Karlsruhe Institute of Technology

Software Engineering II

Prof. Dr. Raffaela Mirandola

Topic 12
Agile Development

SASIS – SELF-ADAPTIVE SOFTWARE-INTENSIVE SYSTEMS
KASTEL – INSTITUTE OF INFORMATION SECURITY AND DEPENDABILITY

sasis.kastel.kit.edu

Overview on the Agile Lecture

Content

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

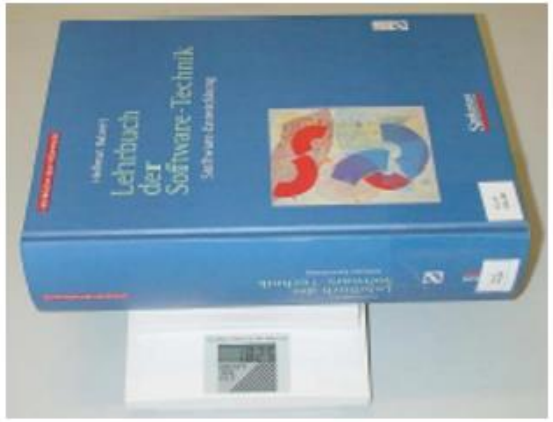
Learning Goals

- 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 choose and apply helpful techniques for a given project context

Motivation

- Environment
  - Complicated software systems
  - High market competition
  - Time and budget pressure
  - …
- Problems
  - Too “heavyweight” processes
  - Not able to reflect changes
  - Too many artifacts
  - → Inflexible, high risk
- → Solutions
  - Extreme Programming (XP)
  - Scrum
  - Crystal
  - Feature-Driven Development …

Traditional Methods
-> Heavyweight: 1825 g



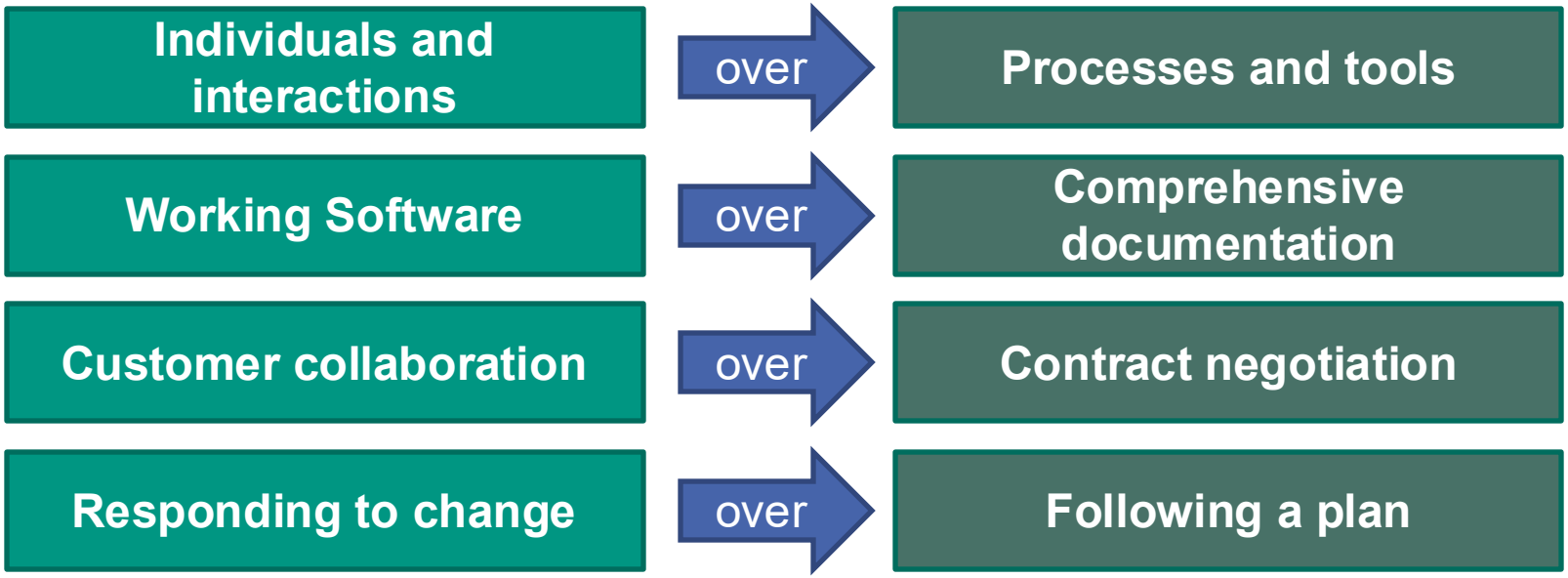
Agile Methods
-> Lightweight: 346 g

Agile Manifesto

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”

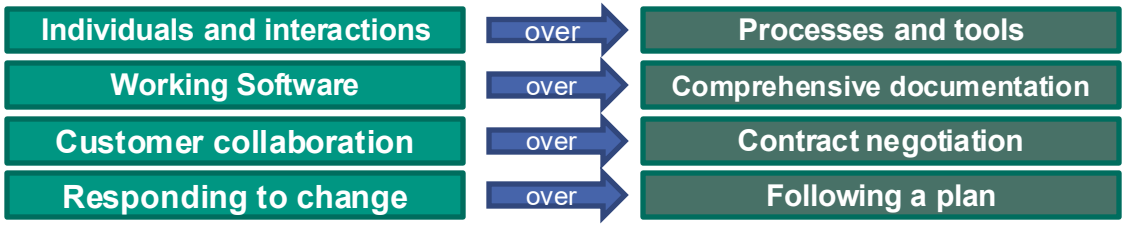


- These ideas may also be used in “non agile” methods!
- Is this really so different? Who was willing to neglect left hand side?

[AgileManifesto.org]

Manifest critically reviewed

- Was the world so bad before?
- Are really the opposites the real-world experience?
  Or just a good enemy image?



However, two important insights:

1. Accept that you cannot plan ahead, dues to changing requirements, unforeseen complexity
2. You need to monitor continously,
   - that you are creating what the customer needs
   - that product quality is satisfying (earlier fixes are cheaper than later ones)

[AgileManifesto.org]

Exercise: Agile vs. „Before“ Agile

- *Create conceptually opposite statements:*
  Which observations lead to the following statements?

