



**北京理工大学**  
BEIJING INSTITUTE OF TECHNOLOGY

# Advanced Software Engineering

---

**Lecture 7: Agile Development**

*by*

**Prof. Harold Liu**

# Content

---

- Agile Development
- XP
- Scrum

# Agile Process

---

- Agile Process represents a category of software development lifecycles/processes
- Common features:
  - Customers and developments form a close team, because
    - Customers cannot clearly and fully written down all requirement and functionalities
    - Changes always happen
  - Working software is the FINAL GOAL of any project, and thus all intermediate releases needs careful testing, reviews, and evaluations
  - Employ iterative and incremental ways for development
  - Employ intensive review meetings
  - Strict planning and executions
  - Team work
  - Team can make changes/decisions of the taken responsibilities<sub>3</sub>

# What is Agile Development?

---

- ❑ Human-centric, iterative, and incremental software development method
- ❑ Project is composed of a series of sub-projects
- ❑ Each sub-project needs to be carefully tested before integration

# Core Value

---

Individuals and interactions

over

Process and tools

Working software

over

Comprehensive documentation

Customer collaboration

over

Contract negotiation

Responding to change

over

Following a plan

# Principles

---

1. Rapid and frequent software release to meet customer requirements
2. Release from ~weeks to ~months
3. Correct software is important
4. Proactive to accept changes
5. Customers and developers together
6. Trust to all members
7. Fact-to-face communications
8. Best software architecture and design comes from a self-organizing team; allow all members to raise questions and make comments
9. Constantly improving the design and coding
10. Encourage normal working hours, NO OT!
11. Simple design

# Abbay the S.O.L.I.D. Design Principle

---

- ❑ SRP: Single Responsibility Principle
- ❑ OCP: Open — Closed Principle
- ❑ LSP: Liskov Substitution Principle
- ❑ DIP: Dependency Inversion Principle
- ❑ ISP: Interface Separate Principle

# Content

---

1. XP

2. Scrum



# Outline

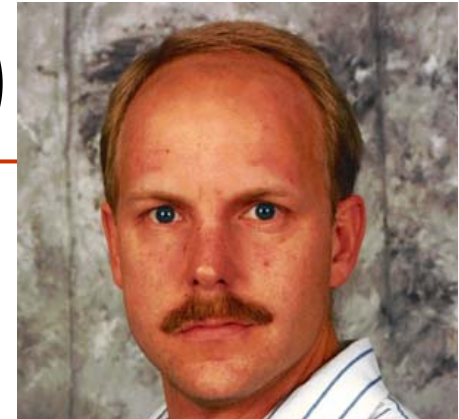
---

- Traditional life cycle vs. XP
- XP motto: “embrace change”
  - How does this attitude compare with that implicit with traditional waterfall software life cycle?
- XP values
- XP practices
- Pair programming
- An XP development road map



# eXtreme Programming (XP)

- Developed by
  - Kent Beck (earlier helped create CRC cards):
  - XP is “a light-weight methodology for small to medium-sized teams developing software in the face of vague or rapidly changing requirements.”
- Alternative to “heavy-weight” software development models (which tend to avoid change and customers)
  - "Extreme Programming turns the conventional software process sideways. Rather than planning, analyzing, and designing for the far-flung future, XP programmers do all of these activities a little at a time throughout development."  
-- *IEEE Computer* , October 1999



so extreme he never smiles?!?

# Successes in Industry

---

- Chrysler Comprehensive Compensation system
  - After finding significant, initial development problems, Beck and Jeffries restarted this development using XP principles
  - The payroll system pays some 10,000 monthly-paid employees and has 2,000 classes and 30,000 methods, went into production almost on schedule, and is still operational today (Anderson 1998)
- Ford Motor Company VCAPS system
  - Spent four unsuccessful years trying to build the Vehicle Cost and Profit System using traditional waterfall methodology
  - XP developers successfully implemented that system in less than a year using Extreme Programming (Beck 2000).

# Embrace change

---

- In traditional software life cycle models, the cost of changing a program rises exponentially over time
  - Why would it cost more to make large changes during testing than during requirements specification?
- A key assumption of XP is that the cost of changing a program can be hold mostly constant over time
- Hence XP is a lightweight (agile) process:
  - Instead of lots of documentation nailing down what customer wants up front, XP emphasizes plenty of feedback
  - Embrace change: iterate often, design and redesign, code and test frequently, keep the customer involved
  - Deliver software to the customer in short (2 week) iterations
  - Eliminate defects early, thus reducing costs

# Four Core Values of XP

---

- Communication
- Simplicity
- Feedback
- Courage

# Communication

---

- XP emphasizes value of communication in many of its practices:
  - On-site customer, user stories, pair programming, collective ownership (popular with open source developers), daily standup meetings, etc.
- XP employs a *coach* whose job is noticing when people aren't communicating and reintroduce them

# Simplicity

---

- ❑ "Do the simplest thing that could possibly work"  
(DTSTTCPW) principle
  - Elsewhere known as KISS
- ❑ A coach may say DTSTTCPW when he sees an XP developer doing something needlessly complicated
- ❑ YAGNI principle ("You ain't gonna need it")
  
- ❑ How do simplicity and communication support each other?

# Feedback

---

- ❑ Feedback at different time scales
- ❑ Unit tests tell programmers status of the system
- ❑ When customers write new *user stories*, programmers estimate time required to deliver changes
- ❑ Programmers produce new releases every 2-3 weeks for customers to review



# Courage

---

- ❑ The courage to communicate and accept feedback
- ❑ The courage to throw code away (prototypes)
- ❑ The courage to refactor the architecture of a system



# 12 XP Practices

---

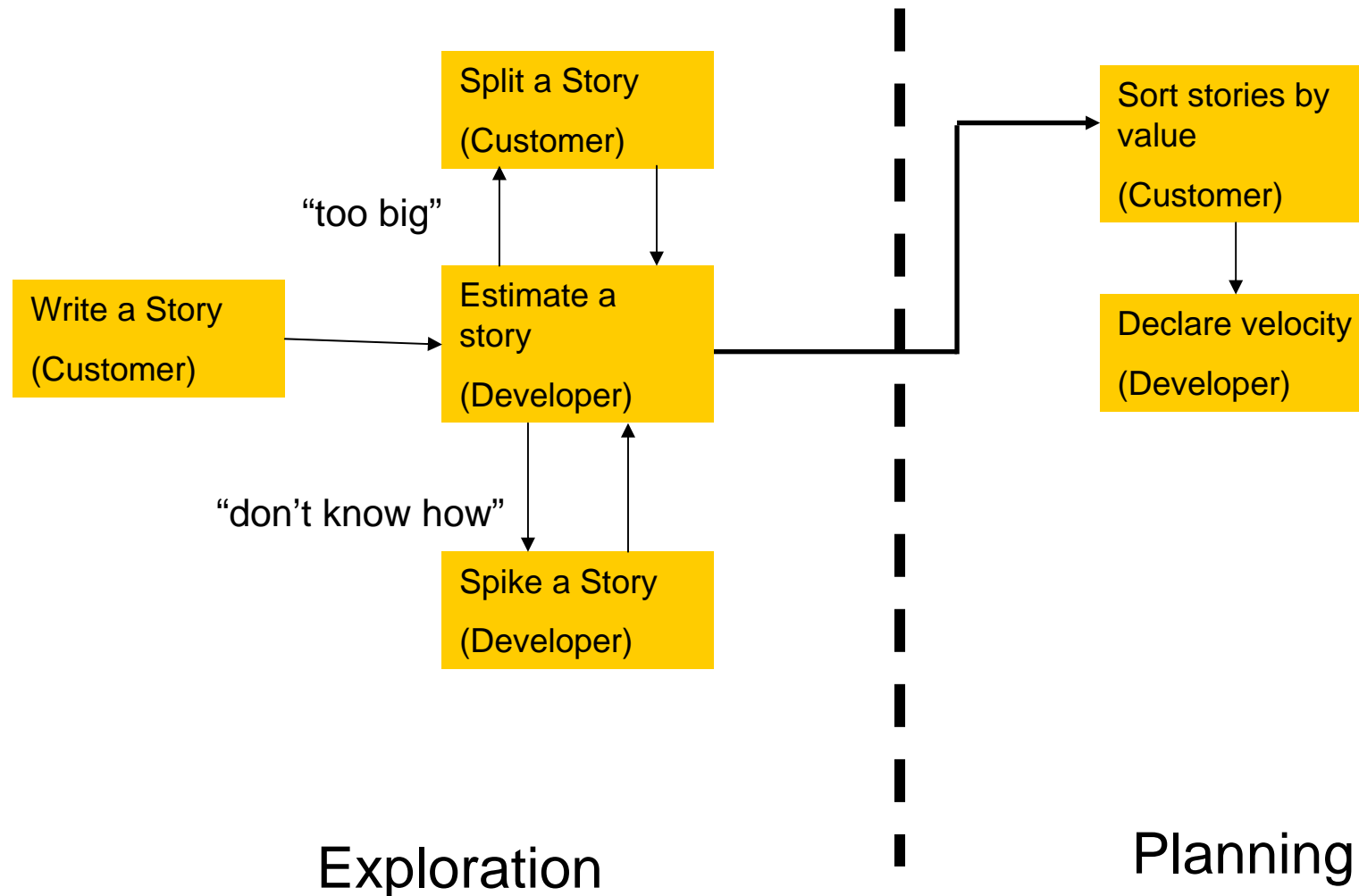
- The Planning Game
- Small Releases
- Metaphor
- Simple Design
- Test-driven development
- Refactoring
- Pair Programming
- Collective Ownership
- Continuous Integration
- 40-Hours a Week
- On-Site Customer
- Coding Standards

# Planning Game

---

- Customer comes up with a list of desired features for the system
- Each feature is written out as a **user story**
  - Typically written in 2-3 sentences on 4x6 story cards
- Developers estimate how much effort each story will take, and how much effort the team can produce in a given time interval (iteration)
- **Project velocity** = how many days can be committed to a project per week
- Given developer estimates and project velocity, the customer prioritizes which stories to implement
  - Why let the customer (rather than developer) set the priorities?

