CS4507: Advanced Software Engineering

Lectures 4 and 5: Agile Methods

A. O'Riordan, 2014

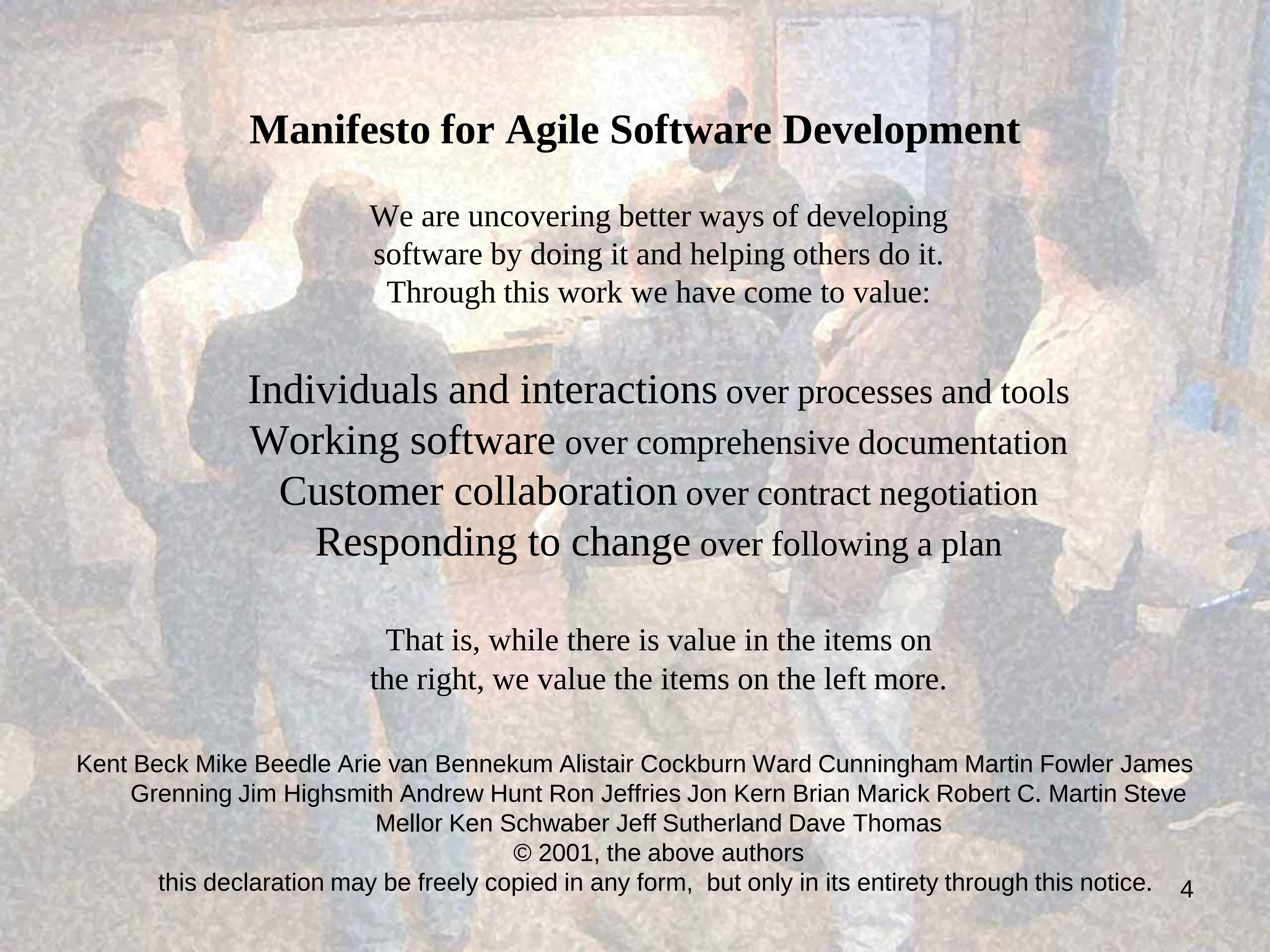
Some slides based on Sommerville

Agile Process

- Agile approach is a *practice-based* methodology for modelling and documentation software-based systems.
- Not a prescriptive process; it does not define detailed procedures for how to create a given type of model, instead it provides *advice* for how to be effective as a modeller.
- *Iterative* and *incremental* with *rapid cycles*
- *Cooperative*: customers and developers working constantly together
- The strengths of an agile process include the greater *flexibility* and the emphasis on the actual *product*.

We need a manifesto

- In 2001, 17 software developers met at Utah resort to discuss lightweight development methods
  - included leading names in software design such as Kent Beck, Robert C. Martin, Jim Highsmith, Steve Mellor, Andrew Hunt, Alistair Cockburn, Ron Jeffries, Ward Cunningham, Ken Schwaber and Martin Fowler http://www.agilemanifesto.org/authors.html
- They published a *Manifesto for Agile Software Development* www.agilemanifesto.org
- Agile methods have a light process but the “Agile movement is not anti-methodology” Jim Highsmith, *History: The Agile Manifesto*



Manifesto for Agile Software Development

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.

