Agile Development for Multimedia Projects

Praktikum Multimedia-Programmierung
Wintersemester 2012/13

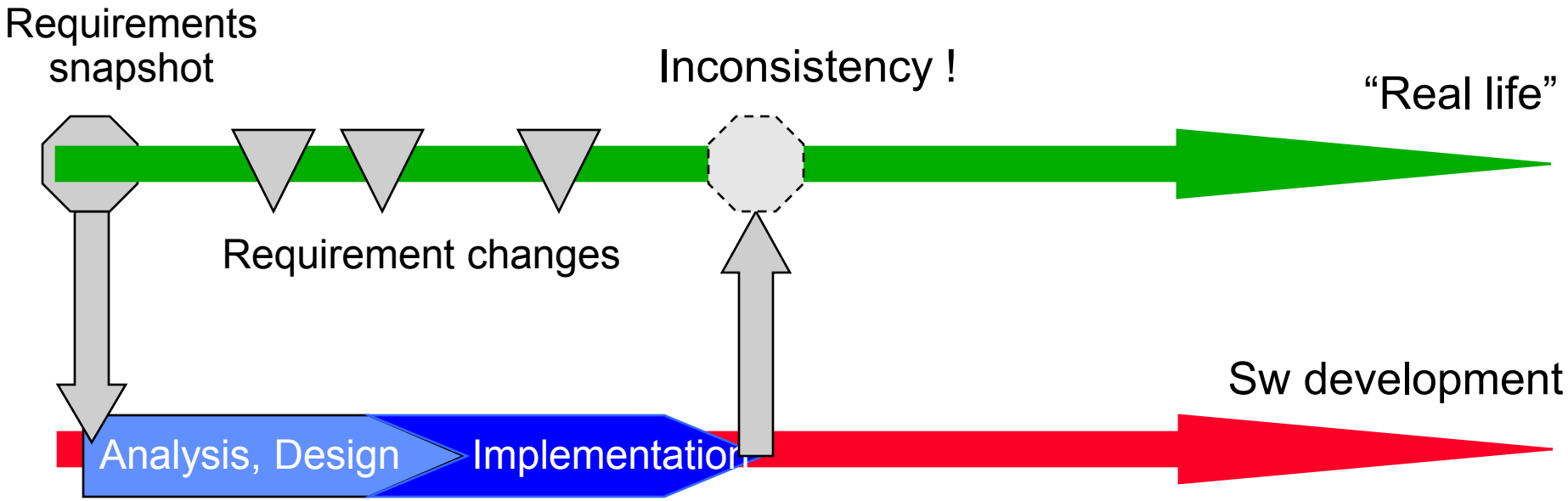
Heinrich Hußmann
LMU München

Literature:

- Mary & Tom Poppendieck: Lean Software Development – An Agile Toolkit, Addison-Wesley 2003
- Chris Sims, Hillary Louise Johnson: Scrum: A Breathtakingly Brief and Agile Introduction, Dymaxicon 2012
- Kent Beck: Extreme Programming Explained – Embrace Change, Addison-Wesley 2000
- Web: www.mountaingoatsoftware.com (Mike Cohn)

Changing Requirements

- Key problem in software development
  - Requirements change during course of project



Specific drivers for requirement changes in multimedia projects:

- New technologies & devices, new (corporate) design rules, new services, …
- Feedback from non-technical reviewers
  (designers, executive management, customers)

Cost of Change

K. Beck, Extreme Programming Explained, p. 21 ff:

“I can remember sitting in a big linoleum-floored classroom as a college junior and seeing the professor draw on the board the curve found in Figure 1.” ...

“The software engineering community has spent enormous resources in recent decades trying to reduce the cost of change – better languages, better database technology, better programming practices, better environments and tools, new notations. What would we do if all that investment paid off?”