# Planning Game



# Small and simple

---

## □ Small releases

- Start with the smallest useful feature set
- Release early and often, adding a few features each time
- Releases can be date driven or user story driven

## □ Simple design

- Always use the simplest possible design that gets the job done
- The requirements will change tomorrow, so only do what's needed to meet today's requirements (remember, YAGNI)



# Test-Driven Development (TDD)

---

- ❑ **Test first:** before adding a feature, write a test for it!
  - If code has no automated test case, it is assumed it does not work
- ❑ When the complete test suite passes 100%, the feature is accepted
- ❑ Tests come in two basic flavors...
- ❑ **Unit Tests** automate testing of functionality as developers write it
  - Each UT typically tests only a single class, or a small cluster of classes
  - UTs typically use a framework, such as JUnit (xUnit)
  - Experiments show that TDD reduces debugging time
  - Increases confidence that new features work, and work with everything
  - If a bug is discovered during development, add a test case to make sure it doesn't come back!
- ❑ **Acceptance Tests (or Functional Tests)** are specified by the customer to test that the overall system is functioning as specified
  - When all acceptance tests pass, that user story is considered complete
  - Could be a script of user interface actions and expected results
  - Ideally acceptance tests should be automated, either using a unit testing framework, or a separate acceptance testing framework

# Sustainable Pace

---

- ❑ Coding is a marathon, not a sprint.
- ❑ Team works 40 hours a week - MAXIMUM!
- ❑ Tired people aren't productive



# Pair programming

- ❑ Two programmers work together at one machine
- ❑ **Driver** enters code, while **navigator** critiques it
- ❑ Periodically switch roles
- ❑ Research results:
  - Pair programming increases productivity
  - Higher quality code (15% fewer defects) in about half the time (58%)
  - Williams, L., Kessler, R., Cunningham, W., & Jeffries, R. Strengthening the case for pair programming. *IEEE Software*, 17(3), July/August 2000
  - Requires proximity in lab or work environment



ijustrealized.com



# More XP practices

---

## □ Refactoring

- Refactor out any duplicate code generated in a coding session
- You can do this with confidence that you didn't break anything because you have the tests

## □ Collective code ownership

- No single person "owns" a module
- Any developer can work on any part of the code base at any time

## □ Continuous integration

- All changes are integrated into the code base at least daily
- Tests have to run 100% both before and after integration

# More XP Practices

---

- On-site customer
  - Development team has continuous access to a real live customer, that is, someone who will actually be using the system, or a proxy
- Coding standards
  - Everyone codes to the same standards
  - Ideally, you shouldn't be able to tell by looking at it who on the team has touched a specific piece of code

# Daily Standup Meeting

---

- Goal: Identify items to be accomplished for the day and raise issues
- Everyone attends, including the customer
- Not a discussion forum
- Take discussions offline
- Everyone gets to speak
- 15 minutes



# Kindergarten lessons

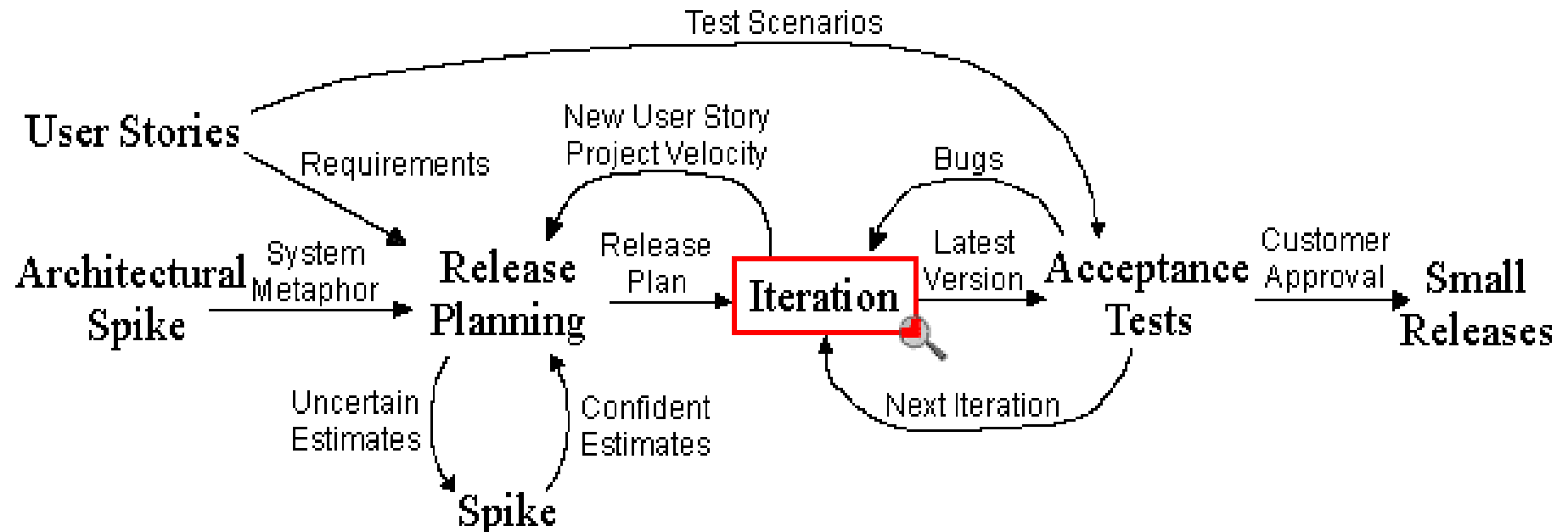
---

- Williams, L. and Kessler, R., “All I Really Need to Know about Pair Programming I Learned In Kindergarten,” *Communications of the ACM* (May 2000)
  - *Share everything.* (Collective code ownership)
  - *Play fair.* (Pair programming—navigator must not be passive)
  - *Don't hit people.* (Give and receive feedback. Stay on track.)
  - *Clean up your own mess.* (Unit testing.)
  - *Flush.* (Test-driven development, refactoring.)
  - *Take a nap every afternoon.* (40-hour week.)
  - *Be aware of wonder.* (Ego-less programming, metaphor.)

# XP practices — A Road Map

(from [www.extremeprogramming.org](http://www.extremeprogramming.org))