Kent Beck Mike Beedle Arie van Bennekum Alistair Cockburn Ward Cunningham Martin Fowler James Grenning Jim Highsmith Andrew Hunt Ron Jeffries Jon Kern Brian Marick Robert C. Martin Steve Mellor Ken Schwaber Jeff Sutherland Dave Thomas

12 Principles of Agile Approach

1. Highest priority is to *satisfy the customer* through early and continuous delivery of software

2. *Welcome changing requirements*, even late in development ─ 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. Customer and developers *work together daily* throughout the project

5. Build projects around motivated *individuals* ─ trust them; Give them the environment and support they need

6. Most efficient and effective method of conveying information to and within a development team is *face-to-face*

12 Principles of Agile Approach *continued*

7. *Working software* is the primary measure of progress

8. Agile processes promote *sustainable* development

9. Continuous attention to technical excellence and good design

10. *Simplicity* – the art of maximizing the amount of work not done

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 behaviour accordingly

http://www.agilemanifesto.org/principles.html

Some Common Practices

- Cross-functional teams: people with different expertise working together
- Just enough documentation (Alistair Cockburn)
- Story-driven modelling see next slides
- Continuous integration
- Test-driven development
- Daily meetings (stand up)

See http://guide.agilealliance.org/

User stories

- User stories are the main vehicle of incremental software delivery
- User stories originated with Extreme Programming
- *Role-feature-reason* template is commonly used

  As a ...

  I want ...

  So that ...

  An example:

  As a bank customer

  I want to withdraw money from an ATM

  So that I'm not constrained by opening hours or lines at the teller's

Three C's

- Three C's: Card, Conversation, Confirmation (from Ron Jeffries)
  - card (Post-It note), a physical token
  - conversation taking place at different time and places during a project between the various people concerned by a given feature of a software product
  - confirmation that the objectives the conversation revolved around have been reached

- Limitations
  - scale up
  - vague, informal and incomplete
  - lack of non-functional requirements

Benefits of User Stories

- Brevity: represent small chunks of business value which a programmer can implement in a period of days to weeks
- Allow a developer and the client representative to discuss requirements throughout the project lifetime
- Enable breaking of projects into small increments
- Suitability for projects which have volatile or poorly understood requirements

Test-Driven Development (TDD)

- Write automated *test* that defines a desired improvement or new function using use cases/user stories
- Run all tests and see if the new one fails
- Write code (sufficient to pass test)
- Run tests again (pass/fail)?
- If pass, Refactor
- Run tests again (pass/fail)?

- Creating a unit test helps a developer to really consider what needs to be done. Requirements are nailed down firmly by tests.

- Developers use testing frameworks such jUnit to create and automatically run the test cases

“Classical” vs. Agile

“Classical”	Agile
process heavy	value individual/interactions
contract with customer	collaborate with customer
plan in advance	incremental planning/respond to change
express requirements in writing	welcome changing requirements
document all elements of design	value working code
large teams	small teams

or http://melltoomom.blogspot.ie/2014/02/agile-development-for-dummies-aka-non.html

Pros and Cons of Agile Process

- Advantages
  - projects have *demonstrable software* at end of each iteration
  - develops tend to be more motivated; prefer to be creating *working artefacts* and not writing documentation
  - customers are involved throughout so can provide better requirements

- Drawbacks
  - lack of documentations (for future employees, *etc.*)
  - more difficult to manage
  - may lack skilled personnel with suitable personalities
  - lack of independent validation
  - not suitable for all project types see next slide

This slide is mostly based on Sommerville, Chapter 3

Unsuitable for Agile Approach

- There is still some debate on what projects are and are not suitable for agile methods
  - projects with teams at different locations
  - embedded and real-time systems
  - safety-critical systems
  - large systems
    - large systems and their development processes are often constrained by external rules and regulations limiting the way that they can be developed
    - large systems have a long procurement and development time; difficult to maintain coherent teams who know about the system over that period
    - large systems usually have diverse set of stakeholders; practically impossible to involve all of these in development process
- Fight back: scaling strategies; distributed development efforts (non-colocated teams).

