北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

高级软件工程

第七节课：敏捷开发

主讲：刘 驰

Content

- Agile Development 介绍
- XP
- Scrum

Agile Process - 敏捷的开发流程

- Agile Process (敏捷开发流程) 是一种软件开发流程的泛称
- 共通的特性：
  - 客户与开发人员形成密切合作的团队，因为客户无法于初期定义完整的规格和功能，而开发人员于开发过程中也常常无法知悉外在环境或业务的变动，所以需要两者密切合作方能开发适用的软件。
  - 项目最终的目标是可执行的程序，因此所有的中间产品必须经过审慎评估，确认有助于最终目标，才需要制作中间产品。
  - 采用 Iterative与Incremental 方式分阶段进行，密集review是否符合需求。
  - 流程可以简单，但规划与执行必须严谨。
  - 强调团队合作，赋予高度的责任，团队有自主权得以因应变化做调整

敏捷开发的概念

- 敏捷开发是一种以人为核心、迭代、循序渐进的开发方法
- 在敏捷开发中，项目的构建被切分成多个子项目，各个子项目的成果都经过测试，具备集成和可运行的特征 。

北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

敏捷开发的核心价值 (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

敏捷开发原则(Principles)

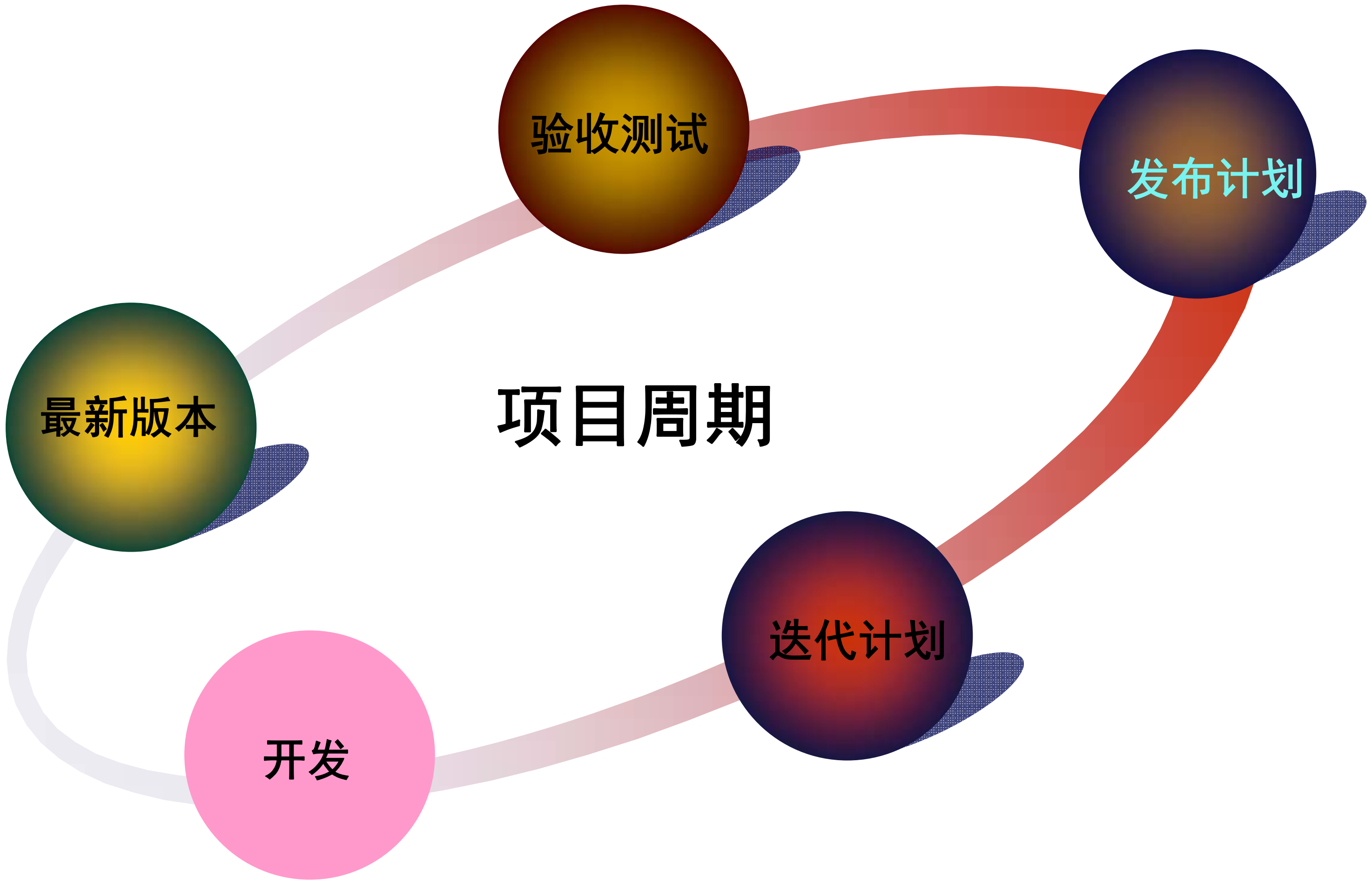
1. 最高目标是通过快速的和经常的发布软件满足客户的需要
2. 提交软件的周期为几个星期到几个月
3. 产生正确的软件是衡量进度的首要标准
4. 主动接受需求的改变而不是拒绝
5. 商务人员和开发人员工作在一起
6. 个人必须有动力，要创造环境支持他们的要求，信任他们
7. 最有效的交流方法是面对面的交流
8. 最好的结构，需求和设计来自于自组织的团队（self-organizing team），允许任何人提出想法和建议
9. 持续改进设计和编码
10. 鼓励正常工作，减少长时间加班
11. 保持简单，减少不必要的部分，认识到简单的设计比复杂的设计更难（simple design is harder to produce）
12. 定期调整过程，获得更高效率

设计原则沿袭设计模式的SOLID原则

- SRP
  - 单一职责原则SRP：Single Responsibility Principle
- OCP
  - 开放封闭原则OCP：Open—Closed Principle
- LSP
  - Liskov替换原则LSP：Liskov Substitution Principle
- DIP
  - 依赖倒置原则DIP：Dependency Invertion Principle
- ISP
  - 接口隔离原则ISP：Interface Separate Principle

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

敏捷开发-迭代计划



Content

1. XP

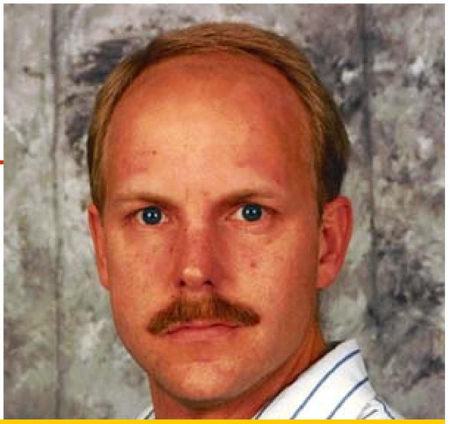
2. Scrum

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

BEIJING INSTITUTE OF TECHNOLOGY 北京理工大學

Extreme Programming (XP)


so extreme he never smiles?!?

Developed by

- Kent Beck
- "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*

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

- What does lack of communication do to projects?
- 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

ALL OF MY BAD DECISIONS ARE CATCHING UP TO ME. COULD WE DO ANOTHER REORG TO COVER MY TRACKS?
YEAH, I'VE GOT SOME BODIES TO BURY, TOO.

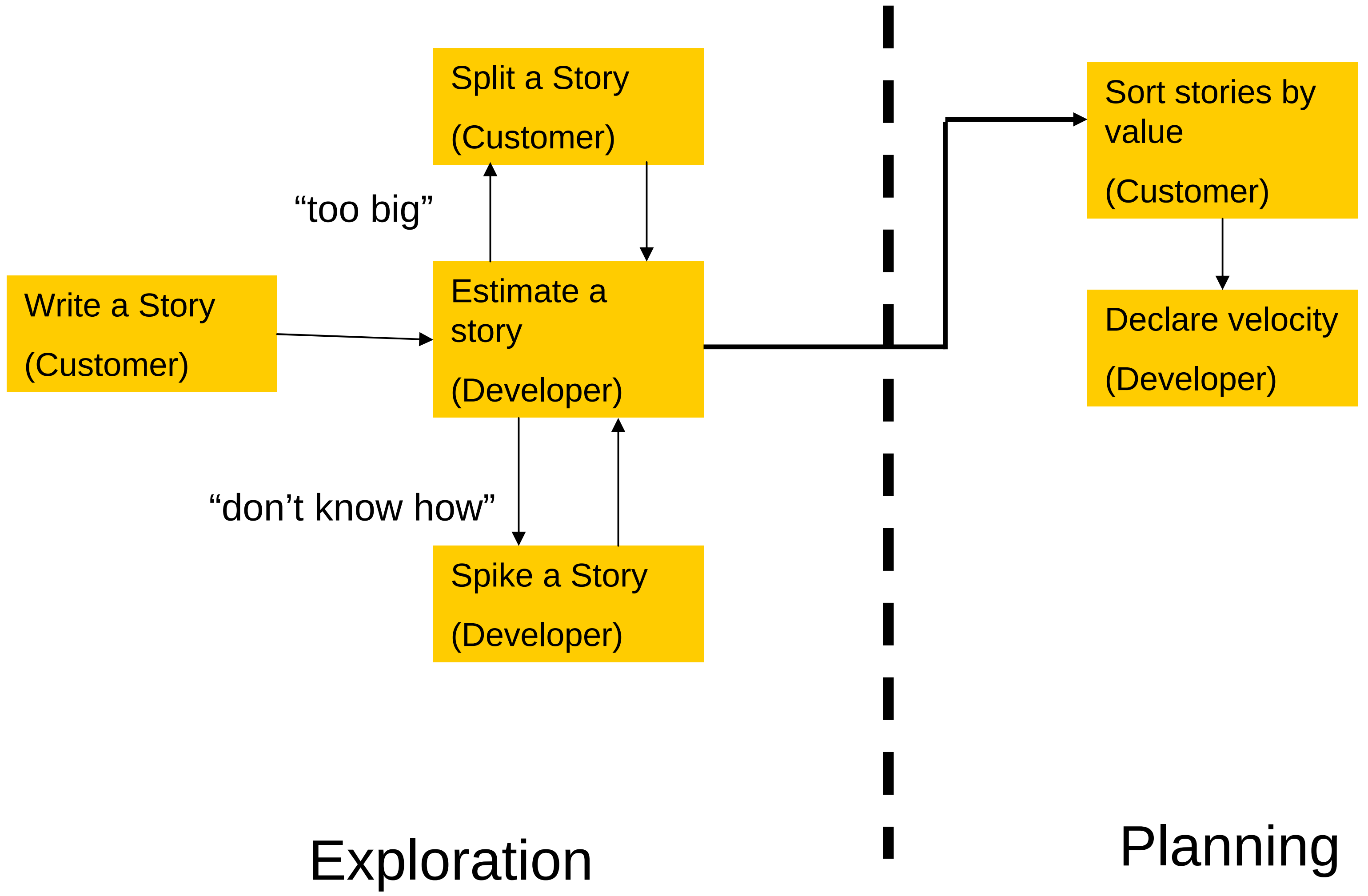
12 XP Practices

- 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 developers' estimates and project velocity, the customer prioritizes which user stories to implement

