# Agile Methods

Nguyen V. Vu

with some materials adapted from Boehm 2003

# Outline

- Agile Manifesto
- Extreme Programming
- Scrum
- Summary

# The Agile Manifesto

**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.**

# The Agile Manifesto (cont'd)

- Our highest priority is to **satisfy** the customer through **early and continuous delivery** of valuable software.
- **Welcome changing** requirements, even late in development
  - Agile processes harness change for the customer's competitive advantage.
- Deliver working software **frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale
- Business people and developers must **work together daily** throughout the project

# The Agile Manifesto (cont'd)

- Build projects around **motivated individuals**
  - Give them the environment and support they need, and trust them to get the job done
- The most efficient and effective method of conveying information to and within a development team is **face-to-face conversation**
- **Working software** is the primary measure of progress
- Agile processes promote **sustainable development**
  - The sponsors, developers, and users should be able to maintain a constant pace indefinitely

# The Agile Manifesto (cont'd)

- Continuous attention to **technical excellence and good design** enhances agility
- **Simplicity** – the art of maximizing the amount of work not done – is essential
- The best architectures, requirements, and designs emerge from **self-organizing teams**
- At regular intervals, the team **reflects** on how to become more effective, then tunes and **adjusts** its behavior accordingly

# Outline

- Introduction to Agile Methods
- Agile Manifesto
- Extreme Programming Method
  - Principles
  - The 12 practices
- Scrum Method
- Summary

# XP Principles

- Philosophy: *Take known good practices and push them to extremes*
- “If code reviews are good, we’ll review code all the time”
- “If testing is good, we’ll test all the time”
- “If design is good, we’ll make it part of everybody’s daily business”

# XP Principles (cont'd)

- "If simplicity is good, we'll always leave the system with the simplest design that supports its current functionality"
- "If architecture is important, everybody will work defining and refining the architecture all the time"
- "If integration testing is important, then we'll integrate and test several times a day"

# XP Principles (cont'd)

- "If short iterations are good, we'll make the iterations really, really short – seconds and minutes and hours, not weeks and months and years"
- "If customer involvement is good, we'll make them full-time participants"

# XP: The 12 Practices

- The Planning Game
- Small Releases
- Metaphor
- Simple Design
- Testing
- Refactoring
- Pair Programming
- Collective Ownership
- Continuous Integration
- 40-hour Week
- On-site Customer
- Coding Standards

✍ **Use these practices generatively, not imperatively**

# The Planning Game

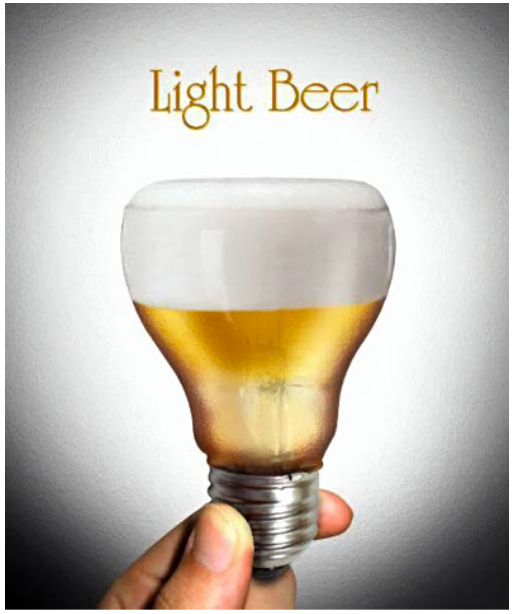
- Use stories to facilitate knowledge transfer
- Put decisions in the hands of the person with the best knowledge:
  - business decisions → Customer
  - software decisions → Developer
- Plan only as far as your knowledge allows
  - next iteration or next release

# Small Releases

- Supports quick feedback from users
- Simplify the tracking of metrics
  - stories per iteration → project velocity
- Increase the manageability of the project for the customer
  - But complicate user conservation of familiarity

# Metaphor

- Ground all discussions in a single shared story of how the whole system works
- Provide an overarching view of the project
- Connect program to work process



# Simple Design

- Design embodies only the needed complexity and no more
  - emphasis on top-down or bottom-up design as needed to meet this iteration's stories
  - extra complexity removed when discovered
- Simpler designs are easier to modify, maintain, and describe
  - decreases the cost of design changes
  - but no notion of product line architecture