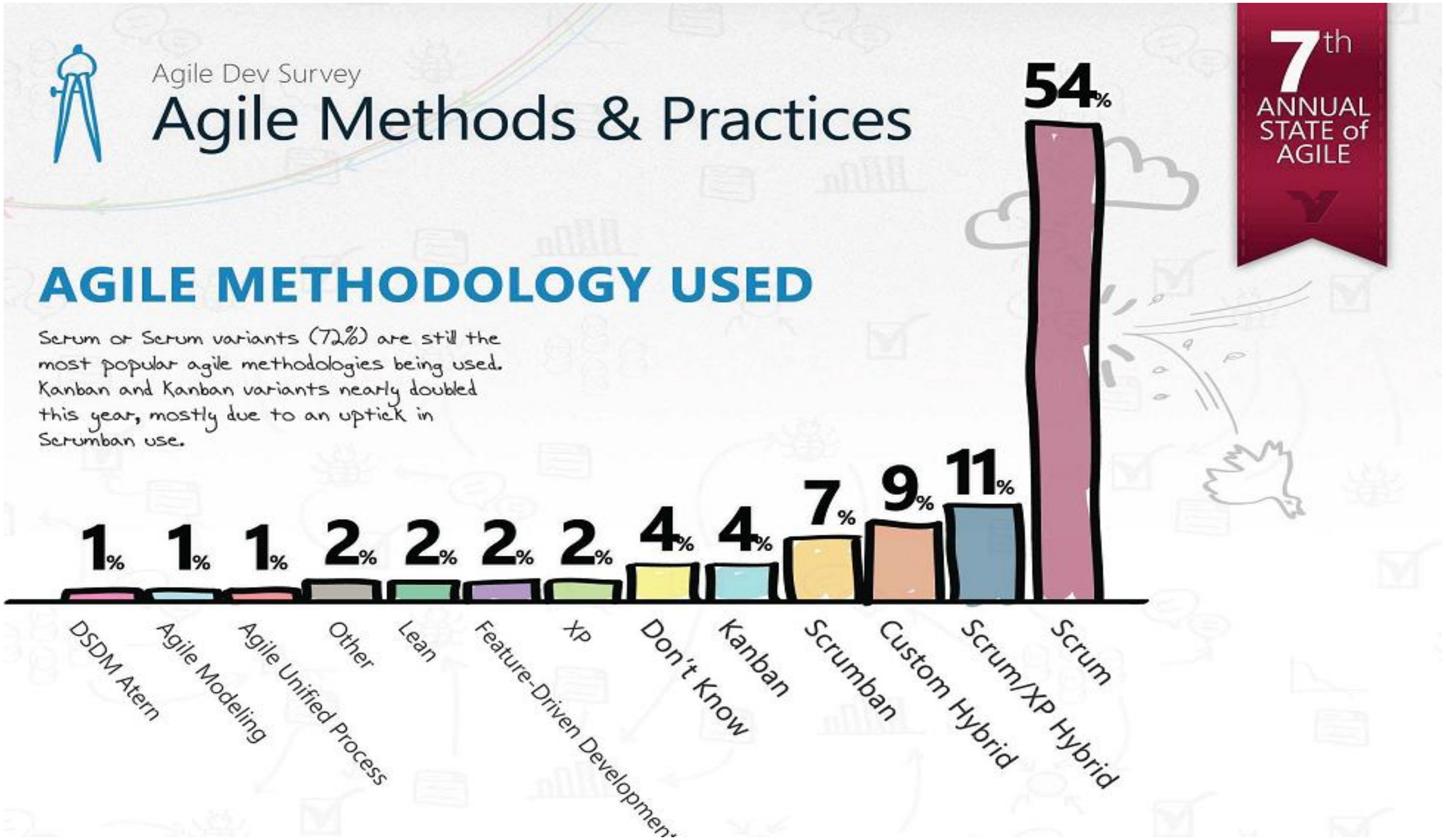
This slide is mostly based on Sommerville, Chapter 3

Agile Methods

- Extreme Programming www.extremeprogramming.org
  - developed by Kent Beck in 90s

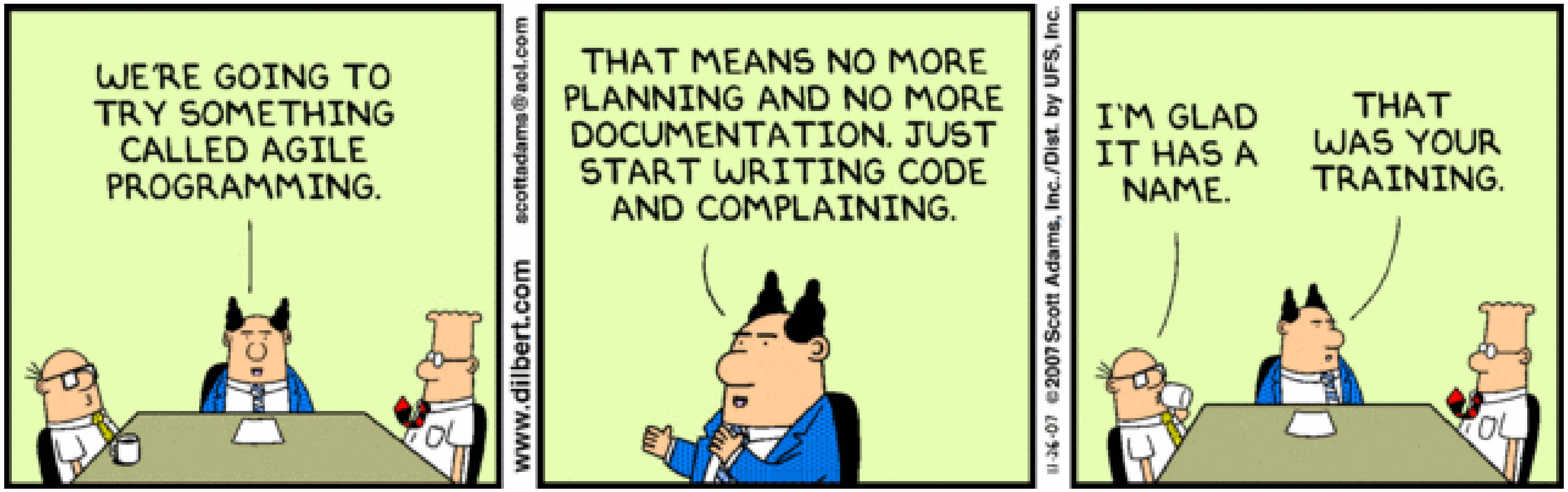
- Scrum https://www.scrum.org/
  - developed for software development by Ken Schwaber in 90s