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

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

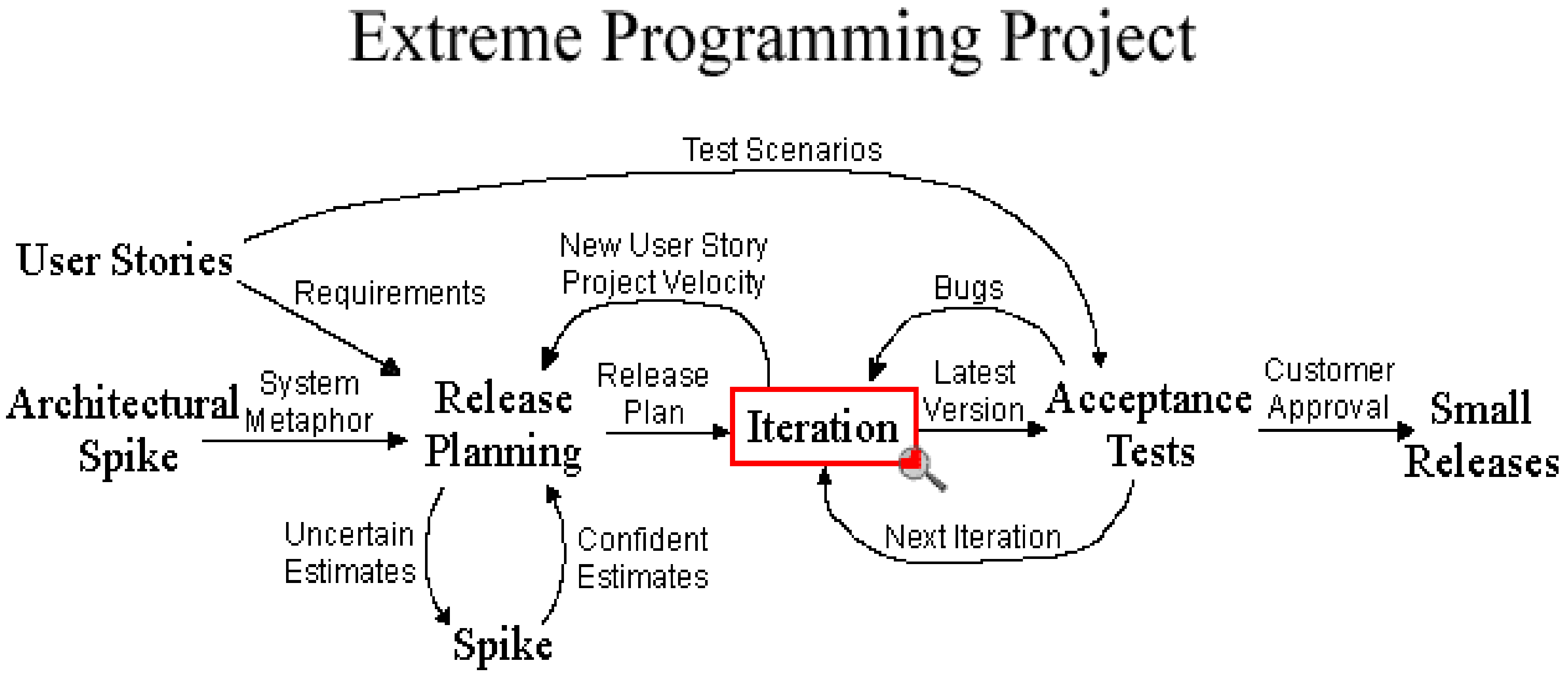
- Everyone attends, including the customer
- Not a discussion forum
- Take discussions offline
- Everyone gets to speak
- 15 minutes

Kindergarten lessons

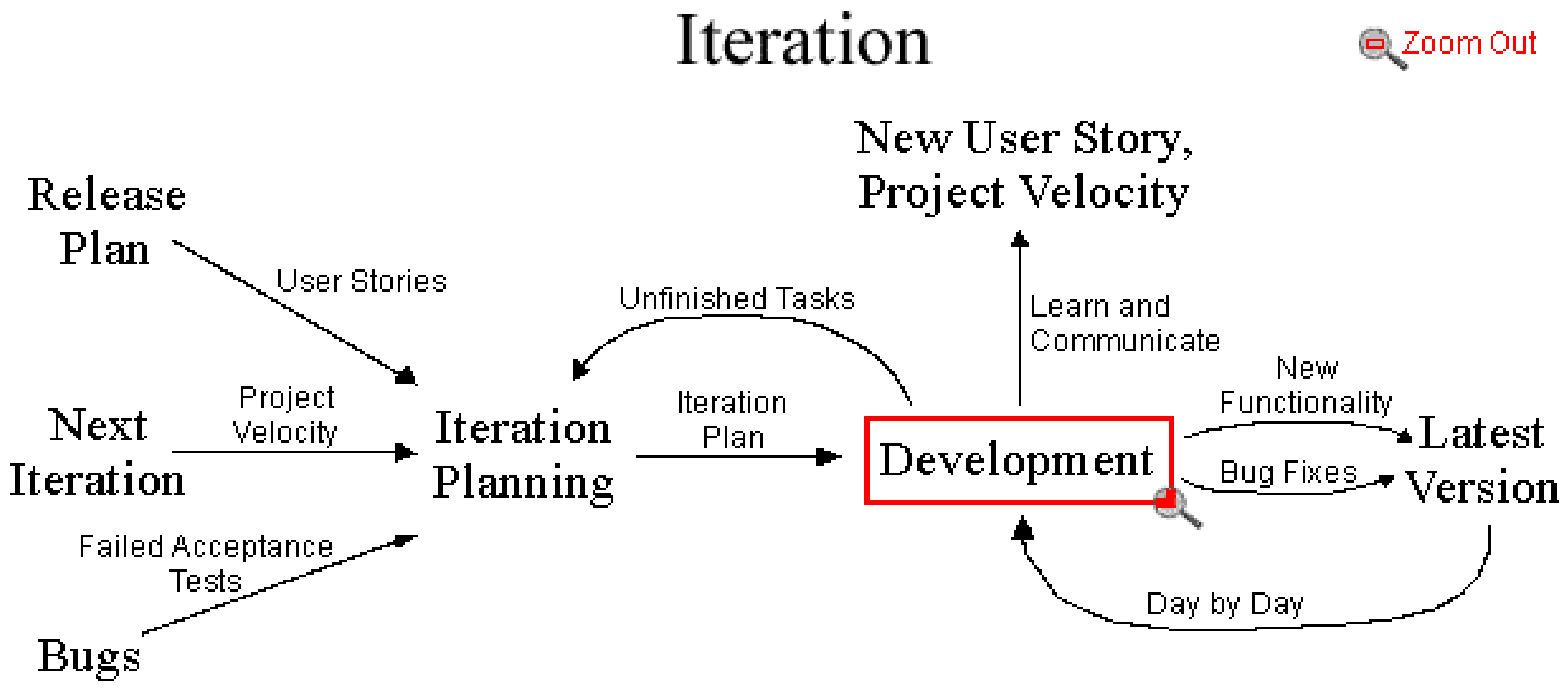
- Williams, L. and Kessler, R., "All I Really Need to Know about Pair Programming I Learned In Kindergarten," *Communications of the ACM* (May 2000)
  - *Share everything.* (Collective code ownership)
  - *Play fair.* (Pair programming—navigator must not be passive)
  - *Don't hit people.* (Give and receive feedback. Stay on track.)
  - *Clean up your own mess.* (Unit testing.)
  - *Wash your hands before you eat.* (Wash your hands of skepticism: buy-in is crucial to pair programming.)
  - *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)