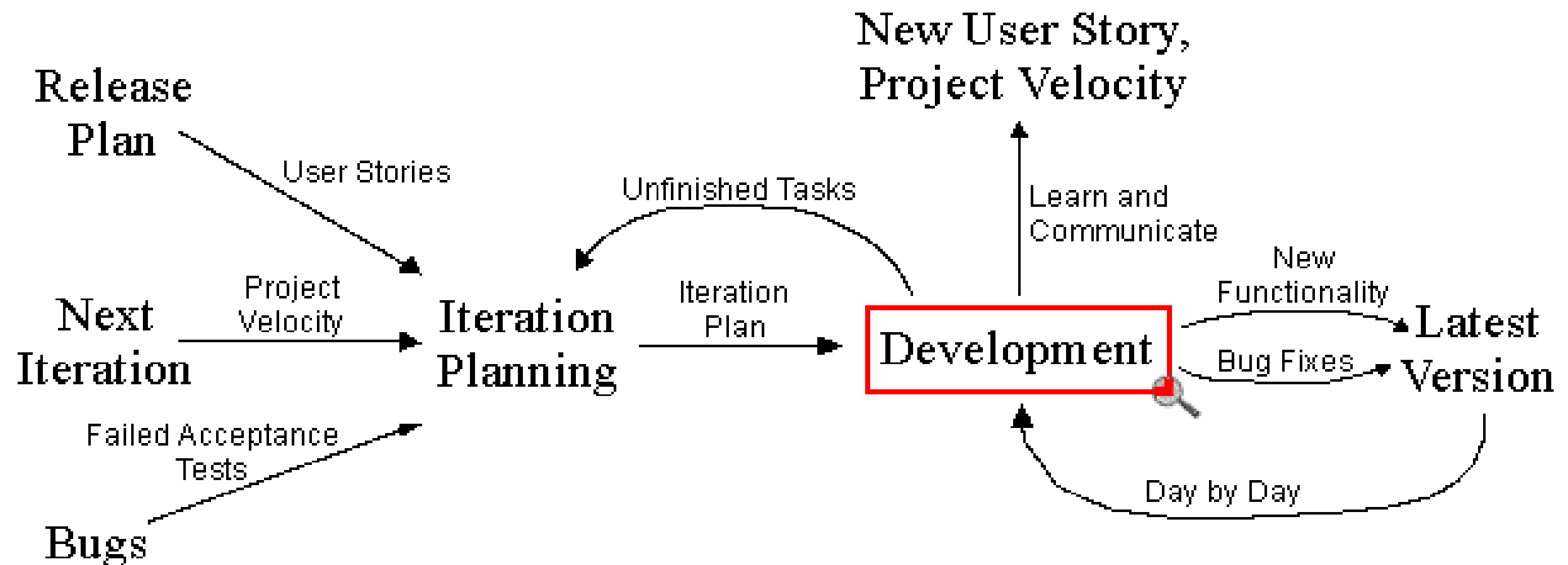
## Extreme Programming Project



# XP emphasizes iteration

## Iteration

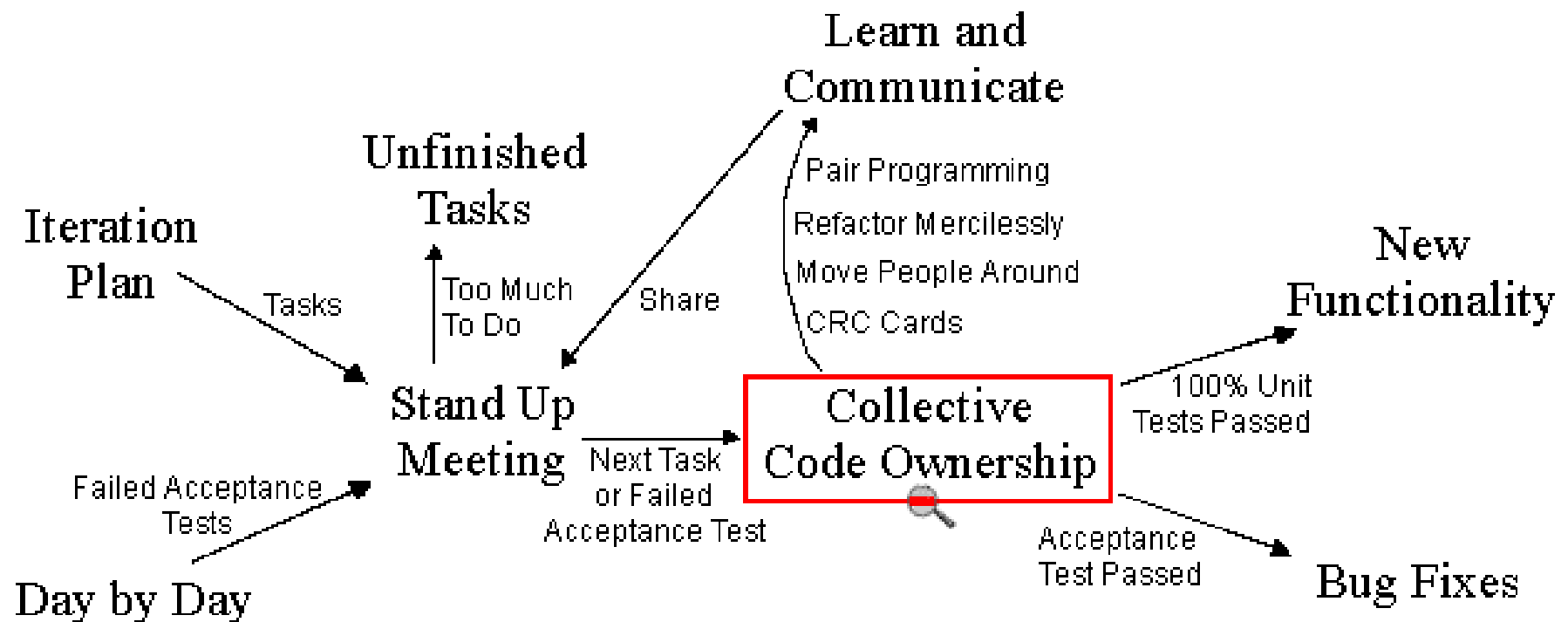
 Zoom Out



# XP emphasizes communication

## Development

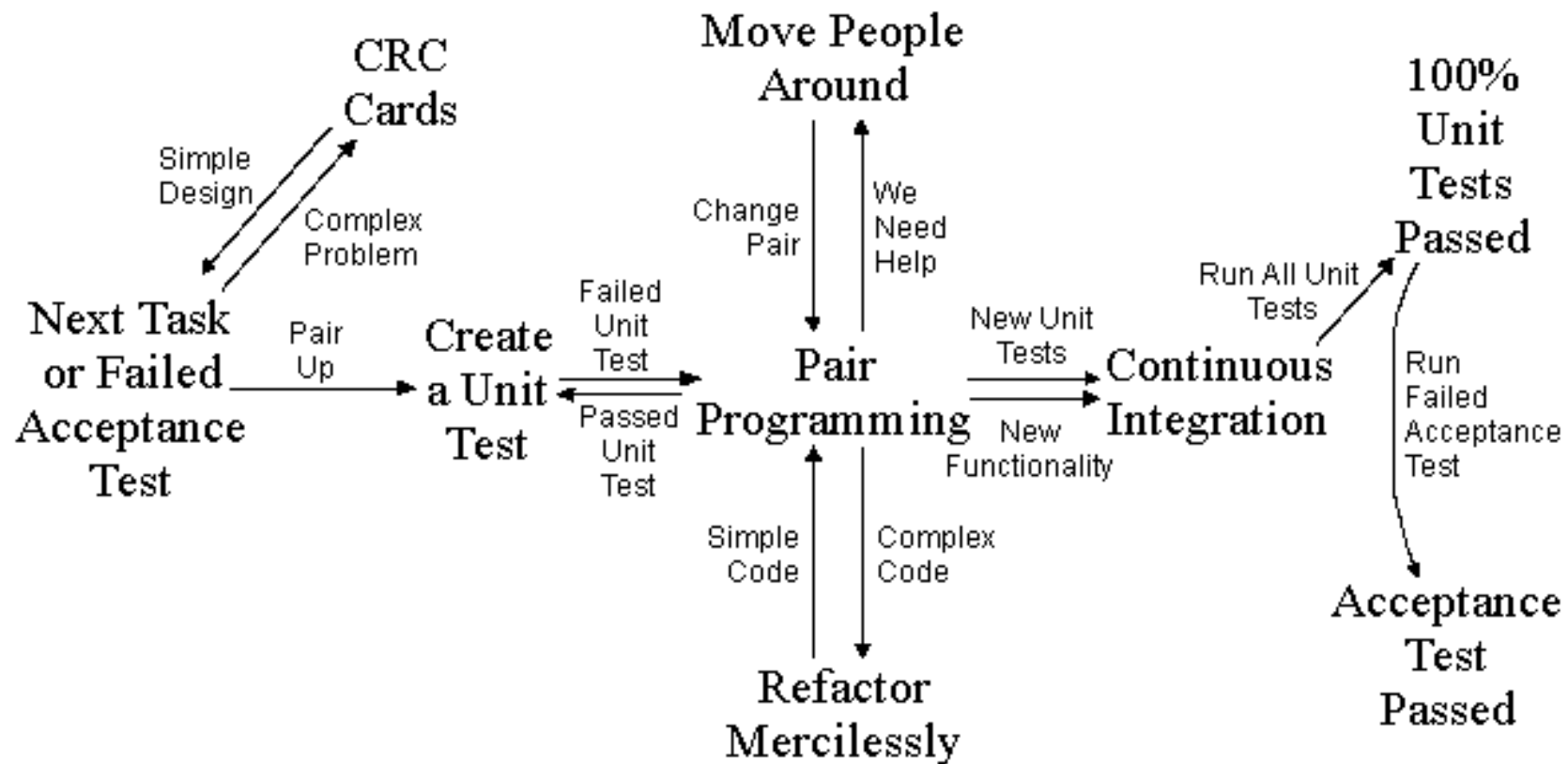
 Zoom Out



# TDD

## Collective Code Ownership

 Zoom Out





# Content

---

1. XP

2. Scrum

## Scrum in 100 words

- Scrum is an agile process that allows us to focus on delivering the highest business value in the shortest time.
- It allows us to rapidly and repeatedly inspect actual working software (every two weeks to one month).
- The business sets the priorities. Teams self-organize to determine the best way to deliver the highest priority features.
- Every two weeks to a month anyone can see real working software and decide to release it as is or continue to enhance it for another sprint.

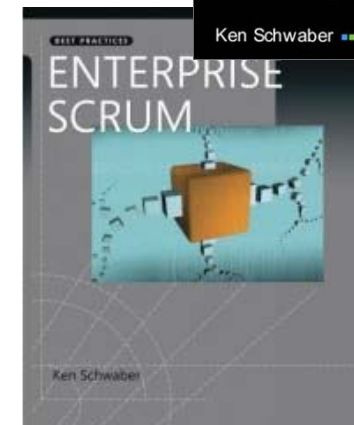
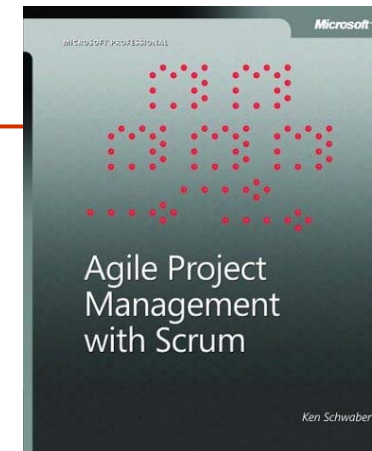
# Scrum

---



# Scrum Origins

- Jeff Sutherland
  - Initial scrums at Easel Corp in 1993
  - IDX and 500+ people doing Scrum
- Ken Schwaber
  - ADM
  - Scrum presented at OOPSLA 96 with Sutherland
  - Author of three books on Scrum
- Mike Beedle
  - Scrum patterns in PLOPD4
- Ken Schwaber and Mike Cohn
  - Co-founded Scrum Alliance in 2002, initially within the Agile Alliance



# Scrum has been used by:

---

- Microsoft
- Yahoo
- Google
- Electronic Arts
- High Moon Studios
- Lockheed Martin
- Philips
- Siemens
- Nokia
- Capital One
- BBC
- Intuit
- Intuit
- Nielsen Media
- First American Real Estate
- BMC Software
- Ipswitch
- John Deere
- Lexis Nexis
- Sabre
- Salesforce.com
- Time Warner
- Turner Broadcasting
- Oce

# Scrum has been used for:

---

- ❑ Commercial software
- ❑ In-house development
- ❑ Contract development
- ❑ Fixed-price projects
- ❑ Financial applications
- ❑ ISO 9001-certified applications
- ❑ Embedded systems
- ❑ 24x7 systems with 99.999% uptime requirements
- ❑ the Joint Strike Fighter
- ❑ Video game development
- ❑ FDA-approved, life-critical systems
- ❑ Satellite-control software
- ❑ Websites
- ❑ Handheld software
- ❑ Mobile phones
- ❑ Network switching applications
- ❑ ISV applications
- ❑ Some of the largest applications in use

# Scrum Characteristics

---

- ❑ Self-organizing teams
- ❑ Product progresses in a series of month-long “sprints”
- ❑ Requirements are captured as items in a list of “product backlog”
- ❑ No specific engineering practices prescribed
- ❑ Uses generative rules to create an agile environment for delivering projects
- ❑ One of the “agile processes”

