From waterfall to SCRUM:
Becoming Agile

Claudio Scordino

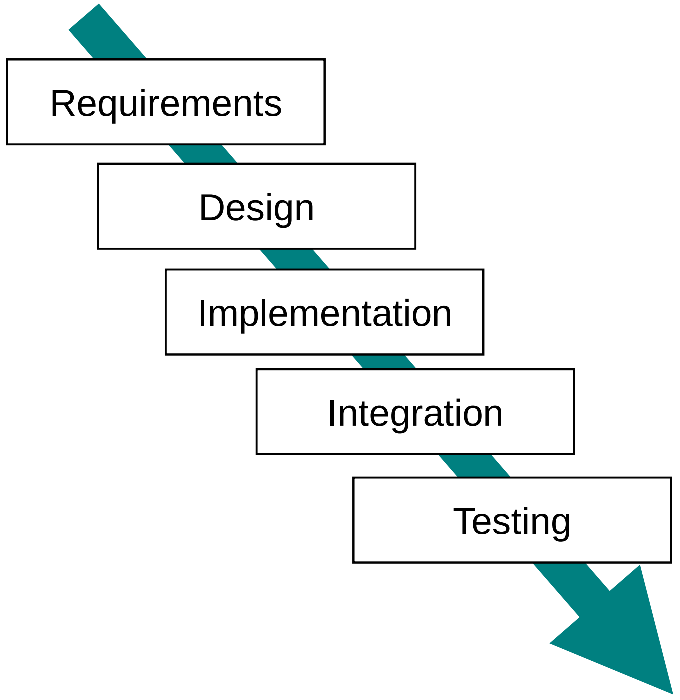
Certified Scrum Master (CSM), Evidence Srl

claudio@evidence.eu.com

Version 1.9

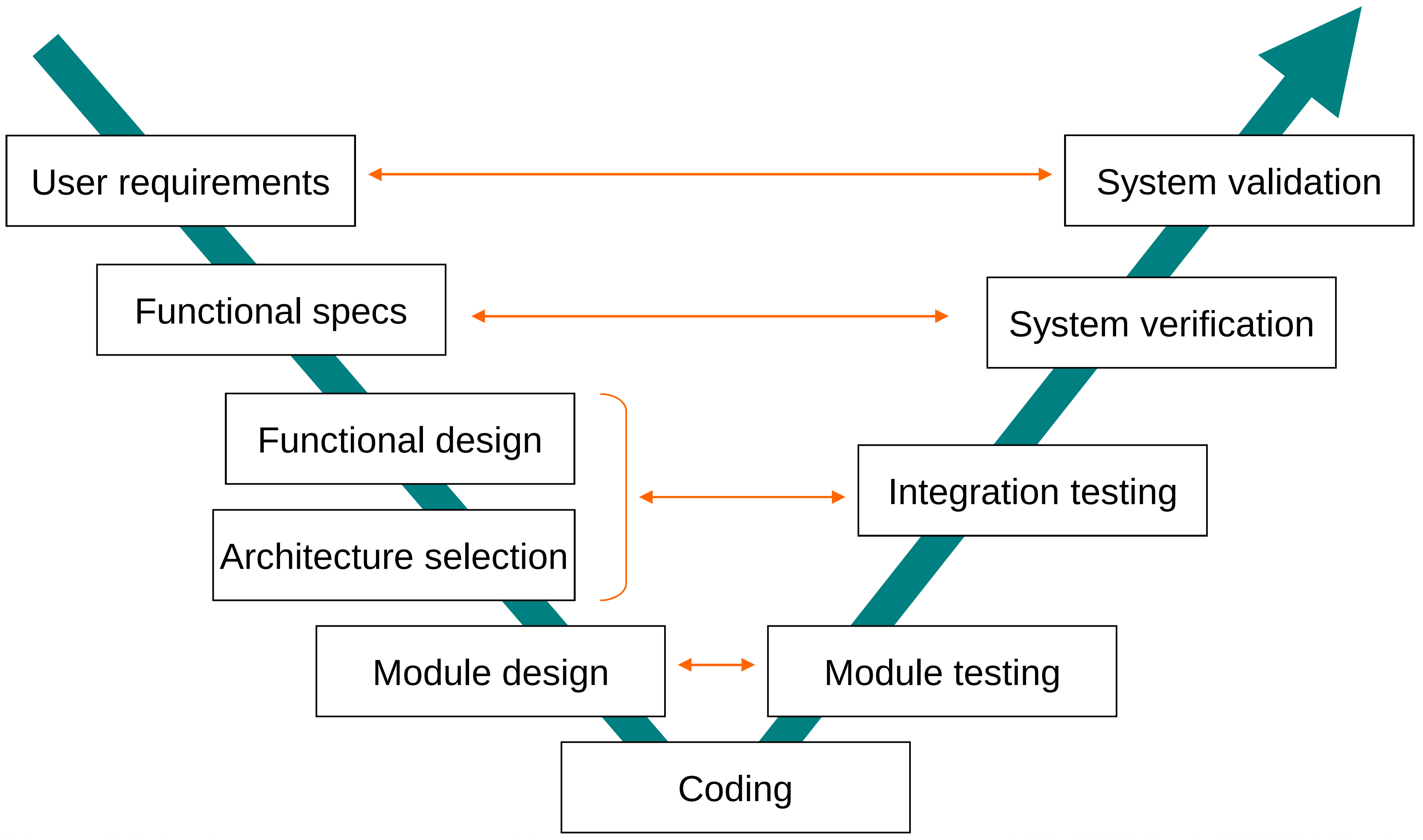
Waterfall methodologies

- Historically, software development has been done following a waterfall methodology:



V-Model

- Even V-Model is a variant of the waterfall methodology:



Software is different

Software development has been threated as manufacturing...

... but software is different from products of manufacturing:

- Software development is an intellectual and R&D activity
- Complexity of the whole system increases much more than linearly with respect to size
- Software can be changed easily and with a low cost
- Large variability
- Software is invisible

Something wrong with waterfall

- The waterfall model
  - has been introduced around 1970
  - is very rigid

- During the course of the years, people realized that waterfall failed in many circumstances:
  - Late delivery:
    - Projects didn't meet deadlines
    - Delays of months or even years
    - Too long times to react to market demands
    - Some projects canceled even before the first release!

    => Big waste of money

Something wrong with waterfall (2)

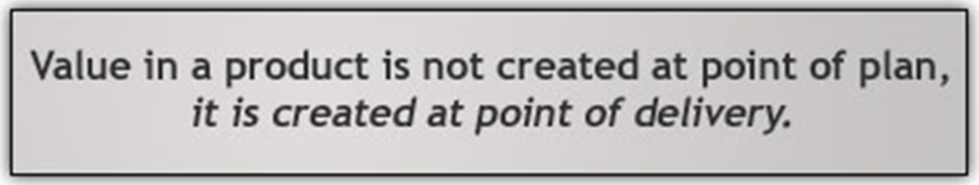
...

- Developers worked 12+ hours/day and weekends
  - Job dissatisfaction: people couldn't enjoy their job
  - People got stressed and unsatisfied by their job
- Final product too different from customers' expectations:
  - Only discovered at release time
  - Further time/money to fix the product

"The software has the power to become a monster capable of missed schedules, blown budgets and flawed components" (F.Brooks)

An example

- Example: see the $788,217 Printer Stand
  - http://theagileadvisors.com/agile-projects/in-agile-value-is-created-at-point-of-delivery


Value in a product is not created at point of plan,
it is created at point of delivery.

Some issues with waterfall

- Requirements issues: requirements changing or not available
  - Customers don't have idea of what they want
  - Customers change requirements after design
    - Lack of flexibility or problems with deadlines
  - Requirements available only when customer "see something"
  - It is difficult to specify some kind of requirements (e.g., Look&Feel of a GUI)

Some issues with waterfall (2)

- Implementation issues: technology unknown
  - Partial ignorance of the technology that is going to be used
  - Only at implementation, developers discover what they can do (and what not) with the technology
    - Unanticipated technical problems
    - This can affect requirements (e.g., some requirement canceled)
    - This may impose a change of technology
  - Only at implementation, developers discover the actual time needed for development

Some issues with waterfall (3)

- Release issues: market and technology has changed
  - Market changes continuously, even during our development
  - e.g., a competitor releases a better product before us
  - New technology evolves or becomes available during development
  - The company changes direction

Assumptions behind waterfall

The waterfall model assumes that after the phase of requirements:

1. Change will not happen
2. Everything has been understood

Both assumptions are not true!

Therefore, our model must assume that:

1. Change will not be small and manageable
2. Suppliers or customers cannot fully understand all the requirements up front

Waterfall: just an illusion

- With waterfall you only have the illusion that the development process is under control
  - The supplier gives this illusion to the customer, and uses gantt charts to prove that everything is under control... but actually it is not.