- *Others:* not covered
  - Crystal (Clear) developed by Alistair Cockburn
  - Adaptive Software Development (ASD) developed by Jim Highsmith
  - Feature-Driven Development (FDD)
  - Dynamic Systems Development Method (DSDM)
  - Kanban, Lean, ...



Agile Development Survey 2013

http://www.versionone.com/pdf/7th-Annual-State-of-Agile-Development-Survey.pdf


WE'RE GOING TO TRY SOMETHING CALLED AGILE PROGRAMMING.
www.dilbert.com scottadams@aol.com
THAT MEANS NO MORE PLANNING AND NO MORE DOCUMENTATION. JUST START WRITING CODE AND COMPLAINING.
11-26-07 © 2007 Scott Adams, Inc./Dist. by UFS, Inc.
I'M GLAD IT HAS A NAME.
THAT WAS YOUR TRAINING.

Extreme Programming (XP)

- A new approach to software development, Extreme Programming was created by Kent Beck in 1990s
  - published as book *Extreme Programming Explained* in 1999
  - discussions of method developed on Ward Cunningham's WikiWikiWeb (the very first Wiki)

- XP stresses *customer satisfaction* and also emphasizes *team work*

- Managers, customers, and developers are all part of a team taking *collective responsibility* dedicated to delivering quality software

XP Values

- Based on four key values: *communication*, *simplicity*, *feedback*, and *courage*. A fifth was later added: respect
  - XP programmers communicate with their customers and fellow programmers
  - they keep their design simple and clean
  - they get feedback by testing their software starting on day one
  - courageously respond to changing requirements and technology