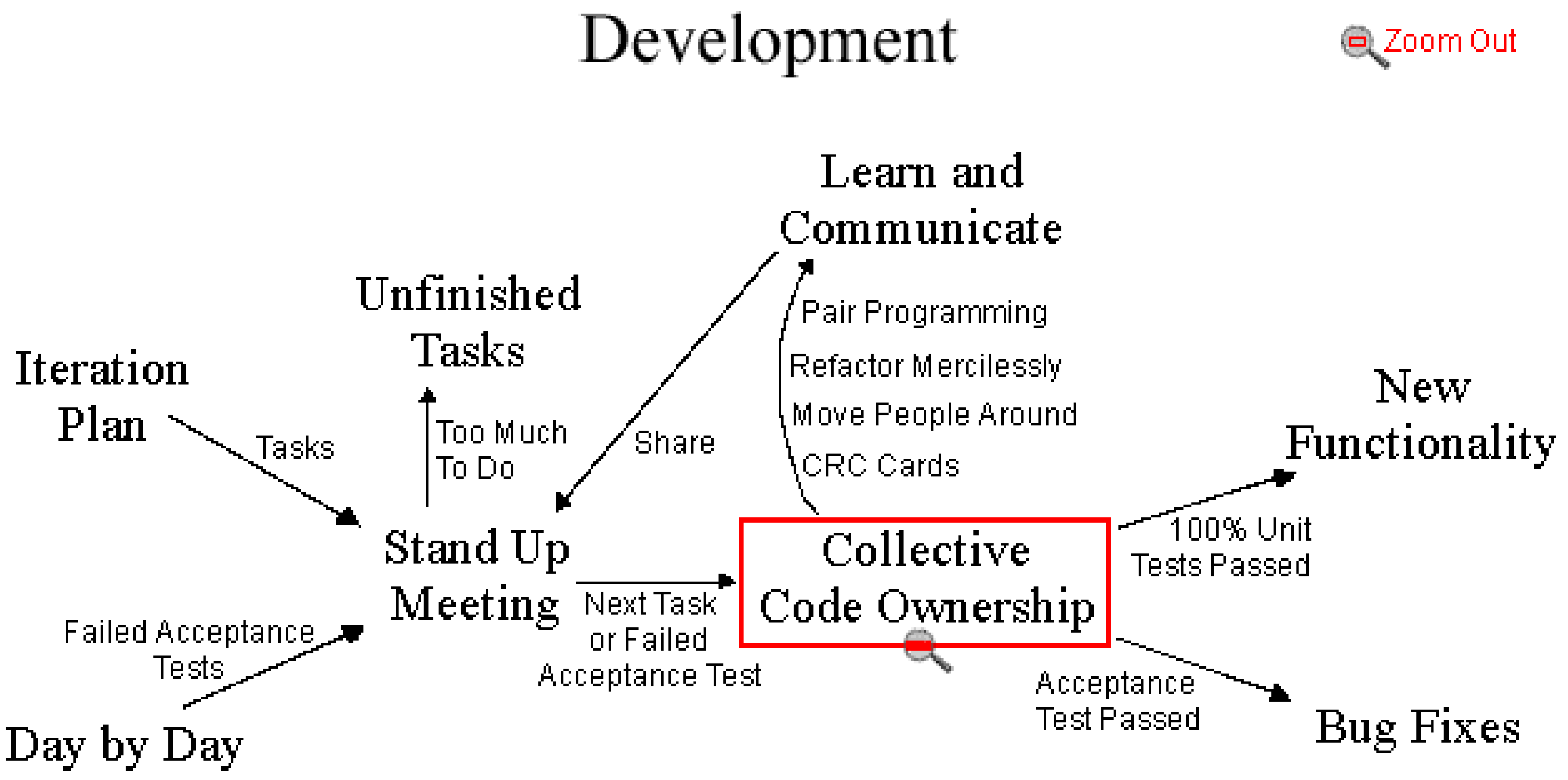


XP emphasizes iteration

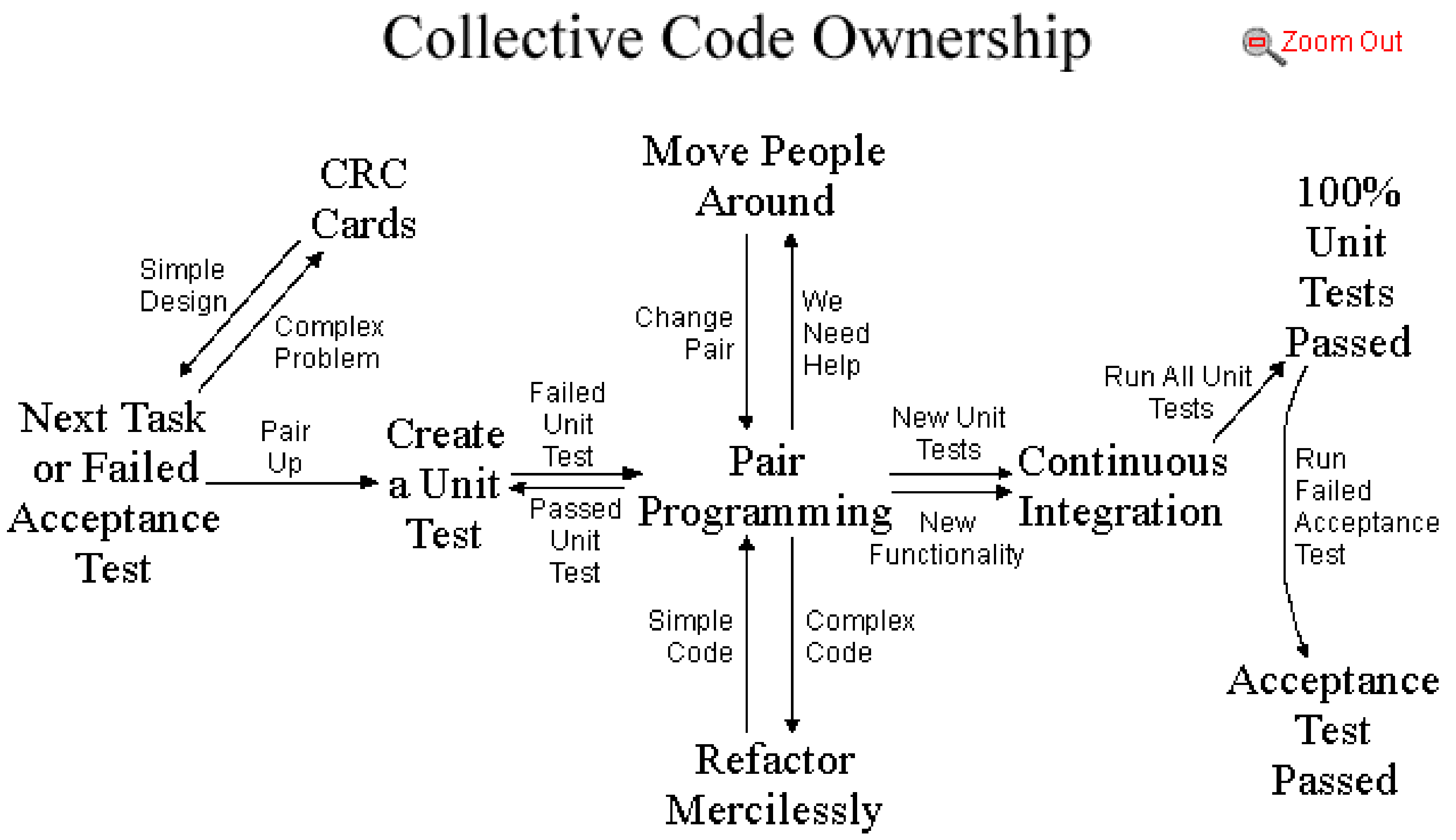


北京理工大學 BEIJING INSTITUTE OF TECHNOLOGY

XP emphasizes communication



TDD



Collective Code Ownership
Zoom Out
CRC Cards
Simple Design
Complex Problem
Move People Around
Change Pair
We Need Help
100% Unit Tests Passed
Run All Unit Tests
Next Task or Failed Acceptance Test
Pair Up
Create a Unit Test
Failed Unit Test
Passed Unit Test
Pair Programming
New Unit Tests
New Functionality
Continuous Integration
Run Failed Acceptance Test
Simple Code
Complex Code
Refactor Mercilessly
Acceptance Test Passed

Content

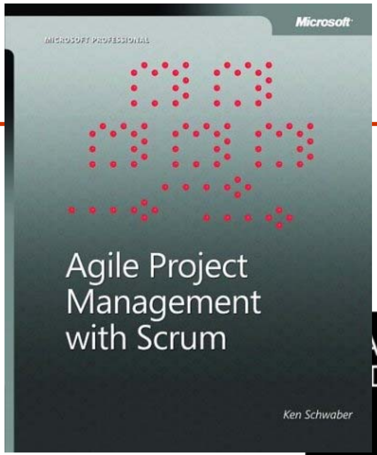
Scrum in 100 words

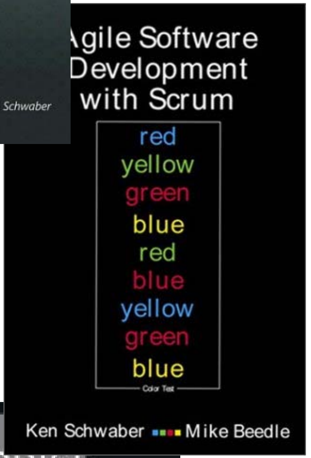
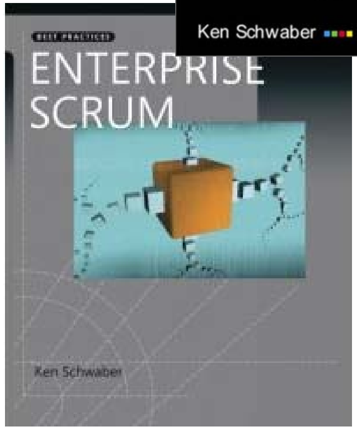
- Scrum is an agile process that allows us to focus on delivering the highest business value in the shortest time.
- It allows us to rapidly and repeatedly inspect actual working software (every two weeks to one month).
- The business sets the priorities. Teams self-organize to determine the best way 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




Agile Software Development with Scrum
red yellow green blue red blue yellow green blue
Ken Schwaber Mike Beedle


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特点

- 自我管理的团队
- “sprint”为周期迭代的产品开发
- 一系列“产品 Backlog”记录了产品需求
- 没有特定的工程实践惯例
- 在以生成规则创造的敏捷开发环境交付产品
- 是其中一种“敏捷方法”

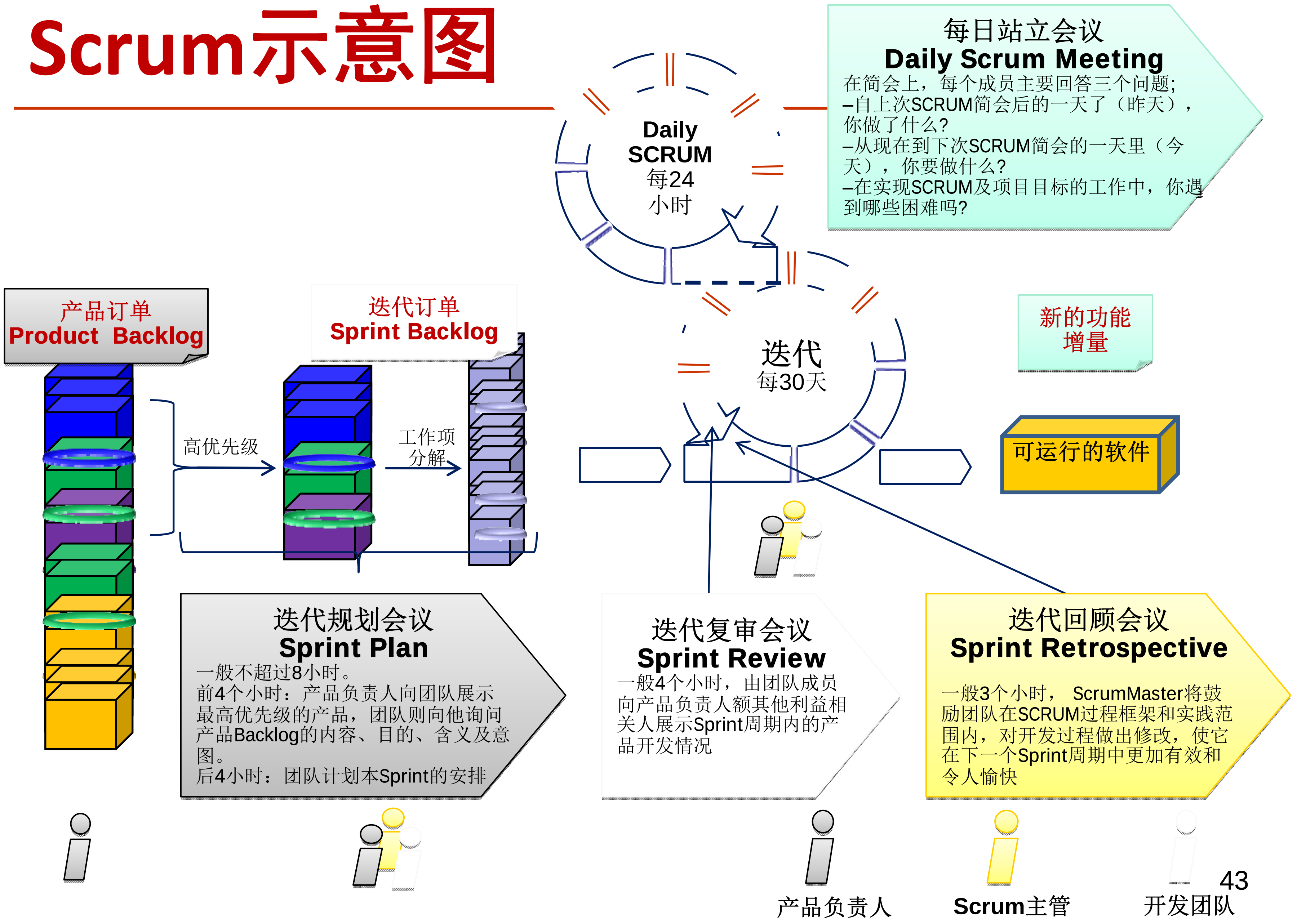
SCRUM 开发流程

- 基本假设是『开发软件就像开发新产品，无法一开始就能定义 Final Product 的规程，过程中需要研发、创意、尝试错误，所以没有一种固定的流程可以保证项目成功』
- Scrum 将软件开发团队比拟成橄榄球队
  - 有明确的最高目标
  - 熟悉开发流程中所需具备的最佳典范与技术
  - 具有高度自主权
  - 紧密地沟通合作
  - 以高度弹性解决各种挑战
  - 確保每天、每个阶段都朝向目标有明确的推进
- 因此 SCRUM 非常适用于产品开发项目。

SCRUM 开发流程

- 通常以 30 天为一个迭代周期（Sprint）
- 由客户提供新产品的SRS开始，开发团队与客户于每一个阶段开始时挑选该完成的规格部份，开发团队必须尽力于 30 天后交付成果
- 团队每天用 15 分钟开会检查每个成员的进度与计划，了解困难并设法排除，决定第二天的任务安排 .
- SCRUM特别强调开发队伍和管理层的交流协作。每天，开发队伍都会向管理层汇报进度，如有问题会向管理层求助。

