

Chapter 3

■ Agile Development

Slide Set to accompany

Software Engineering: A Practitioner's Approach, 7/e

by Roger S. Pressman

Slides copyright © 1996, 2001, 2005, 2009 by Roger S. Pressman

For non-profit educational use only

May be reproduced ONLY for student use at the university level when used in conjunction with *Software Engineering: A Practitioner's Approach, 7/e*. Any other reproduction or use is prohibited without the express written permission of the author.

All copyright information MUST appear if these slides are posted on a website for student use.

The 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 et al

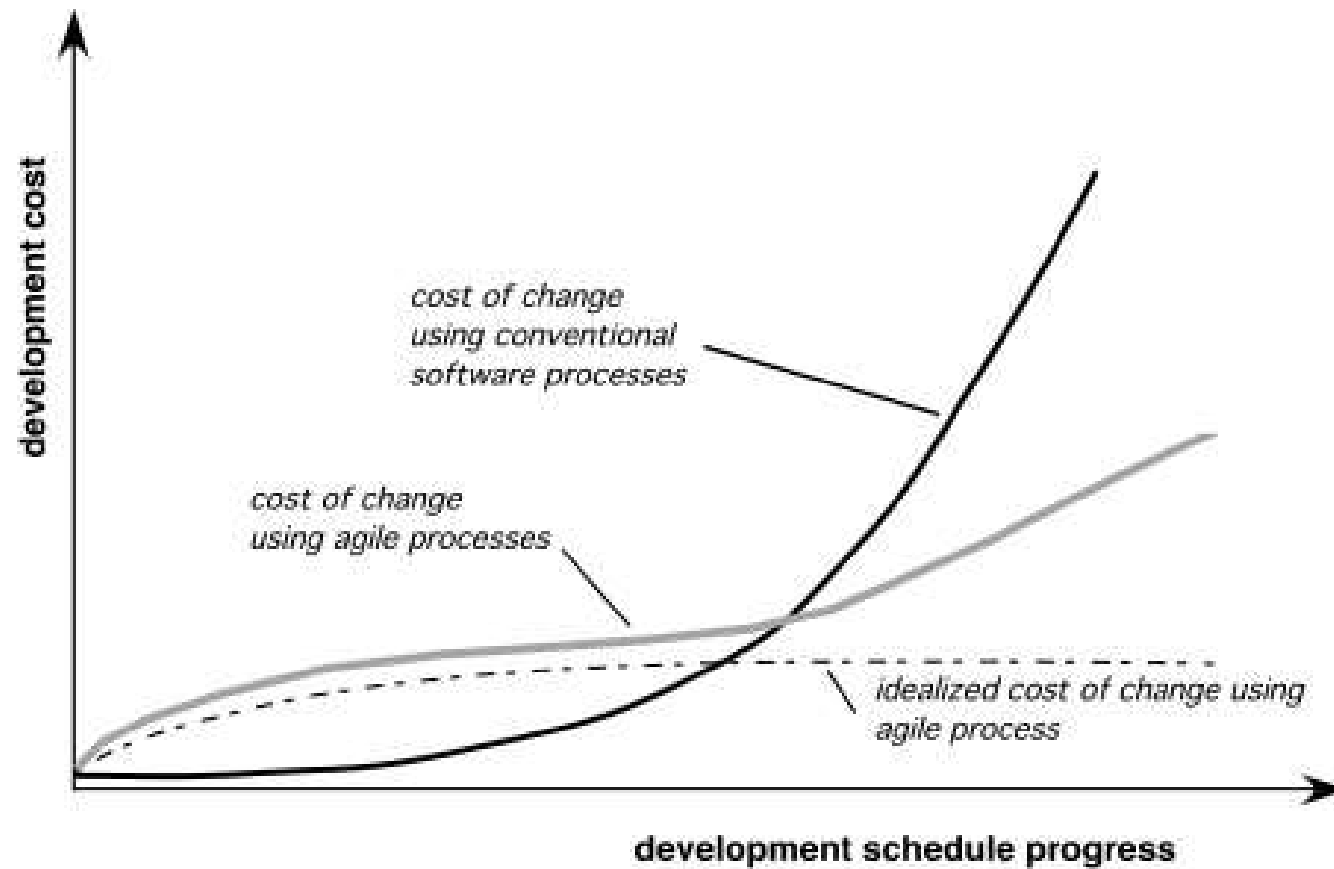
What is “Agility”?