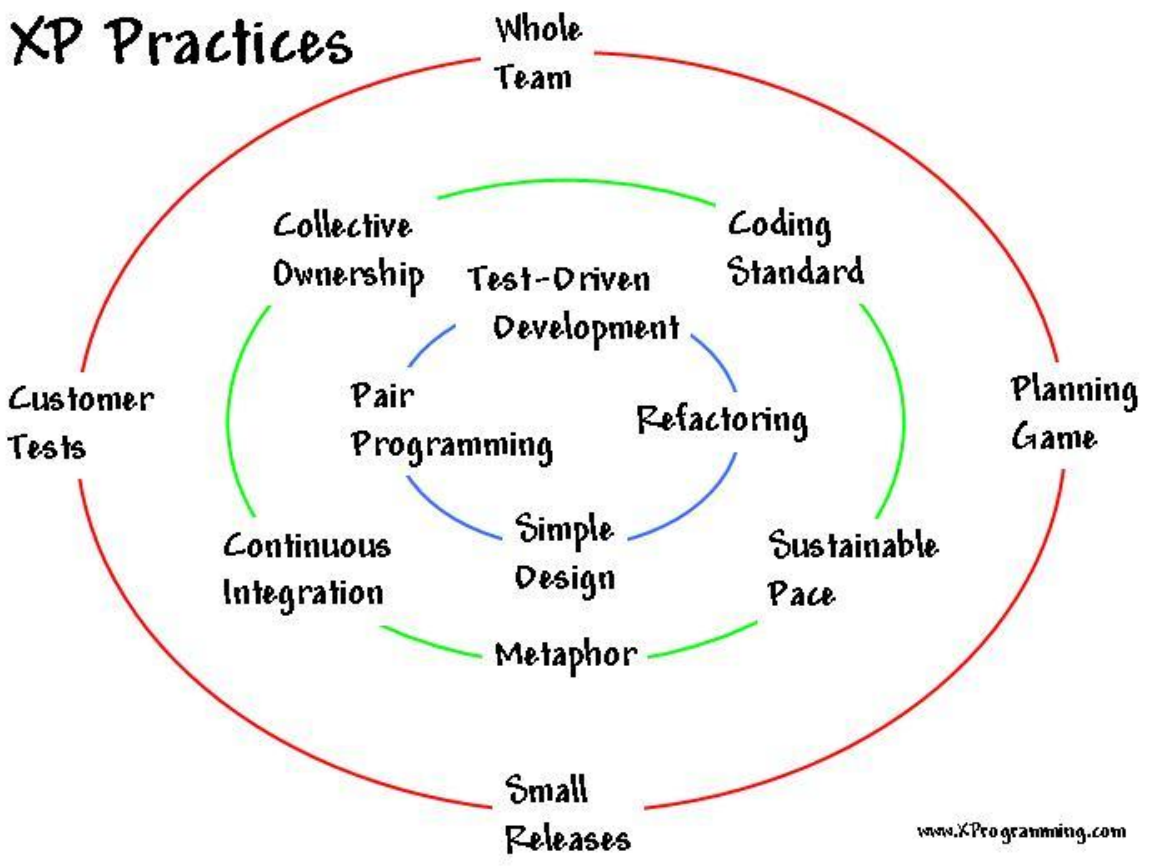
XP Practices

- ‘Extreme’ approach to iterative development
  - increments are delivered to customers every 2 weeks;
  - all tests must be run for every build and the build is only accepted if tests run successfully
- *Incremental planning*
  - user requirements are expressed as scenarios called *user stories* written on cards
  - stories to be included in a release are determined by the *time available* and their relative *priority*
  - *project velocity* is a measure of how much work is getting done on your project
- *Enough design* is carried out to meet the current requirements and no more.

XP Practices *continued*

- *Pair Programming*: all software is built by two programmers, sitting side by side, at the same machine
  - ensures that all production code is reviewed by at least one other programmer
- XP teams practice *test-driven development*; tests written first
- XP uses a process of continuous design improvement called *refactoring*
- The XP team keeps the system *integrated* and running all the time
- XP teams follow a common *coding standard*
  - so that all the code in the system looks as if it was written by a single very competent individual

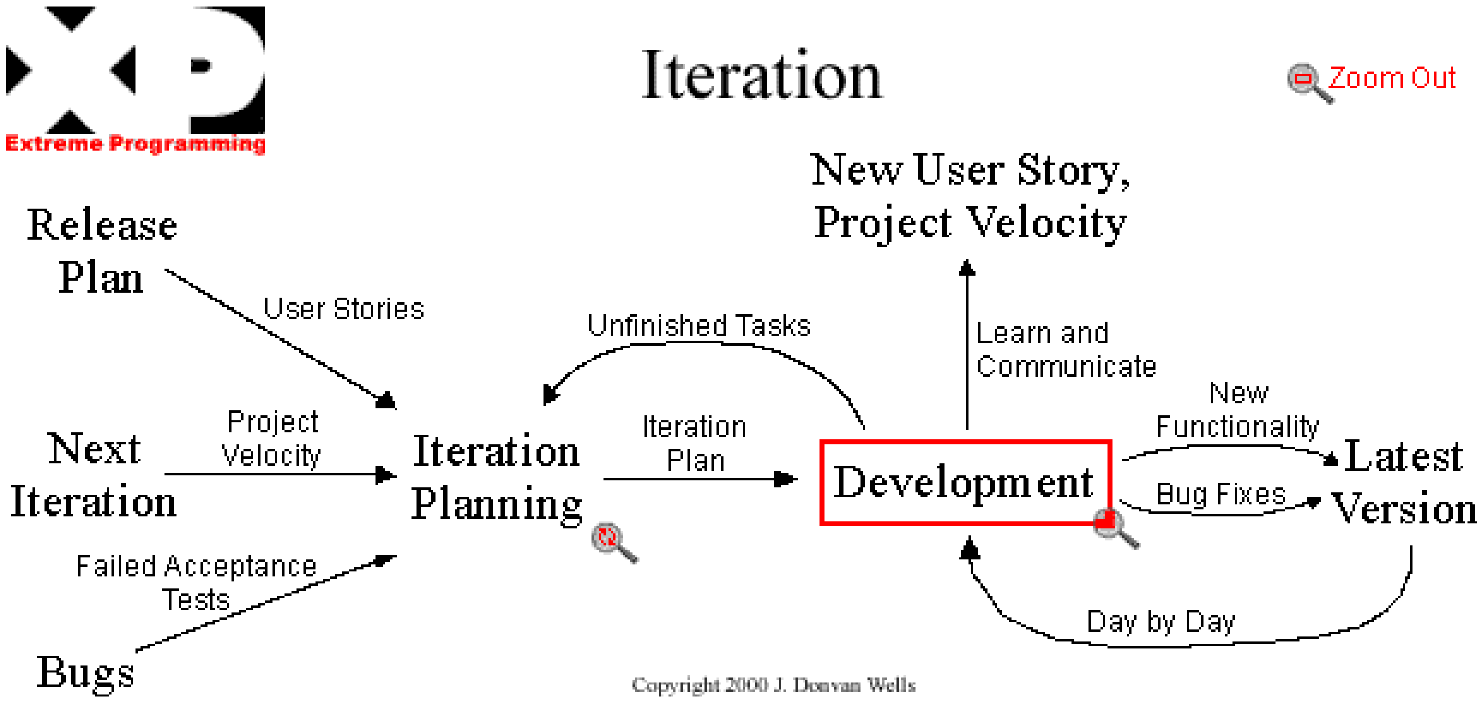
XP Practices Summary



Whole Team

- Every contributor to the project is an integral part of the *Whole team*
- Team forms around a business representative called "the Customer", who sits with the team and works with them daily
  - team must include the Customer, who provides the requirements, sets the priorities, and steers the project
  - analysts may serve as helpers to the Customer, helping to define the requirements
  - may additionally be a manager, providing resources, handling external communication, coordinating activities