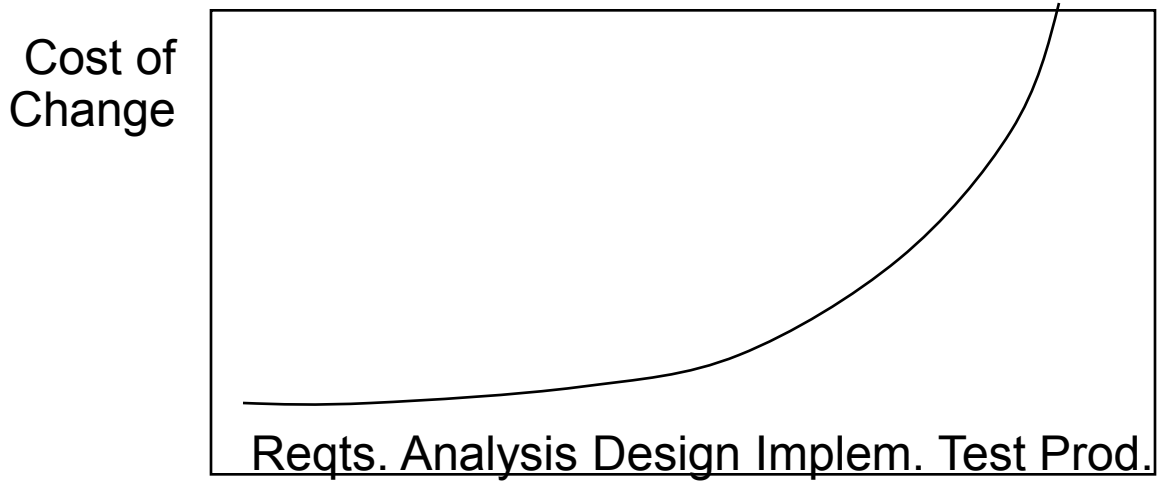


Fig. 1

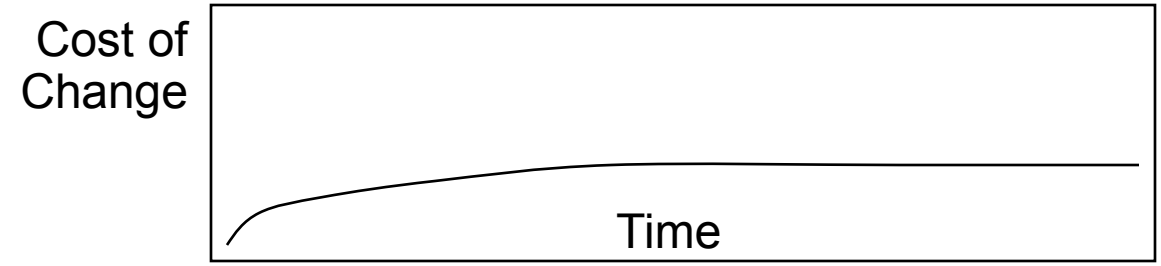


Fig. 2

“What if tomorrow’s software engineering professor draws Figure 2 on the board?”

Origins of “Lean Thinking”

- Late 1940s, Taiichi Ohno, Toyota Corp.:
  - How to produce cars in small quantities as inexpensive as mass-produced cars?
  - Fundamental lean principle: *Eliminate waste*.
    Anything that does not create value for the customer is waste.
    - » The Seven Wastes of Manufacturing (Shigeo Shingo):
      Inventory, extra processing, overproduction, transportation, waiting, motion, defects
  - Transfer from production process onto development process:
    - » How much value to the customer is delivered by an ongoing development project, a design, a prototype?
- James Womack, Daniel Jones, 1996: *Value Stream Mapping*
  - Which part of the production time is spent in actually adding value, which part is waste?
    - » Production of a Coca-Cola can: 319 days from mine to consumption, 3 hours spent on production, i.e. 0.04%

The Seven Wastes of Software Development

- Partially done work
  - Requirements, design documents, not yet integrated software modules
  - Idea: Try to get it into working state immediately
- Extra Processes
  - Idea: Remove all unnecessary paperwork
- Extra Features
  - Are they really needed in the end?
  - Idea: Resist the temptation, keep it simple.
- Task Switching
  - Idea: Assign a developer to one and only one project
- Waiting (for decisions mainly)
  - Idea: Try to delay decisions as long as possible and keep working
- Motion
  - Idea: Keep developers closely together, enable informal communication, try to keep a customer representative on the project team
- Defects

Poppendieck

Agile Software Development Methods

- Most famous authors promoting agile methods:
  Kent Beck, Alistair Cockburn
- Various methods, covered under the umbrella “agile”:
  - Extreme Programming “XP” (Kent Beck, ca. 1998), Crystal, Scrum, Adaptive Software Development
- Common to all agile methods:
  - Evolutionary development
  - Specific techniques of communication

agilemanifesto.org

- Extreme “marketing” effort – lots of “hype”
  - Relatively little empirical evidence for success
  - Increasingly popular within software industry

- In the following: Basics of “Scrum”, plus some elements from XP
  - Scrum: Jeff Sutherland, Ken Schwaber, 1995-2003

Agile Values

Individuals and interactions
over processes and tools

Working software
over comprehensive documentation

Customer collaboration
over contract negotiation

Responding to change
over following a plan

agilemanifesto.org

Selected Agile Principles

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

Welcome changing requirements, even late in development.

Business people and developers must work together daily.

Working software is the primary measure of progress.