- When change happens, the supplier usually hides the problem wishful thinking that everything will be fixed by:
  - Increased overtime
  - Dirty fixes
  - More working people

Waterfall: just an illusion (2)

- Consequences:
  - Increased overtime
    - Atttition increases and morale decreases
      - ⇒ People leave the company
    - Less cognitive efficiency
      - ⇒ More defects (bugs)
      - ⇒ Not seeing/dealing with weaknesses
  - Dirty fixes
    - ⇒ Less quality
    - ⇒ More defects (bugs)
  - More working people
    - ⇒ Increased costs

Lister's law: people under time pressure don't think faster

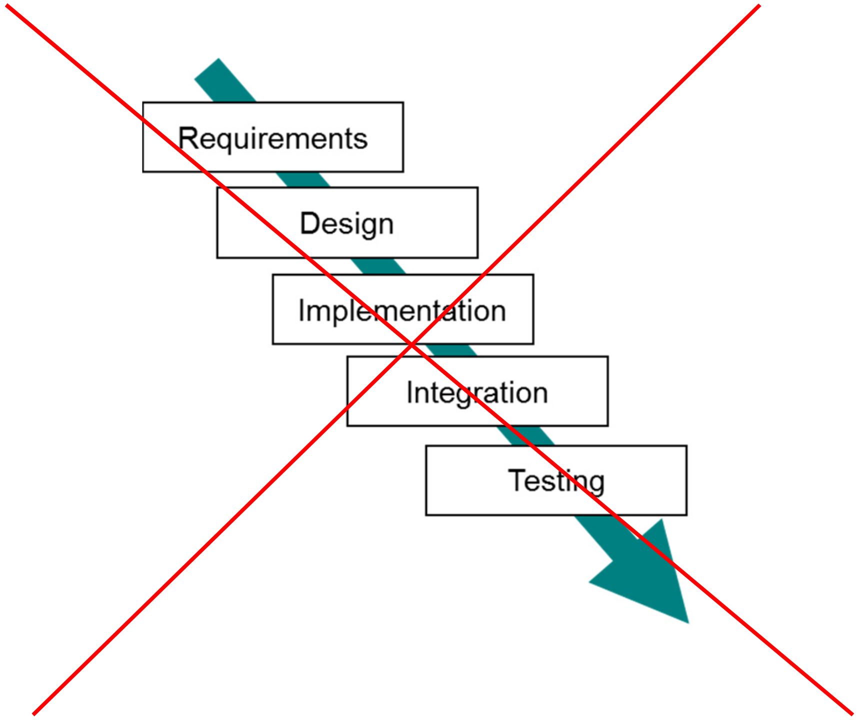
Waterfall doesn't work with software

- There isn't <u>any</u> proof in literature that waterfall works
- Instead, some works clearly show that waterfall with software can't work
  - F.P.Brooks *"No Silver Bullet – Essence and Accident in Software Engineering"* (1986 !) which suggests
    - "rapid prototyping"
    - "extensive iteration between designer and customer"
    - "impossible for customers to specify completely, precisely and correctly the exact requirements of a modern software product before having tried some versions"
    - "Software system grown by incremental design", adding "more and more functions as they are run, used and tested"
    - "Have at every stage in the process a working system"

Waterfall doesn't work with software (2)

- Barry Boehm, *"Anchoring the Software Process"*, 1996
- Statistical evidence: Larman, *"Agile and Iterative Development: A Manager's Guide"*, 2004
- D.Leffingwell, *"Scaling Software Agility: Best Practices for Large Enterprises"*, Chapter 2, 2007

Waterfal doesn't work with software (3)



Something new is needed...

- *"Change begins with an awareness that the status quo is no longer desirable"* (M.Cohn)

- In 80's, people realized that something was wrong with current methodologies, and start thinking about new methodologies for industrial software development...

Manifesto for Agile Software Development

Agile: term coined in 2001 when the Agile Manifesto was formulated

– Kent Beck and Martin Fowler were among the authors

" *We are uncovering better ways of developing software by doing it and helping others do it [...]* "

Agile Manifesto

http://agilemanifesto.org

Agile Manifesto: 4 values

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.

Agile Manifesto: 12 principles

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.

Agile Manifesto: 12 principles (2)

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity - the art of maximizing the amount of work not done - is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile methods

- Software development methodologies based on iterative development
  - See http://en.wikipedia.org/wiki/Agile_software_development
- They promote a disciplined project management process that encourages adapting to change through:
  - Iterations with frequent inspections
  - Teamwork, self-organization and collaboration
  - Development aligned with customer needs
- Gartner predicts that by 2012 “agile development methods will be utilized in 80% of all software development projects”

What is Agile for ?

- Agile is <u>not</u> for
  - Reducing time-to-market
  - Reducing costs
  - Faster development

- AGILE = ADAPT TO CHANGE
  - Motto: *"Agile is for Agile"*
  - Be sure management understands what agile is for
  - Need to be clearly explained to customers

Agile = ready for changes

- In traditional process development (waterfall) there is the assumption that change is a problem, because it increases costs
- But changes can't be avoided
- What Agile methods do is to lower the cost of change

Agile methods (2)

- Tasks are broken into small increments with minimal planning, and do not directly involve long-term planning
- Iterations: short time frames ranging from 1 to 4 weeks
- Benefits:
  - Minimize overall risk
  - Let the project adapt to changes quickly
  - Reduced time-to-market
- Extreme Programming (XP) is one Agile method
  - Created by Kent Beck in 1996
  - Practices: TDD, pair programming, continuous integration
- Another Agile method is SCRUM

Extreme Programming (XP)

- Popular and somewhat controversial agile method consisting in values, principles and practices
  - It can be used together with SCRUM

- Sit together (Open-Space and communication)
- Whole team (People allocated full time to a project)
- Informative workspace (Blackboard to draw how project proceeds)
- Energized work
  - Work only as many hours as you can be productive
  - When you're sick, rest and get well
  - When coding, turn off phone and email notifications
- Pair programming with pairs rotating every couple of hours
- Ten-minute build (build and test shouldn't take more than 10 min.)

Extreme Programming (XP) (2)

- Continuous integration:
  - Commit many times a day
  - Let the build system automatically test the code and report errors through some kind of notification (e.g., a display or email)
  - See Hudson
- Incremental design:
  - Daily attention to design
  - The most effective time to design is in the light of experience
- Team continuity (people work mostly with those they know and trust)
- Single code base: (keep only one code stream: no multiple branches)

SCRUM

- Management framework for iterative incremental product development
  - Approach to optimize predictability and control risk
- Usually associated with object-oriented programming
- Scrum provides a structure of roles, meetings, rules, etc.
  - Scrum doesn't provide practices!

SCRUM structure

Roles:

- Product Owner
- Team
- ScrumMaster

Artifacts:

- Sprint Burndown
- Release Burndown

Meetings:

- Release Planning
- Sprint Planning
- Daily Scrum
- Sprint Refinement
- Sprint Review
- Sprint Retrospective

SCRUM: Brief history

- SCRUM is based on the research by Takeuchi and Nonaka
  - Takeuchi, Nonaka, "The new new Product Development Game", 1986
  - Main ideas:
    - No sequential life cycle (i.e., no waterfall)
    - No traditional division of labour (self-organizing /multilearning teams)
- Note: SCRUM originally started in hardware (not in software) development
- Formally presented by Sutherland and Schwaber at OOPSLA 1995
- Now used by large and small companies, including Yahoo!, Microsoft, Google, Nokia, Siemens, Motorola, SAP, Cisco, Alcatel.

Simple rules, very deep implications

- SCRUM has a set of simple rules
- These simple rules, however, have very deep implications in the organization
  - Change of roles
  - Change of mindset of
    - Management
    - Developers
    - Customers
  - Change of development process

Be sure that management understands the impacts of Scrum (change in organization's roles and responsibilities) !

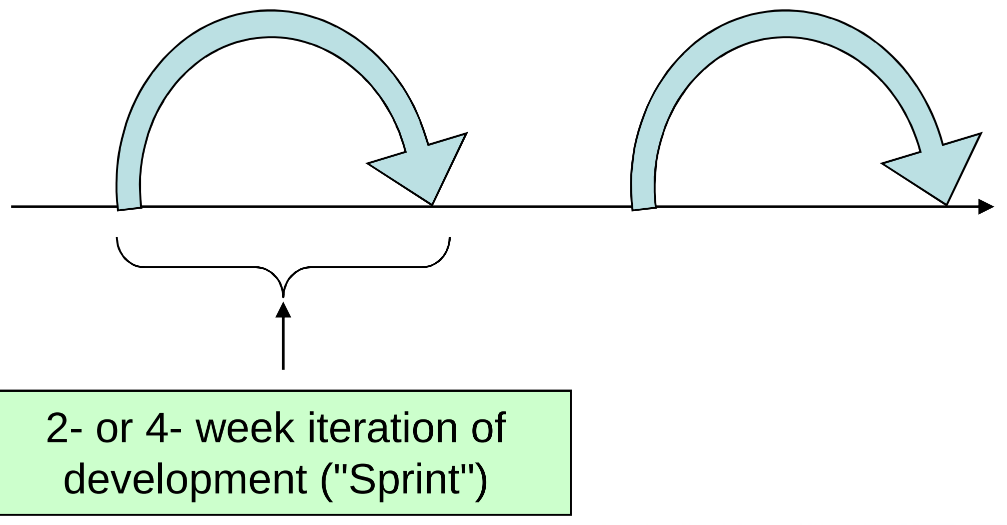
SCRUM is a mirror

"Scrum is a mirror" (A.Cockburn)

- SCRUM does not solve weaknesses. It's not "the solution"
- SCRUM just exposes problems and weaknesses
  - When you first adopt SCRUM, everything will get "worse"

Sprints

- The development is divided in fixed-length iterations, called ``Sprints''
- Fixed timeboxes: the length of the sprint is fixed
  - It ranges from 2 to 4 weeks
  - The length is chosen by the Team



Sprints (2)

- At the end of each Sprint the Team must have created a Potentially Shippable Product Increment (PSPI)
  - The system should be of production quality
  - This means properly tested and documented
- The end can't be deferred: if all functionalities can't be build, then functionalities are deferred
- No one is allowed to add work to a Sprint once it is underway
- A Sprint can be canceled if it no longer makes sense given the circumstances
  - Very uncommon because can be traumatic to the Team

The predictability of the project is controlled at least each Sprint, so that the risk that the project may go out of control or become unpredictable is contained.

