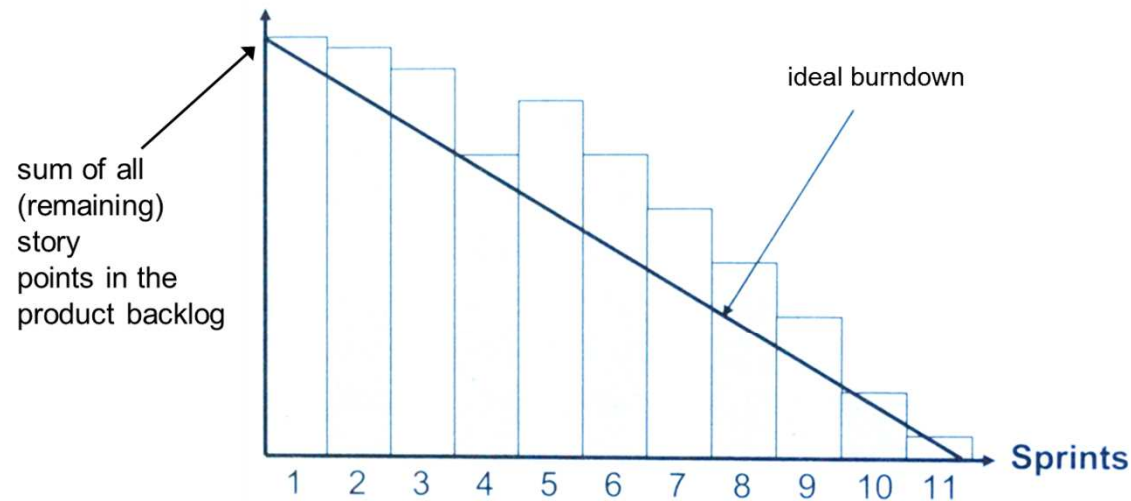
- How can we fill the sprint backlog during **sprint planning**?
 - ➔ take the “next” backlog entry (user story) according to the product backlog or the story map
- Prerequisite is a common understanding of that entry
 - then the team has to identify all activities required to implement it
 - usually –
 - requirements analysis, design, testing, documentation
 - but other activities may also be possible such as –
 - reviewing, integration, prototyping, refactoring etc.
- Each activity must not take longer than roughly 16 hours
 - thus they are well assessable
 - and estimatable in **ideal personhours**
 - ideally in 2 or 4 hour increments only
 - **do not include buffers**
 - **timebox explorative and prototyping activities**

How much is enough?

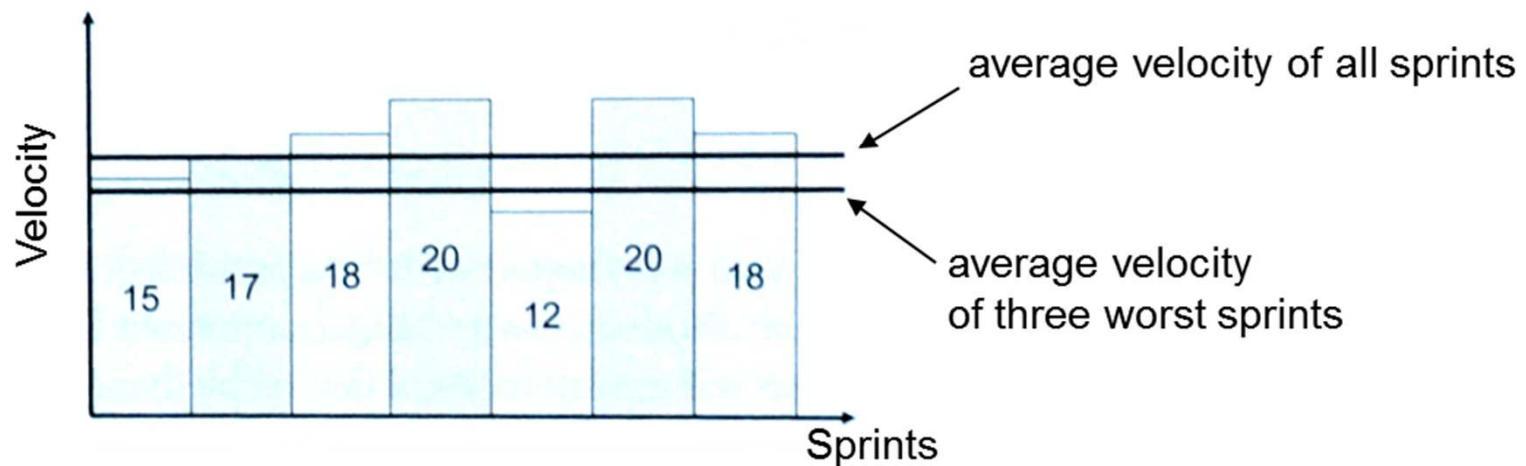
- Only add activities as long as **85% of the net capacity** of the team is not reached
 - determine the total presence of the team members for the iteration
 - minus holidays, professional education etc.
 - reduce it by about 25–35% for the “usual” distractions
 - phone calls, emails etc.
 - try to calibrate this for your organization
- For example
 - we have a team of 5 developers available for 18 days each
 - is it possible to add another requirement with 72 personhours of effort given that the team has already an estimated load of 400 personhours to deal with?