The most efficient and effective method of conveying information is face-to-face conversation.

agilemanifesto.org

The Scrum Team

- Small group,
  around seven persons, “plus or minus two”
- Group should be co-located
  for the whole working time
- Scrum roles:
  - **Product Owner**
    - » Representative of the customer
    - » Is available to the team “on-site”
    - » Creates requirements in form of “user stories
  - **Scrum Master**
    - » Coach, helps in applying Scrum practices, facilitator
  - **Team Member**
    - » Completes user stories to incrementally increase product value
    - » Self-organizes, creates effort estimates
- Additional involved roles: Management, customer, user

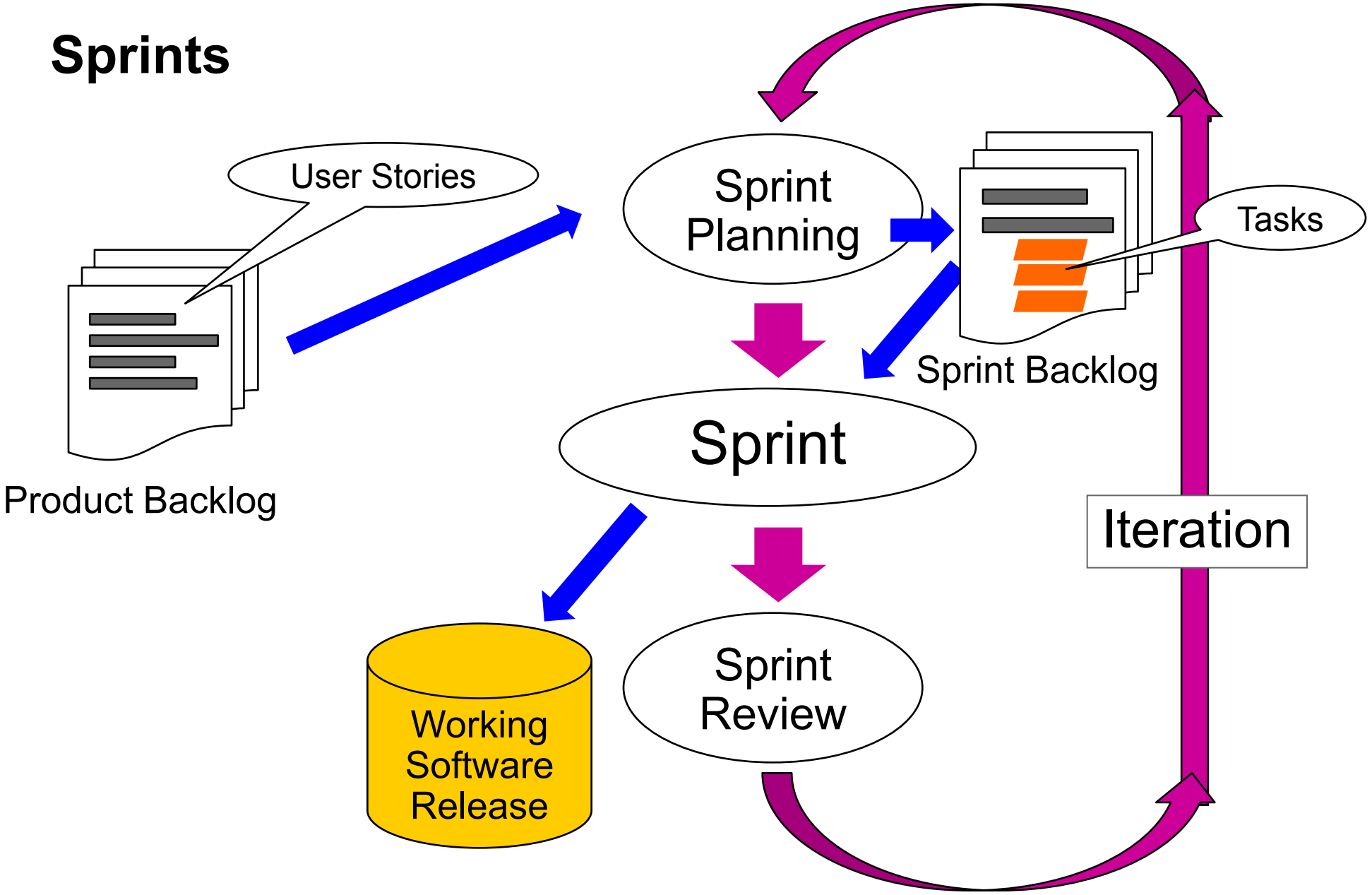
Scrum Artifact: Product Backlog

- **User story**
  - Functional requirement
  - "As a *<type of user>*, I want to *<do something>*, so that *<some value is created>*"
- Backlog items are *User Stories*
  - Short name of *User Story*
  - Sorted according to priority (for customer)
  - Includes effort estimate for implementation
- Additional information for each *User Story*:
  - Who is it for
  - What needs to be built
  - Why we should do it
  - Acceptance criteria

Backlog item	Estimate
Allow a guest to make a reservation	3
As a guest, I want to cancel a reservation.	5
As a guest, I want to change the dates of a reservation.	3
As a hotel employee, I can run RevPAR reports (revenue-per-available-room)	8
Improve exception handling	8
...	30
...	50