What an Iteration looks Like



from http://www.extremeprogramming.org/map/iteration.html

WE'RE GOING TO TRY SOMETHING CALLED EXTREME PROGRAMMING.
FIRST, PICK A PARTNER. THE TWO OF YOU WILL WORK AT ONE COMPUTER FOR FORTY HOURS A WEEK.
THE NEW SYSTEM IS A MINUTE OLD AND I ALREADY HATE EVERYONE.
www.dilbert.com
scottadams@aol.com
© 2002 United Feature Syndicate, Inc.
1/9/03

Scrum

- Iterative and incremental agile approach to *managing* product development

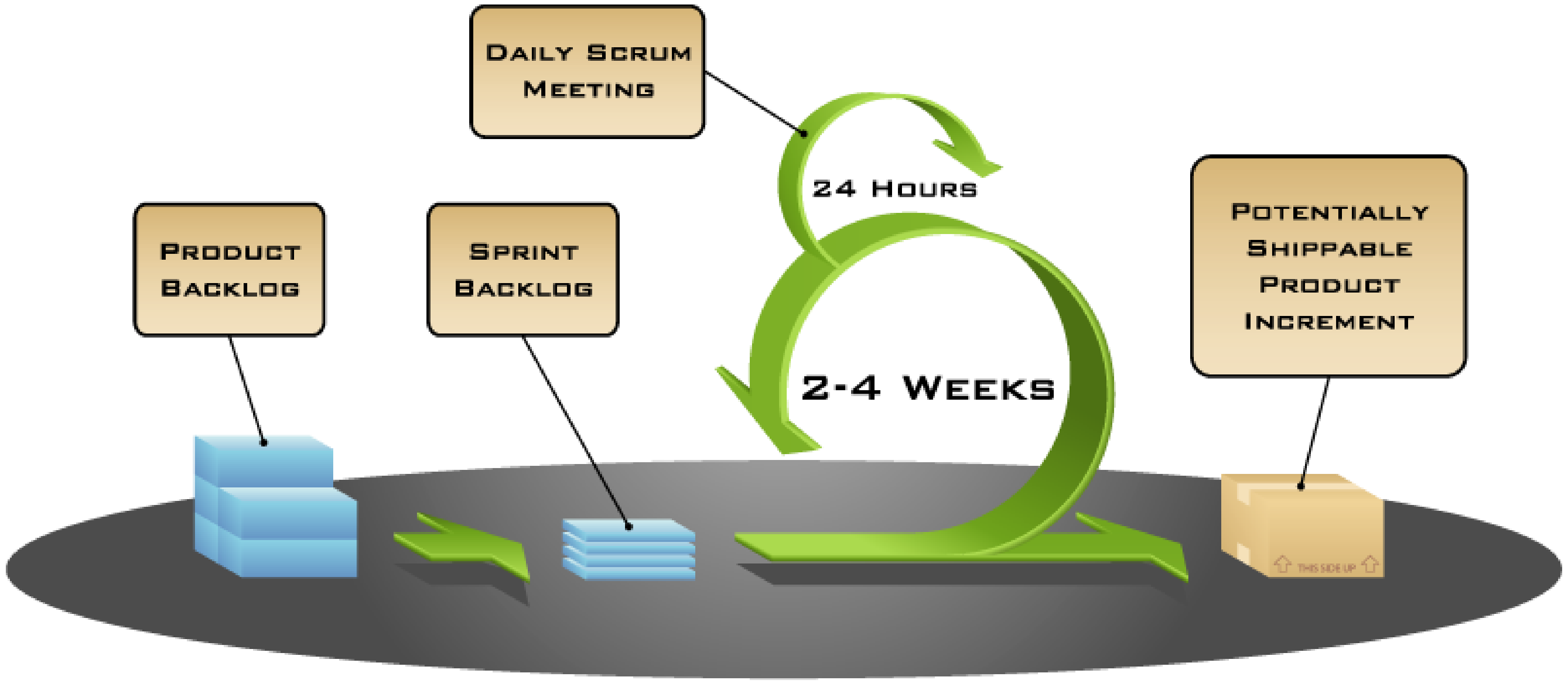
- Ken Schwaber and Jeff Sutherland developed the Scrum method for software development based on a product development methodology from Japan
  - Scrum methodology presented at *OOPSLA conference* in 1995
  - In 2001 book *Agile Software Development with Scrum* (Schwaber and Beedle)
  - Scrum Alliance and the Certified Scrum Master programme