Question

- What prevents our company from delivering a PSPI every 2 weeks ?
  - Automated test ?
  - Automated documentation ?
  - Team size ?
  - Mindset of
    - Developers ?
    - Management ?
    - Customers ?

Sprint Length

- Time boxing is proven to have higher productivity than content-boxing
- Sprint length ranges from 2 to 4 weeks
- 2 weeks is the industry average and it is the suggested lenght
  - Smaller goals in time create the urgency for cooperation
    - Longer-term goals don't create the same amount of cooperation
  - Shorter iterations force people to not use a mini-waterfall

Benefits of short iterations

Technical benefits:

- More quality
- More stability
- Less defects
- Less waste
- Higher team coesion
- More focusing
- Less product complexity

Business benefits:

- Shorter time-to-market
- More competition (adaptation to market change)
- Reduced costs
- More job satisfaction
- More transparency
- More feedback from customers
- More cost control
- Better reputation

Sprint length (2)

If the Team sees it can't accomplish its goal in a Sprint, we can't:

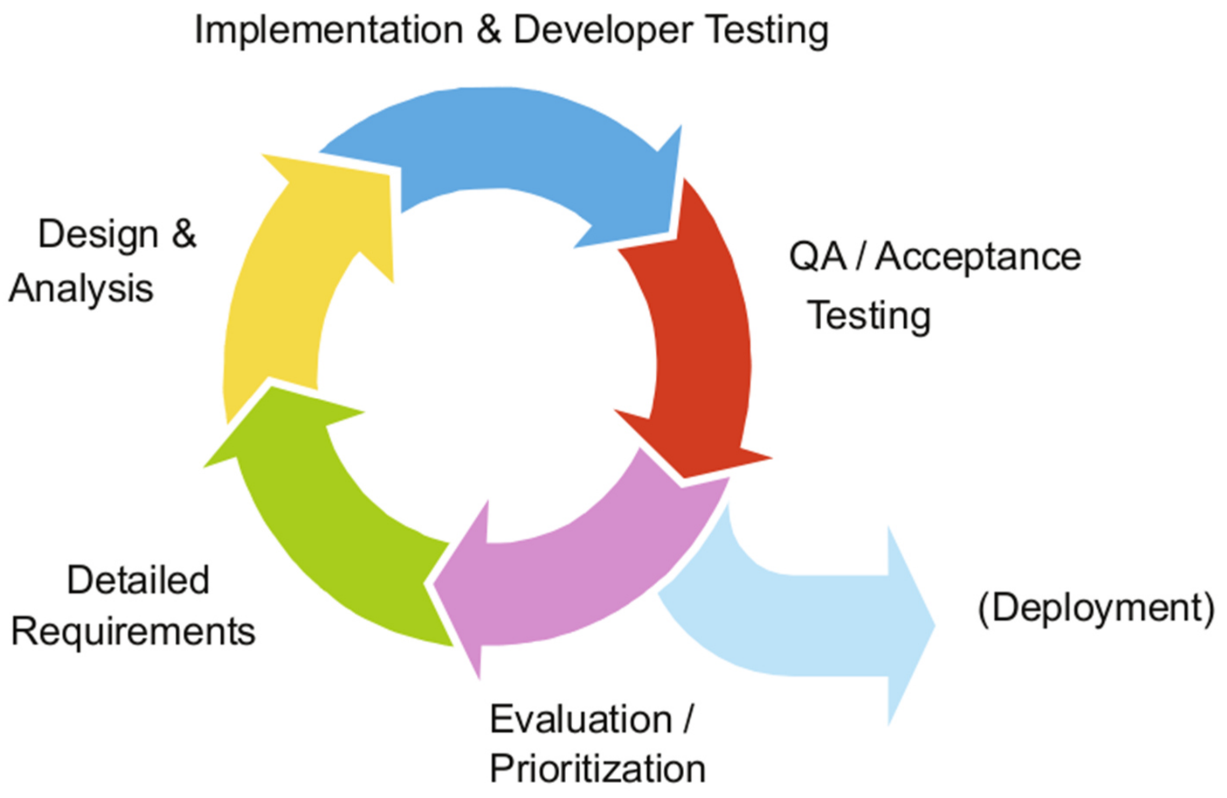
- Extend the timebox
- Reduce quality of practices
  Quality is <u>not</u> a control variable, and cannot be reduced!

but we can

- Add more resources
- Change scope of the item (item "descoping")
  - Descope items, not tasks!
- Switch an item with another that fits

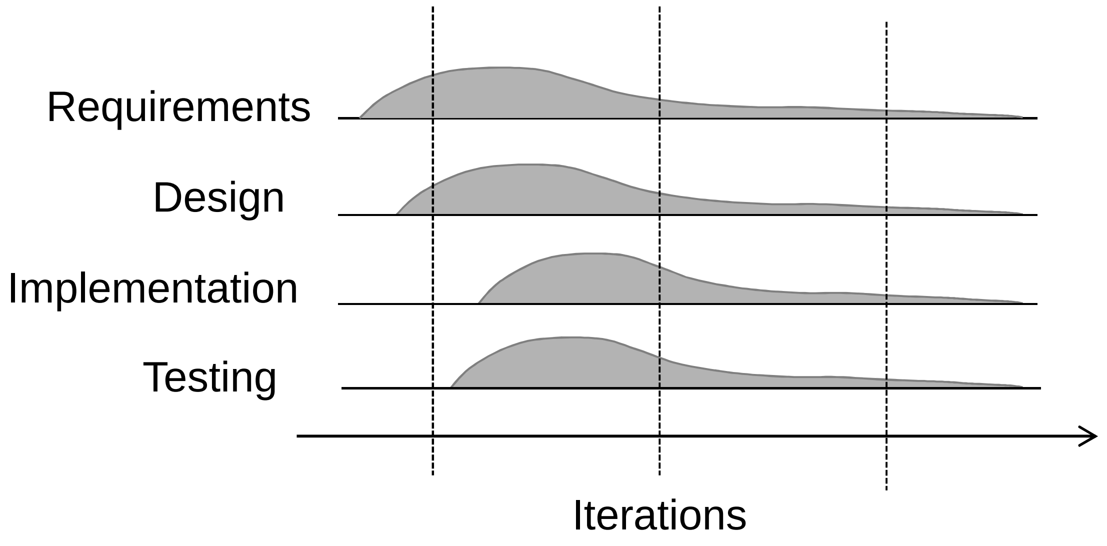
Note: Since the Team is autonomous, it has complete autonomy of changing activities without asking permission to anybody.

The SCRUM Framework



SCRUM: not just a "mini waterfall"

- Scrum is not just a "mini waterfall" but it is concurrent engineering
  - Sprints are "feature-sprints" and not "component-sprints"!
  - When explaining Scrum, start by talking about PSPI



These are concurrent activities and not phases!

Definition of "Done"

- Since every 2 weeks the released software must be "done", Team and Product Owner must agree about what this means
- What does "done" include ?
  - Coded ?
  - Tested ?
    - Functional testing ?
    - Performance testing ?
    - Stability testing ?
    - Usability testing ?
    - User acceptance ?
  - Documented ?
  - Integrated ?
  - Packaged ?

Whatever "done" means, the objective of SCRUM is to extend the definition so that all these activities are done each Sprint.

Definition of "Done" (2)

- The definition of done
  - is chosen before the first Sprint (e.g., at Release Planning)
  - is verified and updated at Review/Retrospective meetings
  - must be completely unrelated to requirements

- Put a poster on the wall that makes clear what "done" means!

"Undone work"

- If "Release done" ≠ "Sprint done", then there is some "Undone Work"
  - Official SCRUM name
  - Not related to undone tasks! Usually activities related to a Release (e.g. documentation, testing integration)
- It must be recorded in the Product Backlog
- Create a "Release Sprint" to do this work
  - Otherwise use the "Rule of 3" rule: to avoid accumulating undone work, create a "Semi-Release" Sprint to eliminate as much undone work as possible
- Should not be done by a separate Team !