Scrum示意图


北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY
每日站立会议
Daily Scrum Meeting
在简会上，每个成员主要回答三个问题;
–自上次SCRUM简会后的一天了（昨天），你做了什么?
–从现在到下次SCRUM简会的一天里（今天），你要做什么?
–在实现SCRUM及项目目标的工作中，你遇到哪些困难吗?
Daily SCRUM 每24小时
产品订单 Product Backlog
迭代订单 Sprint Backlog
高优先级
工作项分解
迭代 每30天
新的功能增量
可运行的软件
迭代规划会议
Sprint Plan
一般不超过8小时。
前4个小时：产品负责人向团队展示最高优先级的产品，团队则向他询问产品Backlog的内容、目的、含义及意图。
后4小时：团队计划本Sprint的安排
迭代复审会议
Sprint Review
一般4个小时，由团队成员向产品负责人额其他利益相关人展示Sprint周期内的产品开发情况
迭代回顾会议
Sprint Retrospective
一般3个小时， ScrumMaster将鼓励团队在SCRUM过程框架和实践范围内，对开发过程做出修改，使它在下一个Sprint周期中更加有效和令人愉快
产品负责人
Scrum主管
开发团队

Sprint

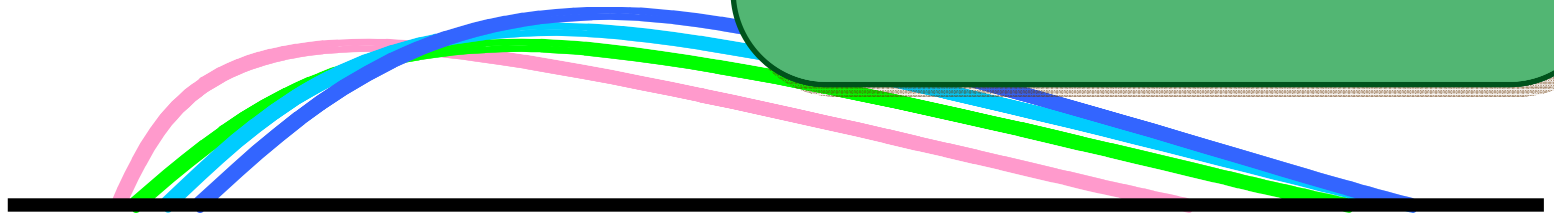
- 交付产品的固定的时间箱，建议2-3周
- 一次迭代包括design, coding, testing, and documentation
- 一旦迭代开始，仅 Scrum Team 能增加或者删除Sprint backlog中的items
- 当迭代的目标变的没有意义的时候，才能终止迭代开发
- 对于Product Backlog，不允许迭代内的需求变更：需求优先级调整仅存在于迭代之间

顺序 vs. 迭代开发过程

需求　设计　代码　测试

Scrum并非以一段时间集中完成一个过程

...而是将所有过程中必须的每一部分集中在这段时间内完成



北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

Scrum 框架

Roles
- Product owner
- ScrumMaster
- Team

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

Artifacts
- Product backlog
- Sprint backlog
- Burndown charts

Scrum Framework

Roles

- Product owner
- ScrumMaster
- Team

nies

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

Artifacts

- Product backlog
- Sprint backlog
- Burndown charts

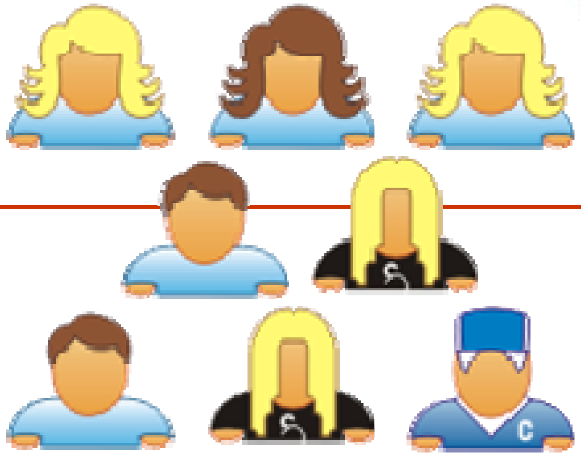
Product Owner (PO)

- Backlog优先级制定、开发版本规划
- 必须确保驱动开发的SRS只有一份
- 受管理层,客户等影响但仅PO有权调整Backlog
- 与相关成员协同工作来确定Product Backlog的items
- 消除关于Backlog的疑惑,确保理解一致,消除干扰

Scrum Master

- 项目管理的代表
- 确保SCRUM的成功
- 实施 SCRUM
- 保护团队以免受干扰

Scrum Team



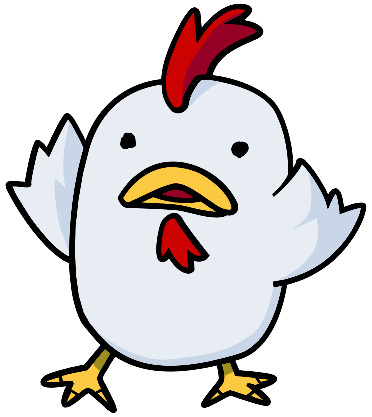
- 5-10人
- 自组织的团队
- 跨功能团队没有角色区分 (QA, Programmers, UI Designers, etc.)
- 全职、对工作交付负责
- 只有sprint之间能换团队
- 只要交付工作需求, 授权做任何事情.

Scrum角色及职责

火腿鸡蛋！
三思过后我决定不和你开餐馆了。因为我全身心投入，而你只牵涉入内！
BINGO!

Chickens & Pigs

- **Pigs:** Scrum Team 成员: 他们承诺对迭代目标交付负责



- **Chickens :** 项目组有关的成员,但并不专注于项目
- 以观察者的形式参加Scrum meeting

BEIJING INSTITUTE OF TECHNOLOGY

Scrum Framework

Roles

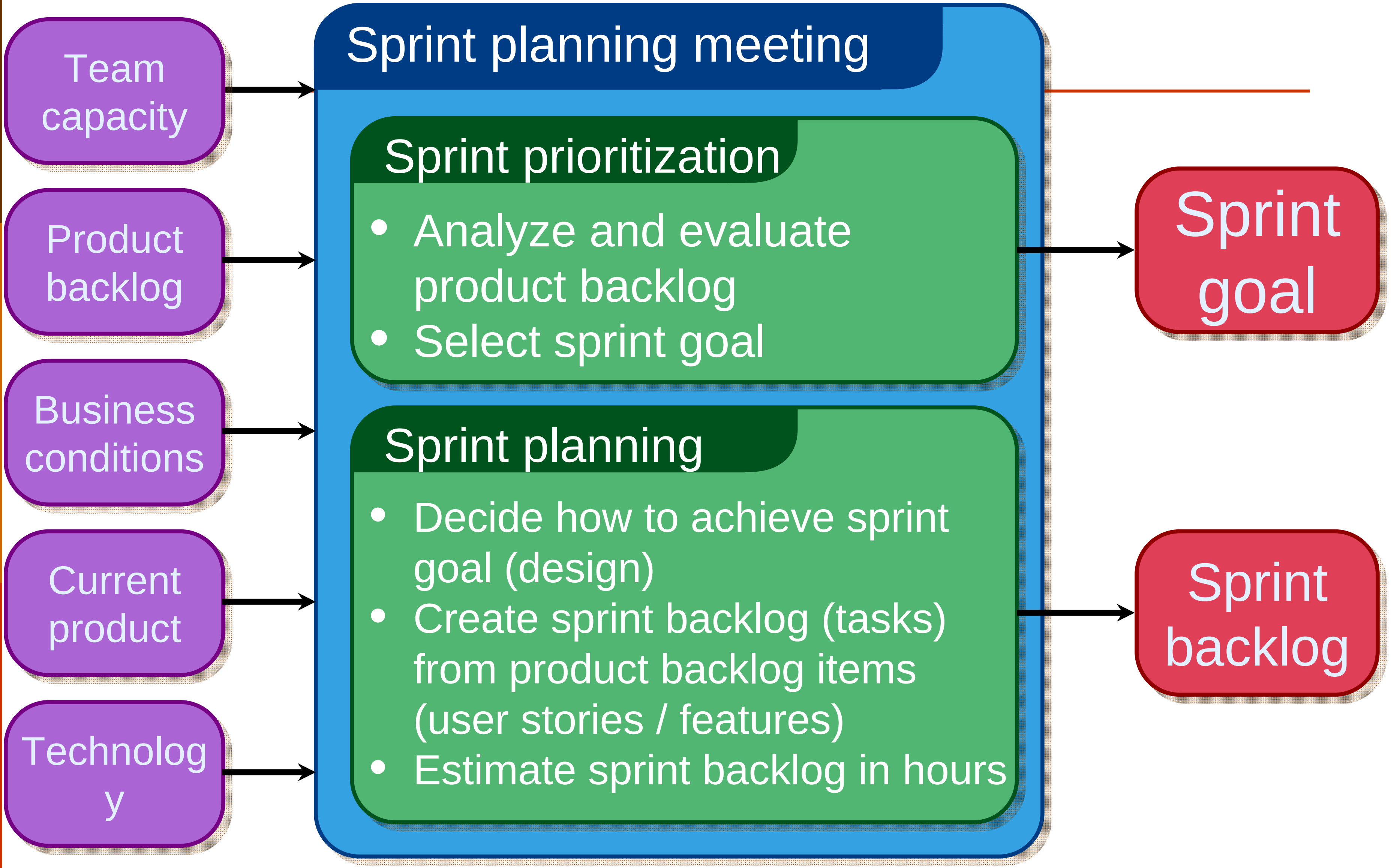
- Product owner
- ScrumMaster
- Team

Ceremonies

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

Artifacts

- Product backlog
- Sprint backlog
- Burndown charts



Sprint planning meeting

Sprint prioritization

- Analyze and evaluate product backlog
- Select sprint goal

Sprint planning

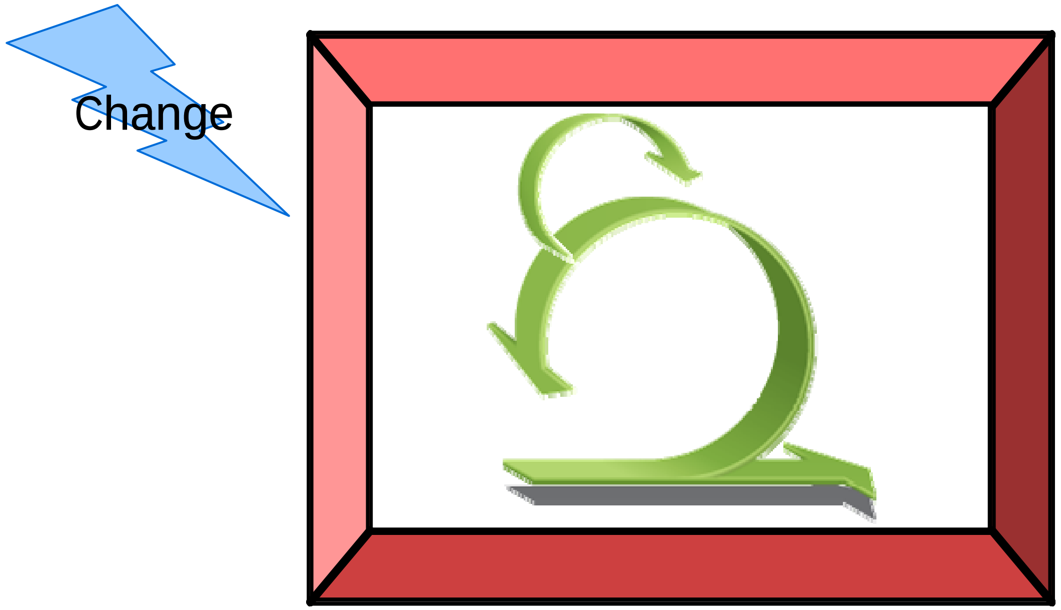
- Decide how to achieve sprint goal (design)
- Create sprint backlog (tasks) from product backlog items (user stories / features)
- Estimate sprint backlog in hours