# SCRUM Development Process

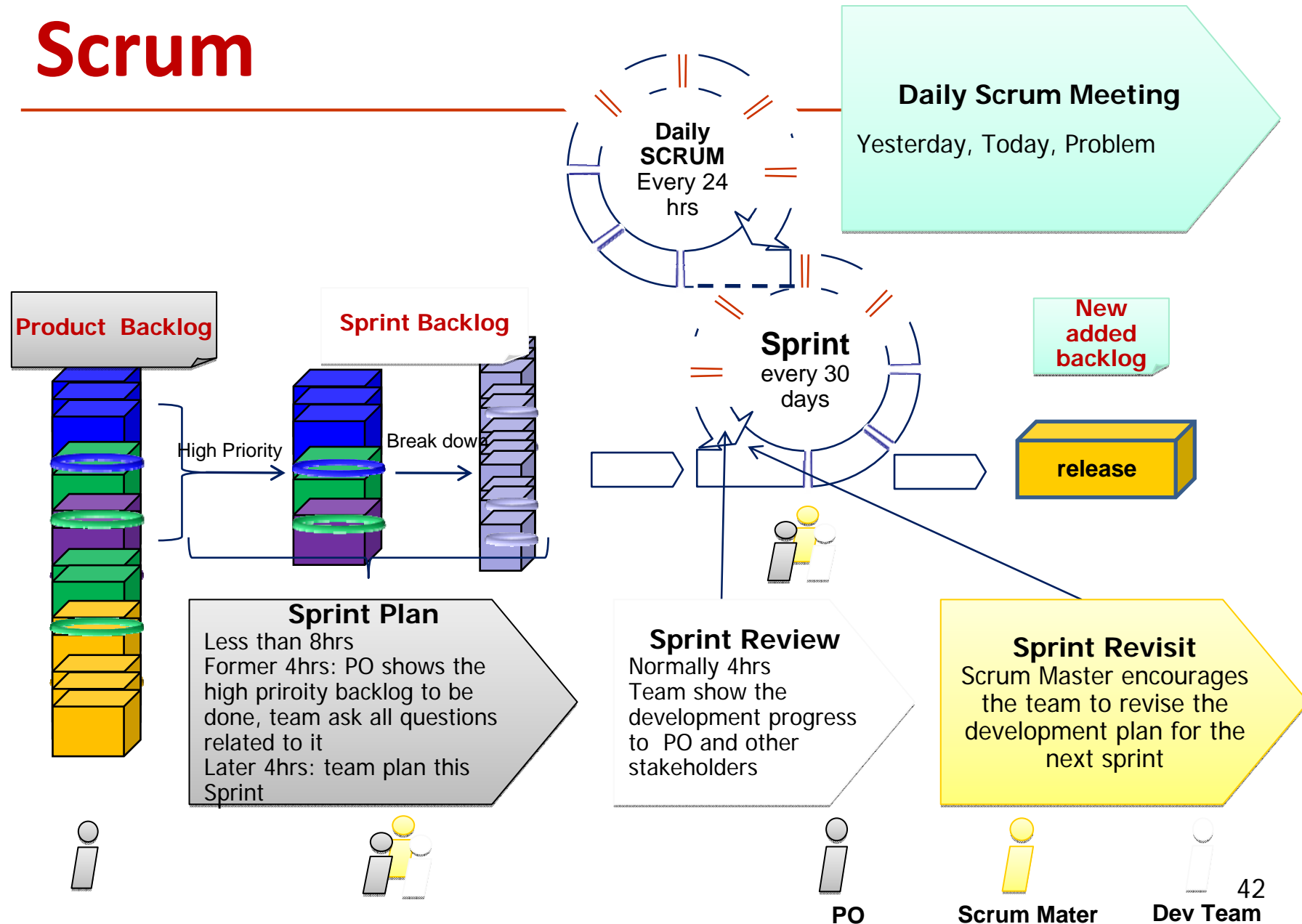
---

- Assumption: One cannot pre-define the specs of the Final Product, but in a try-and-error process
- Scrum treats dev team as the American football team
  - Clear final goal
  - Familiar with all techniques for development
  - A high degree of autonomy
  - Close collaborations
  - Flexibility to solve any challenge
  - Make sure the everyday's progress towards the final goal



- 
- Iteration duration: 30 days, as a “Sprint”
  - Daily stand-up meeting everyday: 15mins to report the yesterday’s progress, problems, and plan for today
  - SCRUM emphasizes the communications between the development and management teams, through daily report and resort to their help if any

# Scrum

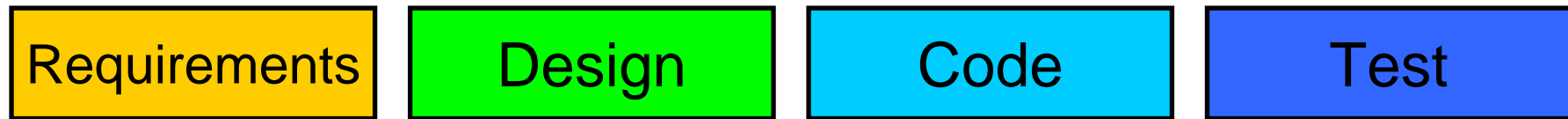


# Sprint

---

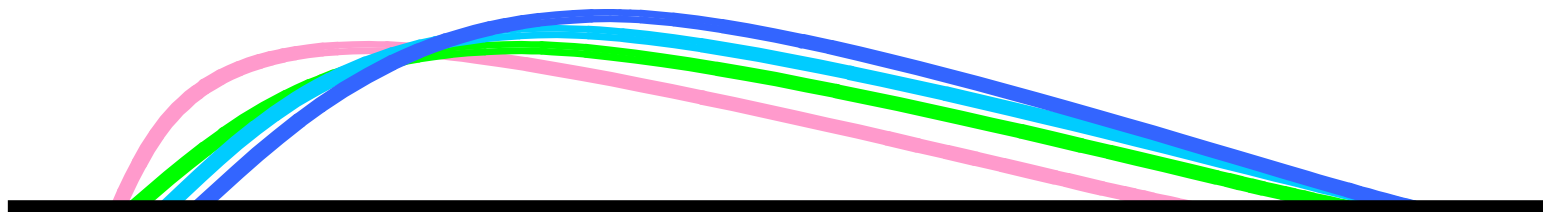
- ❑ Scrum projects make progress in a series of “sprints”, analogous to Extreme Programming iterations
- ❑ Normally 2-3 weeks
- ❑ A constant duration leads to a better rhythm
- ❑ One sprint includes: design, coding, testing, and documentation
- ❑ Once a sprint starts, only Scrum Team can add/remove items in a Sprint backlog
- ❑ Only if when the sprint goal becomes meaning less, to terminate this sprint
- ❑ Does not allow requirement change within a Sprint, priority change only happens between two Sprints

# Sequential vs. Overlapping Development



Rather than doing all of one thing at a time...

...Scrum teams do a little of everything all the time



## Roles

- Product owner
- ScrumMaster
- Team

# Scrum Framework

## Ceremonies

- Sprint planning
- Sprint review
- Sprint retrospective
- Daily scrum meeting

## Artifacts

- Product backlog
- Sprint backlog
- Burndown charts

# Scrum Framework

## Roles

- Product owner
- ScrumMaster
- Team

## Events

- Sprint planning
- Sprint review
- Sprint retrospective
- Daily scrum meeting

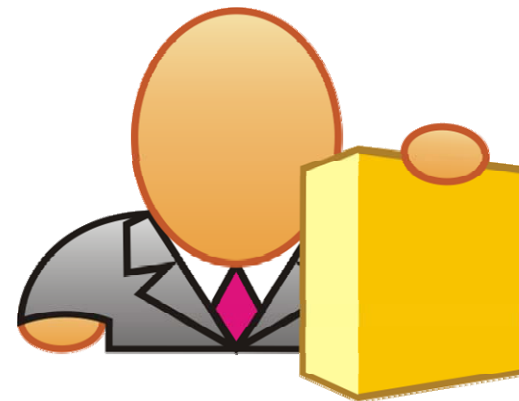
## Artifacts

- Product backlog
- Sprint backlog
- Burndown charts

# Product Owner (PO)

---

- ❑ Define the features of the product
- ❑ Decide on release date and content
- ❑ Be responsible for the profitability of the product (ROI)
- ❑ Prioritize features according to market value
- ❑ Adjust features and priority every iteration, as needed
- ❑ Accept or reject work results



# Scrum Master

---

- ❑ Represents management to the project
- ❑ Responsible for enacting Scrum values and practices
- ❑ Removes impediments
- ❑ Ensure that the team is fully functional and productive
- ❑ Enable close cooperation across all roles and functions
- ❑ Shield the team from external interferences