Illustrating Velocity



How might a Sprint Burndown look like?



[Pichler]

Scrum Framework: Critical Evaluation?

- People/Roles
- Artefacts
- Documentation
- Activities
- Scalability
- Architecture
- Quality-critical software



Tips for large Projects

- Projects with more than one team require special measures –
 - observe **Brook's Law**
 - adding new people reduces the velocity in the first place
 - **start small** and *grow organically*
 - one team with talented people acquires the project's foundation in a few sprints
 - this team is split at the end of a sprint and new people are added
 - prefer to add one team at a time (every 2-3 sprints)
 - a more rapid growth is probably feasible, but is **riskier as well**
 - **minimize dependencies** between teams
 - feature over component teams
 - try to have Area Product Owners
 - meet for a daily **Scrum of Scrums**
 - which is a daily scrum with representatives from each team

→ *Interesting Video with Craig Larman:* http://www.youtube.com/watch?v=HmdGvq_8rVQ

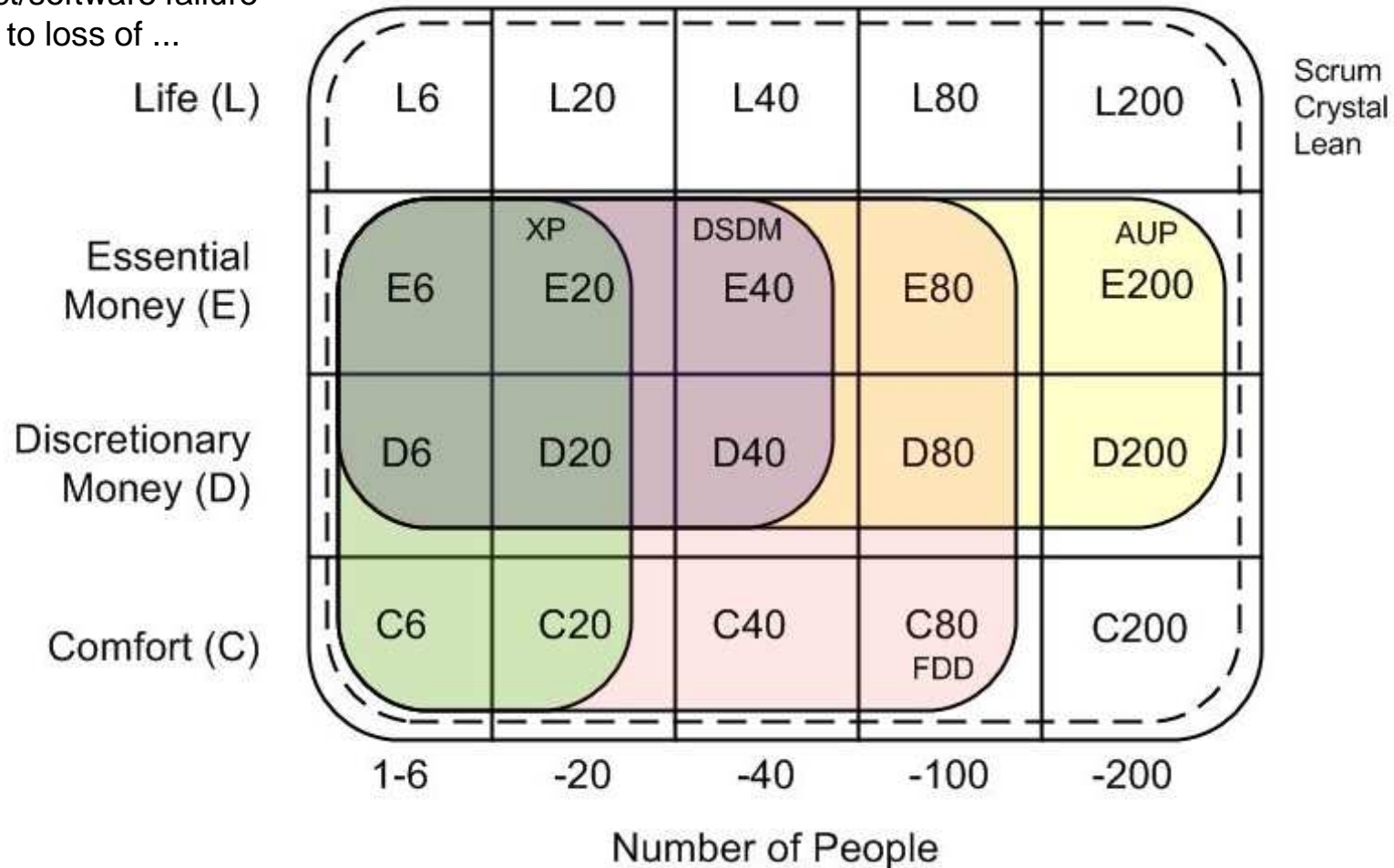
Tips for distributed projects

- A project is considered being distributed as soon as team members work at different locations
 - and **cannot attend one common Daily Scrum** easily
- Some important hints for these projects are –
 - there is nothing like personal communication
 - can't you avoid the distribution?
 - **never separate** the ScrumMaster and the team
 - try to have a product owner with the team as well
 - **distribute teams stepwise** (analogous to the growth in large projects)
 - have a core team that works together for a few sprints
 - its members become the nuclei for two distributed teams
 - **exchange delegates** between the teams from time to time
 - in order to facilitate collaboration
 - and allow the people to get acquainted personally

Comparison of Agile Methods

- ... on the “Cockburn Scale” [<http://www.devx.com/architect/Article/32836/0/page/4>]

project/software failure
leads to loss of ...

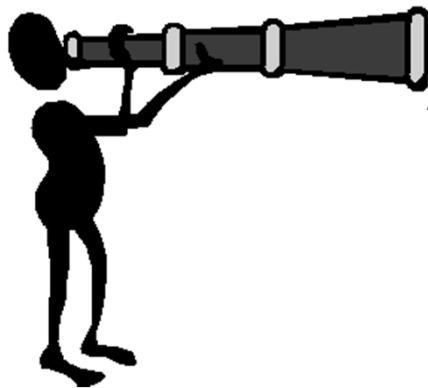


No Silver Bullet!