The SCRUM Team

- The SCRUM Team has 3 roles:
  1. The Scrum Master
  2. The Product Owner
  3. The Team

SCRUM roles: the Product Owner

- He represents the voice of the customer
  - Responsible for the budget
  - Responsible for maximizing the Return On Investment (ROI) of the development effort
  - He chooses release content and date
  - He provides product vision and boundaries
- His job consists of:
  - Talking to customers
  - Following market trends
  - Working with the Team

SCRUM roles: the Product Owner (2)

- Single person
  - Cannot be a committee
- He constantly re-prioritizes and refines the Product Backlog
- Final arbiter of requirements questions
- Accepts or rejects each product increment
- Decides whether to continue development
- May contribute as a team member
- Can be a Team Member, but can never be the same person doing the ScrumMaster
- If the customer/business refuses to do the Product Owner, use an internal "proxy Product Owner"

Attributes of a good Product Owner

1. Available (when needed)
2. Business-savy
   - Deep understanding of business, market conditions, customers
3. Communicative
4. Decisive
   - Capability of making hard decisions
   - A good product owner doesn't reverse prior decisions without a good reason
   - Allows wrong decisions to persist until the end of the sprint; then decides if it should be changed.
5. Empowered
   - Authority to make decisions

SCRUM roles: the Team

- People who do the actual work (design, development, doc, test, etc.)
- Historically called "feature teams" to make clear that they produce features and not "steps"
- Designed to optimize flexibility and productivity
  - Cross-functional skills (e.g., it may include business analysts, domain experts, etc.)
  - Self-organizing/self-managing (no assigned roles)
  - It has autonomy about how to reach commitments
  - People who refuse to code because they are architects or designers are not good fits for Teams: everyone must chip in, even if that requires learning new skills

SCRUM roles: the Team (2)

- Size: 7 ± 2 (Product Owner and Scrum Master not included)
- Intensely collaborative
- Most successful when located in the same room
- Most successful with full-time membership (i.e., no members splitted between teams)
  - Multitasking is one of the biggest drains on team performance!
- The Team is responsible for communication and coordination with the external world (no managers needed)

In Scrum there isn't any project/product management role: all project management must be solved by a self-managing team and the Product Owner

Self-managing teams

- "There can't be individual failures, only team failures"

- Self-managing teams mean:
  – More productive people (less managers needed)
  – Less latency in responding to changes

- Cross-functional teams mean:
  – All people are just "team members"
  – Everybody must be ready to learn new skills
  – There are not roles (no functional division)

Attributes of a good Team

1. Include all needed disciplines
2. Balance technical skill levels
   – Put seniors together with juniors
3. Balance domain knowledge
4. Seek diversity
   – Homogeneuos teams reach consensus more quickly than do heterogeneuos teams, but they do so by failing to consider all options
5. Stability
   – Team members need time (i.e., years) to work well together

Better if formed by volunteering

SCRUM: origins of the name

- The word is not an acronym, but comes from the world of RUGBY

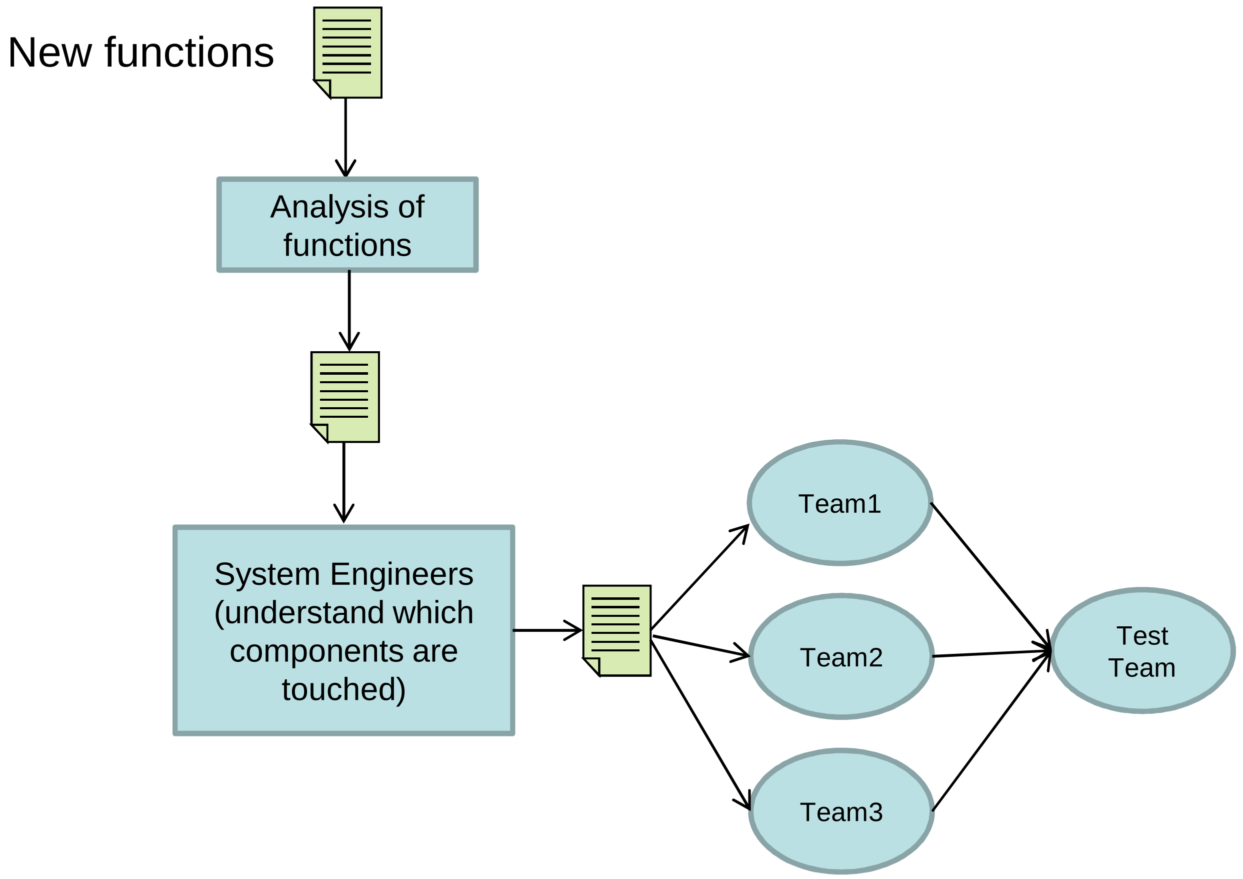
Changing environment

Self-Managing team

All team involved to move the ball together to a common goal (the goal line)

Feature teams vs component teams

- With component teams we create a waterfall:



SCRUM roles: the ScrumMaster

- Responsible for ensuring that the Scrum Team adheres to Scrum values and rules
  - Facilitates the Scrum process
- He is a teacher:
  - He helps the team in learning and applying Scrum
  - He teaches Team, Product Owner and organization
- Removes "impediments" to the ability of the team to deliver the goal
- Shields the team from external interference and distractions
- Enforces timeboxes

SCRUM roles: the ScrumMaster (2)

- Has no management authority over the team
  - Understand power relationships: he can be fired by the Team!
- He provides guidance, not answers
- He is not the leader of the team: a good ScrumMaster does not make decisions for the team
  - He doesn't suggest how to accomplish a goal
- No authority over team members, but only over the process
- If needed, this role can be played by a team member
  - However, this is not suggested
  - Better if he does ScrumMaster full-time, serving more than one Team (especially novice ScrumMasters)

Attributes of a good ScrumMaster

1. Responsible
   - Willing to assume responsibility
2. Humble
   - Not in it for his ego
   - He recognizes the value in all team members
3. Collaborative
   - Ensures a collaborative atmosphere within the team
4. Committed
5. Influential
6. Knowledgeable
   - With technical or business know-how
7. Able of dealing with feelings and emotional problems

What about ex Project Managers ?

- Project Managers are the worst choice to select a person to act as ScrumMaster
- It's better if Project Managers become Product Owners

Other roles

- Stakeholders:
  - Stakeholders, customers and vendors

- Managers:
  - People who will set up the environment for the product development organizations

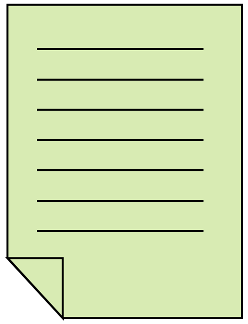
Nobody can tell the Team how to do its work!