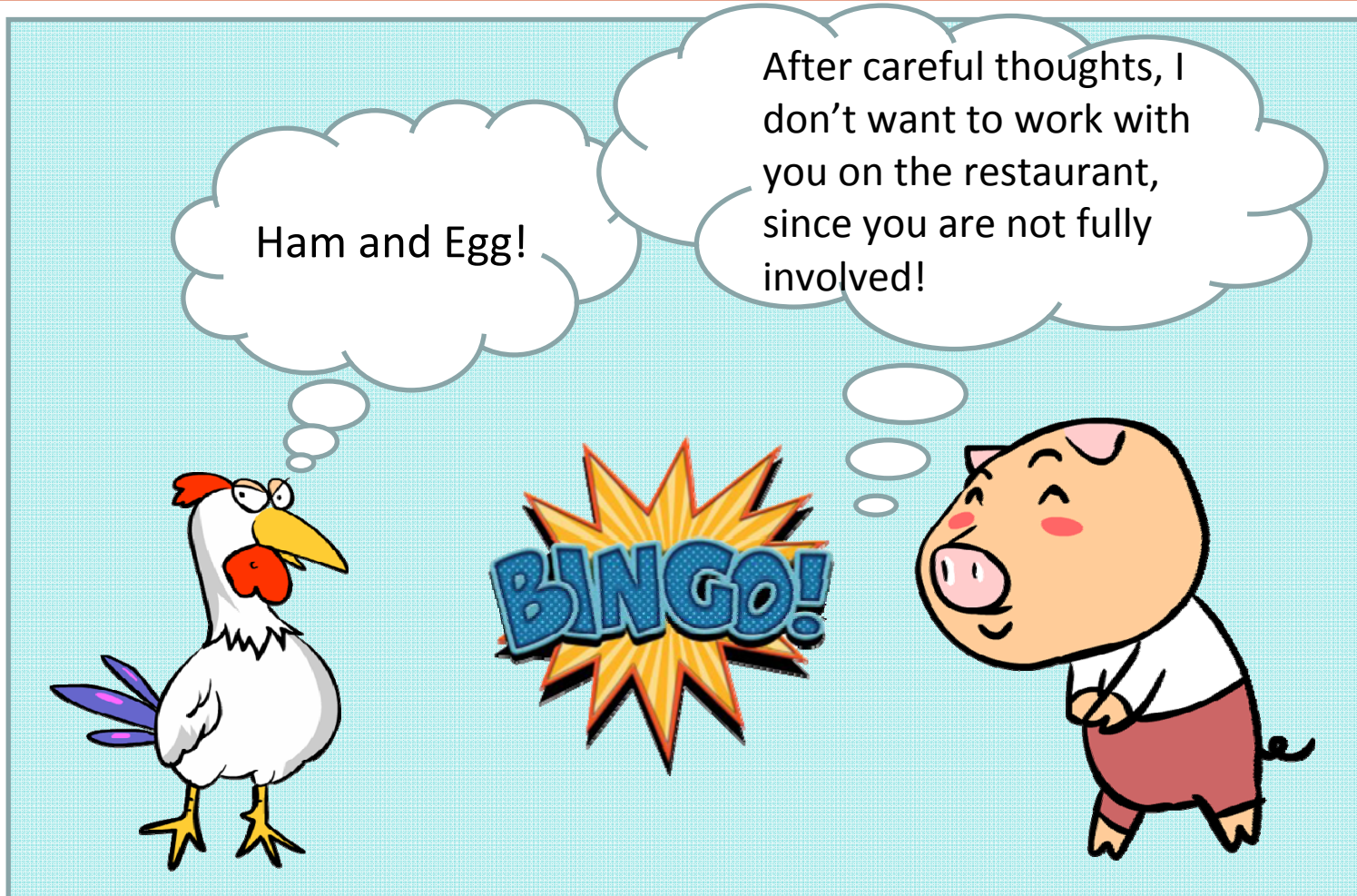
# Scrum Team



- Typically 5-9 people
- Cross-functional:
  - Programmers, testers, user experience designers, etc.
- Members should be full-time
  - May be exceptions  
(e.g., database administrator)
- Teams are self-organizing
  - Ideally, no titles but rarely a possibility
- Membership should change only between sprints



# Scrum Role and Responsibility

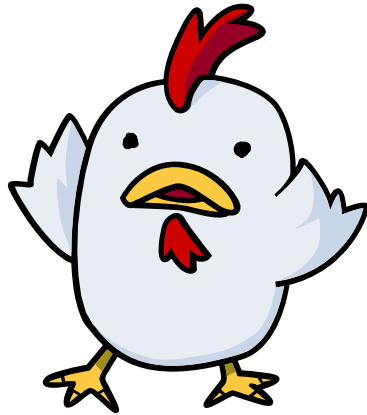


# Chickens & Pigs

---



- **Pigs:** Scrum Team Member: fully responsible for the sprint goal



- **Chickens:** a member related to this project, not fully involved
- can attend the Scrum meeting as a observer

# Scrum Framework

## Roles

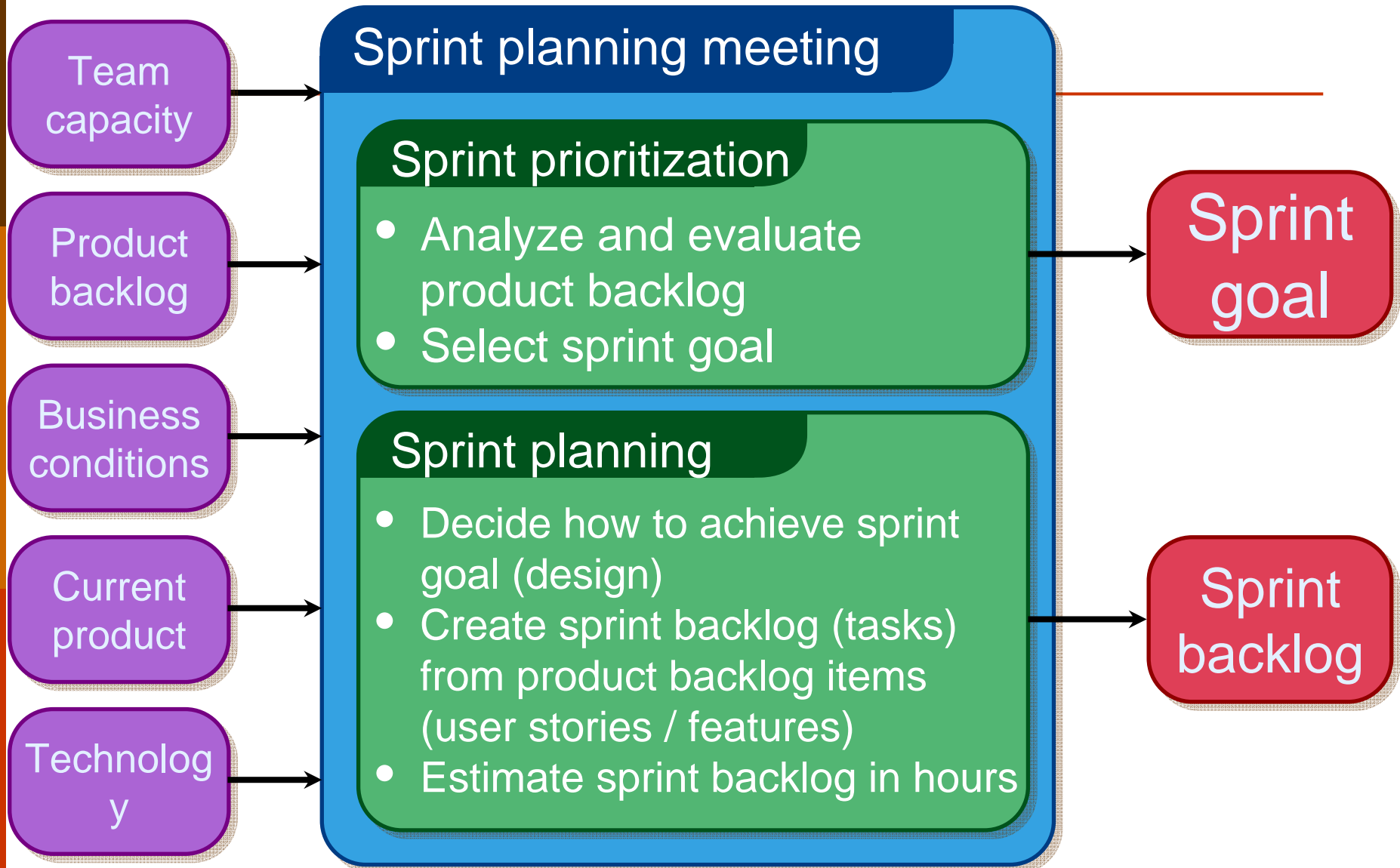
- Product owner
- ScrumMaster
- Team

## Ceremonies

- Sprint planning
- Sprint review
- Sprint retrospective
- Daily scrum meeting

## Artifacts

- Product backlog
- Sprint backlog
- Burndown charts



# Sprint Planning

---

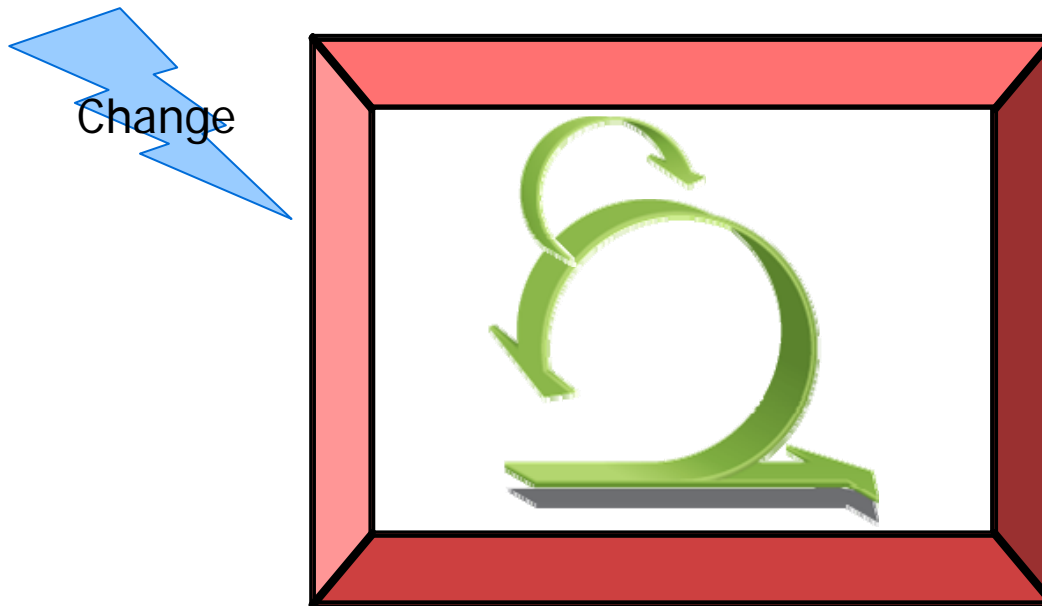
- Team selects items from the product backlog they can commit to completing
- Sprint backlog is created
  - Tasks are identified and each is estimated (1-16 hours)
  - Collaboratively, not done alone by the ScrumMaster
- High-level design is considered

As a vacation planner,  
I want to see photos  
of the hotels.

Code the middle tier (8 hours)  
Code the user interface (4)  
Write test fixtures (4)  
Code the foo class (6)  
Update performance tests (4)

# No changes during a sprint

---



- Plan sprint durations around how long you can commit to keeping change out of the sprint

# Daily Scrum

- Parameters
  - Daily
  - 15-minutes
  - Stand-up
- Not for problem solving
  - Whole world is invited
  - Only team members, ScrumMaster, product owner, can talk
- Helps avoid other unnecessary meetings





# Everyone answers 3 questions

---

1

What did you do yesterday?

2

What will you do today?

3

Is anything in your way?

- These are *not* status for the ScrumMaster
  - They are commitments in front of peers

# Sprint Review

---

- ❑ Team presents what it accomplished during the sprint
- ❑ Typically takes the form of a demo of new features or underlying architecture
- ❑ Informal
  - 2-hour prep time rule
  - No slides
- ❑ Whole team participates
- ❑ Invite the world



# Sprint Retrospective (Revisit)

---

- Periodically take a look at what is and is not working
- Typically 15–30 minutes
- Done after every sprint
- Whole team participates
  - ScrumMaster
  - Product owner
  - Team
  - Possibly customers and others

# Start / Stop / Continue

---

- Whole team gathers and discusses what they'd like to:

Start doing

Stop doing

Continue doing

This is just one of many ways to do a sprint retrospective.