# Testing

- Unit tests verify the programmer's work
  - must be done by programmer
  - constant testing makes finding new bugs faster and easier
- Functional tests verify that a story is complete
  - developed by customer
  - tests define functional requirements

# Refactoring

- Procedure for implementing iterative design
  - behavior-preserving
  - improves communication among developers
  - adds flexibility to the programming process

- Design is important – do it all the time
  - software development process is a design process
  - But redesign much more expensive for large systems

# Pair Programming

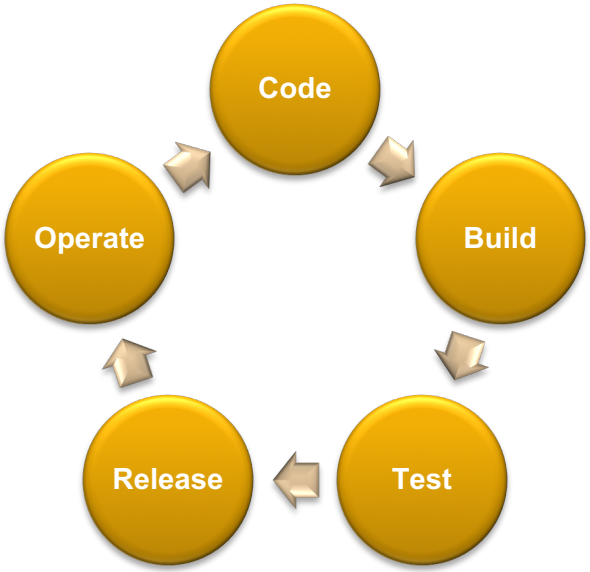
- All code is written by two programmers at a single machine
- Inspections are important, so do them all the time
- Increase implicit knowledge transfer
- Decrease cycle time, but increase effort

# Collective Ownership

- Everyone owns all of the code
  - anyone can change any code anywhere
  - no personal ownership of modules
  - no egoless programming either
- Everyone is permitted access to all the code so everyone has a stake in knowing all of the code (that they will work with)

# Continuous Integration

- The system always works
  - there is always something to be released
- Similar to rapid releases
  - fast feedback to developers on problems
  - no ‘big bang’ integration disasters

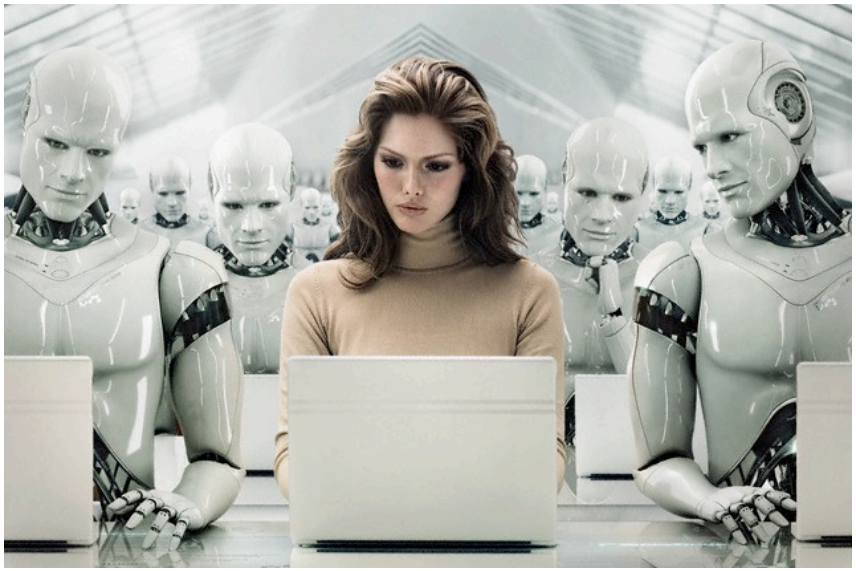


# 40-hour Week

- No heroes
- Knowledge can only be transferred at a limited rate
- Work for sustained speed, not a single sprint
  - never work overtime a second week in a row