SCRUM Artifacts

- SCRUM has 4 artifacts:
  - The Product Backlog
  - The Sprint Backlog
  - The Release Burndown
  - The Sprint Burndown

The Product Backlog

- Also known just as ``Backlog''
- List of high level items to be implemented
- It is never complete
  - It is a "living document": it constantly changes
  - Its content can come from anywhere: users, customers, sales, marketing, customer service, and engineering can all submit items to the backlog
- Keep one list for multiple teams



The Product Backlog: Items

- Items specify the *what* more than the *how* of a customer-centric feature
  - They are just called "items" because it is a generic term, which may include requirements, undone work or major improvements that involve a business decision for investment
  - Can be written in any form (use cases, user stories, plain language)
- Each item is characterized by:
  - Description
  - Estimate: effort estimation (made by the Team)
  - Priority: business value (chosen by the Product Owner)
- Items must be indepentend each other!
- The Product Backlog is sorted in order of priority
  - The higher the priority, the more urgent it is

About "User Stories"

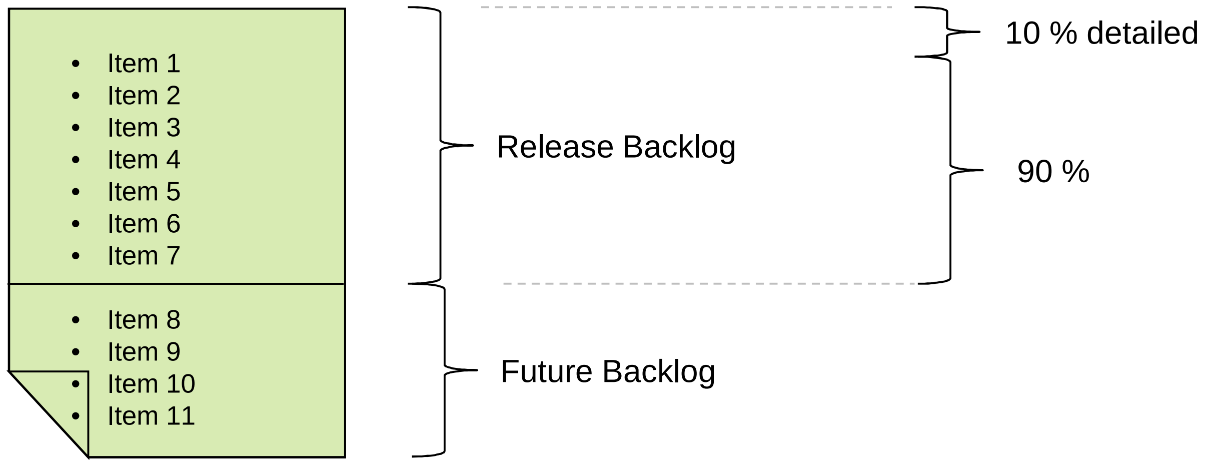
- User Stories is not a special format for writing requirements
- It means to write requirements by conversation
  - It is a behavior
- To avoid misunderstandings, don't use this term

Tools for Product Backlog

- Avoid "Agile Progect Management" tools (e.g., Rally or VersionOne) especially when beginning
- Use a SpreadSheet
  - Better if on-line (e.g. GoogleDocs)
    - You can use Craig's template, if needed
- Use a whiteboard
  - See http://magicwhiteboard.co.uk to get a whiteboard everywhere

Product Backlog structure

- To minimize rework, only the highest priority items need to be detailed out
- At least 10% of Product Backlog broken in small requirements fully analized:



Good Product Backlog

- DEEP:
  - Detailed appropriately
  - Estimated
  - Emergent
  - Prioritized

Product Backlog: Done and WiP

- Organize the Product Backlog in sections:
  - To Do: items that have not been done
  - WiP (Work In Progress): items that have been partially implemented, but that don't meet yet the definition of "done"
  - Done

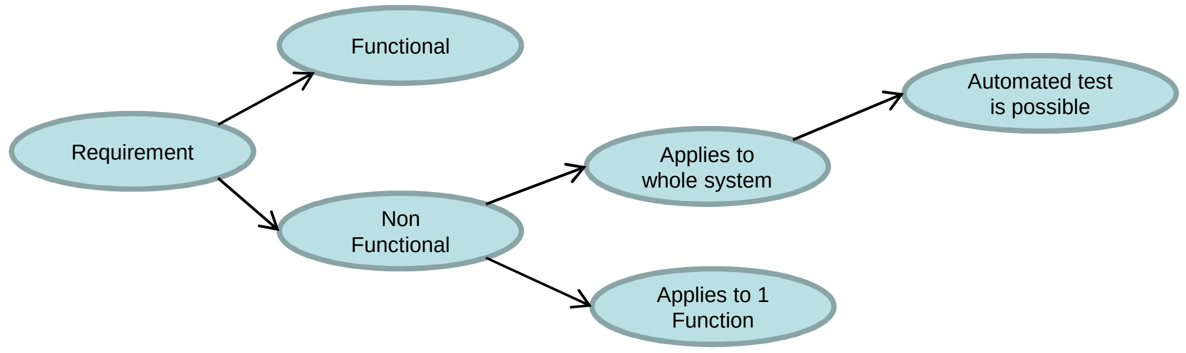
- Note: according to Lean Thinking, WiP is a waste, because it is not shippable

Release Planning meeting

- This meeting is optional
- Purposes:
  - Establish a plan and goals of the Release
  - Create the Product backlog
    - This document is created only once for each project
  - Identify:
    - Highest priority Product backlog
    - Major risks
    - Overall features and functionality that the release will contain
    - Probable delivery date and cost that should hold if nothing changes

Release Planning meeting (2): example

1. Vision workshop: choose product vision and Product Owner
2. High level requirements: suggestion:



3. Estimation
4. 10% Low level requirements
5. Second cycle of estimation
6. Create the physical Product Backlog

Product Backlog estimation

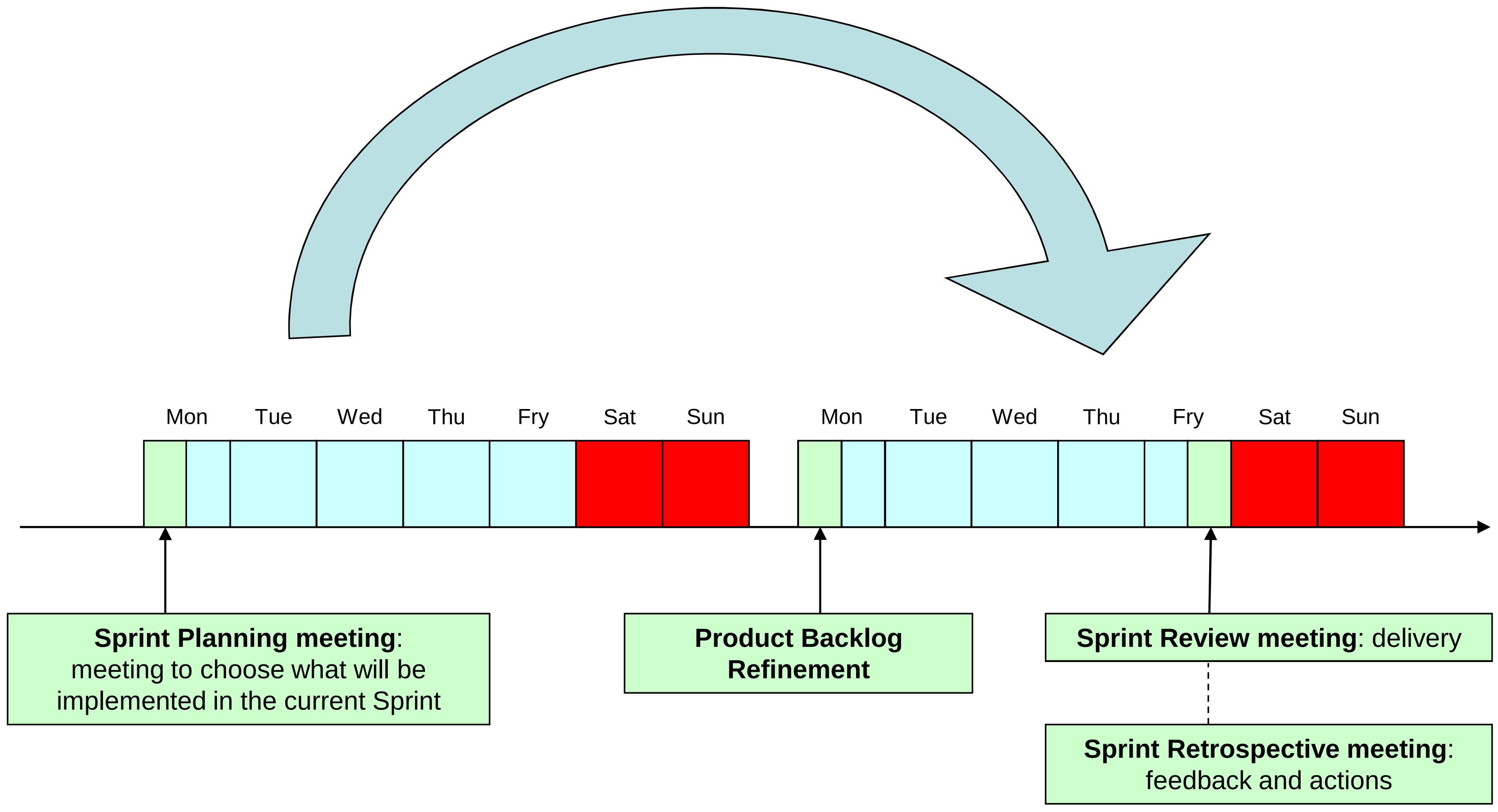
- Estimates of items happen
  - At Release planning meeting
  - At Product Backlog Refinement meeting
    - Usually done after half Sprint
    - The whole Product Backlog is re-estimated
    - Usually takes 5%-10% of time of a Sprint
    - Look forward: think about 2 or 3 Sprints forward

