Agile Methodologies XP and Scrum

Introduction into Software Engineering
Lecture 22

Bernd Bruegge
Applied Software Engineering
Technische Universitaet Muenchen

Problem: How to we Control Software Development?

How can software development best be described?

- Two opinions: Maturity vs agility

1. Through organizational maturity (Humphrey)
   - Repeatable process, Capability Maturity Model (CMM)
2. Through agility (Schwaber):
   - Large parts of software development is empirical in nature; cannot be modeled with a defined process

How do we control software development?

- Software development is a deterministic process
  - with a defined process control model
- Software development is a nondeterministic process
  - with an empirical process control model.

Defined Process Control Model

- The defined Process control models assumes that software development is a deterministic process
  - Given a well-defined set of inputs, the same outputs are generated every time
    - Deviations are seen as errors that need to be corrected
    - All activity-oriented software lifecycle models introduced in the previous lecture are defined process control models
- Precondition to apply the defined process control model:
  - Every piece of work can be completely understood
  - All the activities and tasks are well defined to provide repeatability and predictability
- If the preconditions are not satisfied:
  - Lot of surprises (often too late), loss of control, incomplete or wrong work products.

Empirical Process Control Model

The empirical process control model assumes that many aspects of software development are better described as a nondeterministic process

- Not all pieces of work need to be completely understood or can be understood
- Deviations are seen as opportunities that need to be investigated
- The empirical process “expects the unexpected”

- Control is exercised through frequent inspection
  - Daily inspection in Scrum
- Conditions when to apply this model:
  - Frequent change is expected during the project, inputs are unpredictable and outputs are unrepeatable.

Outline of the Lecture

- Two examples of empirical process control models
  - Extreme Programming (XP)
  - Scrum
- Both are also examples of agile methodologies
  - Agile methodologies use an empirical process model to describe the software lifecycle.

Issues to be addressed by a Methodology (See Lecture 2)

- Methodologies provide guidance, general principles and strategies for selecting methods and tools in a given project environment
- Key questions for which methodologies provide guidance:
  - How much involvement of the customer?
  - How much planning?
  - How much reuse?
  - How much modeling before coding?
  - How much process?
  - How much control and monitoring?

XP (Extreme Programming)

- XP assumes that change is normal
  - XP assumes that software developer must be able react to changing requirements at any point during a project
- XP is an agile software methodology because
  - It places higher priority on adaptability (“empirical process control model”) than on predictability (“defined process control model”)
- XP prescribes a set of day-to-day practices for managers and developers to address change.

History of XP

- Original cast of contributors
  - Kent Beck, Ron Jeffries, Ward Cunningham (also created Wiki)
- Application of XP in the Chrysler Comprehensive Compensation project (C3 Project) in 1995
- Lots of initial excitement but later a lot of problems:
  - Daimler actually shut down the C3 Project in 2000 and even banned XP for some time
    - See Additional References.

XP Day-to-Day Practices

1. Rapid feedback
   - Confronting issues early results in more time for resolving issues. This applies both to client feedback and feedback from testing
2. Simplicity
   - The design should focus on the current requirements
   - Simple designs are easier to understand and change than complex ones
3. Incremental change
   - One change at the time instead of many concurrent changes
   - One change at the time should be integrated with the current baseline.

XP Mantras (continued)

4. Embracing change
   - Change is inevitable and frequent in XP projects
   - Change is normal and not an exception that needs to be avoided
5. Quality work
   - Focus on rapid projects where progress is demonstrated frequently
   - Each change should be implemented carefully and completely.

How much planning in XP?

- Planning is driven by requirements and their relative priorities
  - Requirements are elicited by writing stories with the client (called user stories)
- User stories are high-level scenarios or use cases that encompass a set of coherent features
  - Developers decompose each user story in terms of development tasks that are needed to realize the story
  - Developers estimate the duration of each task in terms of days
  - If a task is planned for more than a couple of weeks, it is further decomposed into smaller tasks.

How much planning in XP?

- Ideal weeks
  - Number of weeks estimated by a developer to implement the story if all work time was dedicated for this single purpose
- Fudge Factor
  - Factor to reflect overhead activities (meetings, holidays, sick days...)
  - Also takes into account uncertainties associated with planning
- Project velocity
  - Inverse of ideal weeks
    - i.e., how many ideal weeks can be accomplished in fixed time.