# Scrum Framework

## Roles

- Product owner
- ScrumMaster
- Team

## Ceremonies

- Sprint planning
- Sprint review
- Sprint retrospective
- Daily scrum meeting

## Artifacts

- Product backlog
- Sprint backlog
- Burndown charts

# Product backlog

---



This is the product backlog

- The requirements
- A list of all desired work on the project
- Ideally expressed such that each item has value to the users or customers of the product
- Prioritized by the product owner
- Reprioritized at the start of each sprint

# A sample product backlog

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

# Sprint goal

---

- A short statement of what the work will be focused on during the sprint

## Database Application

Make the application run on SQL Server in addition to Oracle.

## Life Sciences

Support features necessary for population genetics studies.

## Financial services

Support more technical indicators than company ABC with real-time, streaming data.



# Managing the sprint backlog

---

- Individuals sign up for work of their own choosing
  - Work is never assigned
- Estimated work remaining is updated daily
- Any team member can add, delete or change the sprint backlog
- Work for the sprint emerges
- If work is unclear, define a sprint backlog item with a larger amount of time and break it down later
- Update work remaining as more becomes known

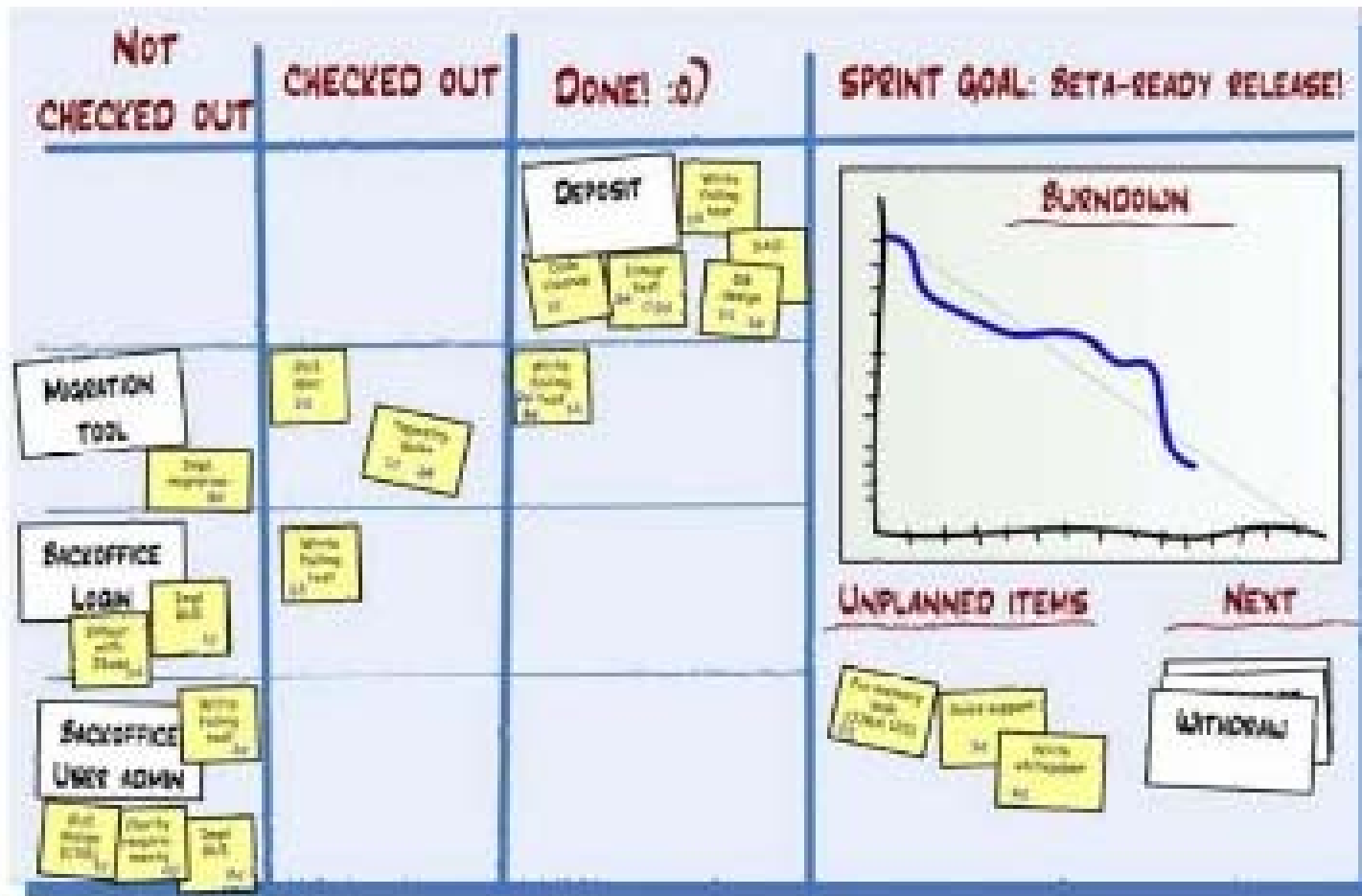
# Example: Sprint Backlog

---

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	

# Burn Down Charts

- 工具：任务板



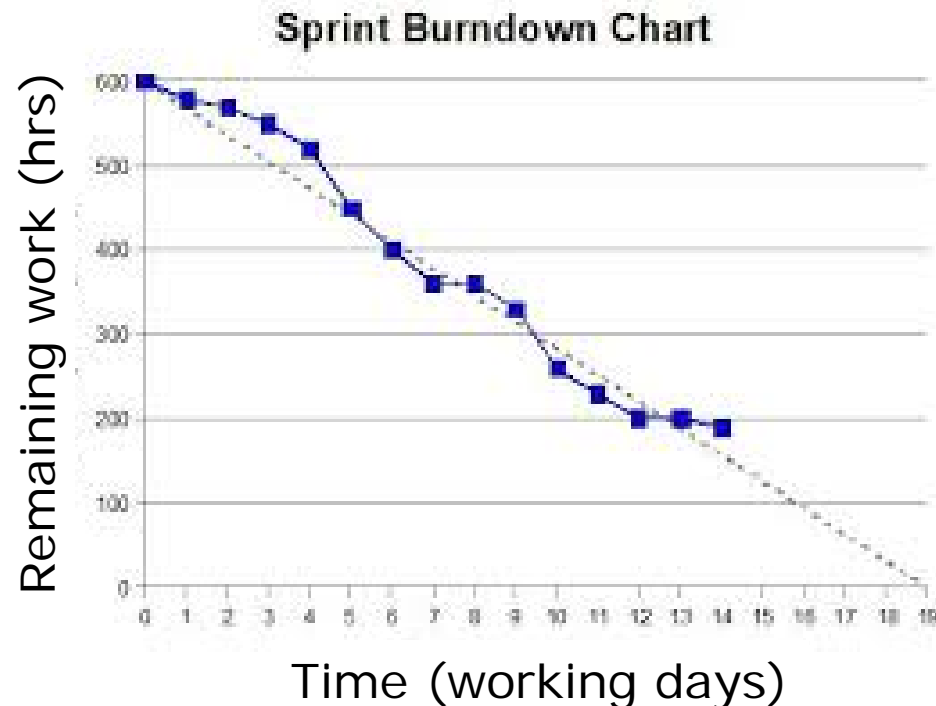
# Burn down Charts

---

- Are used to represent “work done”.
- Are wonderful Information Radiators
- 3 Types:
  - Sprint Burn down Chart (progress of the Sprint)
  - Release Burn down Chart (progress of release)
  - Product Burn down chart (progress of the Product)
- X-Axis: time (usually in days)
- Y-Axis: remaining effort

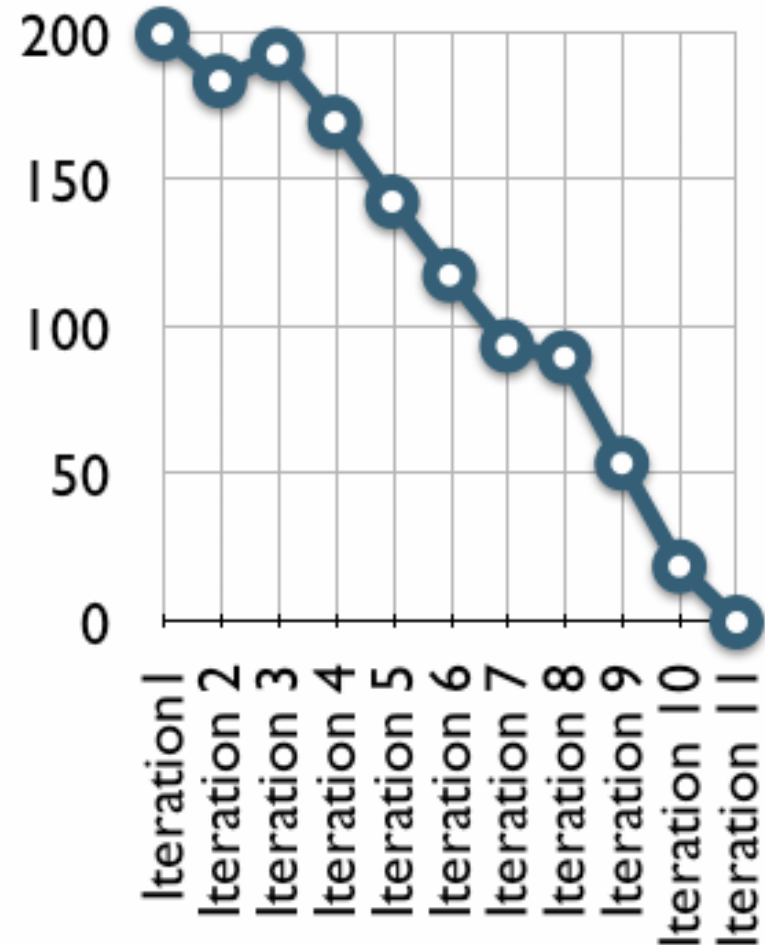
# Sprint Burn down Chart

- ❑ Depicts the total Sprint Backlog hours remaining per day
- ❑ Shows the estimated amount of time to release
- ❑ Ideally should burn down to zero to the end of the Sprint
- ❑ Actually is not a straight line
- ❑ Can bump UP