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.

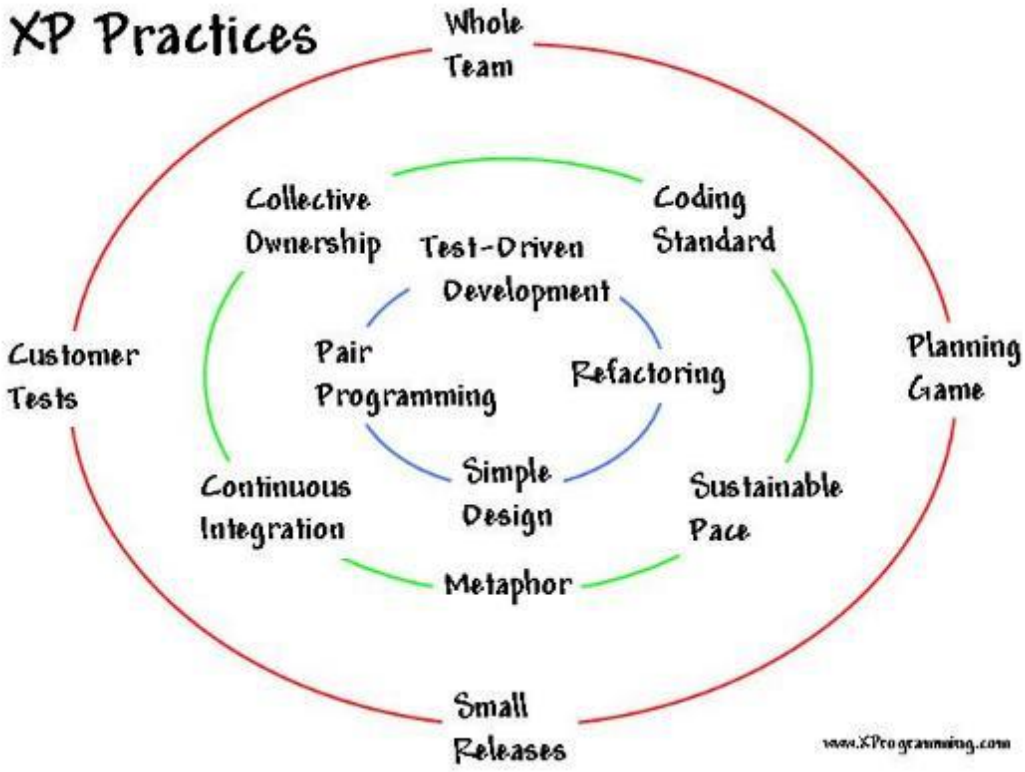
[AgileManifesto.org]

Extreme Programming (XP)

- XP = a set of ***values, principles and practices***
  - practices partially existed before XP, but were consolidated by it

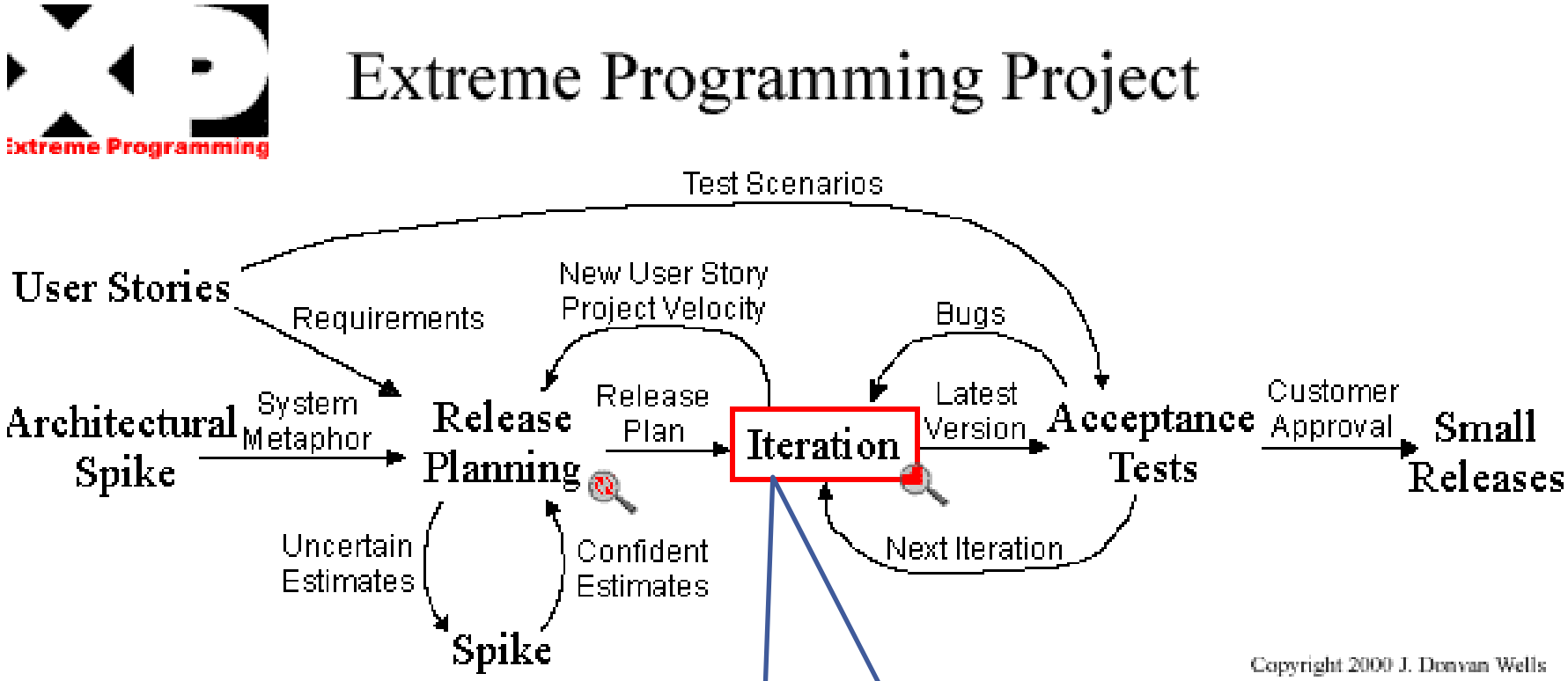
- Main XP values
  - Communication
  - Simplicity
  - Feedback
  - Courage