Example from www.mountaingoatsoftware.com

Sprints

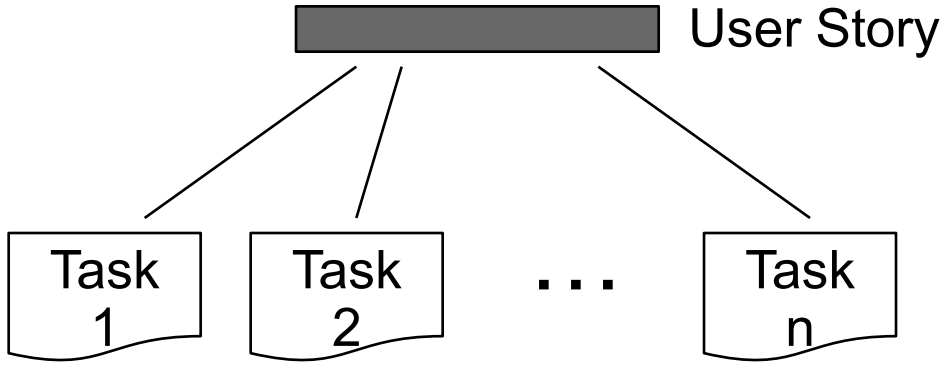

User Stories
Product Backlog
Sprint Planning
Tasks
Sprint Backlog
Sprint
Iteration
Working Software Release
Sprint Review

Scrum Artifact: Sprint Backlog

- Belongs to current *sprint*
  - Limited lifespan
- To-do list of the team for a single *sprint*

Tasks	Mon	Tues	Wed	Thur	Fri
Code the user interface	8	4	8		
Code the middle tier	16	12	10	4	
Test the middle tier	8	16	16	11	8
Write online help	12				
Write the foo class	8	8	8	8	8
Add error logging			8	4	

- For all *user stories* to be delivered in the *sprint:*
- *Sprint backlog* lists the required *tasks*:
  - *Task* = Unit of work, assigned to people carrying it out
  - A *user story* is implemented by carrying out a number of tasks

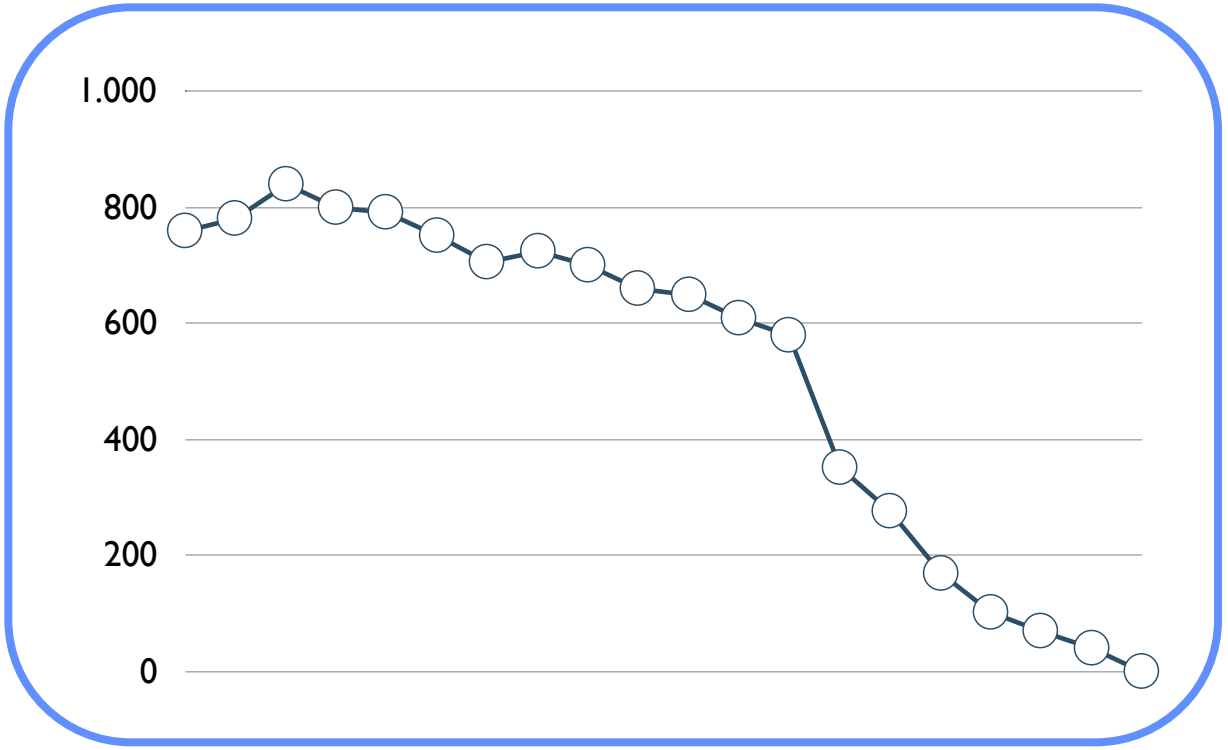


Example from
www.mountaingoatsoftware.com

Duration of a Sprint

- Typical length: Open!
  - Shorter and more frequent cycles are better
  - Early papers on Scrum: One-month sprints
  - Current practice two-week sprints
  - Tendency to one-week sprints
- Sprint length depends on:
  - Type of project
  - Interrelationship of features
- For the “Blockpraktikum”, likely:
  - Very few sprints
  - Sprints of a few days duration
  - Decision is up to the team!