如何定义Sprint

- 团队自己从产品的backlog中选择一些他们能够完成的任务作为此次迭代的backlog
- 任务被确认并且每一任务估计工作量应该在1-16小时左右
- 迭代的backlog的确定是团队协作的结果，而不只由scrum master决定
- 概要设计已经讨论过

编写后台和中间层(8 小时)
编写界面(4小时)
编写测试用例(4小时)
写类foo(6小时)
更新性能测试用例(4小时)

No changes during a sprint



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

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

Daily Scrum

Scrum
过程

在立会上，每个团队成员需要回答以下三个问题：
- 从上次会议到现在你完成了哪些工作？
- 下次每日站会之前准备完成什么？
- 工作中遇到了哪些障碍？

团队成员移动任务板上的贴纸，将贴纸置于任务板相应的位置下。当一条item完成时，挑选新的item。

- 会议：每日立会

- 每天15分钟
- 团队面对面站立成圈
- 为项目信息同步可视化
- 避免无关的讨论

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

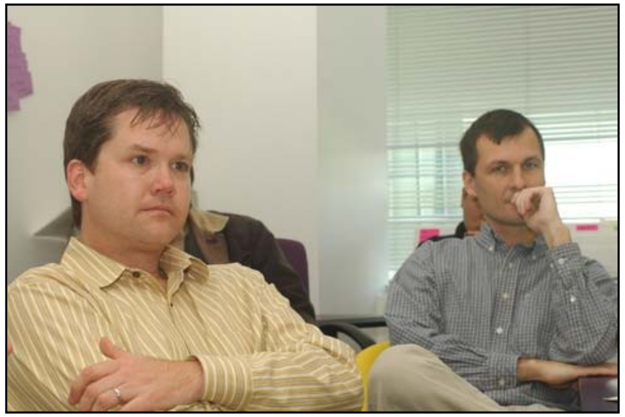
团队成员需要回答3个问题

1. 昨天你做了什么?
2. 今天你将要做什么?
3. 你有需要帮助的地方吗?

- 对于 ScrumMaster来说这些问答不是工作进度报告
- 他们是团队成员彼此的承诺

迭代结果的验收

- 团队需要演示所完成的迭代工作
- 使用演示形式展示新功能或者底层架构的实现
- 非正式
  - 2小时的提前准备
  - 不需要正式演示文档
- 整个团队都需要参加
- 邀请所有关注产品的人参加



有关迭代的回顾

- 周期性的回顾，总结工作中的经验和教训
- 一般 15–30 分钟
- 在每个迭代结束时开始做
- 整个团队都需要参加
  - ScrumMaster
  - 产品所有者
  - 团队
  - 可能还包括客户

Start / Stop / Continue

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

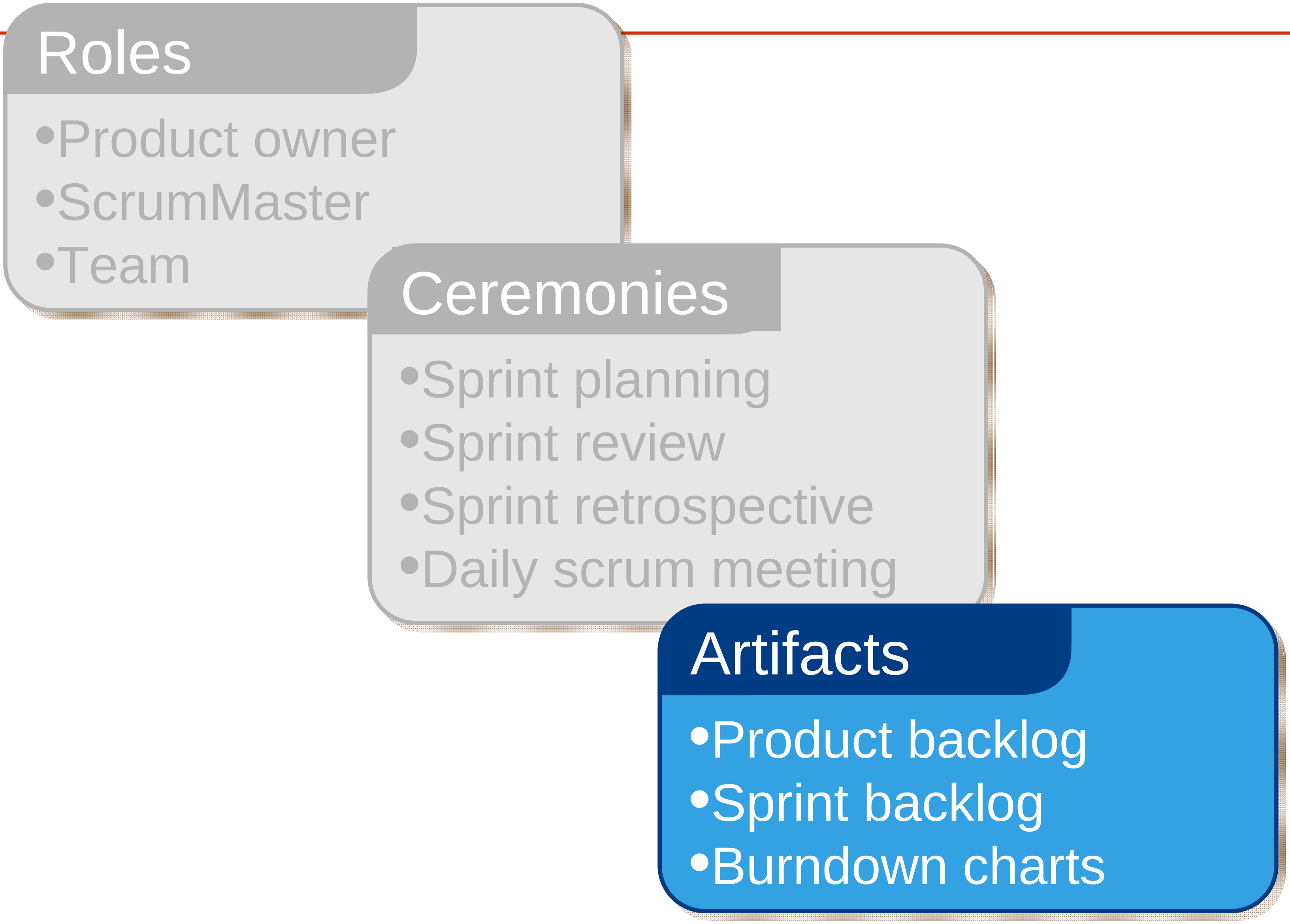
Start doing

Stop doing

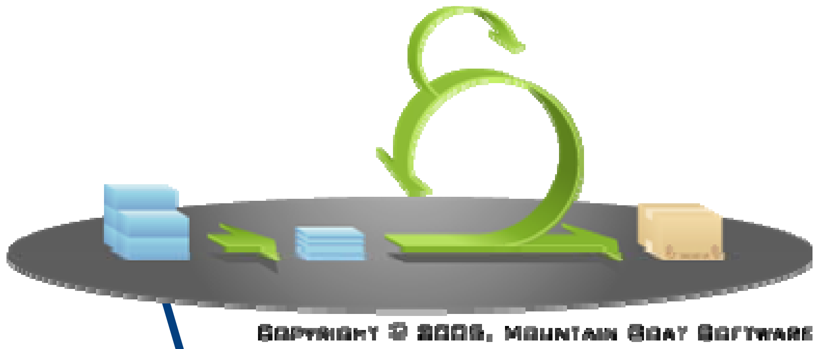
Continue doing

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

Scrum Framework



产品 Backlog



一组产品 backlog

- 需求
- 项目中待完成的工作列表
- 理想的是每一个待完成的工作都将对客户和用户产生价值
- 产品所有者将对这个列表进行优先级排序
- 每个迭代开始前优先级的排序工作还需要再度修正

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

实例：产品 Backlog

Backlog 列表	估计工作量
顾客可以酒店预定	3
顾客可以取消预定.	5
顾客可以提前更改预定的日期.	3
酒店工作人员可以出具RevPAR(revenue-per-available-room)报告	8
提高对突发事情的处理能力	8
…	30
…	50

管理迭代的 Backlog

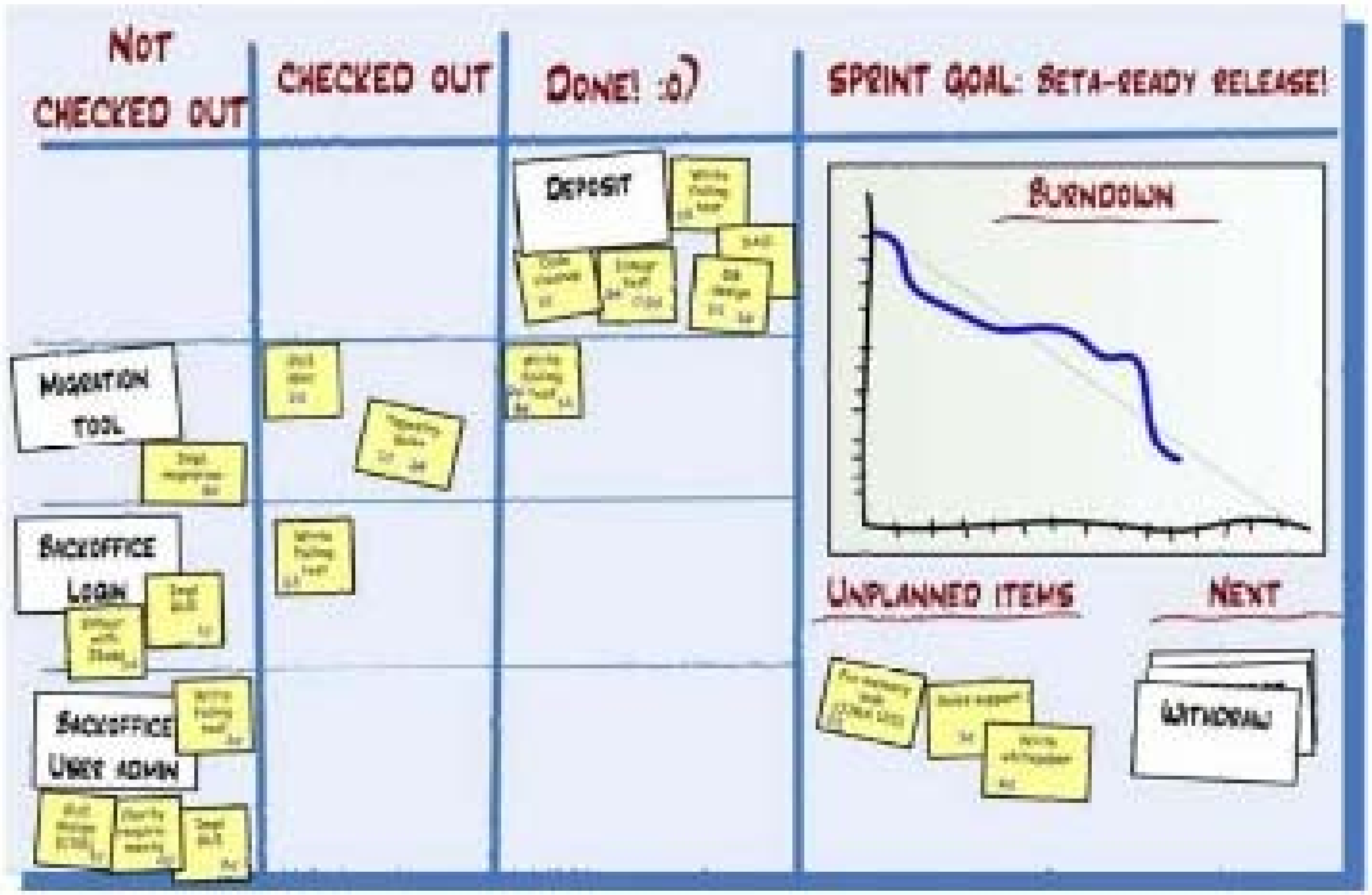
- 团队个人将要签收自己的工作
  - 工作不是单向的分配
- 每天更新个人剩余工作量估计
- 团队中任何人都可以添加、删减或者更改迭代中的工作项目
- 为了迭代目标以及将发布的结果而工作
- 如果对将要面对的困难不清楚，最好先定义一个相对工作量较大的工作项目然后适时在以后将其分散成较小额工作量的几个部分