- Some XP Principles
  - quick delivery
  - rapid feedback
  - keep it simple
  - incremental change
  - embrace change



project-wide practices
team-specific practices
individual (developer-specific) practices

XP Process



May include modelling, such as using Class Responsibility Collaboration (CRC) cards

- Just enough design
- Write tests first
- Implementation

[extremeprogramming.org]

Common XP (Agile) Criticisms

- An ad-hoc process that cannot be replicated
  - ➔ not usable for large projects
- Missing documentation, massing cross-project reuse
- 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]
see also [Beck 2005], Chapter 17

Scrum

- 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 (i.e. object model)***

Scrum at a Glance

Scrum Master

- three roles
- four meetings
- three artefacts

Product Owner

Daily Scrum

Team

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

from [ScrumGuide]

- ***Sprint**:* a time-box of one month or less
  - during which a “Done”, useable, and potentially releasable product increment is created.
- Sprints best have consistent durations
- A new Sprint starts immediately after the conclusion of the previous Sprint.
- Sprints contain and consist of the Sprint Planning, Daily Scrums, the development work, the Sprint Review, and the Sprint Retrospective.
- During the Sprint:
  - No changes are made that would endanger the Sprint Goal;
  - Quality goals do not decrease; and,
  - Scope may be clarified and re-negotiated between the Product Owner and Development Team as more is learned.
- The development process is a sequence 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
  - **Scrum Master**
    - 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 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

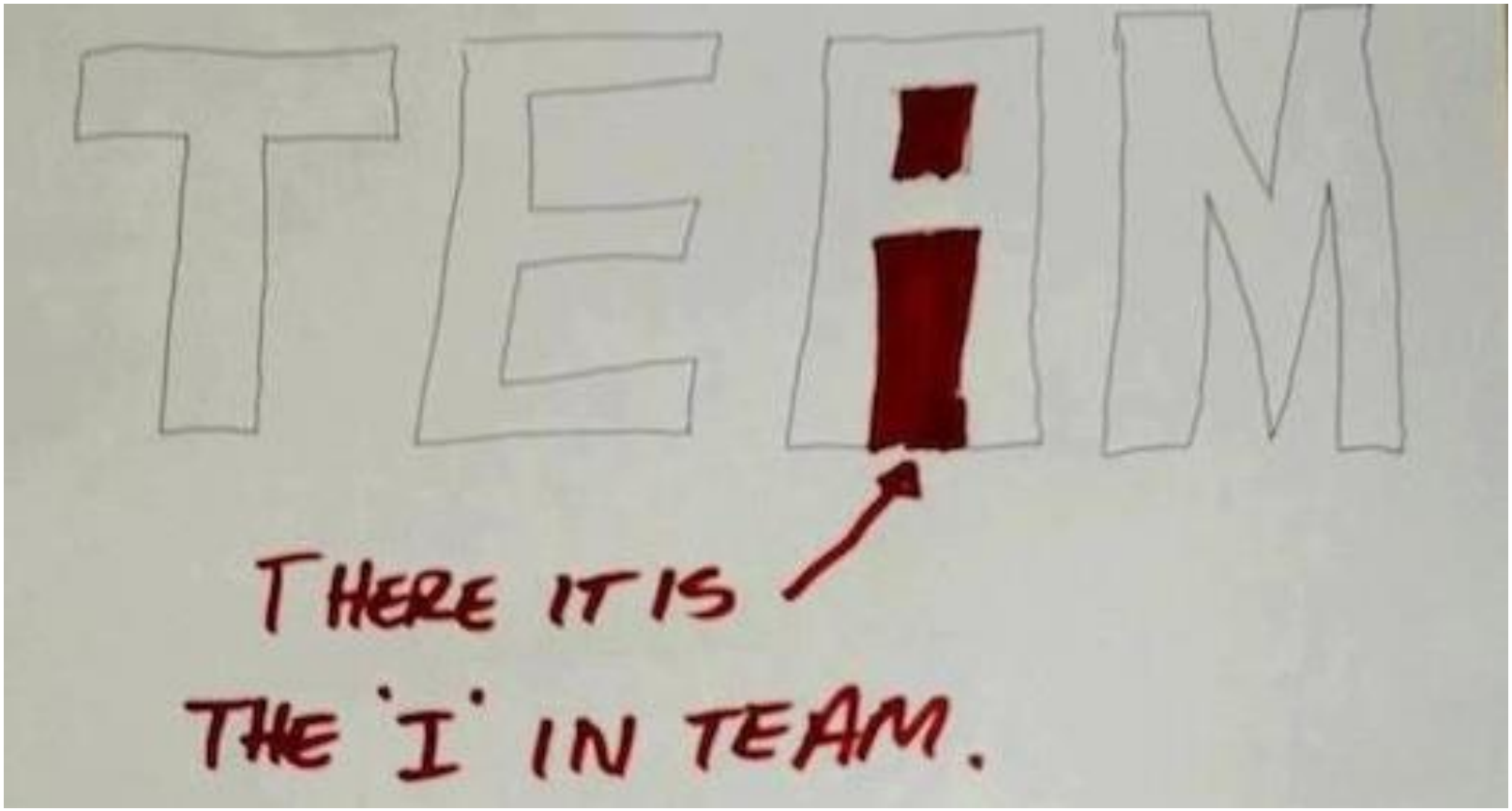
- Solves problems, responsible for the process (and team spirit)
  - *No staff responsibility!*
- Tasks
  - removes impediments, buffers conflicts
  - secures that the Team can work productively
  - supports the communication between all stakeholders
  - protects the Team from outside world
  - secures the commitment to Scrum principles and values
- **The ideal Scrum Master is a moderator, coach and experienced software developer**
- Signs for a good Scrum Muster:
  - 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 practice)
  - 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? ☺