- Don't change estimates during Sprint Planning meeting, because it screwes up the computation of Velocity
- At both meetings, "Planning Poker" may be used for estimation

Product Backlog estimation (2)

- You may proceed by estimating only relative efforts:
  - Humans are more good with relative estimations than with absolute estimations
  - First step: identify the shorter item, and give effort "1"
  - Do not estimate items with more than 1 order of magnitude, because they have a high variability and cannot be estimated
    - Split big items in estimable items

Example of 2-week Sprint



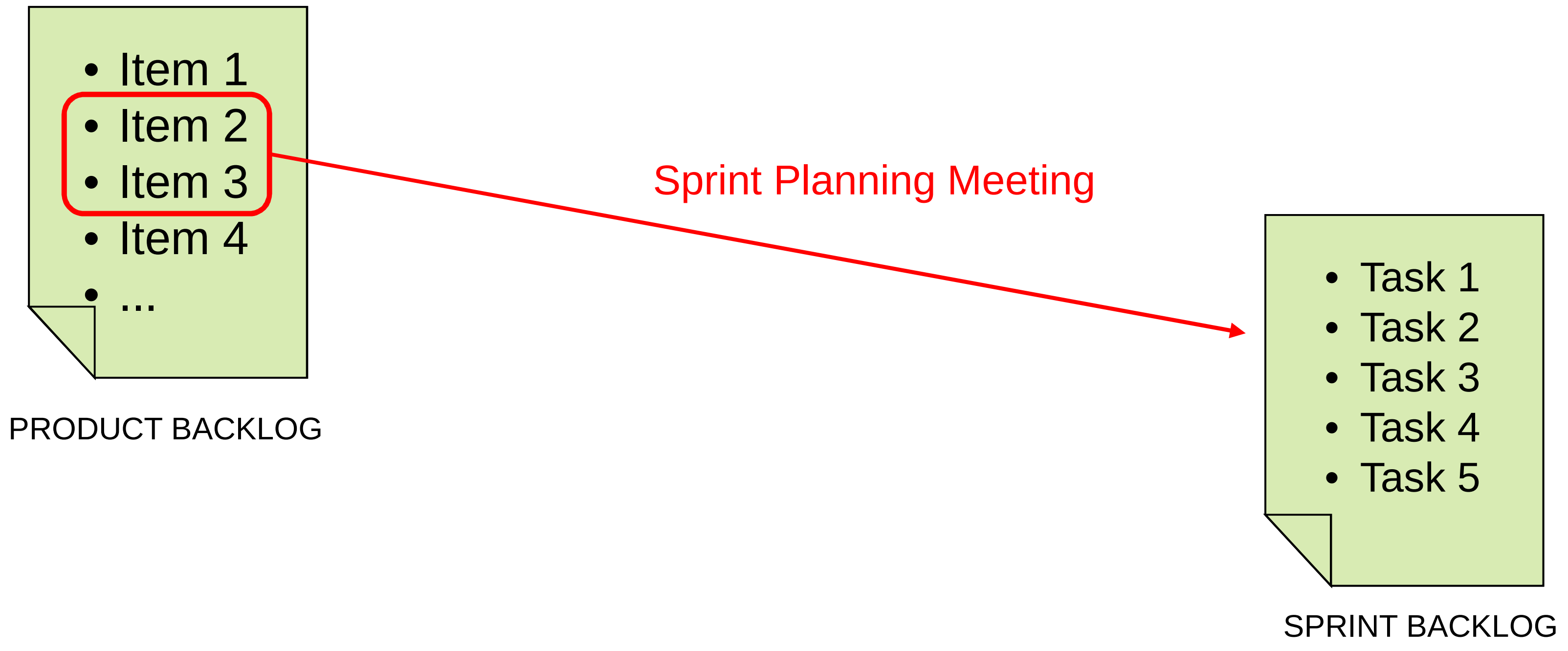
Mon
Tue
Wed
Thu
Fry
Sat
Sun
Mon
Tue
Wed
Thu
Fry
Sat
Sun
Sprint Planning meeting: meeting to choose what will be implemented in the current Sprint
Product Backlog Refinement
Sprint Review meeting: delivery
Sprint Retrospective meeting: feedback and actions

Sprint Planning Meeting

- At the beginning of each sprint, a Sprint Planning meeting is held
- In this meeting, the work to be done in the sprint is selected
- The Product Owner is responsible for declaring which items are the most important for the business
- The team is responsible for selecting the amount of work needed to implement each item

Sprint Planning Meeting: Sprint Backlog

- Features from the Product backlog are put in a document called Sprint Backlog



Important: during the Sprint, no one is allowed to change the Spring Backlog!

Sprint Planning Meeting (2)

- Features are broken down into tasks
- Tasks specify *how* to achieve the item's *what*
- Tasks are normally estimated between 4 and 16 hours of work
- Tasks are never assigned
  - They are signed up for by the team members as needed, according to the team member skills
  - It is better to only volunteer for one task at a time, when it is time to pick up a new task
- Maximum length: 8 hours for a 4-week Sprint (reduced proportionally for a shorter Sprint)

Sprint Planning Meeting: Part 1

It consists of 2 parts:

PART 1:

- Attendees: Product Owner, Scrum Master and Team
- Activities:
  - Review the high-priority items in the Product Backlog
  - Update estimates

Sprint Planning Meeting: Part 2

PART 2:

- Attendees: ScrumMaster and Team
  - Product Owner must be reachable to clarify items and help making trade-offs
  - Other people may be invited to provide domain advice
- Activities:
  - The Team selects the items from the Product Backlog it commits to complete by the end of the Sprint
    - The amount of work is decided by the Team, rather than being assigned by the Product Owner
    - The Tean also breaks down the items into tasks (length: 2-16 person/hours) and decides how to implement each task

Sprint Planning Meeting: commitment

- Once the Team makes its commitment, any additions or changes must be deferred until the next Sprint. If an external circumstance appears that significantly changes priorities, the Product Owner can terminate the Sprint.

- This commitment has nothing to do with traditional contract negotiation: if a problem happens, there is nothing you can do to accomplish a goal. Traditional contract negotiation is just "belief in magic"

Date of release delivery

- How to compute the date in which the software will be released?
- Can be done after one Sprint, evaluating the Velocity
- If you need information earlier:
  - Just after Release Planning, do a "Pretend Sprint Planning meeting", and see the amount of work committed by the Team.
  - This value can be used as Velocity to compute the date.
  - It is only a speculation: real values of Velocity will be available only after the first Sprint.

Sprint Review Meeting

- Held at the end of the Sprint
- Review the work completed according to "done" definition
- Completed work:
  - Present the completed work (i.e. the "live" software) to Product Owner, stakeholders, customers, executives, etc.
  - It's a demonstration, not a report
  - The demo usually runs in a sandbox
  - Make Product Owner and people <u>try</u> the software to collect feedback
  - Management comes to see what the team was able to build with the resources it has been given
  - Customers come to see if they like what the team has built

Sprint Review Meeting (2)

- Uncompleted work:
  - Incomplete work cannot be demonstrated
  - Untested software is considered not done (because it is not shippable)
  - Avoid "smoke and mirror" demos
    - Not demo features that aren't truly "done"
  - If something went wrong, the team may be given more training, the team may be recomposed, or more tools may be brought in.
  - Incomplete items are returned to the Product backlog and ranked according to the Product Owner's revised priorities

Sprint Review Meeting (3)

- People in charge of accepting/rejecting the software must attend this meeting

- No one should prepare extensively for the review meeting.
  - To enforce this rule, PowerPoint presentations are forbidden.
  - If the team feels that it has to spend more than one hour preparing for the meeting, it usually has less to show for the Sprint than it had hoped, and it is trying to obscure this fact with fancy presentation.