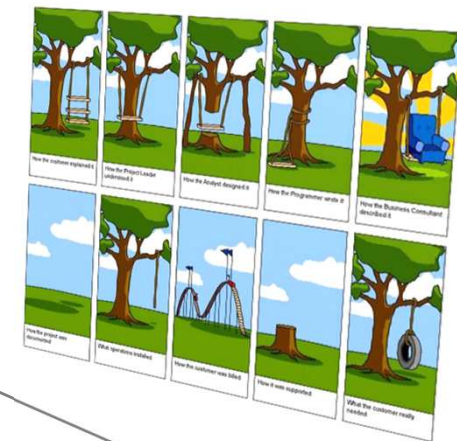
- Although Scrum works in the spirit of lean management and development...
 - ... it is **no silver bullet** that improves your development practices by itself, as it is –
 - hard work
 - it encourages the creativity and self-reliance of staff members
 - by providing few guidelines “Scrum is so trivial” (Ken Schwaber)
 - an empirical process model
 - helping to deal with the unexpected
 - but it requires a continuous learning based on “inspect and adapt”
- Especially in the initial adoption it is often difficult to implement and to “live” Scrum
 - as participants need to learn new rules and behaviours
 - thus, unfortunately, Scrum is often changed inappropriately
 - and not the own habits
 - in other words, traditional behaviours are conserved (*cf. the Dilbert strip*)

Conclusion

- Agile methods are **certainly helpful** in various contexts
 - through their highly flexible approach to SW development
 - that allows early risk recognition through customer collaboration and early feedback
- Nevertheless, they certainly have their **limitations**
 - such as their weakness in architecture & design
 - that should be kept in mind while using them