Scrum: Product Backlog

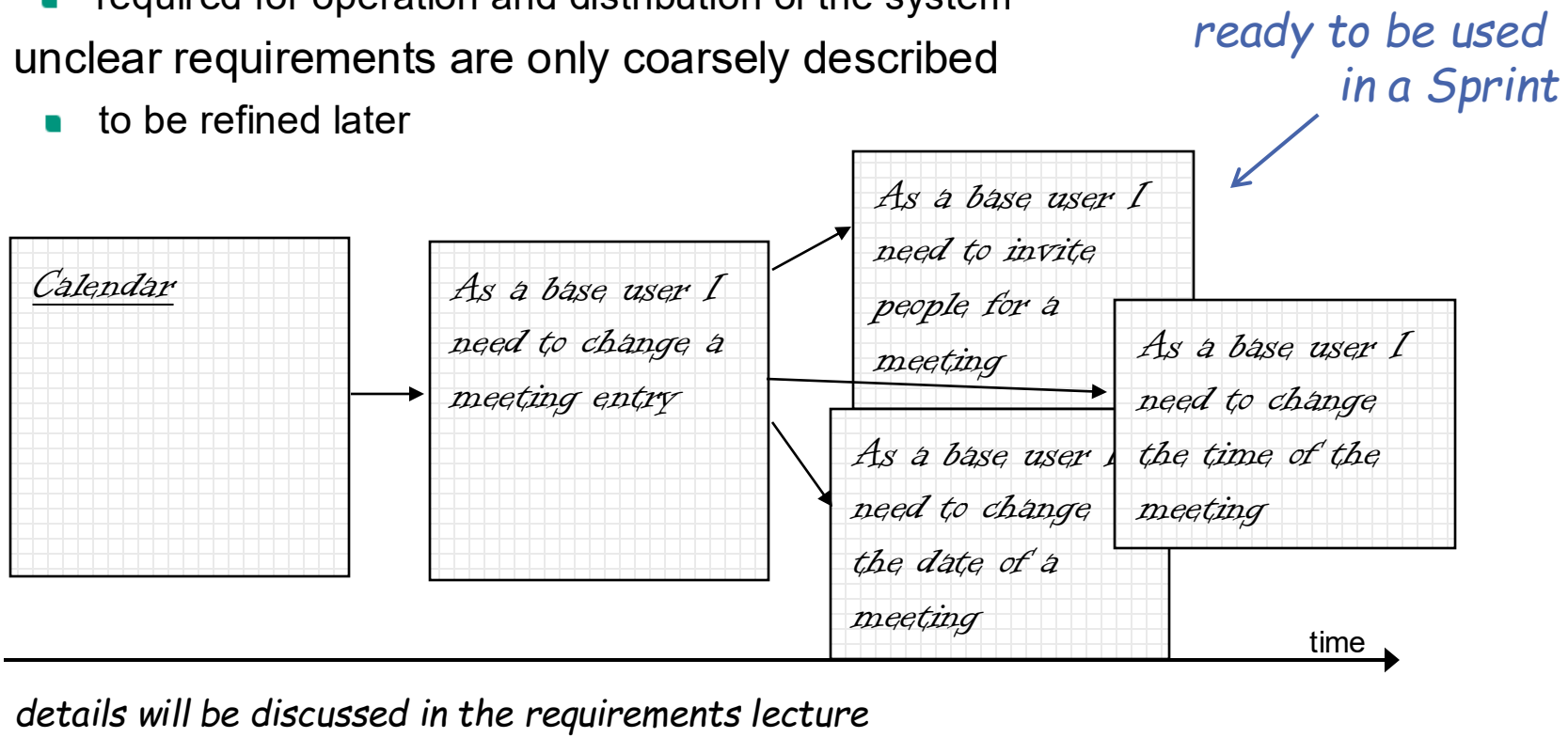
- A high-level collection of all features, etc. prioritized by business value
  - contains all known requirements and work results, which have to be implemented in order to achieve the project goal
    - these Product Backlog Items are usually expressed as user stories
  - *no activities (activities belong to 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
  - are annotated with *story points* for expressing the expected effort

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

Project Planning

- 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

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

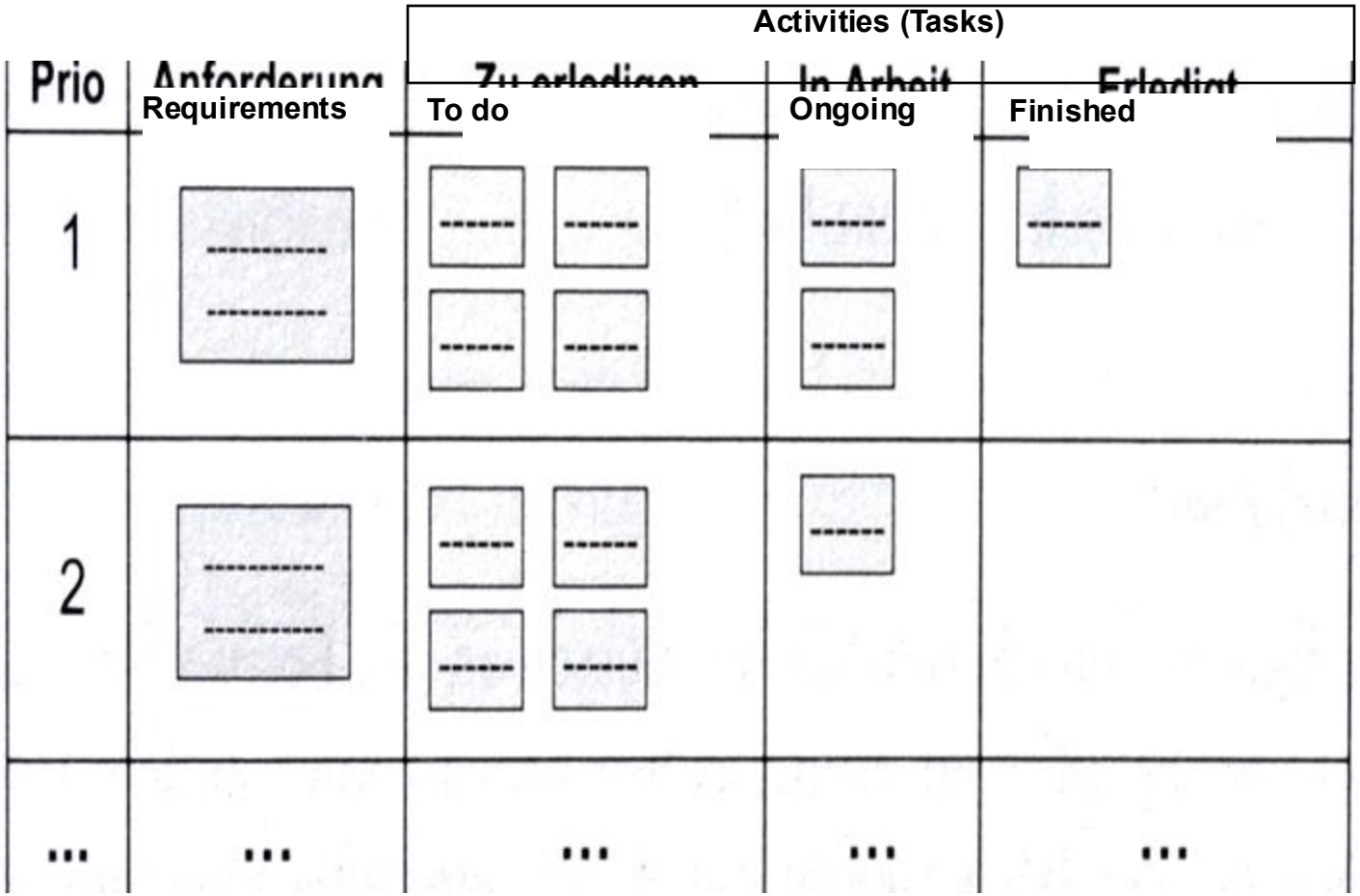
- Scrum thus applies planning on 3 levels –
  - **release planning** which is a coarse forecast based on
    - the product backlog
    - the team's development velocity
  - **sprint planning**
  - **planning of a workday**