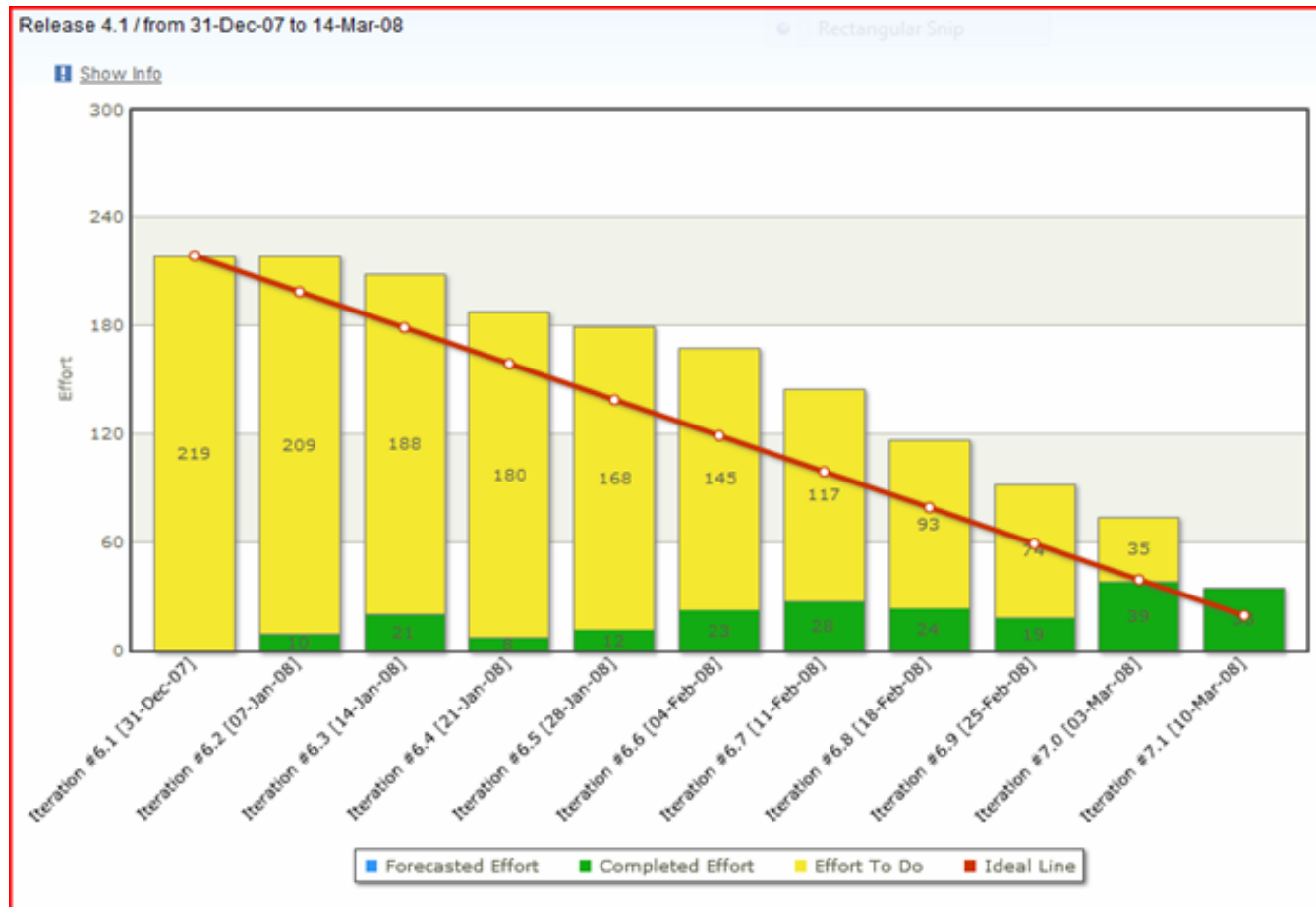
# Release Burn down Chart

- ❑ Will the release be done on right time?
- ❑ X-axis: sprints
- ❑ Y-axis: amount of remaining hrs
- ❑ The estimated work remaining can also burn up



# Alternative Release Burn down Chart

- Consists of bars (one for each sprint)

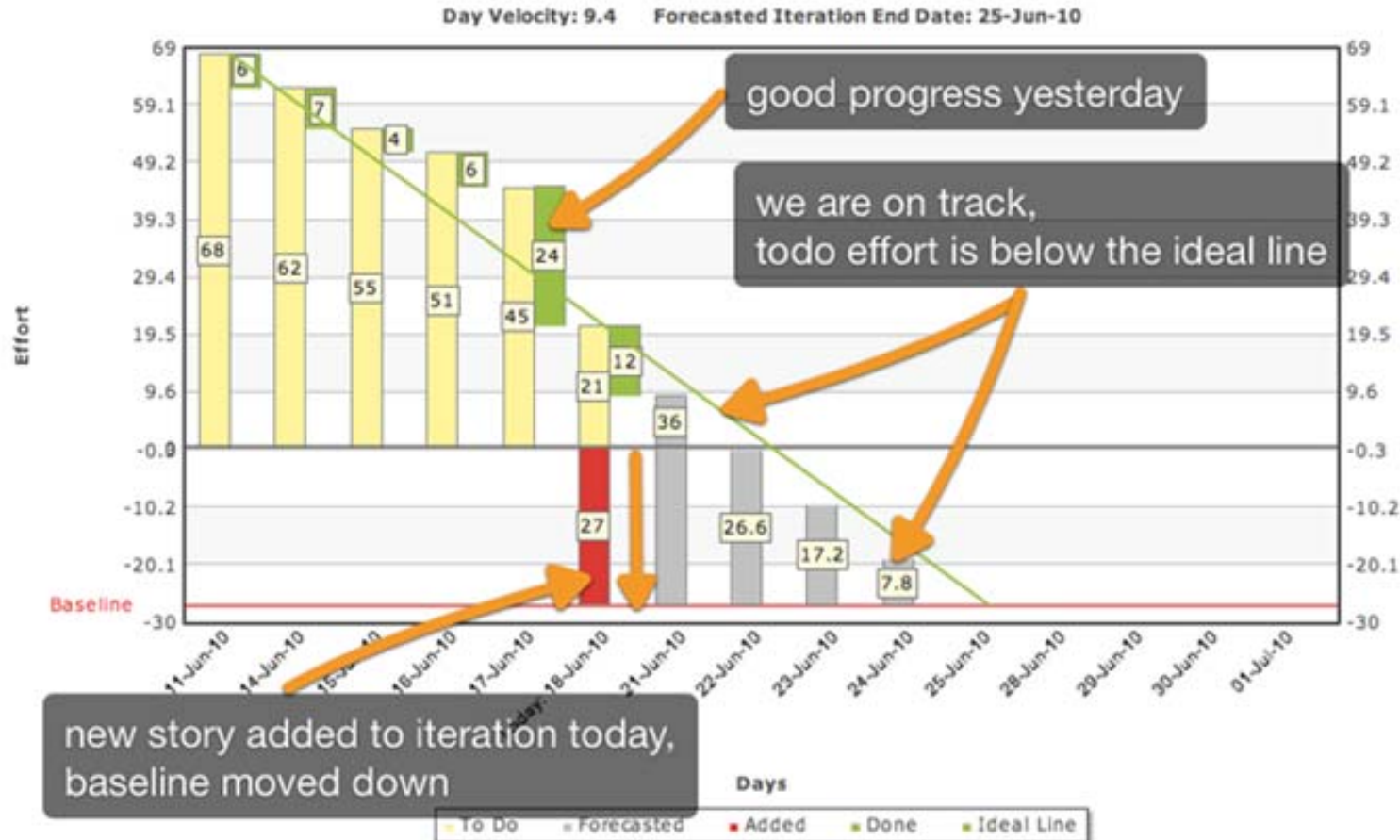


- Values on the Y-axis: **positive AND negative**
- Is more informative than a simple chart

### Iteration Burn Down

Iteration #0.2 (current) Current

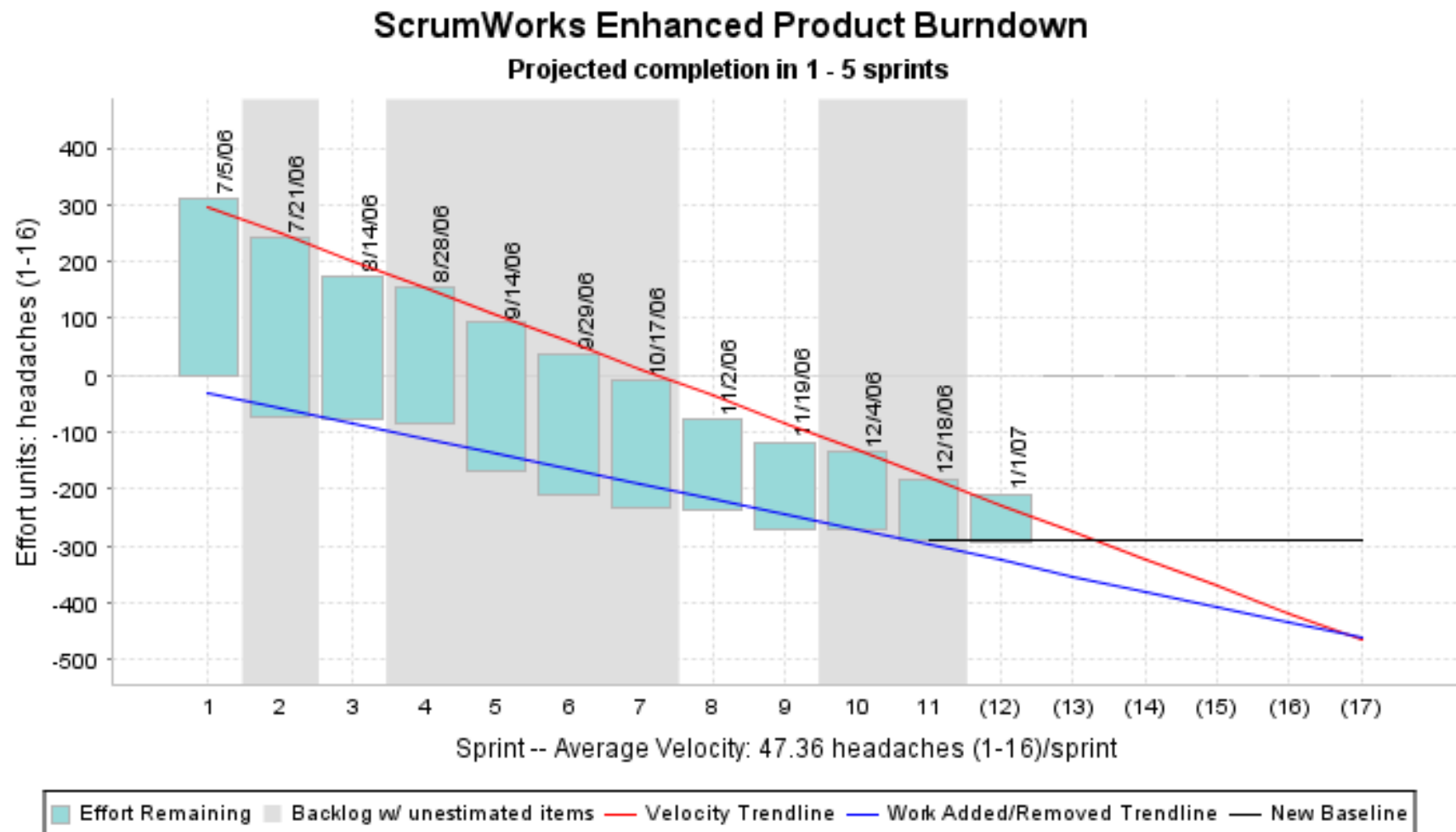
Iteration planning in the first day of iteration  Include completed stories, tasks and bugs only  Weekends  Labels Show