- Scrum team consists of a *product owner*, the *development team*, and a *scrum master*

3 Phases

- Initial phase is an *outline planning phase* where you establish the general objectives for project and design software architecture
  - starting point for planning is the product *backlog*, which is the list of work to be done on the project
- Series of *sprint* cycles, where each cycle develops an increment of the system
  - all of the project team who work with the customer select the features and functionality to be developed during the sprint
- *Project closure* phase wraps up the project, completes required documentation and assesses the lessons learned

Scum in a diagram



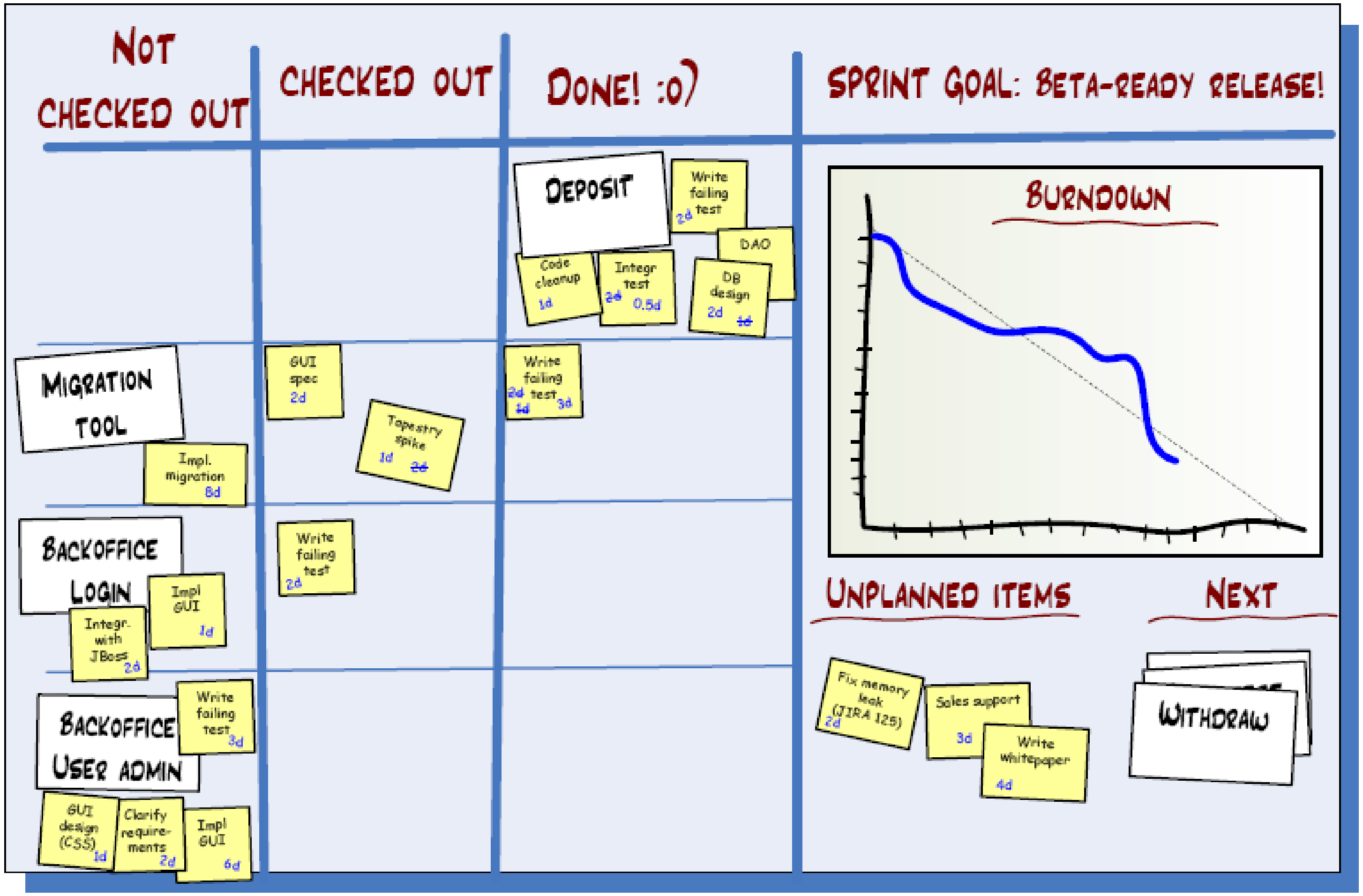
In practice

- Sprint
  - during this stage the team is isolated from the customer and the organization, with all communications channelled through *Scrum master*
  - *sprint burn down* charts daily progress for sprint over the sprint's length
  - at the end of the sprint, the work done is *reviewed* and presented to stakeholders

- Scrum master is facilitator
  - arranges daily meetings, tracks the backlog of work to be done, records decisions, measures progress against the backlog and communicates with customers and management outside of the team