Sprint Review Meeting (4)

- The meeting is very informal: what matters is the product the team has been able to create.
- It is a working meeting: questions, observations, discussions and suggestions are allowed and encouraged.
- The Scrum Master is responsible for coordinating and conducting the meeting. He:
  - meets with the team to establish the agenda and discuss how the Sprint results will be presented and by whom.
  - item sends all attendants a reminder a week before the meeting, confirming the time, date, location and agenda
- Maximum length: 4 hours

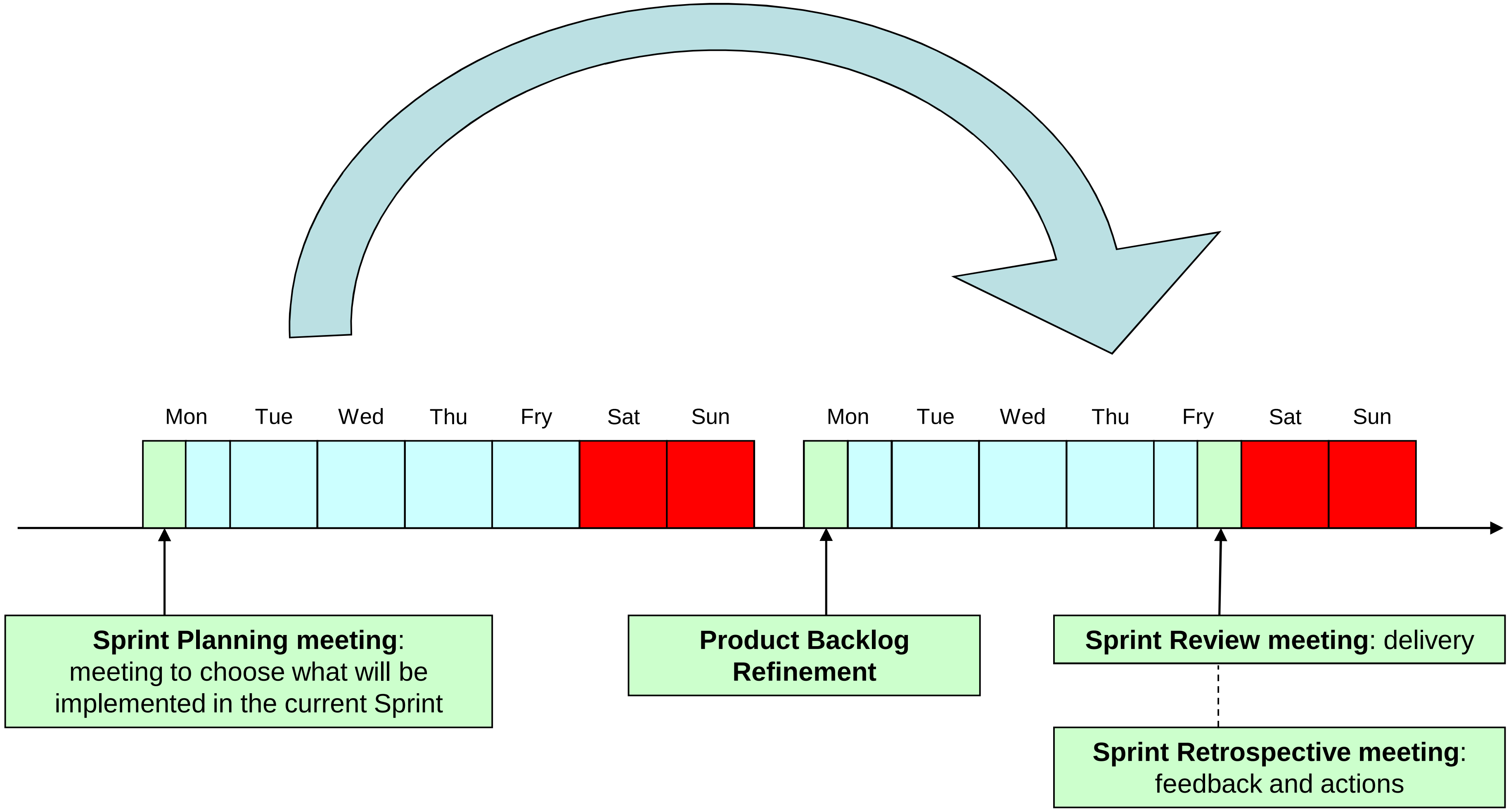
Sprint Review Meeting (5)

- The Sprint Review meeting includes at least the following elements:
  1. The Product Owner identifies what has been done and what hasn't been done
  2. The Team discusses what went well, what problems it ran into and how they have been solved
  3. The team demonstrates the work done and answers questions
  4. The Product Owner projects likely completion dates with various velocity assumptions

Sprint Review Meeting (6)

- It's not just a demo:
  - Review of the product and of the process
  - Review the definition of "done"
  - The Product Owner should share business information

Example of 2-week Sprint



Sprint Planning meeting: meeting to choose what will be implemented in the current Sprint

Product Backlog Refinement

Sprint Review meeting: delivery

Sprint Retrospective meeting: feedback and actions

Other meetings: Daily Scrum

- Daily meeting, also known as ``*Daily Standup*''
- Public event: all are welcome, but only Scrum Master and the team may speak
  - The Product Owner should attend
  - It is generally recommended *not* to have managers or others in positions of perceived authority attend the Daily Scrum
- Each team member answers 3 questions:
  1. What have you done since yesterday?
  2. What are you planning to do today?
  3. What impediments do you face ?
- You can add a further question: "What have I learnt ?"

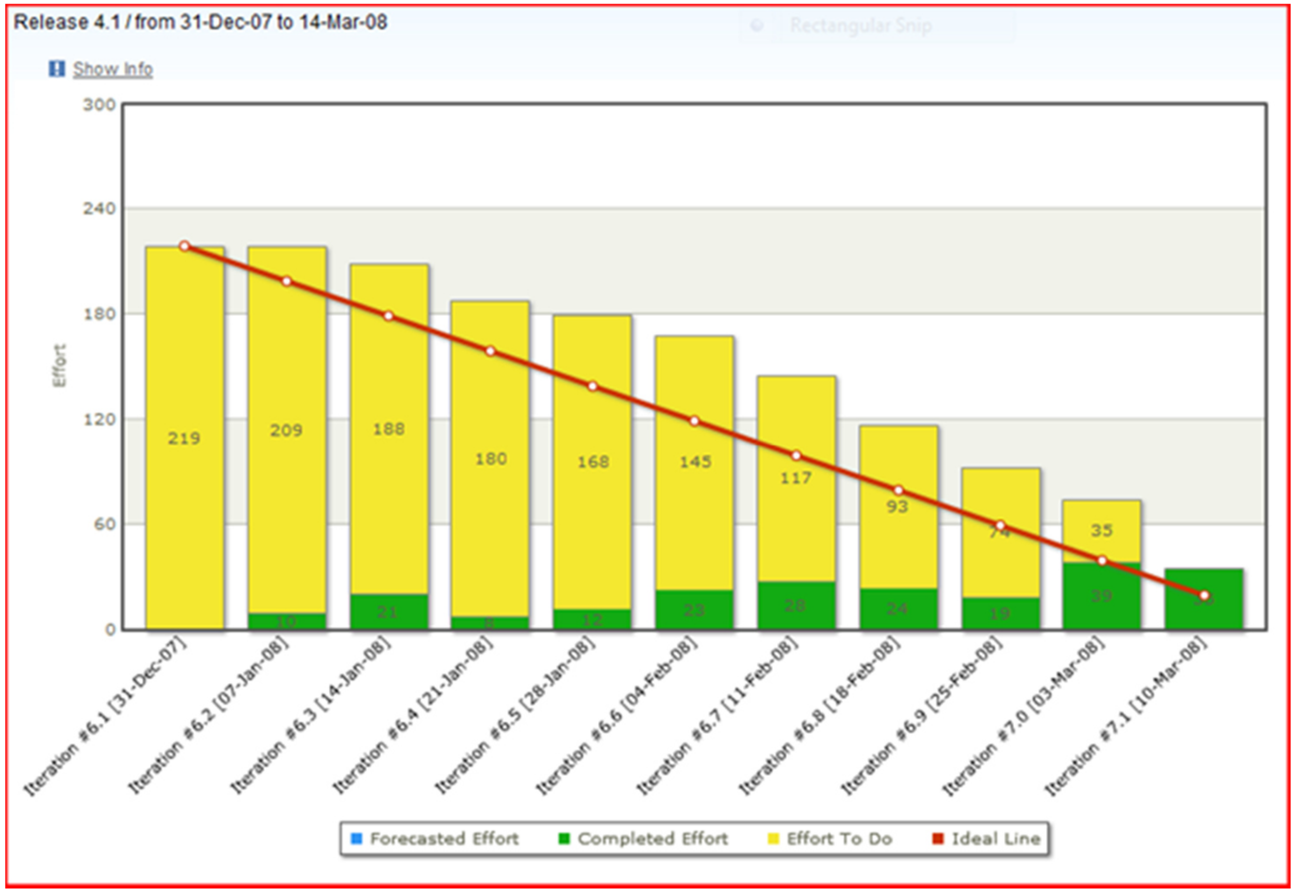
Other meetings: Daily Scrum (2)

- People report to each other, not to ScrumMaster/managers
- If any discussion is needed other than the status provided by answering the three questions, a follow-up meeting may be requested

- The meeting starts precisely on time
- It should happen at the same location and same time every day
- Maximum length: 15 minutes
- At the end of the meeting the Burndown chart is updated (better if updated by a rotating Team member)

Burndown Charts

- Publicly displayed chart showing estimated effort across time



Burndown Charts (2)

Two Charts:

- Sprint Burndown chart:
  - Indicates total remaining task hours within a Sprint
  - Re-estimated daily, thus may go up before going down
  - Updated every day by the Scrum Master
  - It shouldn't be used if it becomes an impediment to team self-organization
  - It gives a simple view of the sprint progress
- Product Burndown Chart:
  - Generated and updated by the Product Owner
  - Depicts forecasts
  - Units of time are usually Sprints

Burndown Charts (3)

- Whenever possible, draw burndown charts on a big sheet of paper displayed in the Team's work area
  - Teams are more likely to see a big, visible chart than they are to look at an Excel chart or a tool