- Effective (rapid and adaptive) response to change
- Effective communication among all stakeholders
- Drawing the customer onto the team
- Organizing a team so that it is in control of the work performed

Yielding ...

- Rapid, incremental delivery of software

Agility and the Cost of Change



An Agile Process

- Is driven by customer descriptions of what is required (scenarios)
- Recognizes that plans are short-lived
- Develops software iteratively with a heavy emphasis on construction activities
- Delivers multiple ‘software increments’
- Adapts as changes occur

Agility Principles - I

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes 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. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Agility Principles - II

7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.
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 behavior accordingly.

Human Factors

- *the process molds to the needs of the people and team, not the other way around*
- key traits must exist among the people on an agile team and the team itself:
 - **Competence.**
 - **Common focus.**
 - **Collaboration.**
 - **Decision-making ability.**
 - **Fuzzy problem-solving ability.**
 - **Mutual trust and respect.**
 - **Self-organization.**

Extreme Programming (XP)

- The most widely used agile process, originally proposed by Kent Beck
- XP Values
 - communication,
 - simplicity,
 - feedback,
 - courage, and
 - respect

XP Process

- XP Planning
 - Begins with the creation of “**user stories**”
 - Agile team assesses each story and assigns a **cost**
 - Stories are grouped to for a **deliverable increment**
 - A **commitment** is made on delivery date
 - After the first increment “**project velocity**” is used to help define subsequent delivery dates for other increments

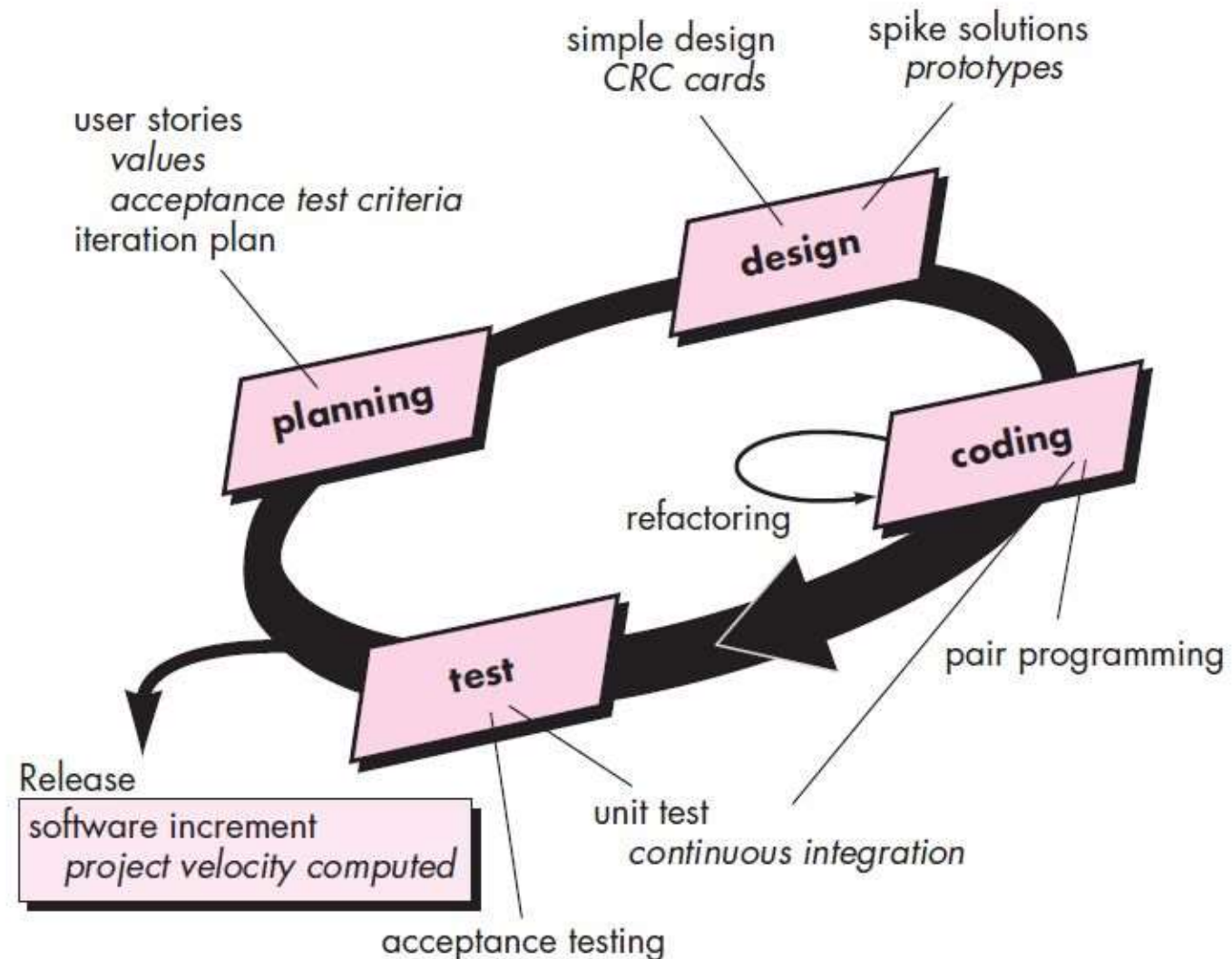
Extreme Programming (XP)