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



[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
  - should be well assessable
  - effort should be estimated in **ideal personhours**
    - ideally in 2 or 4 hour increments only
    - do not include buffers
    - time-box 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 person-hours of effort, if the team already has an estimated load of 400 person-hours to deal with?

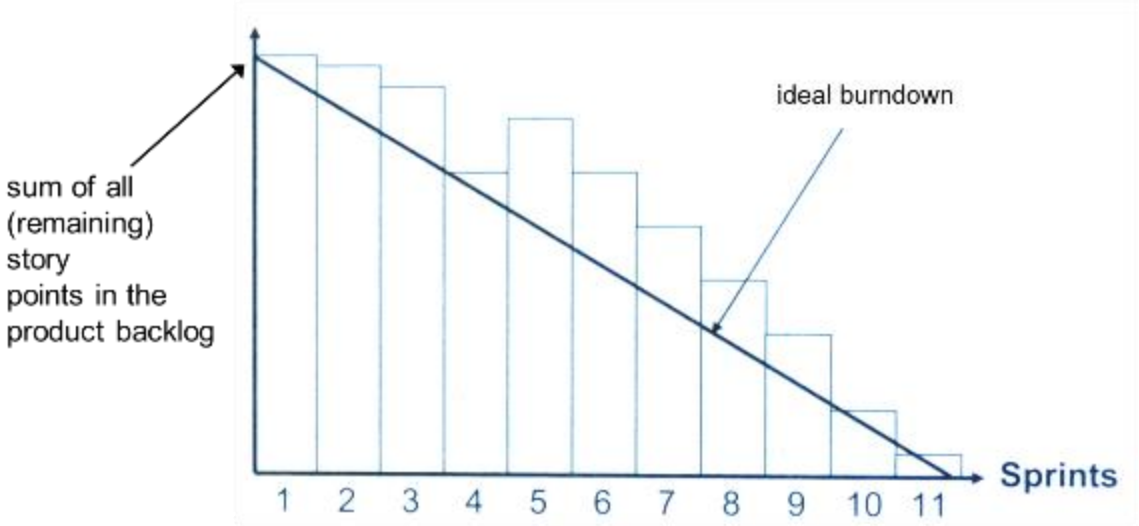
How much is enough?

- 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
  - reduce it by about 25–35% for the “usual” distractions

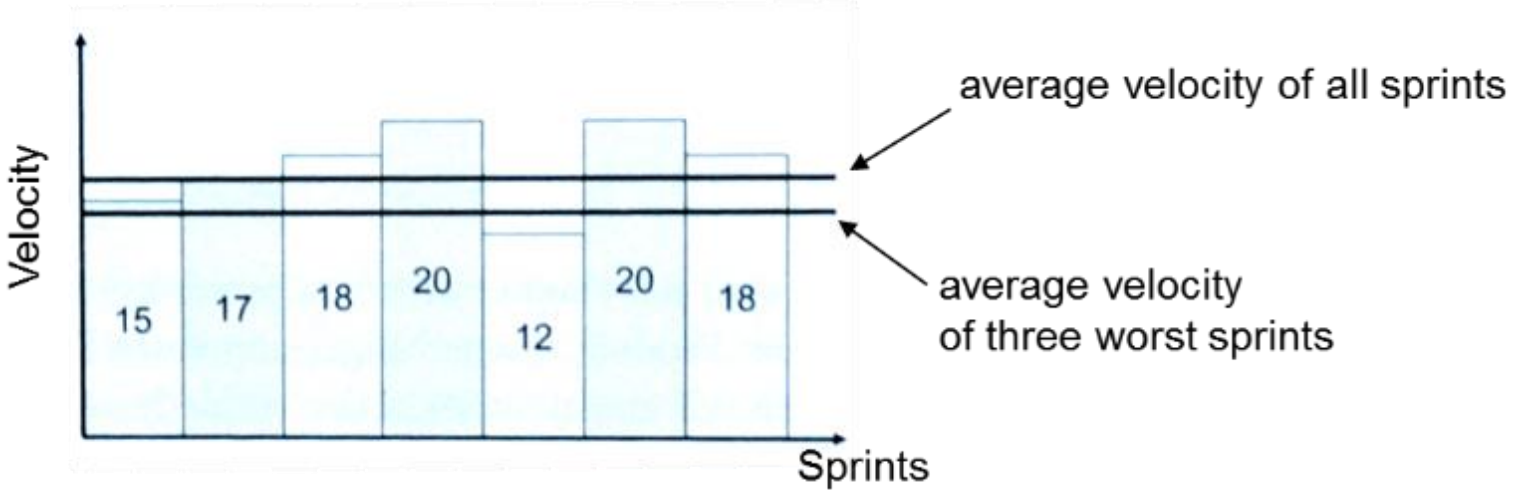
- For example
  - we have a team of 5 developers available for 18 days each
  - Is it possible to add another requirement with 72 person-hours of effort, if the team already has an estimated load of 400 person-hours to deal with?