How much planning in XP? (2)

- Stacks
  - The user stories are organized into stacks of related functionality
- Prioritization of stacks
  - The client prioritizes the stacks so that essential requirements can be addressed early and optional requirements last
- Release Plan
  - Specifies which story will be implemented for which release and when it will be deployed to the end user
- Schedule
  - Releases are scheduled frequently (e.g., every 1–2 months) to ensure rapid feedback from the end users.

Team Organization in XP

- Production code is written in pairs (pair programming)
    - The person typing is called the driver.
    - The person reviewing the code is called the observer or navigator.
- Individual developers may write prototypes for experiments or proof of concepts, but not production code
- Pairs are rotated often to enable a better distribution of knowledge throughout the project
    - The two programmers switch roles frequently, possibly every 30 minutes.

How much reuse in XP?

- Simple design
  - Developers are encouraged to select the most simple solution that addresses the user story being currently implemented
- No design reusability
  - The software architecture can be refined and discovered one story at the time, as the prototype evolves towards the complete system
- Focus on Refactoring
  - Design patterns might be introduced as a result of refactoring, when changes are actually implemented
  - Reuse discovery only during implementation.

How much modeling in XP?

- No explicit analysis/design models
  - Minimize the amount of documentation
  - Fewer deliverables reduce the duplication of issues
- Models are only communicated among participants
  - The client is the “walking specification”
- Source Code is the only external model
  - The system design is made visible in the source code by using descriptive naming schemes
- Refactoring is used to improve the source code
  - Coding standards are used to help developers communicate using only the source code.

How much process in XP?

- Iterative life cycle model with 5 activities: planning, design, coding, testing and integration
  - Planning occurs at the beginning of each iteration
  - Design, coding, and testing are done incrementally
  - Source code is continuously integrated into the main branch, one contribution at the time
  - Unit tests for all integrated units; regression testing
- Constraints on these activities
  - Write the tests first. Unit tests are written before units. They are written by the developer
    - Generalized to test-driven programming
  - Catch the errors: When defects are discovered, another unit test is created to reproduce the defect
  - Refactor before extending the source code.

How much control in XP?

- Reduced number of formal meetings
  - Daily stand up meeting with the co-located client for status communication
  - No discussions to keep the meeting short
- No inspections and no peer reviews
  - Pair programming is used instead
  - Production code is written in pairs, review during coding.
- Self-organizing system with a leader:
  - The leader communicates the vision of the system
  - The leader does not plan, schedule or budget
  - The leader establishes an environment based on collaboration, shared information, and mutual trust
  - The leader ensures that a product is shipped.

Summary of the XP Methodology

Planning	Co-locate the project with the client, write user stories with the client, frequent small releases (1-2 months), create schedule with release planning, kick off an iteration with iteration planning, create programmer pairs, allow rotation of pairs
Modeling	Select the simplest design that addresses the current story; Use a system metaphor to model difficult concepts; Use CRC cards for the initial object identification; Write code that adheres to standards; Refactor whenever possible
Process	Code unit test first, do not release before all unit tests pass, write a unit test for each uncovered bug, integrate one pair at the time
Control	Code is owned collectively. Adjust schedule, Rotate pairs, Daily status stand-up meeting, Run acceptance tests often and publish the results.

Introduction

- Classical software development methodologies have some disadvantages:
  - Huge effort during the planning phase
  - Poor requirements conversion in a rapid changing environment
  - Treatment of staff as a factor of production

- Agile Software Development Methodologies
  - Minimize risk → short iterations
  - Real-time communication (preferable face-to-face) → very little written documentation
  - www.agilealliance.org

Today's Lecture

- Miscellaneous
- What is Scrum?
- Agile Alliance and Manifesto
- History of Scrum
- Definition
- Components of Scrum
  - Scrum Roles
  - The Process
  - Scrum Artifacts
- Conclusion

Miscellaneous

- Next Thursday: 8:15-10:00 HS2 , Sprechstunde ("doctor's hour")
- Final Exam
  - 5 February 2009, Maschinenwesen 0001, 18:00-19:30
- Repeat exam
  - 23 April, Location: TBA, Time: TBA
  - Note: Repeat exam means Repeat exam
- Next Tuesday: Invited Lecture
- iPhone Praktikum Summer 2009