实例：迭代Backlog

任务	Mon	Tues	Wed	Thur	Fri		
编写用户界面	8	4	8				
编写中间层	16	12	10	4			
测试中间层	8	16	16	11	8		
编写在线帮助	12						
编写Foo类	8	8	8	8	8		
增加对错误的日志记录			8	4			

Burn Down 迭代燃尽图表

- 工具：任务板

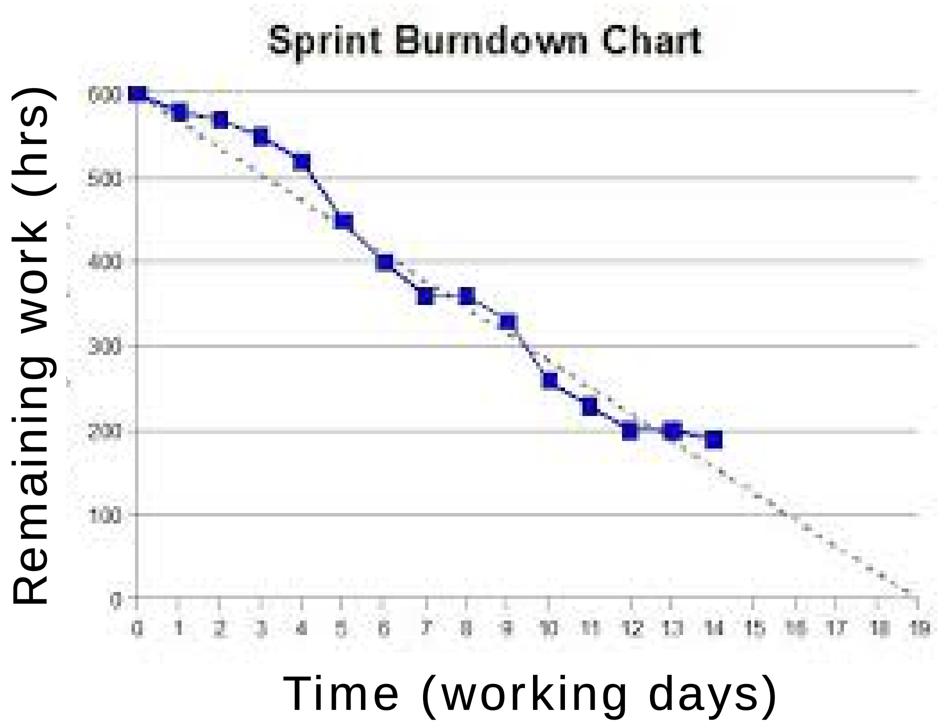


Burn down Charts

- 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

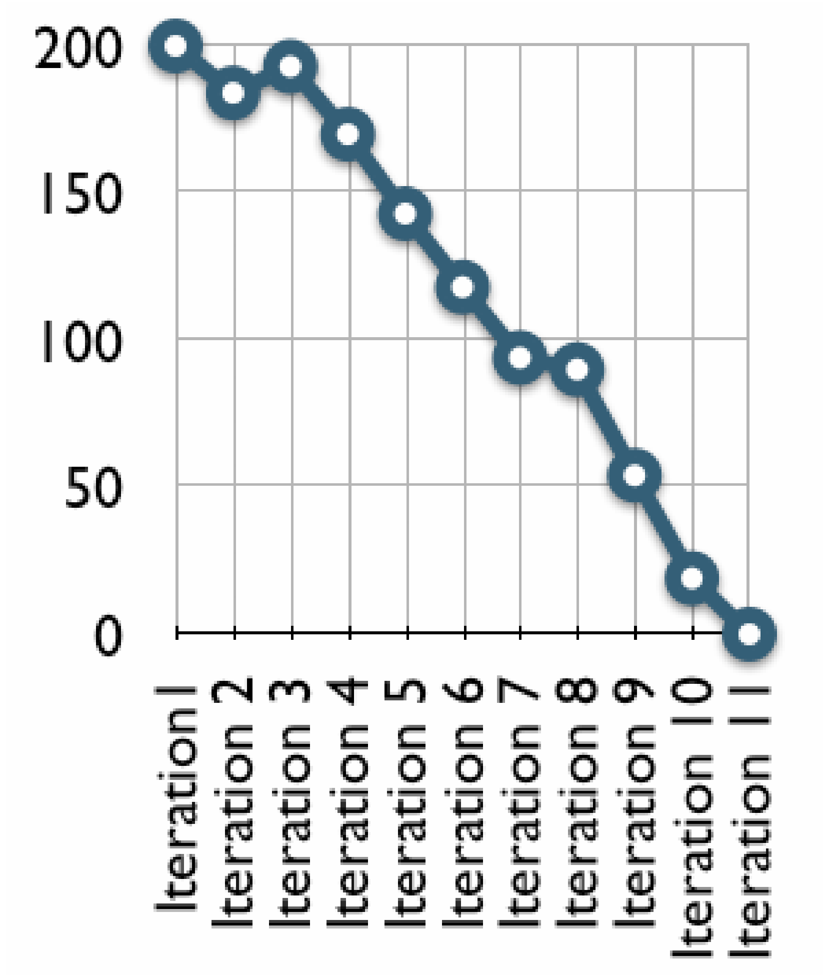
Sprint Burn down Chart

- 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



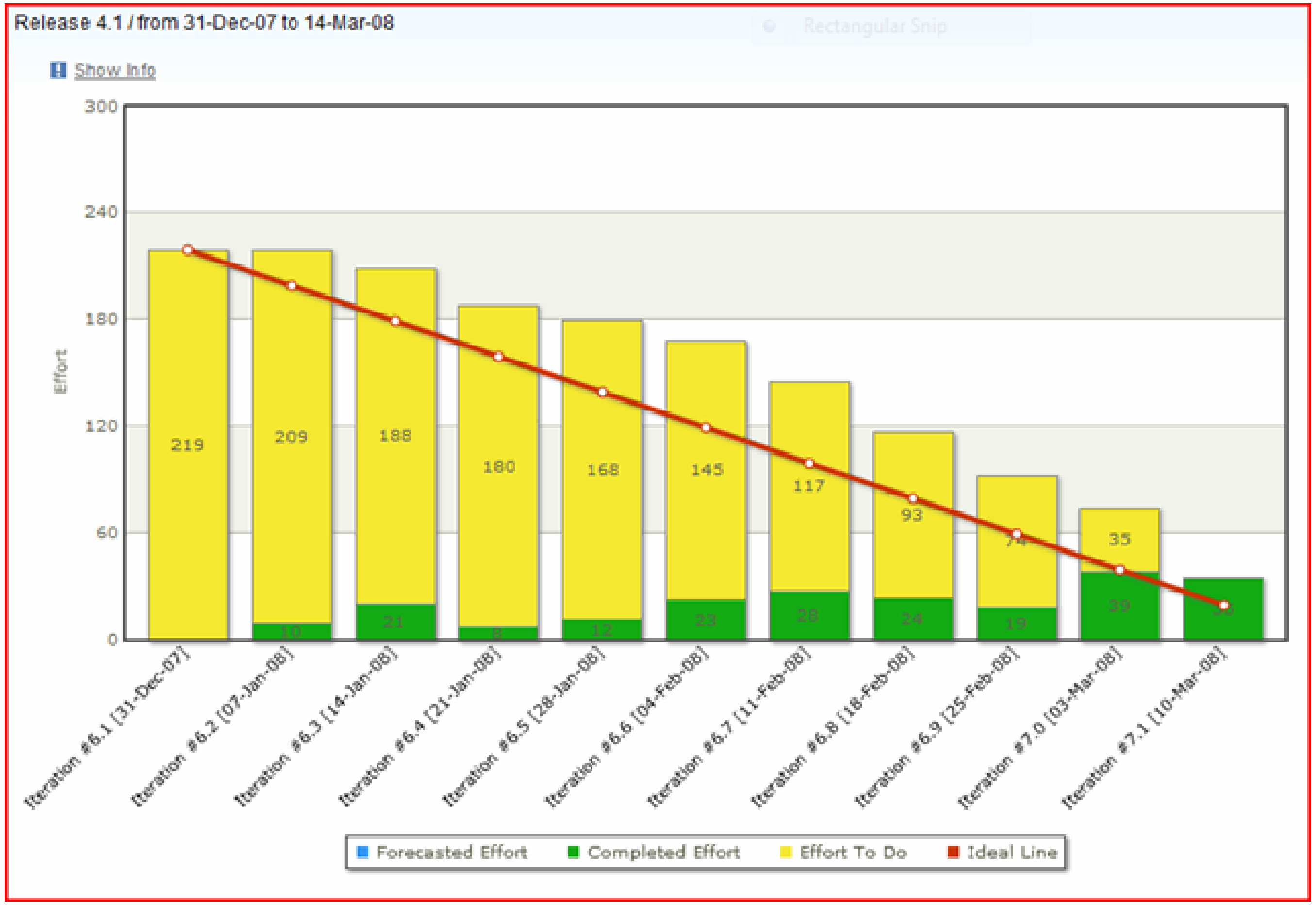
Release Burn down Chart

- 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