Scrum Artifact: Burn Down Chart

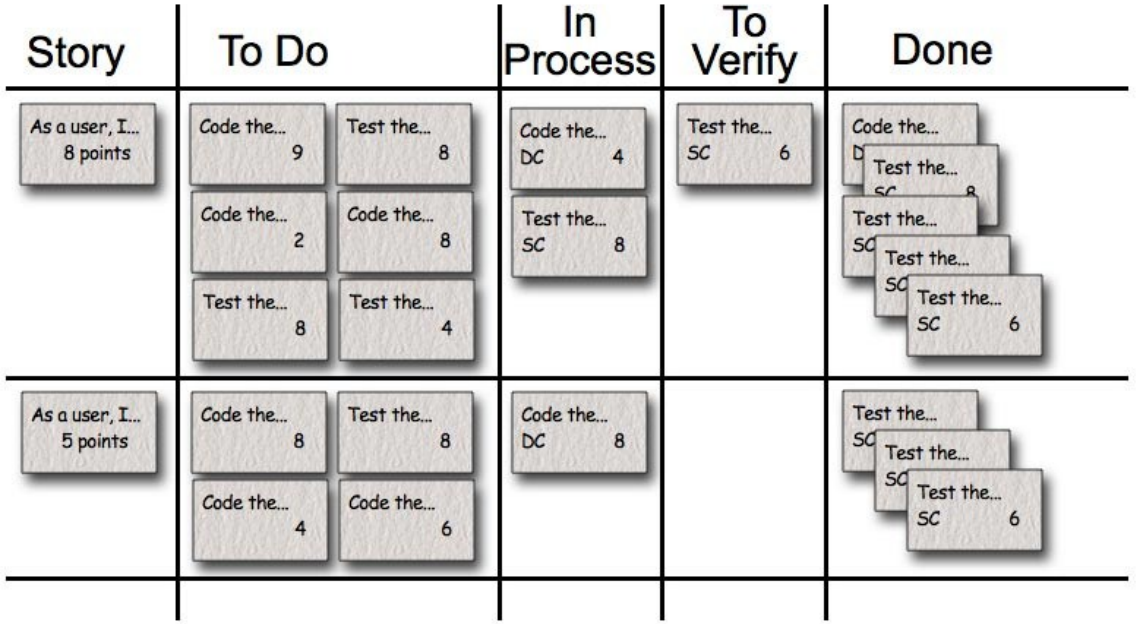


Example from
www.mountaingoatsoftware.com

- X-Axis: Time (sequence of sprints/releases)
- Y-Axis: Scope of undone work (number of tasks, or number of hours)
- Diagonal line downwards from left to right:
  - Each time something is completed, the line goes down
- Vertical “jumps”:
  - Each time the scope is extended or reduced, a vertical line appears