Tuesday 16:15-18:00
Invited Lecture

iteratec

Warum IT-Projekte scheitern

Klaus Eberhardt, iteratec GmbH

3. Februar 2009

iPhone Praktikum Announcement SS 2009

Programming mobile multimedia applications with the iPhone SDK

Kickoff

wwwbruegge.in.tum.de/iphone09
Laboratory Summer 2009

04/23/2009, 5:00 pm,
Room 01.07.014

The new iPhone laboratory 6R

Real customers
Real problems
Real data
Real teamwork
Real project experience
Real deadlines

Manifesto for Agile Software Development

- http://www.agilemanifesto.org/
- Individuals and interactions are preferred over processes and tools
- Working software is preferred over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan.

History of Scrum

- 1995:
  - Jeff Sutherland and Ken Schwaber analyse common software development processes
    - Conclusion: not suitable for empirical, unpredictable and non-repeatable processes
  - Proposal of Scrum
  - Enhancement of Scrum by Mike Beedle
    - Combination of Scrum with Extreme Programming
- 1996: Introduction of Scrum at OOPSLA
- 2001: Publication “Agile Software Development with Scrum” by Ken Schwaber & Mike Beedle
- Founders are also members in the Agile Alliance.

Scrum

- Definition (Rugby): A Scrum is a way to restart the game after an interruption,
  - The forwards of each side come together in a tight formation and struggle to gain possession of the ball when it is tossed in among them
- Definition (Software Development): Scrum is an agile, lightweight process
  - To manage and control software and product development with rapidly changing requirements
  - Based on improved communication and maximizing cooperation.

Why Scrum ?

Traditional methods are like relay races

Agile methods are like rugby

Practicing a Scrum

Scrums in Real Games

Testudo:
Battle Formation used by the Romans

Methodology Issues

- Methodologies provide guidance, general principles and strategies for selecting methods and tools in a given project environment
- Key questions for which methodologies provide guidance:
  - How much involvement of the customer?
  - How much planning?
  - How much reuse?
  - How much modeling before coding?
  - How much process?
  - How much control and monitoring?