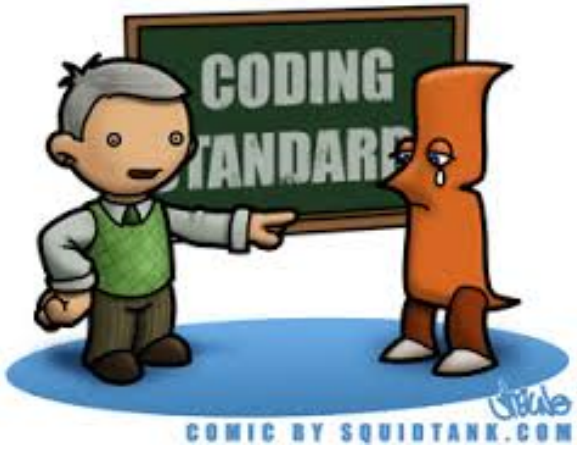
# On-site Customer

- A real, live user available full-time to answer questions as they occur
- Programmers don’t know everything
- Business knowledge is the key to a successful business project



# Coding Standards

- Communication occurs through the code
- Common standard promotes understanding of other developers' code
- Helps promote team focus



# Outline

- Agile Manifesto
- Extreme Programming Method
- Scrum
  - Concepts
  - Practices
- Summary

# What is Scrum?

- An agile method that employs a set of simple practices and rules to incrementally develop products
- Developed by Ken Schwaber and Jeff Sutherland

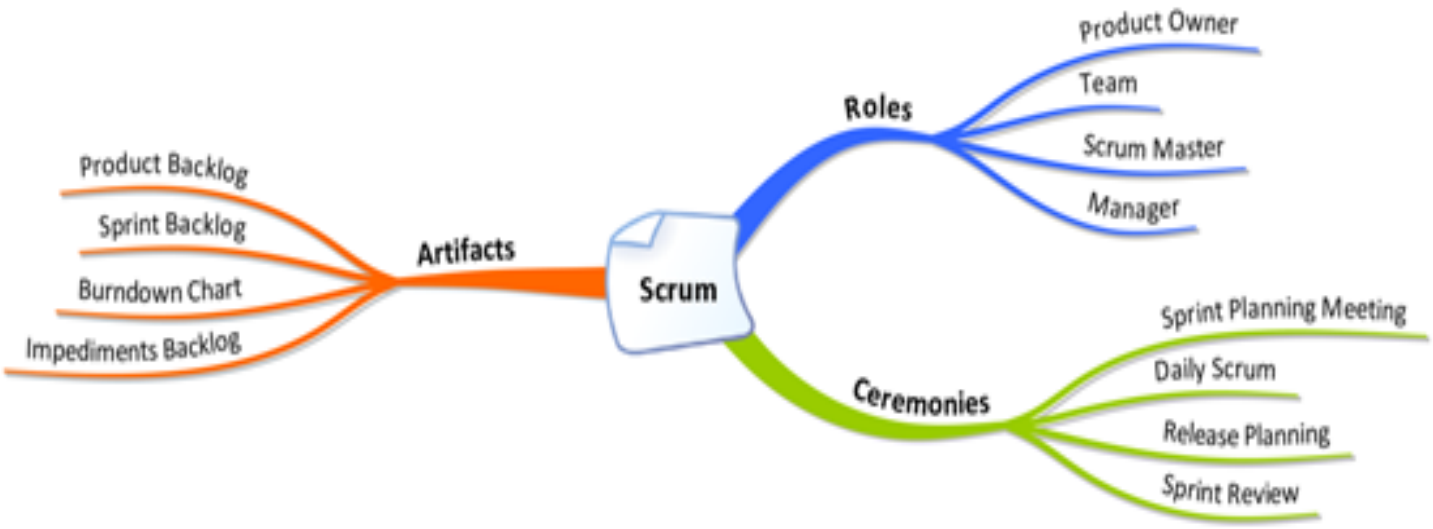
# Scrum Concepts

- Daily Scrum
  - a short daily meeting (less than 30 minutes) for the team to monitor status and communicate problems
- Sprint
  - a development cycle (typically 30 days)
- Backlog
  - Product backlog: prioritized list of product requirements
  - Sprint backlog: prioritized list of requirements allocated to the sprint
  - Impediments backlog: list of issues
- Burndown chart
  - chart showing the progress (backlog items completed)