  - Full capacity: 5*18*8 = 720 ph
  - Net capacity: 720 ph * 2/3 (33%) = 480 ph
  - 480 ph * 0,85 (85%) = 408 ph

Illustrating Velocity



How might a bad Sprint Burndown look like?



[Pichler]

Scrum Framework: Critical Evaluation?

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

Hints 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

How to scale agile approaches is a hot topic!
→ Enterprise Agility, Scaled Professional Scrum, Scrum of Scrums

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

Hints for d • i • s • tr • ib • u • ted 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 Scrum Master 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

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 and needs discipline (no ad-hoc!)
    - 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, even mentality
  - thus, unfortunately, Scrum is often changed inappropriately
    - and not the own habits
    - in other words, traditional behaviours are conserved *(cf. the Dilbert strip)*

General Reflection on Agile Methods

Good:

- Agile projects do not assume a stable world.
- Fast feedback on customer satisfaction and product quality essential
- Disciplined Process (certainly not ad hoc!)
- Shifts responsibility to each team member.

Problematic:

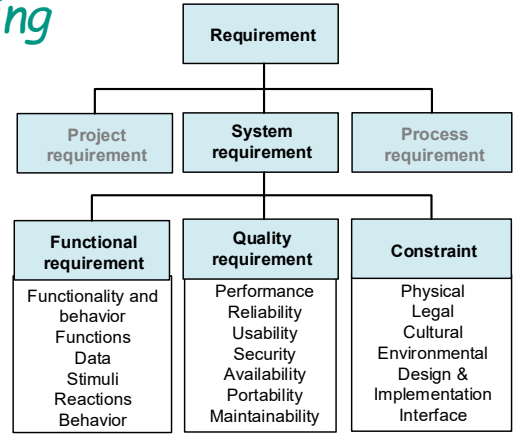
- Unclear how to include architectural design in agile projects.
- Unclear how to scale to larger projects, unclear how to do planned reuse.
- “Keep the architecture flexible” is not the complete answer, as some design decisions are by nature hard to revert.
- Shifts responsibility to each team member (not everyone wants to live with this pressure)
- Some dogmatism: relevance of scientific empirical evidence sometimes neglected: e.g., pair programming helps to improve quality early. However, they are less cost-effective than reviews (which help likewise in quality).
- Agile methods are a good manufacturing approach. But can we become an engineering discipline?

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 on functionality and quality.

- Nevertheless, they certainly have their limitations
  - such as their weakness in architecture & design
  - that should be kept in mind while using them

Requirements Engineering



Literature

- 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

Online Resources

- [AgileManifesto] Agile Manifest, http://agilemanifesto.org/
- [XP] Wells, *Extreme Programming: A gentle introduction* http://www.extremeprogramming.org/
- [ScrumCenter] ScrumCenter.com, 2010, Roberts and Mathis
- [ScrumGuide] Schwaber and Sutherland, *The Scrum Guide* http://www.scrumguides.org/

- Hummel, Sample chapter on Scrum Project Planning:
  - http://www.springer.com/cda/content/document/cda_downloaddocument/9783827427519-c1.pdf

APPENDIX

Release Planning I

- 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

Release Planning II

- *How can this work in software development?*
  - it requires **2 metrics**, namely –

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

Release Planning II

- *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

$$v = \frac{10 \text{ SPs}}{2 \text{ weeks}}$$

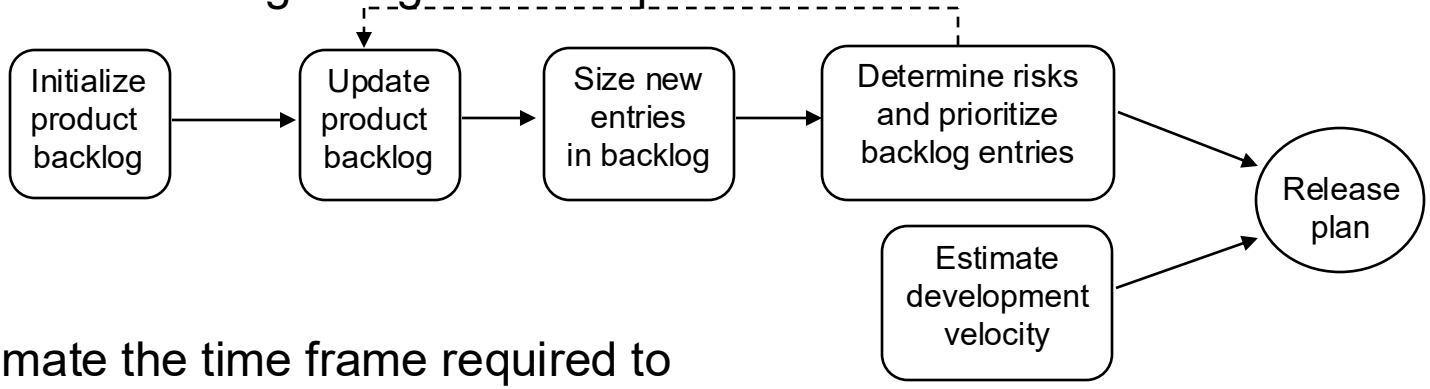
- It is usual that the velocity fluctuates slightly
  - due to holidays, sicknesses, etc.
  - it is typically also slightly rising within the first few sprints

Further Reading

- 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 find 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 implement all requirements in the product backlog
   - ➔ this also allows to predict the cost to be expected
2. prioritize user stories
3. distribute the effort on particular sprints as evenly as possible
4. 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