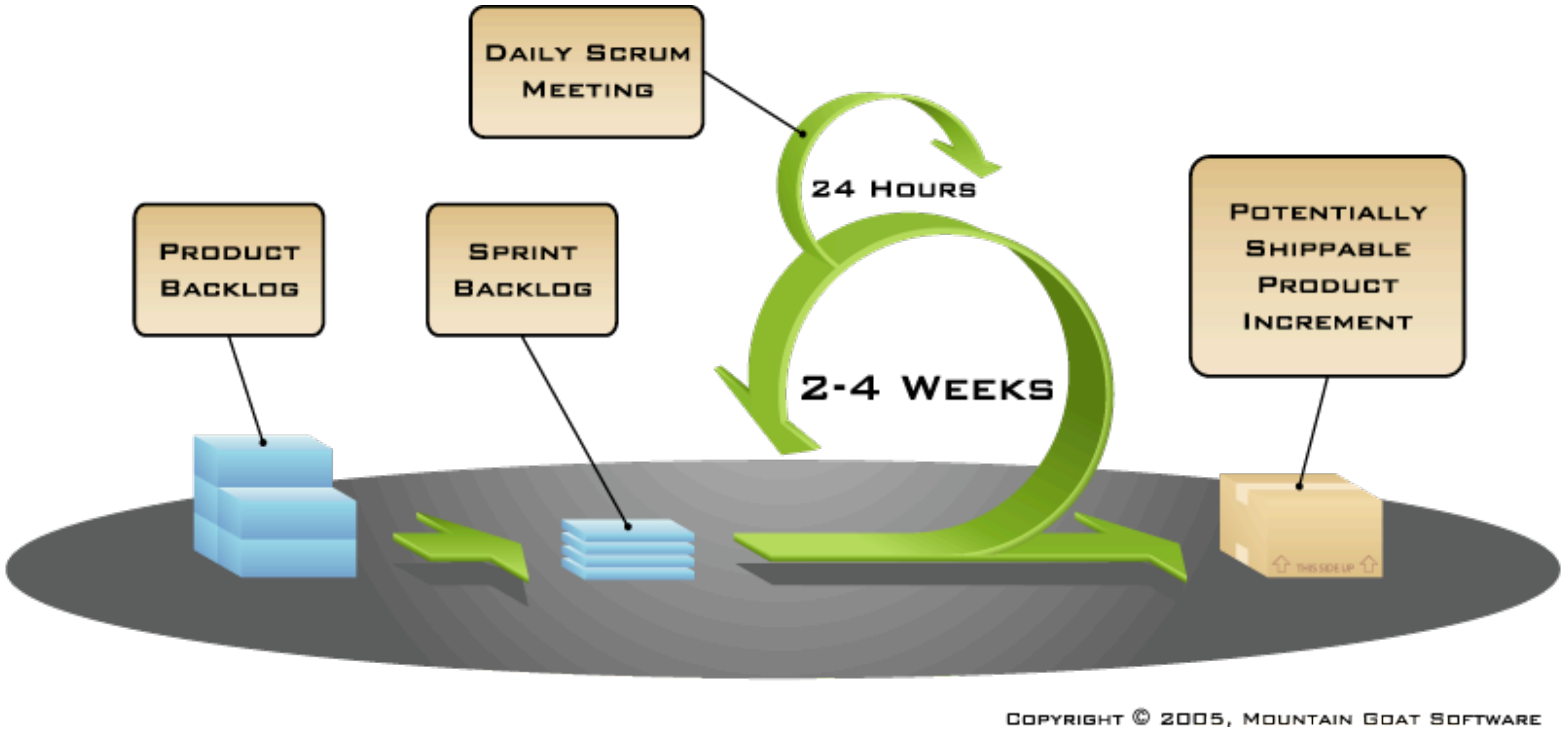
Scrum as Methodology

- Involvement of the customer
  - Onsite customer
- Planning
  - Checklists and incremental daily plans
  - Product backlog, sprint backlogs
- Reuse
  - Checklists from previous projects
- Modeling
  - Models may or may not be used
- Process
  - Iterative, incremental process
- Control and Monitoring
  - Daily meetings.