- The whole team attends short daily meetings
  - all team members share information

Scrum Taskboard

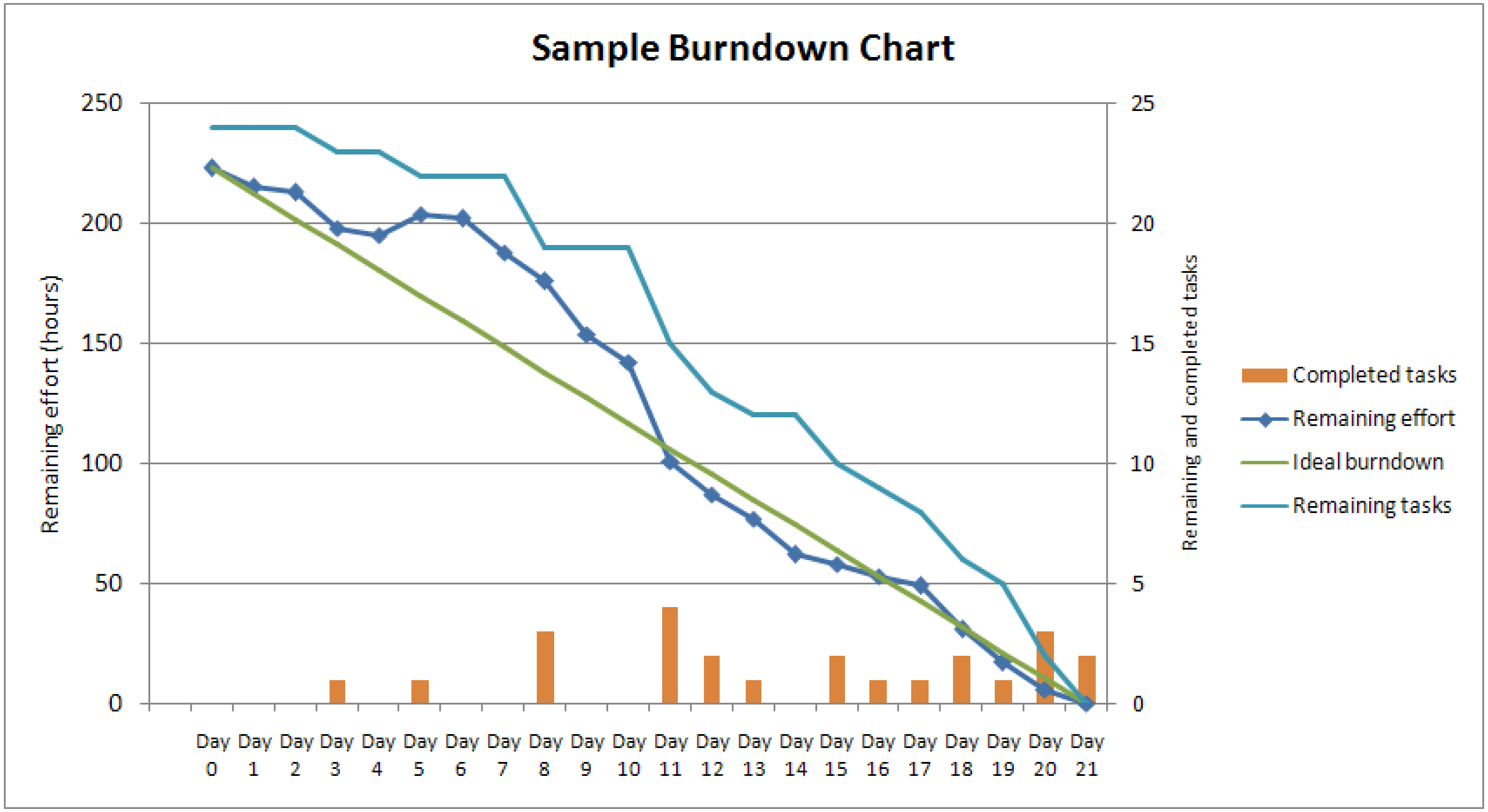


Scrum meeting (Daily Scrum)

- Held at the same time and place each day within a *15-minute time-box*
- Only Development Team members participate in the *Daily Scrum*
- Issues:
  – What did I do yesterday that helped the Development Team meet the Sprint Goal?
  – What will I do today to help the Development Team meet the Sprint Goal?
  – Do I see any impediment that prevents me or the Development Team from meeting the Sprint Goal?

see https://www.scrum.org/Portals/0/Documents/Scrum Guides/2013/Scrum-Guide.pdf

Burn down chart


Sample Burndown Chart
Remaining effort (hours)
Remaining and completed tasks
Day 0
Day 1
Day 2
Day 3
Day 4
Day 5
Day 6
Day 7
Day 8
Day 9
Day 10
Day 11
Day 12
Day 13
Day 14
Day 15
Day 16
Day 17
Day 18
Day 19
Day 20
Day 21
Completed tasks
Remaining effort
Ideal burndown
Remaining tasks