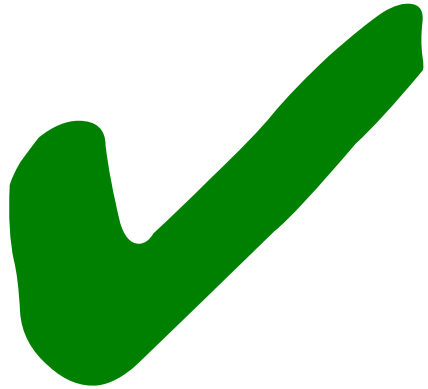
Scrum Artifact: Sprint Task Board

- Optional tool



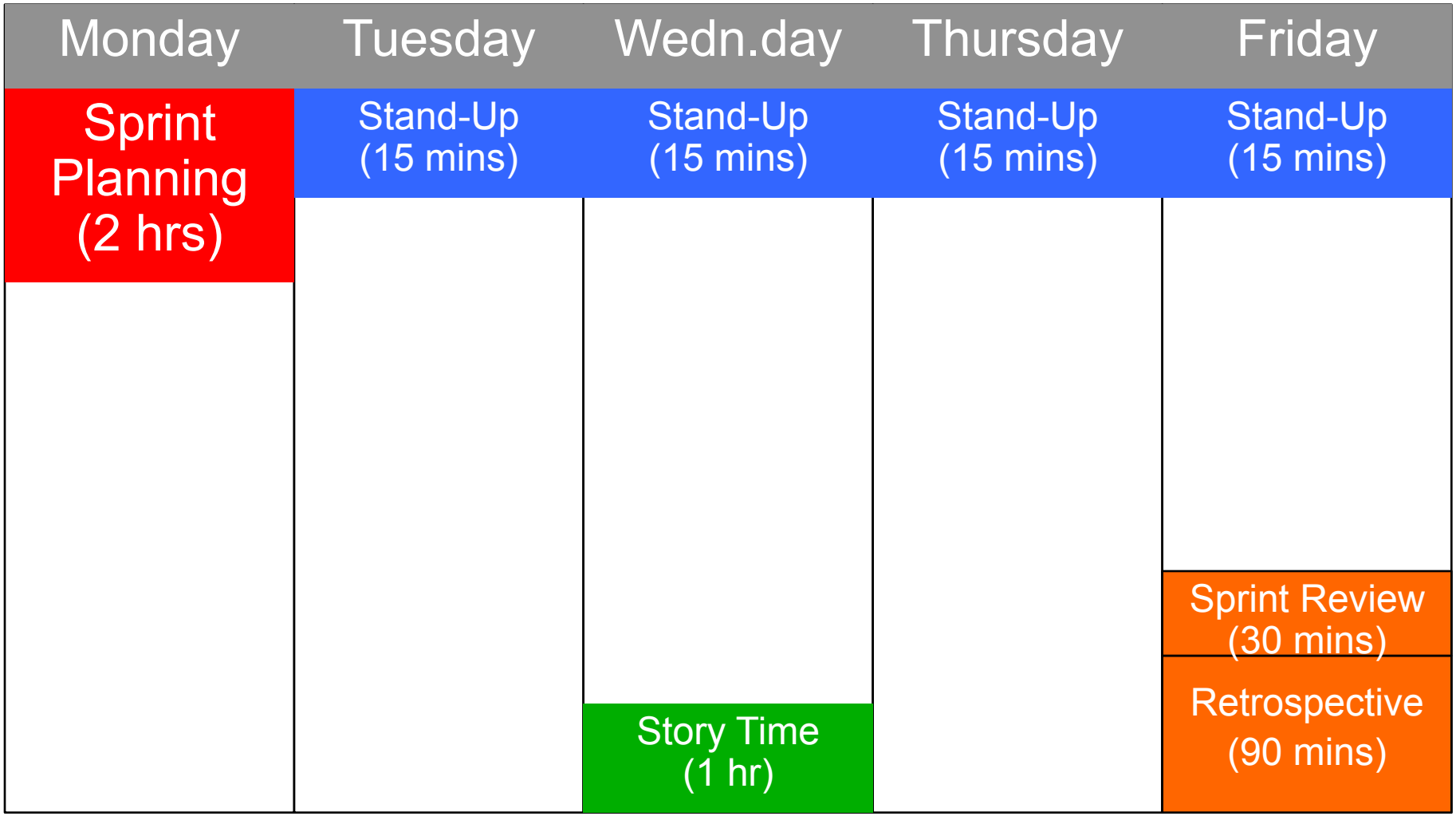
- Visualization of *tasks* within current *sprint*
  - Well visible for team and other stakeholders
  - Creates awareness of progress
- Several categories for task status:
  - To Do, In Process, To Verify, Done, …
- Tasks move between categories
  - E.g. To Do ➔ In Process➔ Done
- Various representations and adaptations are possible

Scrum Artifact: Definition of Done



- When does the team call a user story “done”?
  - Code written
  - Code reviewed
  - Unit tests completed
  - Product owner signed off
- Team writes down its own definition
- Posted publicly in the team room

Typical Schedule for a One-Week Sprint



Scrum Ceremony: Sprint Planning

- Part 1:
  - Team commits to a set of *user stories* to be delivered in the *sprint*
  - Product owner presents stories in priority order
  - Team and Product owner agree on acceptance criteria
  - Team decides, one by one, how much work to commit to
- Part 2:
  - Team identifies tasks to be performed to deliver
- Output:
  - *Sprint backlog*
  - Contents of *sprint task board*

XP Practice: Planning as a Game

- Project owner has to learn about effort to implement stories
  - Proposal of team (how many days will it need), decision of project owner
- Team members have to learn about priorities of the customer
  - Proposal of project owner (what is important), decision by team
- Balanced solution is needed, has to be found in collaboration
  - Takes place in *Sprint Planning* and *Story Time* meetings