# Product Burn down Chart

- Is a “big picture” view of project’s progress (all releases)

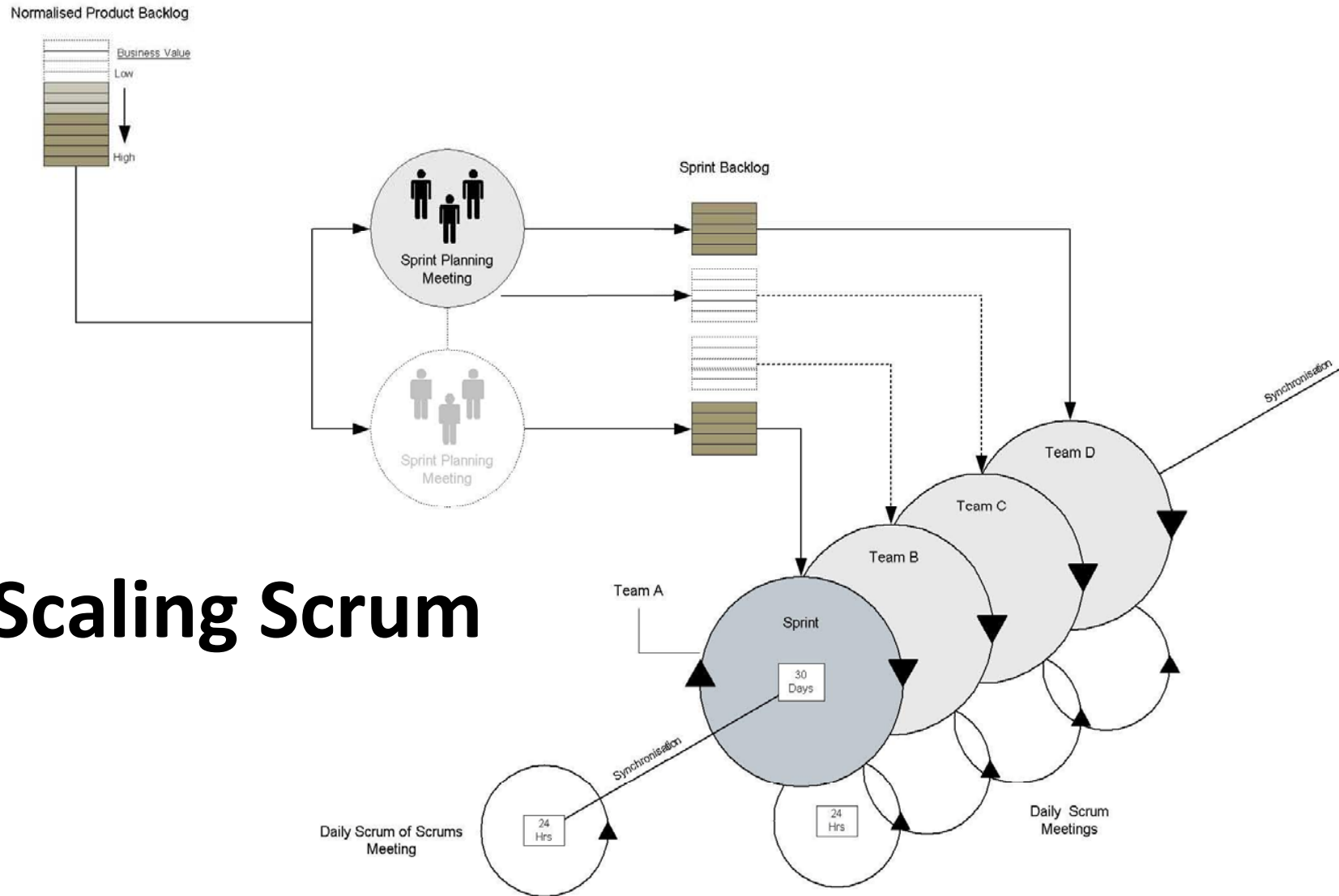



# Scaling Scrum

---

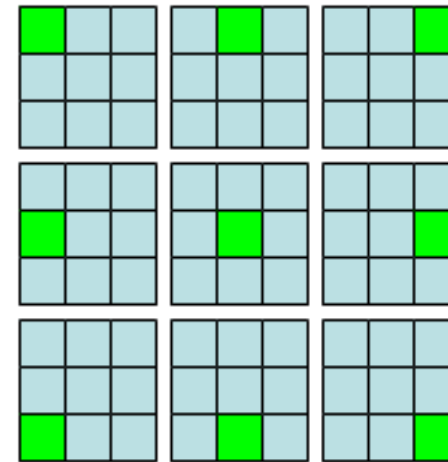
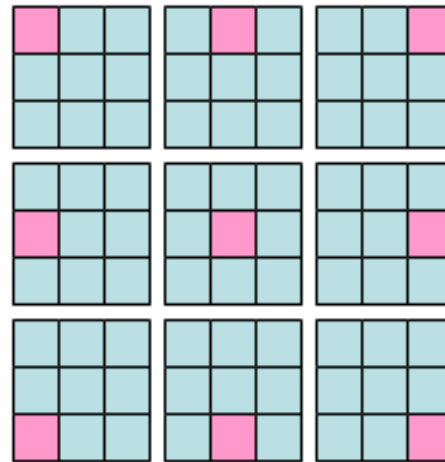
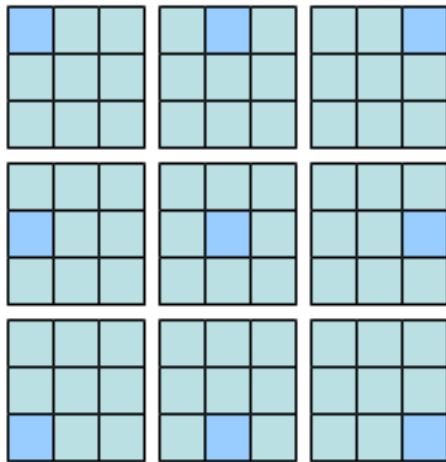
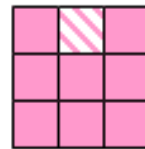
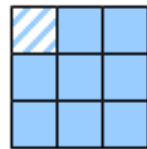
- A typical Scrum team is 6-10 people
- Jeff Sutherland - up to over 800 people
- "Scrum of Scrums" or what called "Meta-Scrum"
- Frequency of meetings is based on the degree of coupling between packets

# Scaling Scrum



	TITLE	Scaled scrum sprints with collocated teams	VERSION	1.00
	PROJECT	Scrum	DATE	24/06/02
	NOTES		INITIALS	SJB
			PAGES	1 / 1

think-box Limited.



# Pro/Con

---

## □ Advantages

- Completely developed and tested features in short iterations
- Simplicity of the process
- Clearly defined rules
- Increasing productivity
- Self-organizing
- each team member carries a lot of responsibility
- Improved communication
- Combination with Extreme Programming

## □ Drawbacks

- “Undisciplined hacking” (no written documentation)
- Violation of responsibility
- Current mainly carried by the inventors

# Review

## Scrum Master

- Manages the process removes obstacles in the teams way communicates with the stakeholders, product owner and senior management

## Product Owner

- Manages development of product functionality, builds business value, plans and executes each iteration or "Sprint"

## Scrum Team

- Manages the product vision, project ROI and plans releases

## Sprint Planing

- Consists of two meetins sprint planing 1 and sprint planing 2 the first meeting all scrum members attend , the second meeting the Product owner is optional, each meeting should take 4 hours, output should be estimated Sprint Backlogd with all story card details

## Daily Scrum

- Status meeting each day of 15 mins
- Team members says what he done and he will do to next meeting and if he have impedments
- any questions should be taken aside after meeting
- All team must attend , Scrum master is optional

## Sprint Review

- On last day of sprint and takes 4 hours
- no special preparation
- team presents what they have one and wasnt able to do and if they done taks right
- Scrum Master, Product Owner, Team Attends some Stakeholders Optional
- Sprint Velocity is calculated at the end of meeting

## Sprint Prospective

- Accurs after the sprint review and takes 3 hours
- Team discusses what went well (+) what went badly (  $\Delta$  ) things needs to be changed
- things went badly should be presented in a solutions to problems not just (-) text
- Scrum Master and Team must attend , Stackholder are optional

## Product Backlog

- List of feature, functionality , Issues that placeholder define later as work
- Emergent, protized, estimated(Effort,Complexity,Risk)
- user centered, value driven
- one list for multiple teams

## Sprint Backlog

- Tasks from Product Backlog to turn into product
- Taks should take max 1 day or else should be divided
- Estimated Work remaining is updated daily

## Sprint Burndown

- shows work remaining for the currentsprint
- should be a good presnataion of the team work and taks vs time remianing to on the sprint

## Product Burndown

- shows work remaining at the start of each sprint includes work completed, added,removed and resized
- Burnup chart : tracks both work completed and total backlog efforts