- XP Design
 - Follows the **KIS principle**
 - Encourage the use of **CRC cards** (see Chapter 8)
 - For difficult design problems, suggests the creation of “**spike solutions**”—a design prototype
 - Encourages “**refactoring**”—an iterative refinement of the internal program design
- XP Coding
 - Recommends the **construction of a unit test** for a store *before* coding commences
 - Encourages “**pair programming**”
- XP Testing
 - All **unit tests are executed daily**
 - “**Acceptance tests**” are defined by the customer and executed to assess customer visible functionality

Extreme Programming (XP)

FIGURE 3.2

The Extreme Programming process



Industrial XP

- IXP is an organic evolution of XP. It is imbued with XP's minimalist, customer-centric, test-driven spirit.
- IXP differs most from the original XP in its greater inclusion of management, its expanded role for customers, and its upgraded technical practices.”
- IXP incorporates six new practices that are designed to help ensure that an XP project works successfully for significant projects within a large organization.

1. Readiness assessment

The assessment ascertains whether

- (1) an appropriate development environment exists to support IXP,
- (2) the team will be populated by the proper set of stakeholders,
- (3) the organization has a distinct quality program and supports continuous improvement,
- (4) the organizational culture will support the new values of an agile team, and
- (5) the broader project community will be populated appropriately.

2. **Project community**
3. **Project chartering**
4. **Test-driven management**
5. **Retrospectives**

a *retrospective*, the review examines “issues, events, and lessons-learned”

6. **Continuous learning**
 - *Story-driven development (SDD)*
 - *Domain-driven design (DDD)*
 - *Pairing*
 - *Iterative usability*

The XP Debate

- *Requirements volatility.*
- *Conflicting customer needs.*
- *Requirements are expressed informally.*
- *Lack of formal design.*