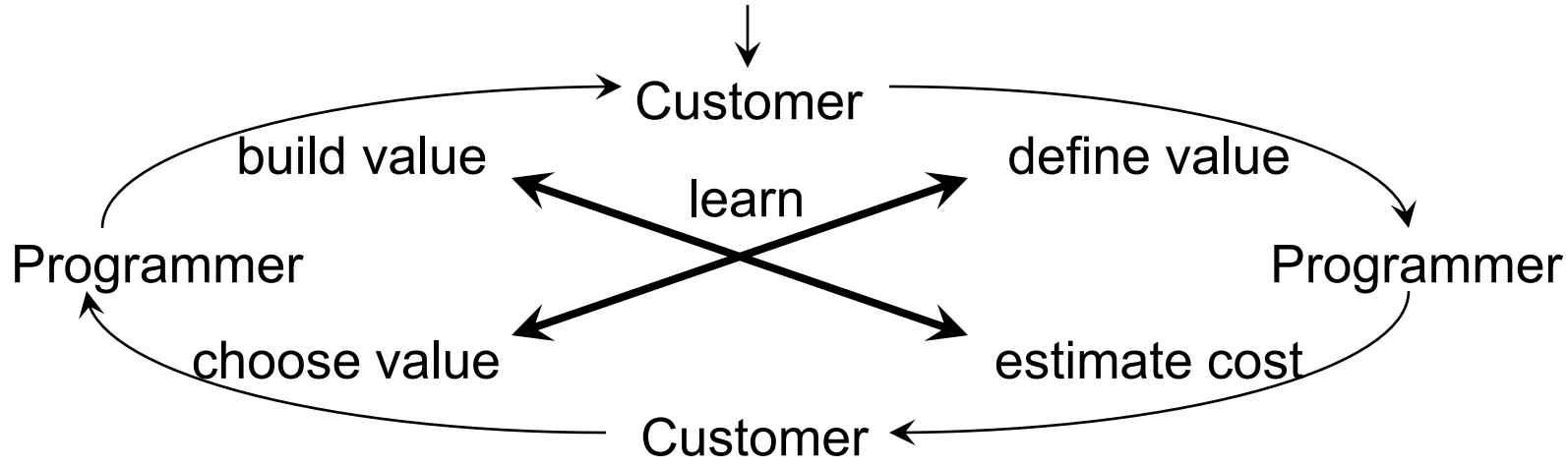


Figure from Jeffries/Anderson/Hendrickson: XP installed, Addison-Wesley 2001

Scrum Ceremony: Daily Scrum (Standup Meeting)

- Daily on each project day
- Brief (15 mins maximum)
  - therefore everybody standing!
- Pointed: Clearly defined scope

- Each participant in turn shares:
  - What tasks I have completed since the last *Daily Scrum*?
  - What tasks I expect to have completed by the next *Daily Scrum*?
  - What obstacles are slowing me down?

Scrum Ceremony: Sprint Review

- Official end of the *sprint*
- Presentation of accomplishments to product owner
- Communication about *user stories* not completed
- “Inspect-and-adapt”: Creation of new ideas

Scrum Ceremony *(not yet official)*: Story Time

- Once per *sprint*, discuss and refine stories in the *product backlog*
- Per story:
  - Define and refine acceptance criteria
  - Assign size (effort estimation)
  - Split big stories in smaller ones
- In particular, jobs high in priority should be small
  - Easier to understand and commit

Scrum Ceremony: Retrospective

- Inspect-and-adapt session not about product, but about the ***process***
- Identify one or two strategic issues to be done better in next *sprint*

XP Practice: Simple Design

- "The right design for the software at any given time is the one that
  1. Runs all the tests
  2. Has no duplicated logic
  3. States every intention important to the programmers
  4. Has the fewest possible classes and methods."

  (K.Beck, p.57)

- XP *mantras*: "The simplest thing that could possibly work.", "You ain't going to need it. (YAGNI)"
  - Erase (or better do not add in the first place) everything unnecessary
- Design is represented in code
  - There is no separate documentation
  - Graphical sketches (e.g. UML) only used for short digression (essentially for finding the right questions)

XP Practice: Testing

- “Any program feature without an automated test simply does not exist.”
  (K. Beck, p. 57)
- Test-First approach:
  - Tests are written *before* the program
  - Tests are used to clarify the usage of an interface, to define the expected effect, to single out problematic special cases, ...
  - Tests are the XP replacement for traditional software specification
- Automated tests:
  - Tests are kept in an executable infrastructure and can be run at any time
  - Tests give feedback and confidence to the programmer
  - “Test infected: Programmers love testing”
    (E. Gamma about the xUnit testing framework)
- Programmer-written unit tests: Must always run to 100%
- Customer-written functional tests based on “stories”: May run only partially for some time

XP Practice: Refactoring

- Adding new features in an arbitrary way will lead to ill-structured code
- When adding a new feature, the structure of the system may need to be adapted *(refactored)*
  - Refactoring may remove code, introduce new code but keeps the functionality unchanged (all tests run 100%)
  - Simple steps like combining parameters of a method into a data structure
  - Complex steps like applying design patterns
- This is done only when necessary to keep the solution simple:
  - Main reason for refactoring: To avoid duplication of logic
  - Possible other reason: Smaller and more elegant design after introduction of new features
- How can we be sure not to destroy working functionality by refactoring?
  - Use automated tests
  - Refactor first, run the tests, then add the feature, run the enhanced tests

XP Practice: Pair Programming

- “All production code is written with two people looking at one machine, with one keyboard and one mouse.” (K. Beck, p. 58)
- Two roles:
  - Keyboard/mouse owner: Thinks tactically about the best way to implement the method under development
  - Observer: Thinks strategically about the overall approach and simplification
  - Roles in the pair may be switched after some time
- Dynamic pairing:
  - Pair partners should change frequently
  - Changes may take place even several times a day