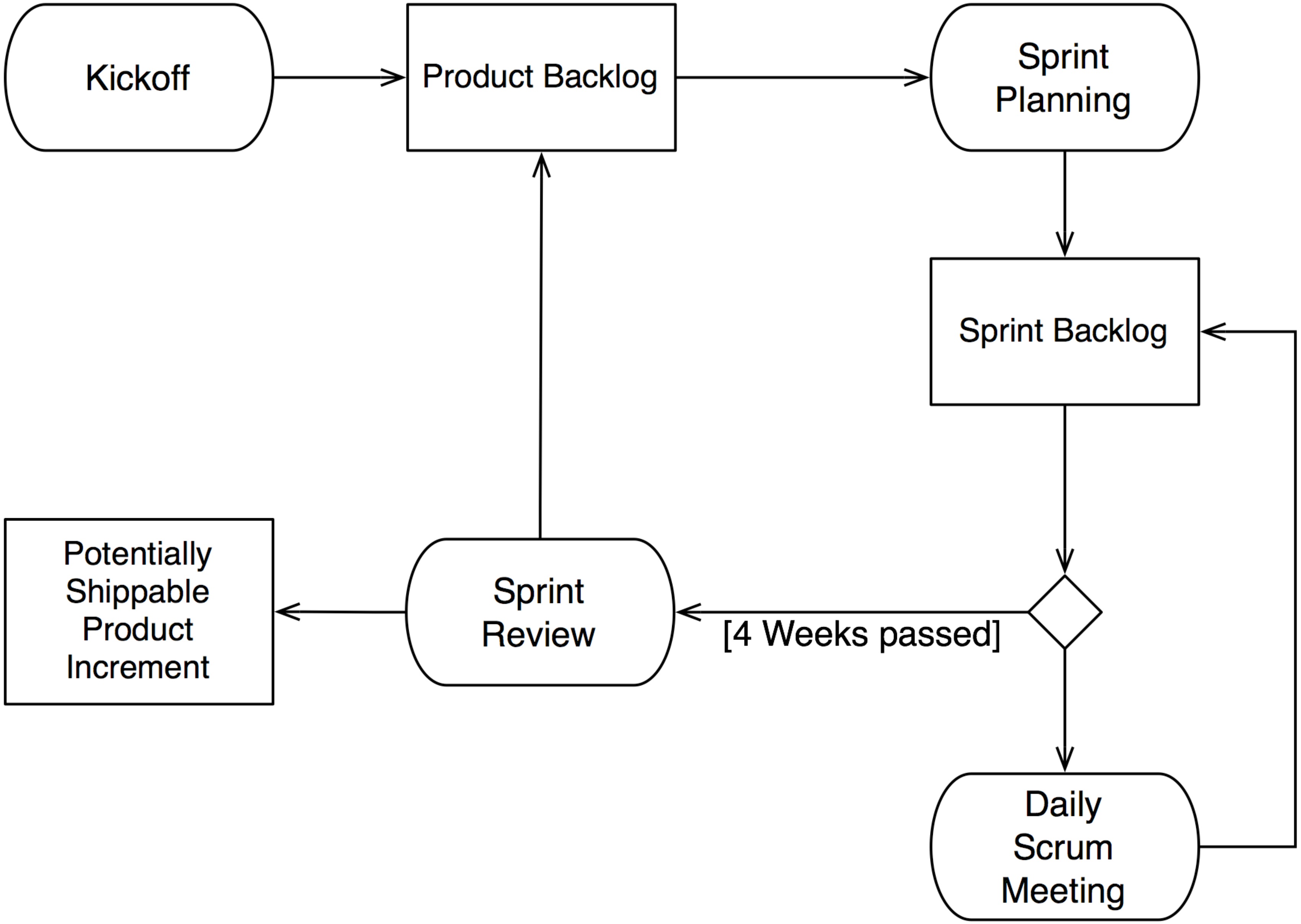
Components of Scrum

- Scrum Roles
  - Scrum Master, Scrum Team, Product Owner
- Process
  - Sprint Planning Meeting
  - Kickoff Meeting
  - Sprint (corresponds to an iteration in a Unified Process, but limited to 30 days)
  - Daily Scrum Meeting
  - Sprint Review Meeting
- Scrum Artifacts
  - Product Backlog, Sprint Backlog
  - Burndown Charts

Overview of Scrum (Napkin View)


Daily Scrum Meeting
24 Hours
Product Backlog
Sprint Backlog
2-4 Weeks
Potentially Shippable Product Increment
This side up
Copyright © 2005, Mountain Goat Software

Overview of Scrum (Activity Diagram)



Scrum Master

- Represents management to the project
- Typically filled by a project manager or team leader
- Responsible for enacting scrum values and practices
- Main job is to remove impediments.

The Scrum Team

- Typically 5-6 people
- Cross-functional (quality assurance, programmers, UI designers, architects)
- Members should work full-time in the team
- Team is self-organizing
- Membership can change only between sprints.

Product Owner

- Knows what needs to be build and in what sequence this should be done
- Traditionally the “Client”
- Typically a product manager

Scrum Process Activities

- Project-Kickoff Meeting
- Sprint Planning Meeting
- Sprint
- Daily Scrum Meeting
- Sprint Review Meeting

Project-Kickoff Meeting

- A collaborative meeting in the beginning of the project
  - Participants: Product Owner, Scrum Master
  - Takes 8 hours and consists of 2 parts ("before lunch and after lunch")
- Goal: Create the Product Backlog

Sprint Planning Meeting

- A collaborative meeting in the beginning of each Sprint
    - Participants: Product Owner, Scrum Master and Scrum Team
- Takes 8 hours and consists of 2 parts (“before lunch and after lunch”)
- Goal: Create the Sprint Backlog

Sprint

- A month-long iteration, during which is incremented a product functionality
- No outside influence can interference with the Scrum team during the Sprint
- Each day in a Sprint begins with the Daily Scrum Meeting

Daily Scrum Meeting

- Is a short (15 minutes long) meeting, which is held every day before the Team starts working
- Participants:
  - Scrum Master (which is the chairman), Scrum Team
- Every Team member should answer on 3 questions

Questions for each Scrum Team Member

1. Status:
   What did I do since the last Scrum meeting?
2. Issues:
   What is stopping me getting on with the work?
3. Action items:
   What am I doing until the next Scrum meeting?

Daily Scrum Meeting

- NOT a problem solving session
- NOT a way to collect information about WHO is behind the schedule

- It is a meeting in which team members make commitments to each other and to the Scrum Master
- Is a good way for a Scrum Master to track the progress of the team.