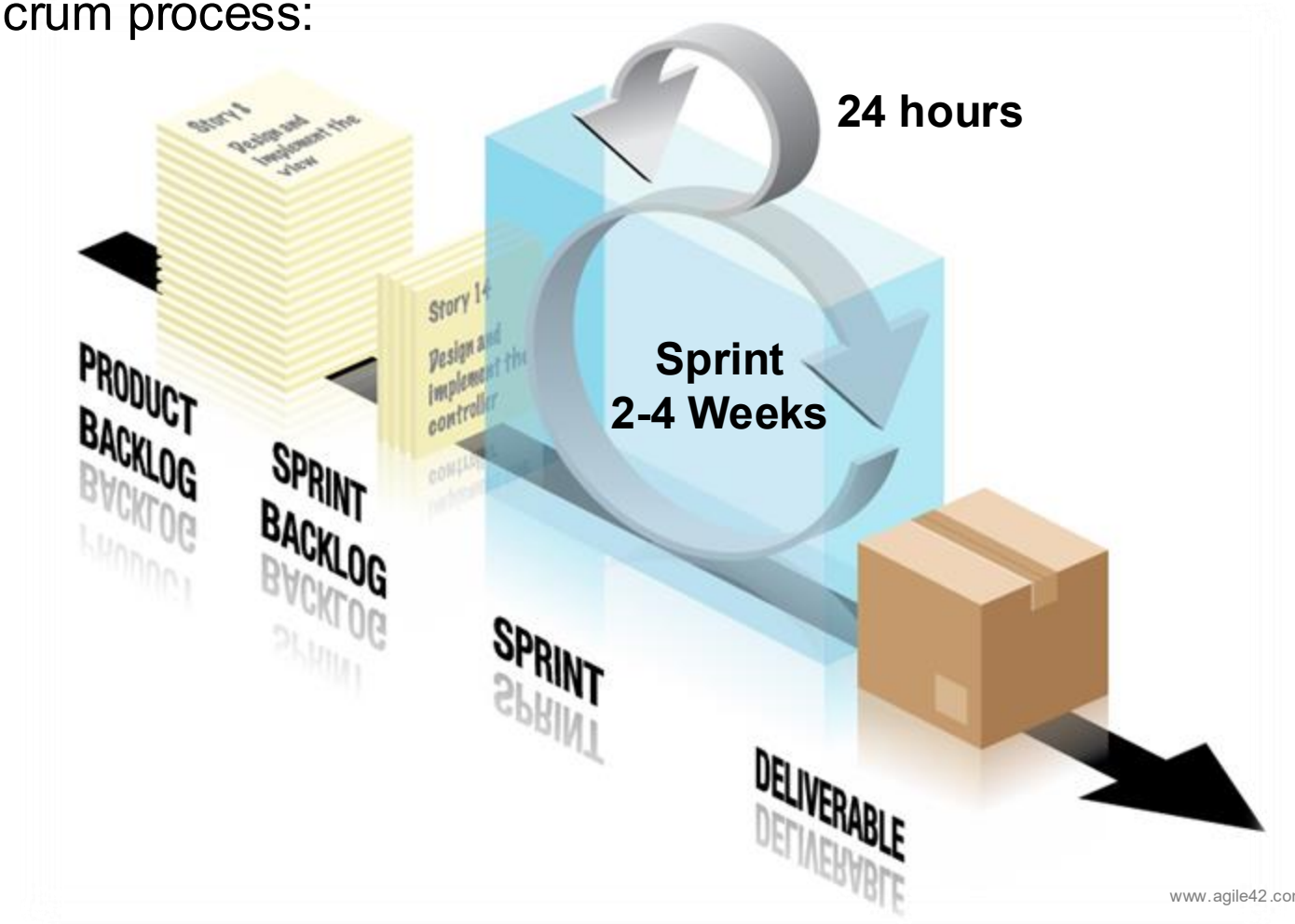
Sprint	3		4		5	
Start and end date	March 7, 2007	March 20, 2007	March 21, 2007	April 3, 2007	April 4, 2007	April 17, 2007
Sprint goal	Address UI risks		Complete Topic A		Complete Topic B	
Requirements catalogue entries (Backlog Entries)	17, 18, 19, 20, 21, 22, 23		25, 28, 30, 31, 32, 33, 34		26, 27, 29, 35	
Velocity	18		20		10	
Comments and assumptions			Maximum velocity		Easter holidays	

Sprint	6		7		8	
Start and end date	April 18, 2007	April 27, 2007	May 2, 2007	May 15, 2007	May16, 2007	May 28, 2007
Sprint goal	Complete Topic C, release functionality A, B, C as beta		Topic D		Topic E	
Requirements catalogue entries (Backlog Entries)	TBD		TBD		TBD	
Velocity	20		20		18	
Comments and assumptions	First release of the software as beta		The first day is a holiday.		Whitsun holidays begin on the 29th May, release V1.0	

[Pichler]