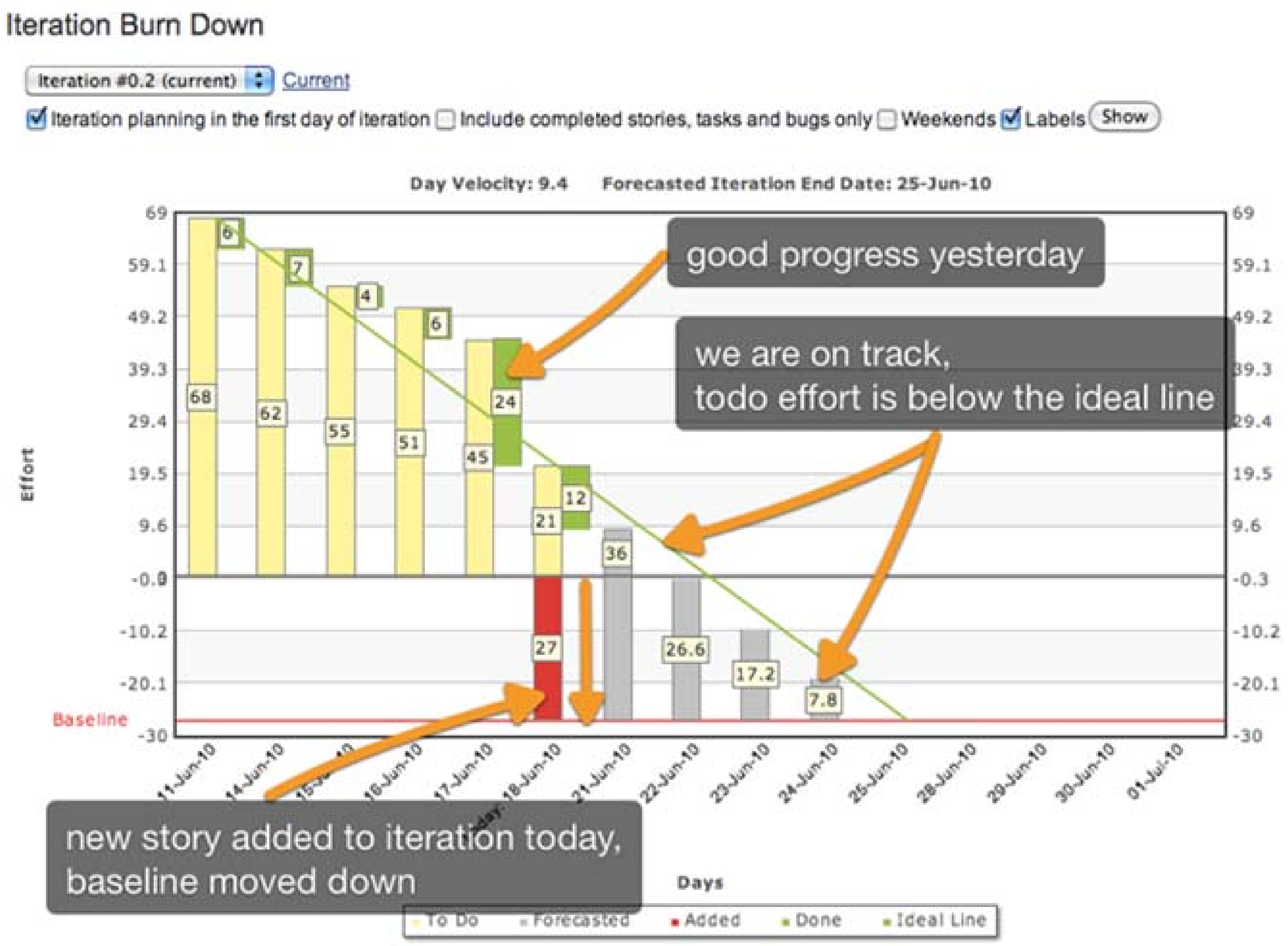


Alternative Release Burn down Chart

- Consists of bars (one for each sprint)



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


Iteration Burn Down

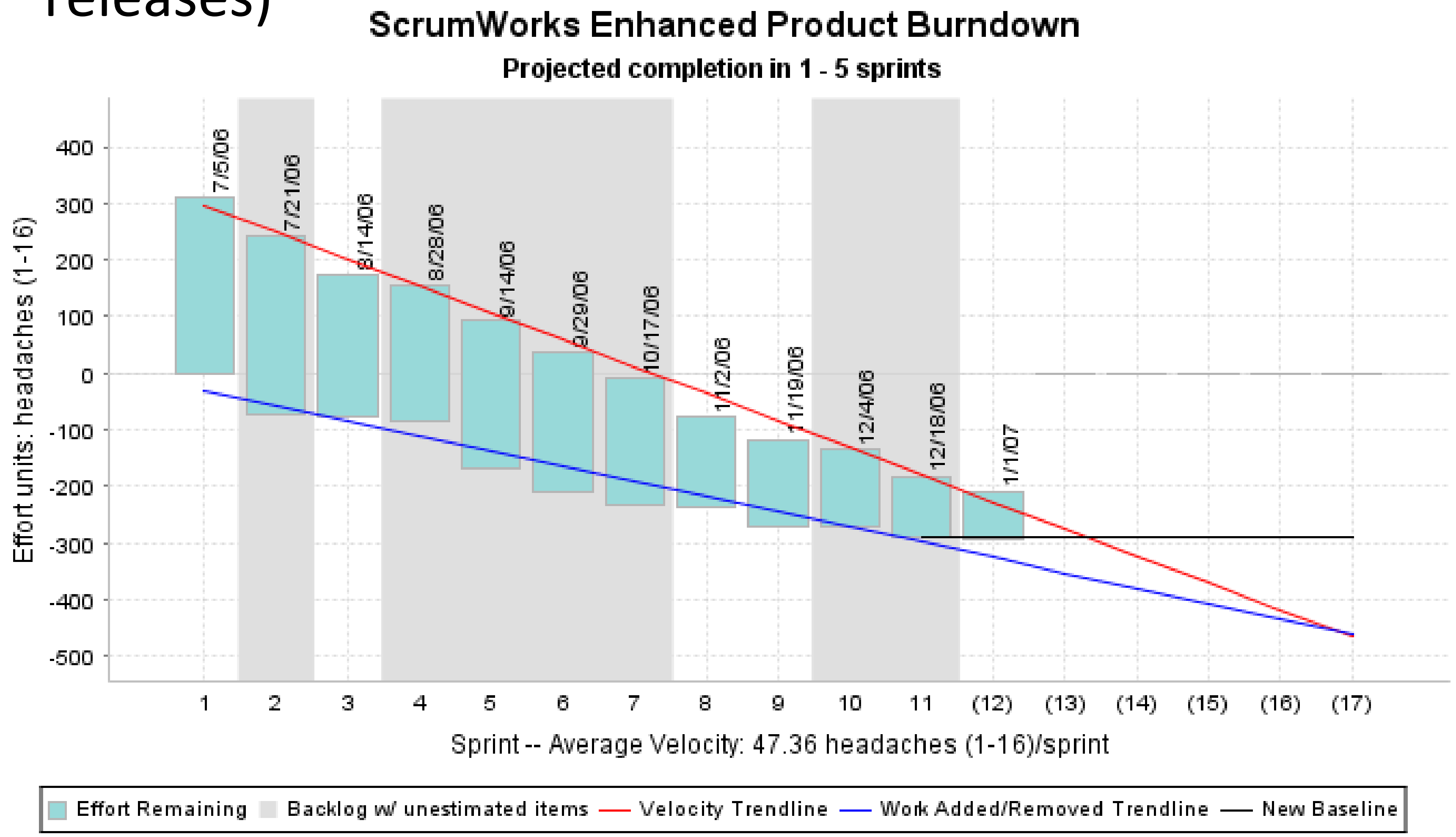
Iteration #0.2 (current) Current

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

Day Velocity: 9.4 Forecasted Iteration End Date: 25-Jun-10

Product Burn down Chart

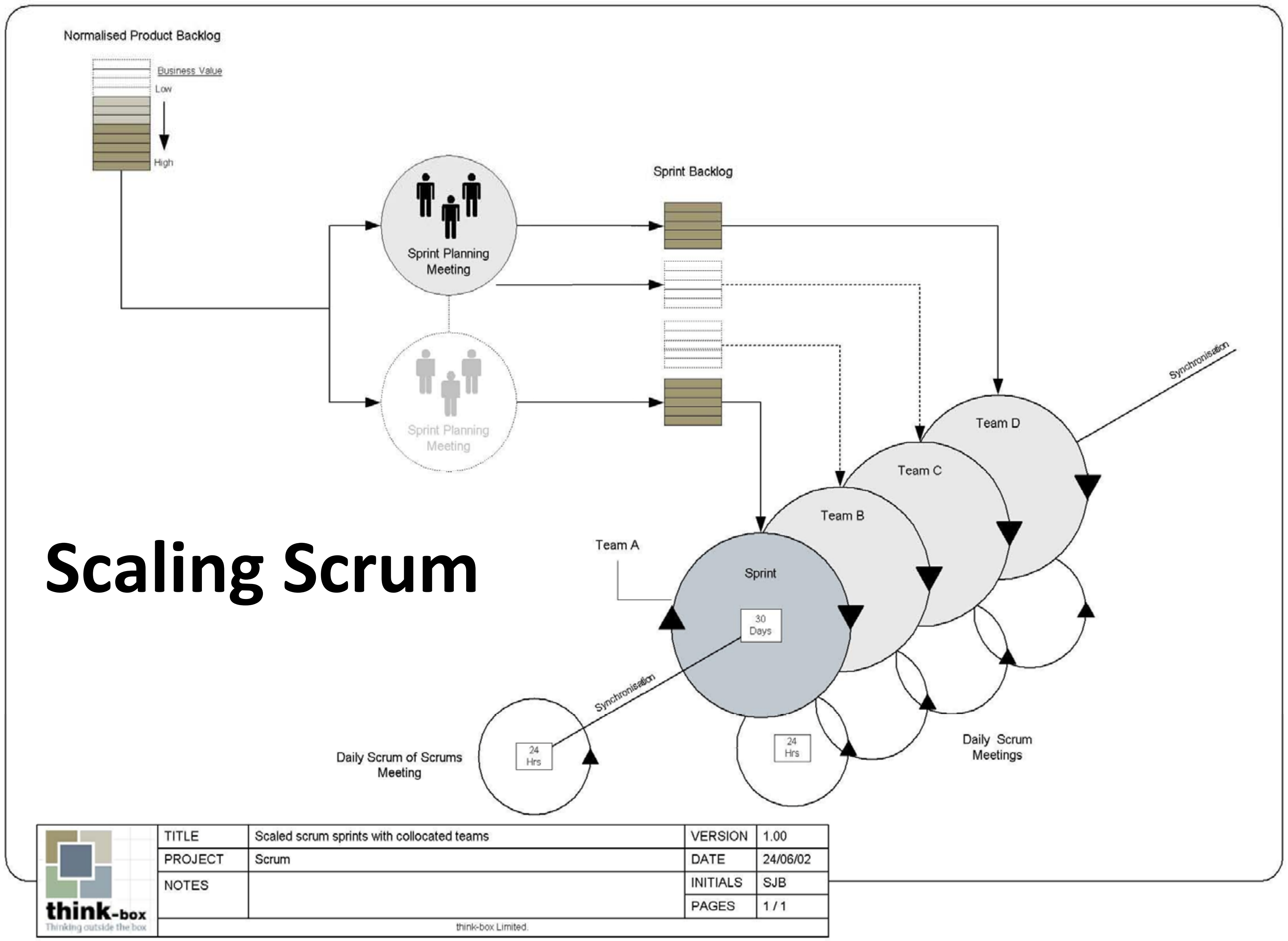
- Is a "big picture" view of project's progress (all the releases)