Sprint Review Meeting

- Is held at the end of each Sprint
- Business process functionality which was created during the Sprint is demonstrated to the Product Owner
- Informal, should not distract Team members of doing their work.

Scrum Artifacts

- Product Backlog
- Sprint Backlog
- Burn down Charts

Product Backlog

- Requirements for a system, expressed as a prioritized list of Backlog Items (“Todos”, requirements, open issues)
  - Managed and owned by a Product Owner
  - Contained in a Spreadsheet (typically)
- Usually created during the Project Kickoff Meeting
- Can be changed and re-prioritized before each Sprint.

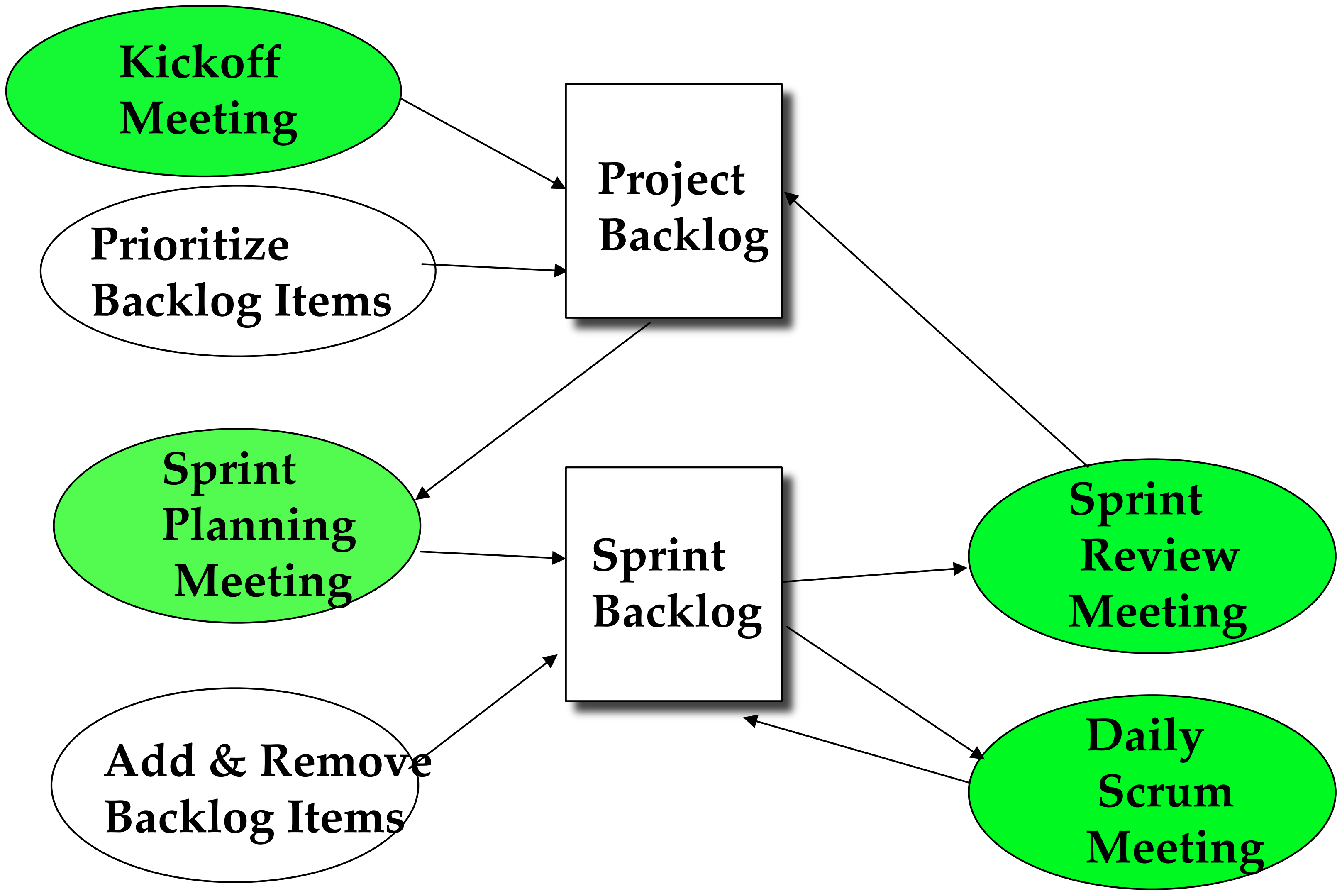
Estimation of Product Backlog Items

- Establishes team's velocity (how much effort a Team can handle in one Sprint)
- Units of complexity
    - Size-category: L, M, S ("T-Shirt size")
    - Story points
    - Work days/work hours
- Methods of estimation:
    - Expert Review
    - Creating a Work Breakdown Structure (WBS)

Sprint Backlog

- A subset of Product Backlog Items, which defines the work to be done in a Sprint
- Is created ONLY by Team members
- Each item has it’s own status
- Should be updated every day.