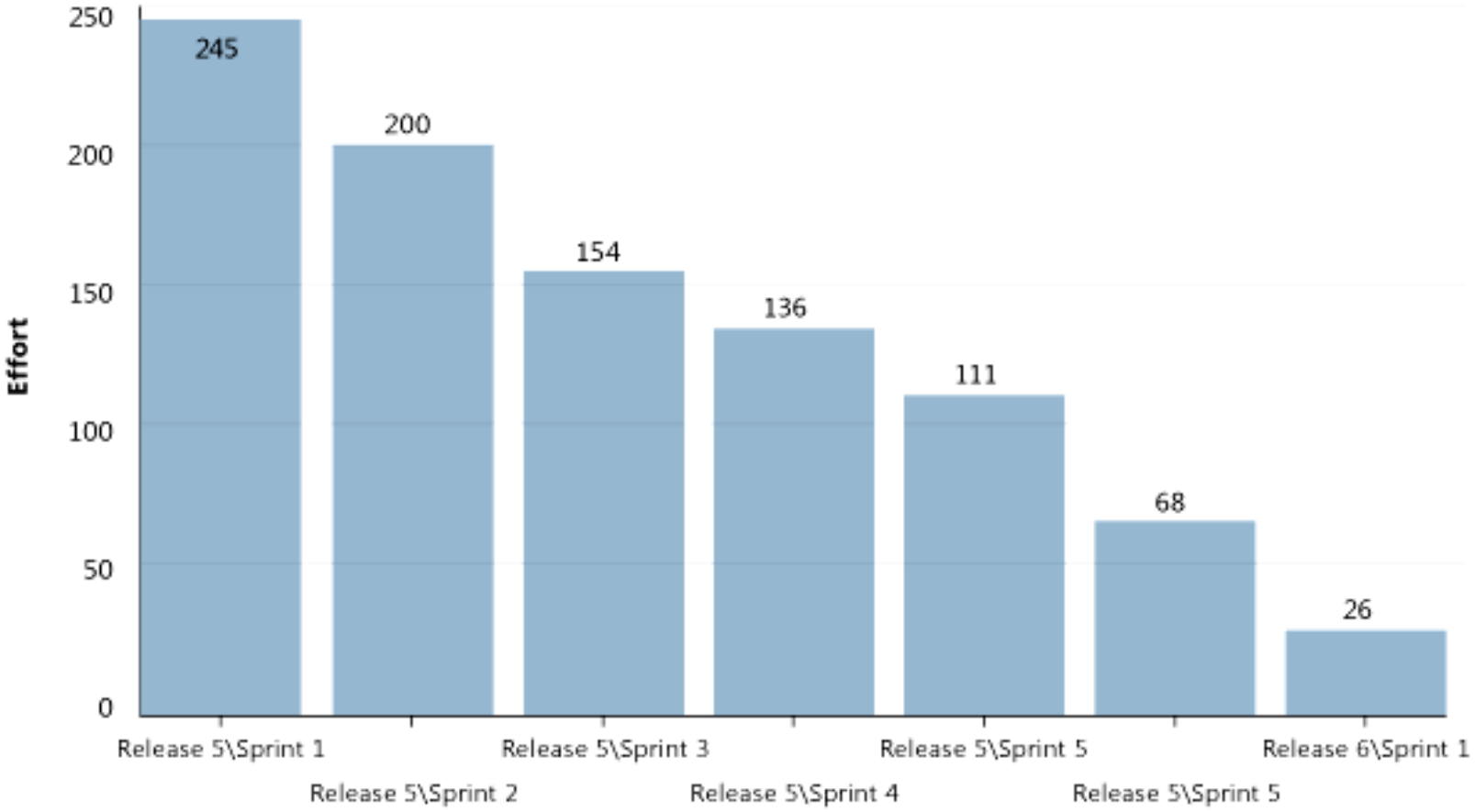
Appendix: Yet Another Scrum Picture

- The Scrum process:

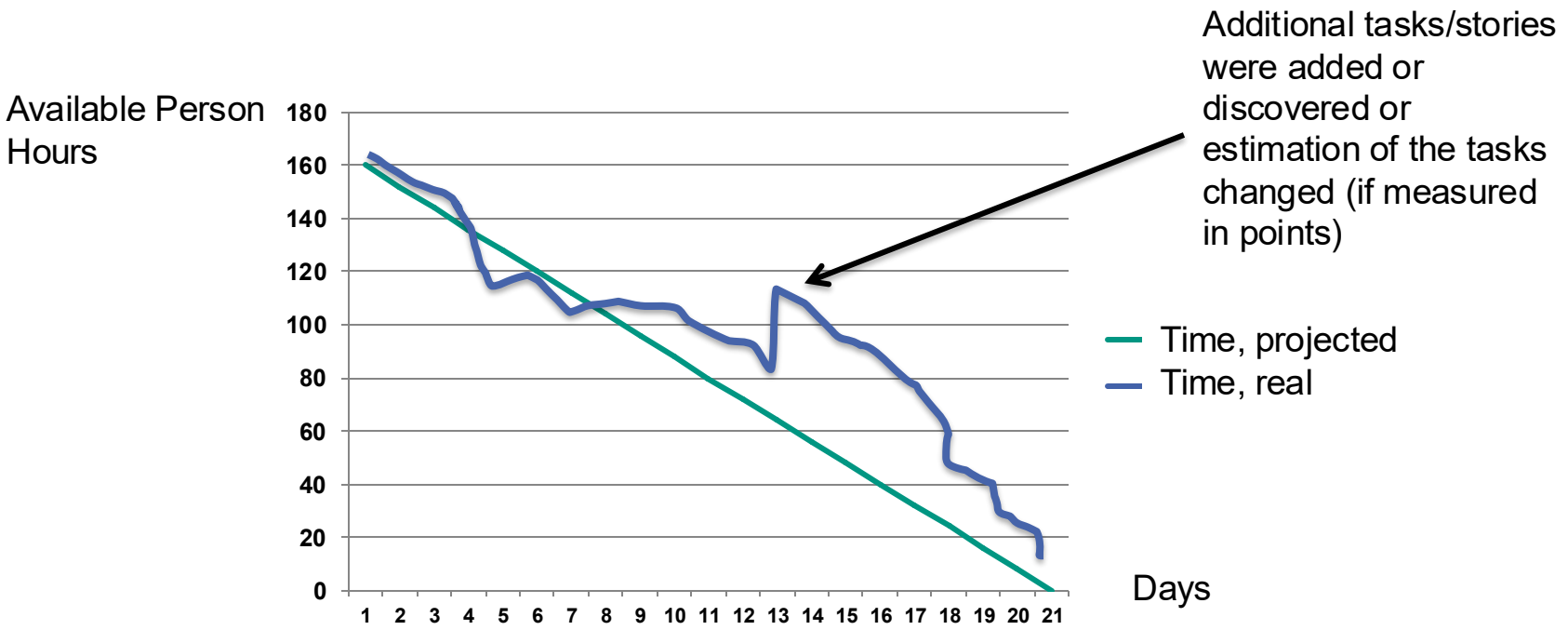


Appendix Burndown Chart

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



Appendix: Sprint Burndown Example



Appendix: Sample Exam Question I

Agile Methods in General (3P)	True	False
Agile methods offer goal-oriented project management.		
Agile methods have comparatively little management overhead with experienced teams.		
Test-driven development has been proven to lead to significantly better software quality.		
Agile methods offer earlier risk identification through collaboration with the customer and earlier development.		
According to the agile manifesto, contract negotiations play a more important role than collaboration with the customer.		
New requirements are always welcome in agile processes and are immediately in the current sprint.		

Appendix: Sample Exam Question II

Extreme Programming (XP) (2P)	True	False
XP is suitable for a rapidly changing environment with a multitude of new requirements.		
XP is suitable for a small development team.		
XP is a non-replicable, ad hoc process.		
The majority of XP practices are extensively validated.		

Scrum (3P)	True	False
XP should be viewed more as a process model, whereas Scrum is a practice.		
Scrum is heavily dependent on the responsibility of individual team members		
Scrum scales well even for very large projects with team sizes more than 1000 developers.		
Scrum should be viewed as a general process model that must be tailored for different projects and domains.		
A Sprint backlog contains a description of all features to be implemented in the project, prioritized (stored) according to their business value.		
During the sprint planning meeting, the Scrum Master assigns tasks to the team members.		