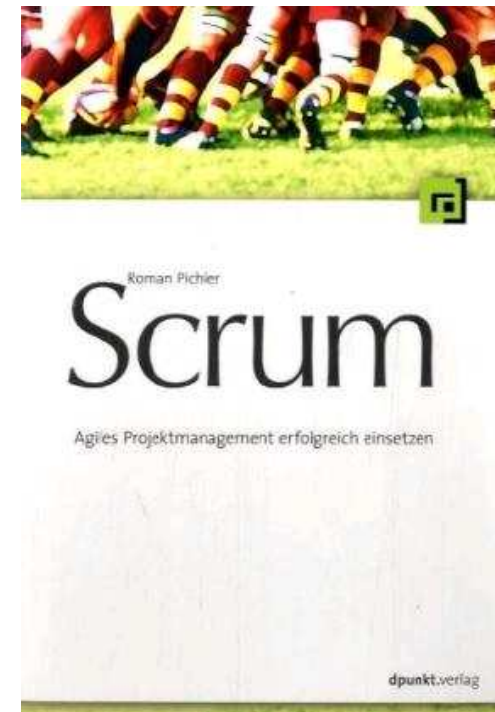


*Requirements capture
and analysis*



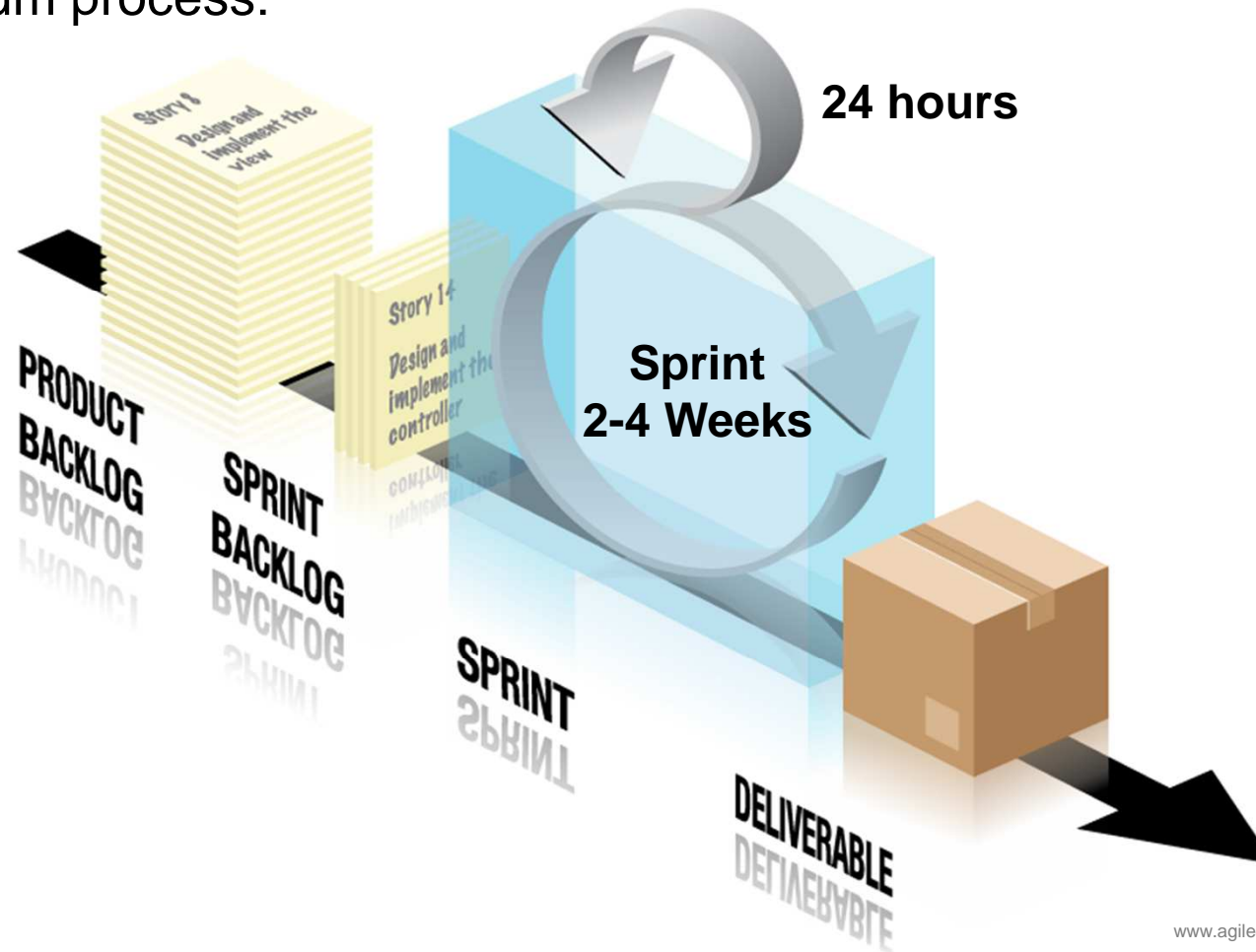
- Thank you for your attention!

- Beck, *Extreme Programming Explained*, 2nd edition, Addison Wesley, 2005
- Cohn, *User Stories Applied: For Agile Software Development*, Addison-Wesley, 2004
- [Pichler], *Scrum - Agiles Projektmanagement erfolgreich einsetzen*, dpunkt.verlag, 2007
 - also see http://www.romanpichler.com/articles/pdfs/pichler_OS_01_05.pdf
- [AgileManifesto] Agile Manifest, <http://agilemanifesto.org/>
- [ScrumCenter] ScrumCenter, 2010, Roberts S., Mathis C.
- Sample chapter on Scrum Project Planning:
 - http://www.springer.com/cda/content/document/cda_downloadaddocument/9783827427519-c1.pdf