Lists, Activities and Meetings in Scrum



Sprint Backlog

- No more then 300 tasks in the list
- If a task requires more than 16 hours, it should be broken down
- Team can add or subtract items from the list
  - Product owner is not allowed to do it.

Sprint Backlog

- Is a FORECAST!
- Is a good warning monitor

Measuring Progress in Scrum

- Project Manager is mostly concerned about
  - Sprint progress: How is the team doing toward meeting their Sprint goal
  - Release progress: Will the release be on time with the quality and functionality desired?
  - Product progress: how is the product filling out compared to what's needed?
- 3 Types of Charts (good information radiators)
  - Sprint Burn down Chart (progress of the sprint)
  - Release Burn down Chart (progress of release)
  - Product Burn down chart (progress of the product)

Burn down Charts

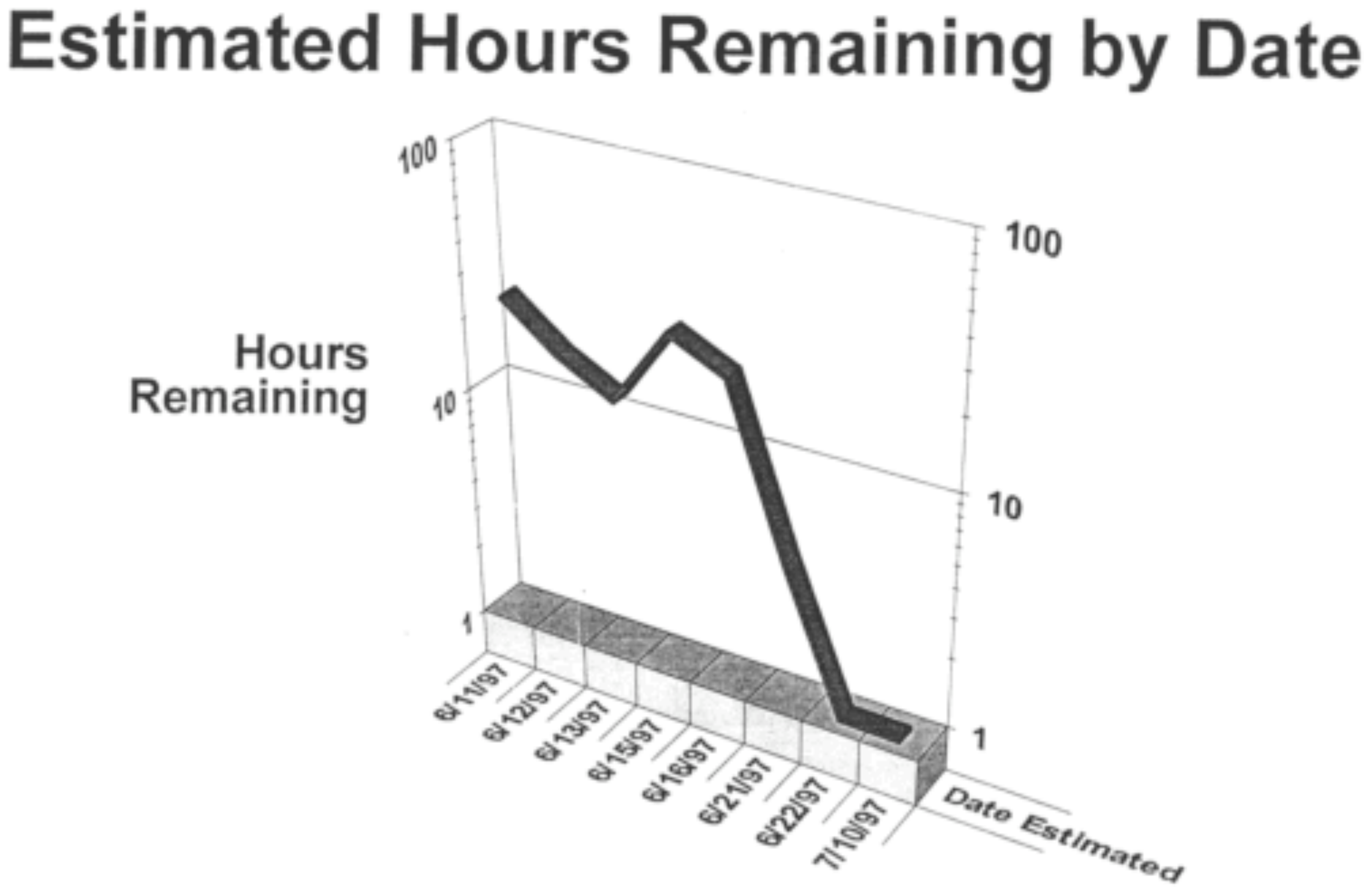
- Schwaber calls them “Information radiators”
- Two characteristics are key
  - The information changes over time
    - This makes it worth a person's while to look at the display...
  - It takes very little energy to view the display.