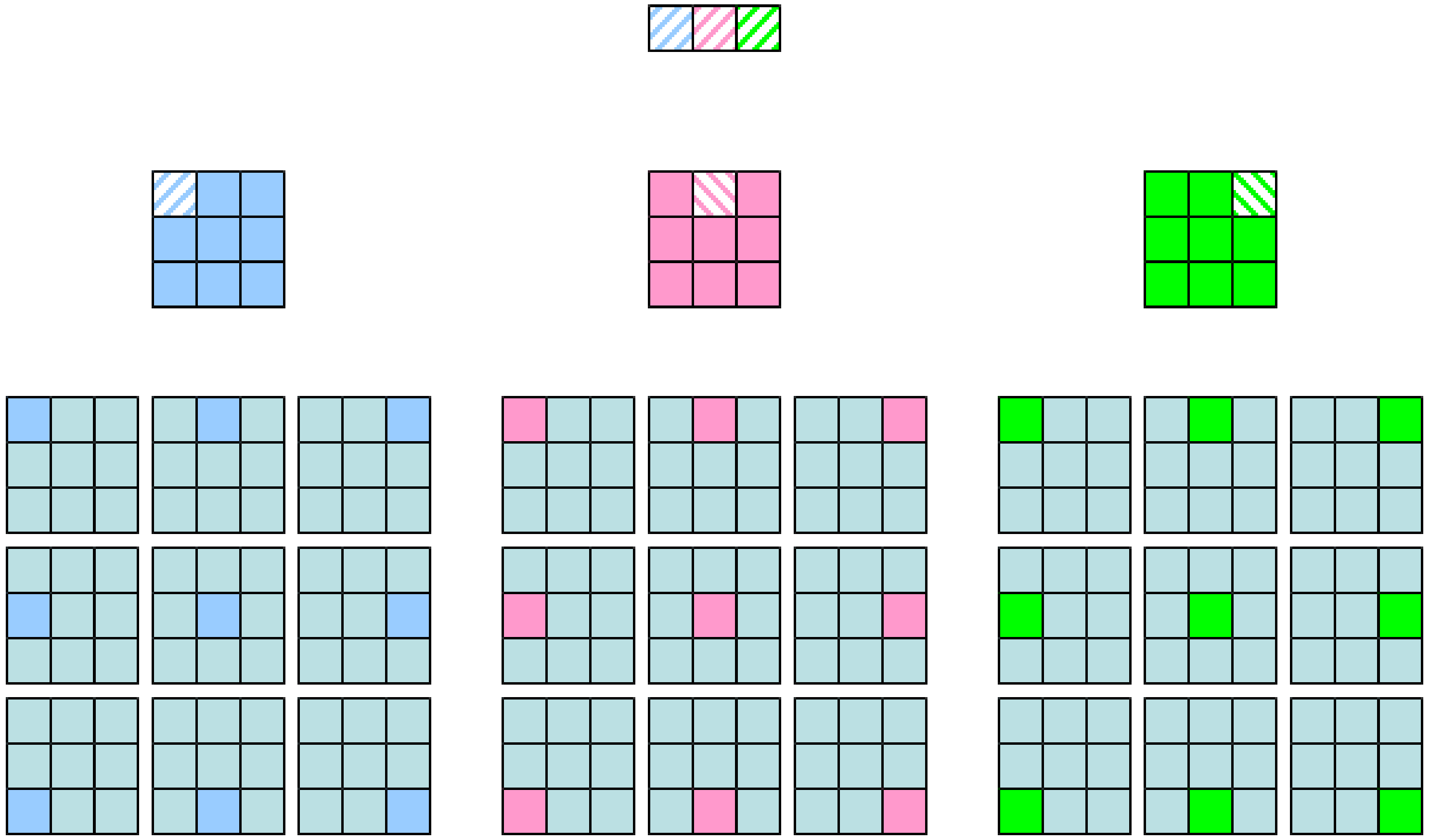
Scaling Scrum

- 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 (Δ) 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 , Stackeholder 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 diveded •Estimated Work remaining is updated daily
Sprint Burndown	•shows work remaining for the current sprint •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

Backup

eXtreme Programming

XP - eXtreme Programming

- 极限编程，最轻量级的开发流程，其最主要的精神是『在客户有系统需求时，给予及时满意的可执行程序』，所以最适合需求快速变动的项目
- 强调客户所要的是 workable 的执行码，所以把与撰写程序无关的工作降至最低，并要求客户与开发人员最好以 side-by-side 的方式一起工作

XP强调4个因素

- 交流(communication)，XP要求程序员之间以及和用户之间有大量而迅速的交流
- 简单（simplicity），XP要求设计和实现简单和干净
- 反馈（feedback）通过测试得到反馈，尽快提交软件并根据反馈修改
- 勇气（courage）。勇敢的面对需求和技术上的变化

XP 开发流程

- 开发人员随时可以和客户进行有效沟通，撰写 user stories 以确认需求。
- 简易快速的系统设计，撰写独立的验证程序以解决特殊困难的问题，找出算法即可丢弃验证程序。
- 规划多次小型阶段的项目计划，以最快速度完成每一阶段的程序交付客户，客户负责 Acceptance tests；
- Coding 前必须完成 Unit Test 与 Acceptance tests 程序，所有模块整合前都须经过 Unit Tests；
- 开发人员必须快速响应 Bug 与需求变更；
- 要求二人一组使用一台计算机设计程序，当一人 coding 时，另一人负责思考与设计；
- 程序必须符合程序规范，并常做程序的重构 (Re-factoring)。

XP原则和实践-Planning-user stories

- user stories
  - User stories类似use case， 描述用户所见的系统功能，但避免使用大量的文档，user stories由用户编写（不仅限于描述用户界面）。User stories使用用户的语言编写，不使用技术性的语言，每个user stories限于几句话。User stories用于在release plan会议上对开发时间进行评估，也用于产生验收测试（acceptance test），必须使用可以自动进行的验收测试保证软件的正确性。User stories与传统的用户需求的区别在于详细的程度，user stories并不会确定需求的每个细节，它只是用来简单的描述系统功能，供开发人员进行估计开发进度，在开发过程中开发人员和用户会不断的交流以讨论细 节问题。User story应该专注于功能，不应该过分注重用户界面等细节。一般一个user storiy在1-3周的时间完成，如果估计超过3周，说明user story太大了，需要细分 .

XP原则和实践-Planning-release plan

- release plan
  - 召开一个 release plan会议，产生release plan。Release plan将用于指定每个iteration的计划。开发人员对每个user story估计开发时间（在不被打断，无其他工作情况下的开发时间，包括测试），用户对user stories给出优先级，release plan会议并不制订每个iteration的计划。Release plan要用户，开发人员和管理者都同意，在完成功能范围（scope），使用资源（resource）,时间（time）和质量(quality)上达 成一致（一般质量是不能改变的）

XP原则和实践-Planning-small release

- small release
  - often and small release是XP的一个原则，每个release完成一些用户有意义的功能集合，尽快的提交给用户以获得反馈，及时调整，提交的越晚，调整越困难。

XP原则和实践-Planning-project velocity

- project velocity
  - 团队在开发过程中要收集数据，以便于对自己的开发速度进行评估，用于以后的release plan

XP原则和实践-Planning-iteration

- iteration
  - 每个small release的周期称为iteration，每个iteration约为1-3周，在一个项目中保持每个iteration的时间相等，不要超前制定计 划，每个iteration的计划在iteration的开始时制定。这样能够更灵活的应付变化。不要急于完成本次iteration没有包括的功能。要注重每个iteration的时间限制，当团队觉得不能按时完成iteration时，召开一次iteration plan会议，重新评估，减少一些user stories。

XP原则和实践-Planning-iteration plan

- iteration plan
  - 在每个 iteration开始的时候召开会议，从release plan中选择还没有实现的用户最迫切需要的user stories。上一个iteration中没有通过验收测试的功能也要在这个iteration中实现。可以根据上一个iteration的实践调整团 队速度。User stories和失败的测试被分解成programming task，task使用技术语言编写，作为iteration plan的详细描述。程序员主动领取task并估计完成时间，每个task应该在1-3天内完成，超过3天的task应该被细分。如果整个团队需要的时间 多于或少于规定的iteration时间，调整user stories。

XP原则和实践-Planning-move people around

- move people around
  - 不要使每个开发人员局限于一项工作，不要使某项工作依赖于一个开发人员，增加知识共享，减少信息孤岛，多进行交流和培训。当项目中的所有人对所有模块都了解并可以进行开发时是效率最高的，鼓励开发人员在不同iteration中开发不同的模块。

XP原则和实践-Planning-stand-up meeting

- stand-up meeting
  - 每天工作开始之前，开5-10分钟的stand-up会议（站立会议），站立的目的是为了强迫节省时间，会议的目的是交流，提出存在的问题，但不要在会议上解决问题。开一个所有人员参加的短会比多个个别人员参加的会议要高效。在会议上提出的问题可以由少数人讨论解决，这个少数人参加的会议如果涉及到代码，可以在计算机前讨论。

XP原则和实践-Planning-fix XP when it breaks

- fix XP when it breaks
  - XP并不是一成不变的，当团队觉得需要修改的时候，可以根据自己的情况进行修改，任何修改都要经过整个团队的讨论和达成一致

XP原则和实践-Designing-1

- Simplicity
  - 保持简单的设计，在完成同样的功能的情况下，选择简单的设计，不要急于设计没有计划的功能，应该认识到：keeping a design simple is a difficult job
- System metaphor
  - 使用统一的术语描述系统，在用户，管理者和开发人员之间使用统一的术语。这将使交流清晰。
- CRC card
  - 使用CRC (Class, Responsibilities, and Collaboration) card进行系统设计，鼓励更多的人参加设计。每个CRC卡片表示系统中一个对象，写上对象的名字，责任和每个责任需要交互的其他对象。可以模拟对象之间 的交互。CRC卡片是易于理解的，但是是非正式的，如果需要正式的文档，要将卡片转换为相应的文档。
- spike solution
- never add function early
- Refactoring whenever and wherever

XP原则和实践-Designing-2

- spike solution
  - 使用spike solution减低风险，当遇到技术难题或设计问题时，使用简单的程序进行测试，找出问题，在不同的解决方法之间进行评估。在早期进行实验可以有效的降低风险。
- never add function early
  - 不要过早的设计没有计划的功能，在一个需求经常变化的环境中，过早的设计经常是没有用的。
- Refactoring whenever and wherever
  - XP鼓励对设计和代码经常进行重构（Refactoring），目的是去除冗余，提高质量，保持设计简单。重构必须以完全测试为检验条件

XP原则和实践-Coding-1

- customer is always available
  - 用户是项目组的成员之一，用户的参加贯穿整个开发过程，用户与开发人员之间的交流是重要的
- coding standard
  - 使用统一的编码标准，这是保持代码整洁和共享代码的基础
- coding unit test first
  - test first是XP的一个特点，在编写代码之前先编写单元测试代码，单元测试代码和代码由同一个程序员完成。先编写测试代码可以使程序员更好的理解需求，避免直接编码造成的理解偏差，对需求的不清晰，可以在编写测试代码时就发现。测试代码也是检验程序是否完成的标准。编码工作应该是以下工作的循环：
    1 编写测试代码
    2 运行测试程序，确认失败
    3 编写实现这个测试程序要求功能的代码，不需要实现其他的功能，只需要实现刚刚满足测试程序的功能
    4 运行测试程序，确认成功
    实践证明，test first方式需要的编码实践少于先编码，后写测试代码

XP原则和实践-Coding-2

- Pair Programming
  - Pair programming是XP的特色，它要求两个程序员在一台计算机上同时进行编程工作。共用鼠标和键盘，通常一个人进行战略上的思考，程序架构和函数， 类之间的关系，一个人进行输入和战术上的思考，完成函数和类。两个人可以经常交换角色。
- sequential integration
  - 要保证源代码库的状态是可标识的，同一时间，只允许一个pair的程序进行整和和测试，这样可以缩小问题产生的范围。不同的pair可以在自己的机器上随时进行整和和测试.
- often integration
  - 只要有可能就进行代码整合，周期可以在几个小时，最好不要超过一天。

XP原则和实践-Coding-3

共同拥有代码

- 鼓励每个人对项目中的任何人提出新的想法，任何开发人员对项目中的任何代码都可以进行增加功能，改正错误和重构。

- 优化工作放在最后
  - 先使系统能够正常工作，不要猜测系统的瓶颈，要实际测量
- NO OT!
  - 不要长时间的加班，大多数加班并不能挽回已有的延迟，连续超过两个星期的加班说明有问题存在。向一个已经延迟的项目填加人员也不是一个好的选择。

XP原则和实践-Testing-1

- 所有的代码都有单元测试
  - 单元测试是XP的基石，XP中的单元测试应该是可以自动进行的，所以可以很快的进行所有的单元测试，单元测试应该在编码之前创建。单元测试的代码和代码一起release， 没有单元测试的代码不能够release。使用自动单元测试可以系统整合时节省大量的发现错误和改正的时间。单元测试越完善，节省的时间越多。
- 代码在release前必须通过所有的单元测试
- 发现bug后，首先增加测试
  - 在实际运行中发现bug,首先增加acceptance test，然后增加unit test，在增加完测试后在查找和修改代码，增加的测试保证同样的错误不会再出现
- acceptance test
  - acceptance test根据user stories在iteration plan会议上创建，它也应该可以自动运行以便可以经常运行。