# Scrum Process

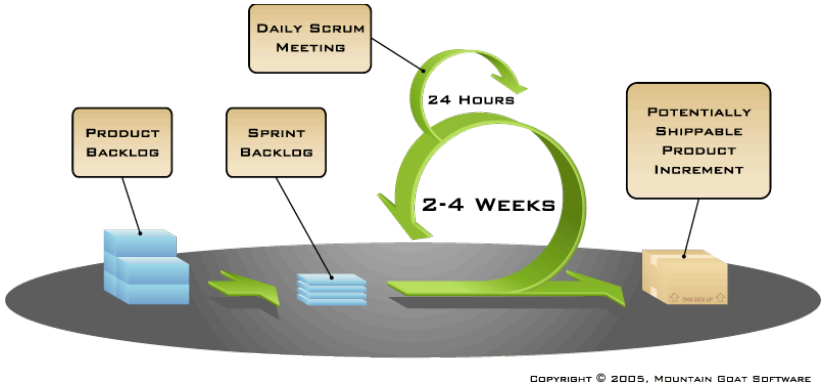

(source: controlchaos.com)

# Scrum Process (cont'd)



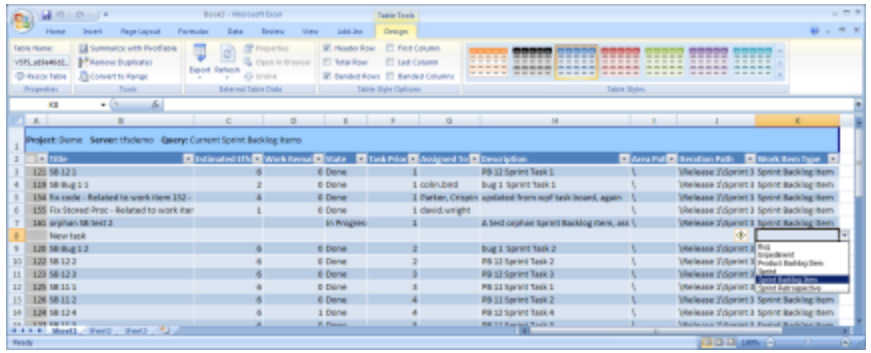
# Artifacts

| # in PM-Pres. | Official P/B-Area Description | PM Rank | IT Rank | Total Rank | Business Value |
|---|---|---|---|---|---|
| 5.1 | Process E | 4 | 4 | 1 | 25 |
| 2.4 | Process B | 6 | 1 | 1 | 25 |
| 2.1 | Process X | 7 | 1 | 3 | 15 |
| RQ0074 | Process D | 2 | | 2 | 15 |
| PJ0060 | Process M | 2 | | 2 | 2 |
| 2.2 | Process N | 8 | 1 | 3 | 25 |
| 3.1/3.3 | Prozess S | 1 | 1 | 5 | 80 |
| 3.6 | Process A | 2 | 2 | 6 | 60 |
| 6.2 | Process Y | 1 | 1 | 4 | 15 |
| 3.1/3.3 | Subprocess S.1 | | 1 | 5 | 50 |
| 3.1/3.3 | Subprocess S.2 | | 1 | 5 | 50 |
| 3.1/3.3 | Subprocess S.3 | | 1 | 5 | 35 |
| 3.1/3.3 | Subprocess S.4 | | 1 | 5 | 51 |
| 3.1/3.3 | Subprocess S.5 | | 1 | 5 | 45 |




## Product Backlog:

- Overall product requirements
- Items are estimated and prioritized



## Sprint Backlog:

- Requirement items allocated for one sprint
- Items are estimated prioritized

# Scrum Roles


- Facilitate Scum practices
- Enforce Scrum principles

Team

- Estimate Product backlog
- Create Sprint backlog
- Implement backlog items

- Create vision and requirements
- Contact point
- Create releases plan

Product Owner

- Ensure resources available
- Resources management
- Team building

Manager

# Activities

- Release planning
  - Prioritize and allocate features to releases
- Sprint planning
  - Set goal and allocate backlog items to Sprint
- Daily Scrum
  - Review daily status
- Sprint review
  - Held at end of Sprint
  - Review features completed
  - Decide what went wrong, right

# Outline

- Introduction to Agile Methods
- Agile Manifesto
- Extreme Programming Method
- Scrum Method
- Summary
  - Strengths and Weaknesses

# Where Agile Methods Work Best?

- Small teams (2 – 20 people)
- Short iterations (1 – 2 weeks)
- Collocated teams – not recommended for distributed teams
- Non safety-critical systems

# Strengths and Weaknesses of Agile Methods

- Strengths
  - Suitable for dynamic, chaotic environments
  - Supportive of emergent requirements, rapid change
  - Avoiding documentation overhead
  - Empowering/encouraging people
- Weaknesses
  - Requiring high-capable, self-motivated people
  - Difficult in safety-critical systems
  - Problem in large projects