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

Burn down Chart Example

- X-Axis: time (usually in days)
- Y-Axis: remaining effort



Release Burn down Chart

- Radiator for the Question:
  - "Will the release be done on right time? "
- X-axis: sprints
- Y-axis: amount of hours remaining
- The estimated work remaining can also burn up

Summary

- XP and Scrum are agile software development methodologies with focus on
  - Empirical process control model
  - Changing requirements are the norm
  - Controlling conflicting interests and needs
- Very simple processes with clearly defined rules
- Self-organizing teams, where each team member carries a lot of responsibility
- No extensive documentation
  - Possibility for “undisciplined hacking”.

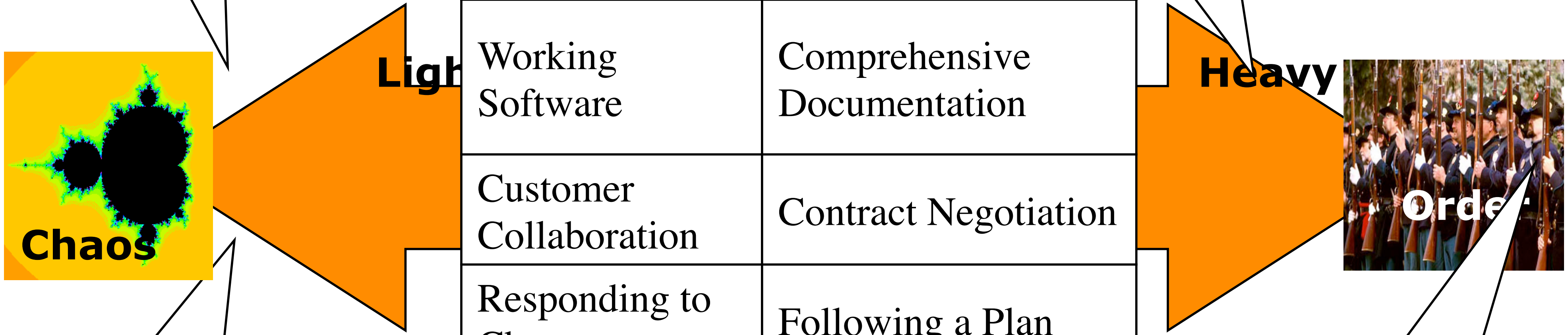
Additional Readings

- Schwaber, Beedle
  - Agile Software Development with Scrum, Addison-Wesley Verlag, 2002.
- Kevin Aguanno (editor)
  - Managing Agile Projects, Multi-Media Publications Inc., 2005.
- Tapscott, Williams
  - Wikinomics, Portfolio Verlag, 2006.

Ways to React to Complexity and Change

Nonhierarchical organization (**Scrum**)

Hierarchical organization
Iterative process
(**Royce**)



Individuals and Interactions	Processes and Tools
Working Software	Comprehensive Documentation
Customer Collaboration	Contract Negotiation
Responding to Change	Following a Plan

Nonlinear process (**XP**)

Linear process (**Waterfall**)

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

Product Burn down Chart

- The “big picture” view of project’s progress
  - Burn down Chart containing all the releases.