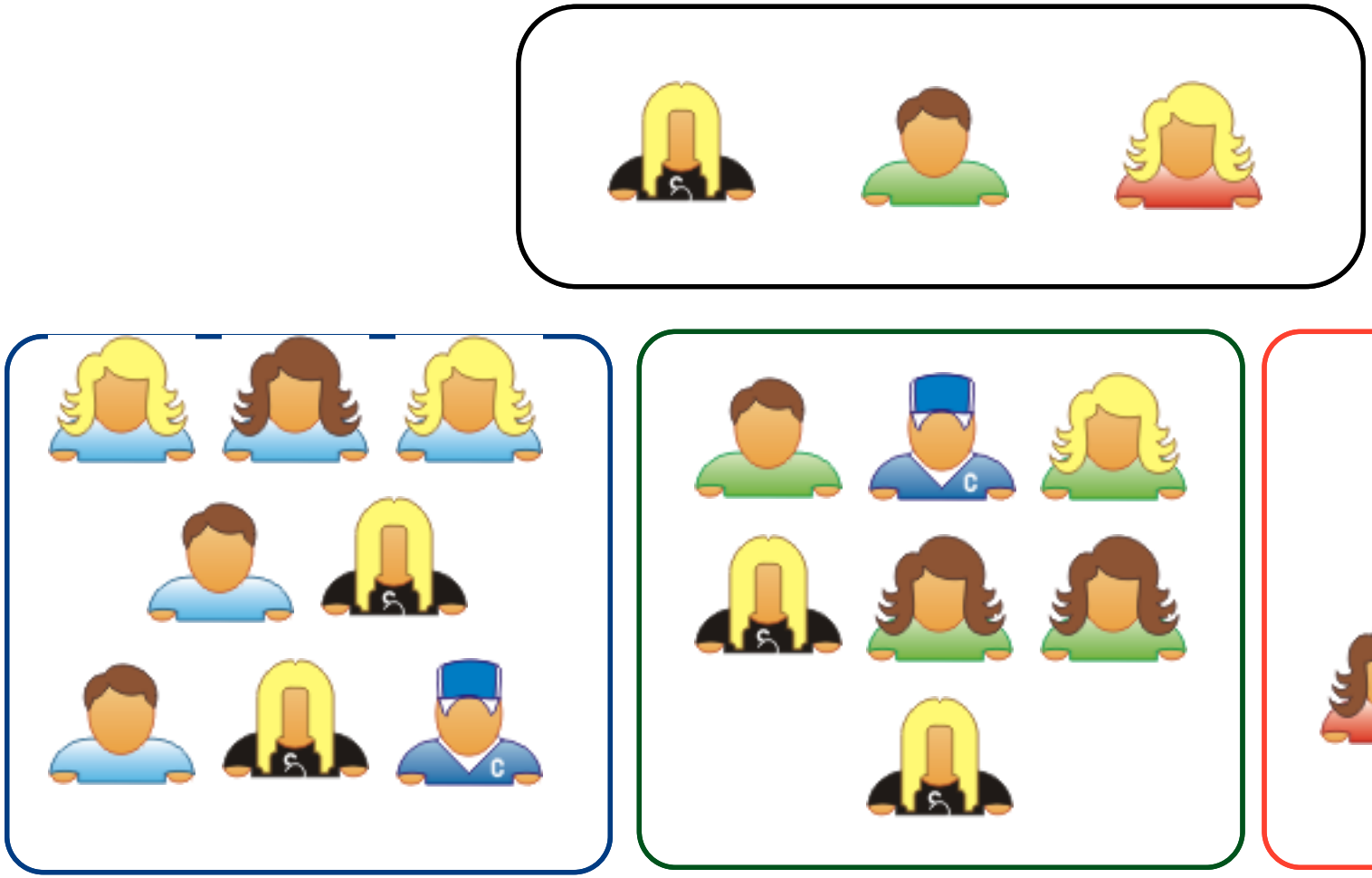
- ***Potential problems:***
  - Complex programming tasks are often better solved by a single person “meditating” over the solution
  - Some people simply do not like the pair situation

XP Practice: Collective Ownership

- Through pair programming, any piece of code has many authors
  - Side effect: Removes too complex code
- *Everybody in the team has the right to add value to any portion of the code at any time.*
- If somebody does not know some part of the system well, he/she should pair up with an expert on this part
- Practical tool for co-ordination: *Stand-Up Meetings (Daily Scrums)*

- ***Potential problems:***
  - Collective ownership means no individual responsibility
  - Collectivism partially contradicts to human nature

Scalability of Scrum: Scrum of Scrums



Picture: www.mountaingoatsoftware.com

Suggestion for Scrum in “Blockpraktikum”

- First day (today):
  - Initial planning meeting (*Story Time* + first *Sprint Planning*) (90 mins)
  - Creation of *Product Backlog*
  - Definition of first *sprint*
- Development days (7+):
  - *Daily Scrum* in the morning (10 mins)
- After each *sprint* (1–2 days):
  - Combined *Sprint Review/Story Time* (with a little *Retrospective*) (60 mins)
- Before each *sprint* (3–4 sprints):
  - *Sprint Planning* (30 mins)
- Final presentation, last day
  - *Sprint Review* + *Retrospective*