Appendix: Yet Another Scrum Picture

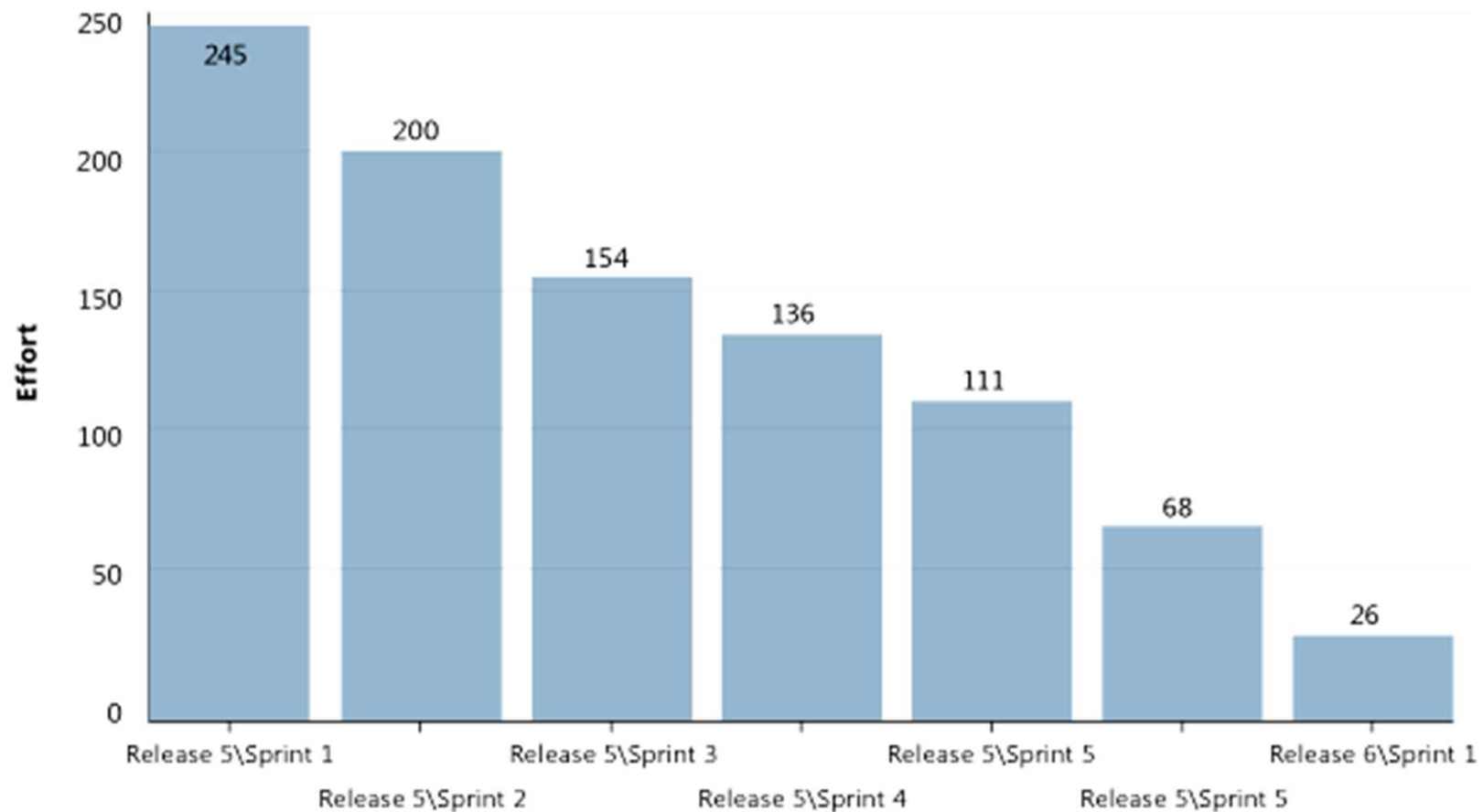
■ The Scrum process:



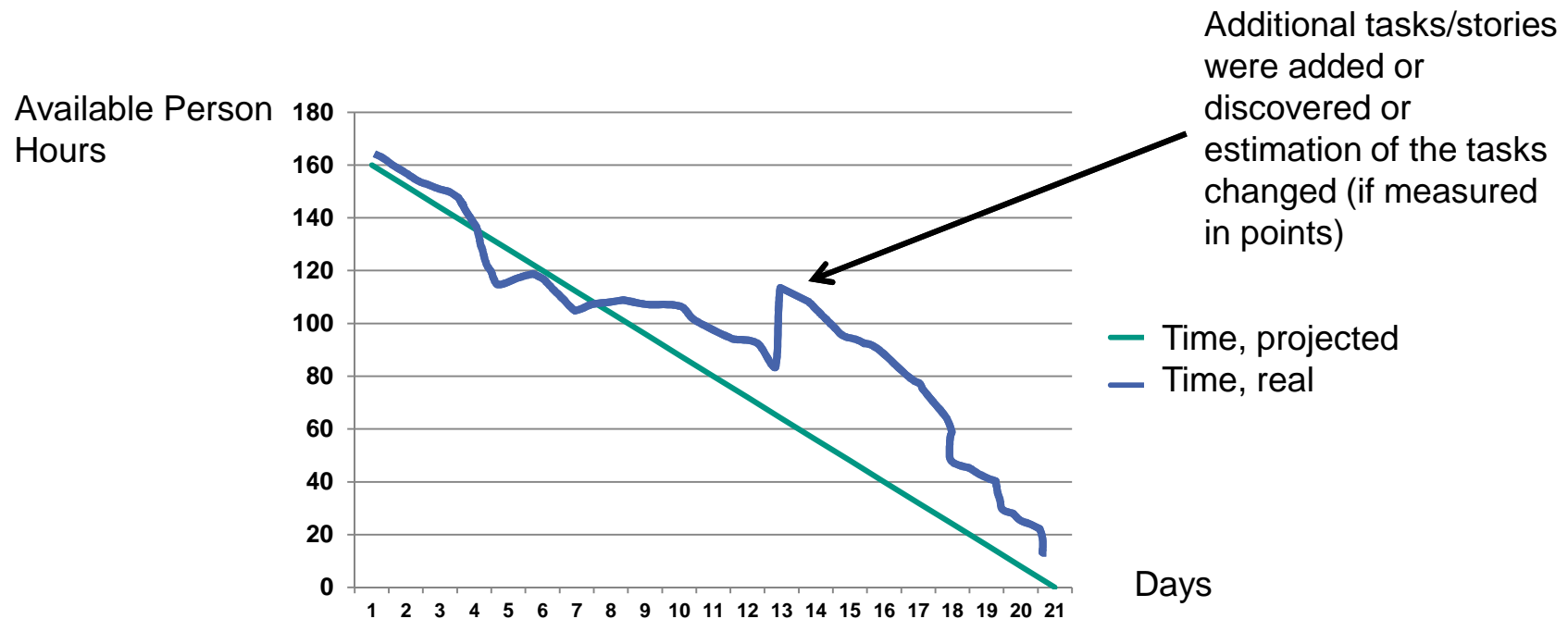
www.agile42.com

Appendix Burndown Chart

Another example of a Product (Release) Burndown Chart [msdn.microsoft.com]:



Appendix: Sprint Burndown Example



Appendix: Sample Exam Question I

Agile Methoden im Allgemeinen (3P)	Richtig	Falsch
Agile Methoden bieten zielgerichtetes Projektmanagement.		
Agile Methoden besitzen vergleichsweise wenig Management-Overhead mit erfahrenen Teams.		
Testgetriebene Entwicklung führt nachgewiesenermaßen zu deutlich besserer Software-Qualität.		
Agile Methoden bieten frühere Risiko-Erkennung durch Kollaboration mit dem Kunden und frühere Entwicklung.		
Die Vertragsverhandlungen spielen laut dem Agilen Manifest eine wichtigere Rolle als eine Kollaboration mit dem Kunden.		
Neue Anforderungen sind in agilen Prozessen jederzeit willkommen und werden sofort in dem derzeitigen Sprint implementiert.		

Appendix: Sample Exam Question II

Extreme Programming (XP) (2P)	Richtig	Falsch
XP ist passend für eine schnell wechselnde Umgebung mit einer Vielzahl neuer Anforderungen.		
XP ist passend für ein kleines Entwicklungs-Team.		
XP ist ein nicht replizierbarer Ad-hoc-Prozess.		
Die Mehrheit der XP-Praktiken ist ausführlich validiert.		

Scrum (3P)	Richtig	Falsch
XP ist eher als ein Vorgehensmodell zu betrachten, Scrum dagegen als eine Praktik.		
Bei Scrum besteht eine starke Abhängigkeit vom Verantwortungsbewusstsein einzelner Team-Mitglieder.		
Scrum skaliert gut auch für sehr große Projekte mit Teamgrößen > 1000 Entwickler.		
Scrum sollte als generelles Vorgehensmodell betrachtet werden, das für unterschiedliche Projekte und Domänen zugeschnitten werden muss.		
Ein Sprint-Backlog enthält die Beschreibung aller im Projekt zu implementierenden Features, die je nach ihrem Geschäftswert priorisiert (sortiert) sind.		
Während des Sprint-Planning-Meetings teilt der Scrum-Master die Aufgaben den Team-Mitgliedern zu.		