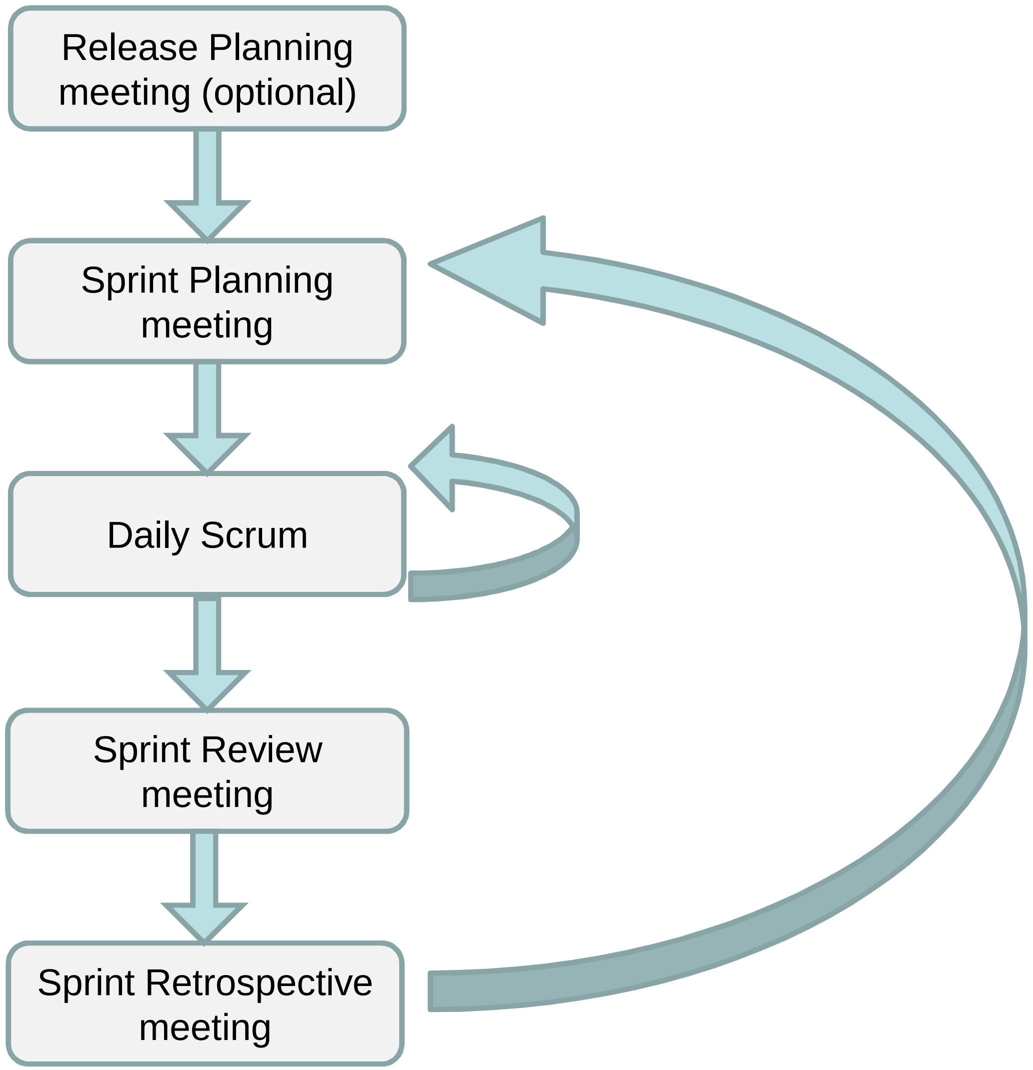
Other meetings: Sprint Retrospective

- It is important not only to generate the feedback, but to use it
  - ScrumMaster must emphasize this part
  - In SCRUM, improvement has higher priority than features
- Sprint retrospectives are officially private meetings
  - It's up to the Team to choose if other people can attend
- The Team reflects on the past Sprint
  - Goal: make continuous process improvements
  - ScrumMaster should play a background role, only facilitating the Team by helping focusing
- Avoid superficial retrospectives
- Length: 45 minutes

Other meetings: Sprint Retrospective (2)

- 3 steps of retrospective:
  1. Analysis of previous work:
     - What went well and what went bad during the last Sprint?
     - You can use a two-column spreadhseet
  2. Analysis of the current status
     - What could be improved in the next Sprint?
  3. Design new actions ("experiments")
     - Identify actions, and really do them
     - Actions will become tasks in the next Sprint Backlog
     - Actions may include:
       – Team composition
       – Meeting arrangements
       – Tools, methods of communication
       – Definition of "done"

Summary of Scrum meetings


Release Planning meeting (optional)
Sprint Planning meeting
Daily Scrum
Sprint Review meeting
Sprint Retrospective meeting

Scrum: some warnings

1. Transition to Scrum without support from the top of the company creates a resistance that cannot be overcome from below
2. To reduce resistance, keep the change process nameless among employees
3. Apply Scrum as proposed for at least 3 Sprints
   – *"If you follow 80% of the process, you get 20% of the benefits"* (K.Beck)
4. You can add engineering practices (e.g., Extreme Programming, pair programming, refactoring, automated tests)
5. Don't start a new project until it can be fully staffed
   – All disciplines must be represented from the beginning

Scrum: some warnings (2)

6. A company is never ``done'' becoming agile: there are always improvements to be made
   – None of the agile processes described by their originators is perfect for your organization. It needs to be tailored and adapted.
7. Do not give performance-based bonuses because they reduce transparency and encourage cheating
8. Do not reward single individuals, but only the whole team
   – A bonus can be a free Sprint to work on whatever they want
9. Do not change ScrumMaster mid-project unless requested by the Team

Scrum: some warnings (3)

10. Do not use external (i.e., contract) ScrumMaster over the long term
11. Scrum shouldn't be used as an excuse to be sloppy
    – Do requirement analysis, effort estimation and design!
12. Don't make good programmers become managers
    – You loose a good programmer
    – Good programmers are not necessarily good managers

Three legs of SCRUM: T, I, A

1. Transparency
2. Inspection
3. Adaption

Three legs of SCRUM: T, I, A (2)

1. Transparency
   - It ensures that aspects of the process that affect the outcome are visible to those managing the outcomes
2. Inspection
   - The various aspects of the process must be inspected frequently enough that unacceptable variances in the process can be detected
3. Adaption
   - When inspection determines that one or more aspects of the process are outside acceptable limits, an adjustment must be made as quicly as possible to minimize further deviation

Inspection and Adaption are achieved through Daily Scrum, Sprint Review, Sprint Planning and Sprint Retrospective meetings.

A new relationship with the customer

- Scrum implies a transparent relationship with the customer
- It keeps everything about a project visible to anyone
  - The customer sees the product costantly
  - This way, it can provide early exposure to possible schedule slips
- When accepting a project, be sure to communicate risks to the customer. Courage is needed!
  - Make <u>him</u> make the decision
  - Just saying "yes" can harm everyone
  - Give him probabilistic answers (about the expected percentage of success) + risk analysis

SCRUM: References

To start:

- SCRUM Alliance: http://scrumalliance.org/
- Wikipedia: http://en.wikipedia.org/wiki/Scrum_(development)
- SCRUM in under 10 minutes: http://www.youtube.com/watch?v=Q5k7a9YEoUI
- SCRUM Reference Card: http://scrumreferencecard.com
- SCRUM Primer: http://scrumtraininginstitute.com/library
- K.Schwaber, "*Agile Software Development with Scrum*"

For advanced users:

- M.Cohn, *"Succeeding with Agile – Software Development Using Scrum"*

Lean Thinking

- Lean Thinking is the Engish name for the "Toyota way"
  - Book *"The machine that changed the world"*
  - Other resource: *LeanPrimer*
- Principles:
  1. Value stream
     Perform only activities that add value for the customer, without pauses or impediments
  2. "Whatch the baton, not the runner"
     Do System-wide optimizations and not local optimizations
     Example: don't give the work to expert developers who easily become bottlenecks; make them only teach

Lean Thinking (2)

- ...

3. Persons are not machines
   Don't call them "resources"; don't threat them as machines
4. "Go see & Gemba"
   Managers should avoid looking at numbers and reports and formal communications. They must be physically with the workers in the place where value is added.
   In software, Gemba is the code, because other things are just "bla bla".
5. "Stop and fix"
   Fix mistakes as soon as possible (e.g., Test Driven Development)

Lean Thinking (3)

- ...

6. Physical visual management and open spaces
   Open-space: no separate offices or cubes
   Visual management: to show the hidden waste of money of Work In Progress hidden in the computer, to start the "visceral response" when you actually see the waste
7. Perfection challenge
   Be never satisfied with the status quo
8. "Have eyes for value and eyes for waste"
   Remove the need to fix: "Build Quality In"

Lean Thinking: 10 sources of waste

1. Overproduction of features and inventory
2. Delays
3. Handoff (e.g., instead of automatic procedures)
4. Extra-processing and extra-processes
5. Partially done work (is not a PSPI)
6. Interrupt task switching
7. Testing and corrections at the end
8. Not using people's full potential
9. Information scattered (e.g., multiple documents)
   Do everything online (e.g., Google docs, wiki)
10. Wishful thinking (in meeting goals promised)