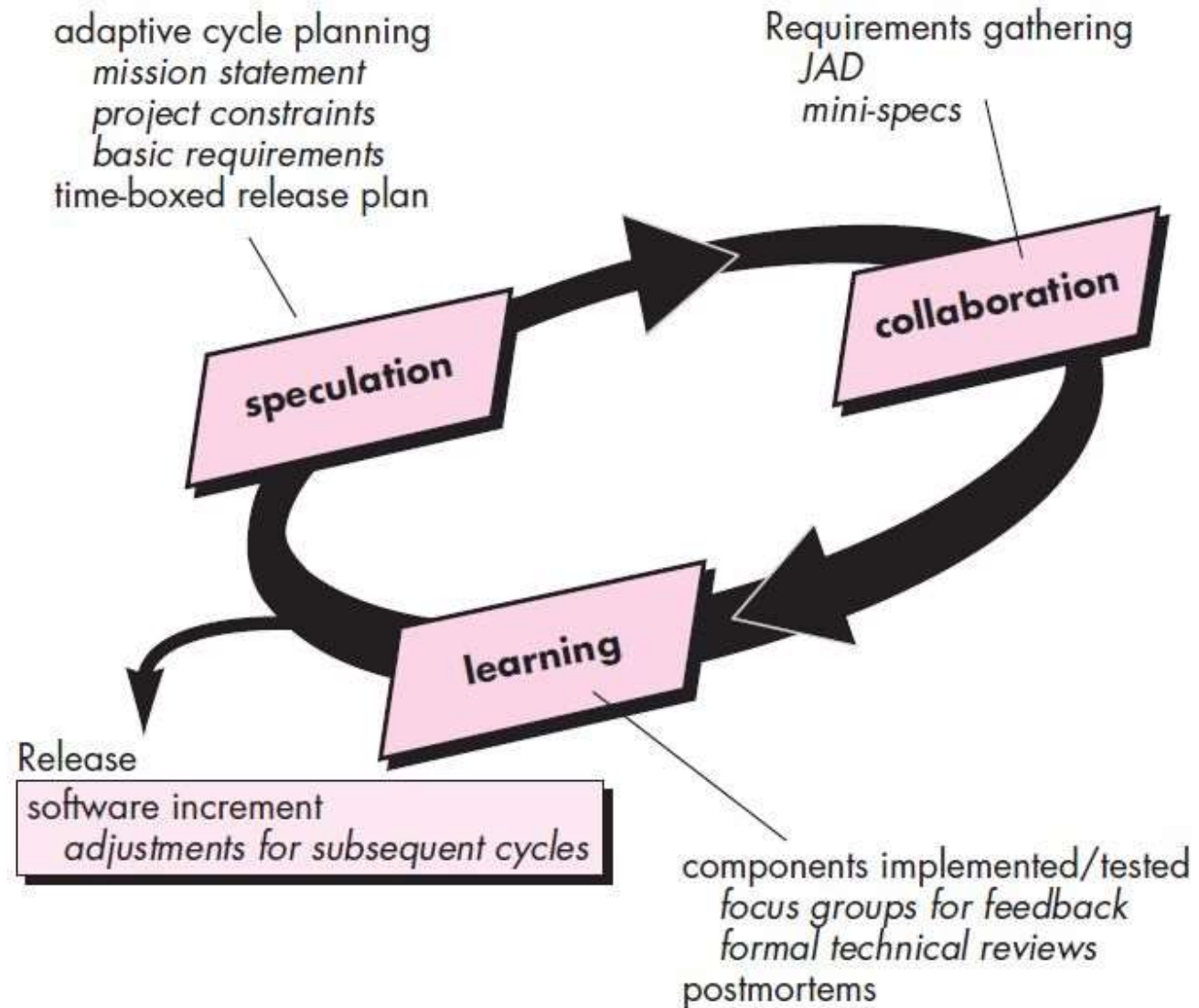
Adaptive Software Development

- Originally proposed by Jim Highsmith
- ASD — distinguishing features
 - Mission-driven planning
 - Component-based focus
 - Uses “time-boxing” (See Chapter 24)
 - Explicit consideration of risks
 - Emphasizes collaboration for requirements gathering
 - Emphasizes “learning” throughout the process

Adaptive Software Development

FIGURE 3.3

Adaptive
software
development



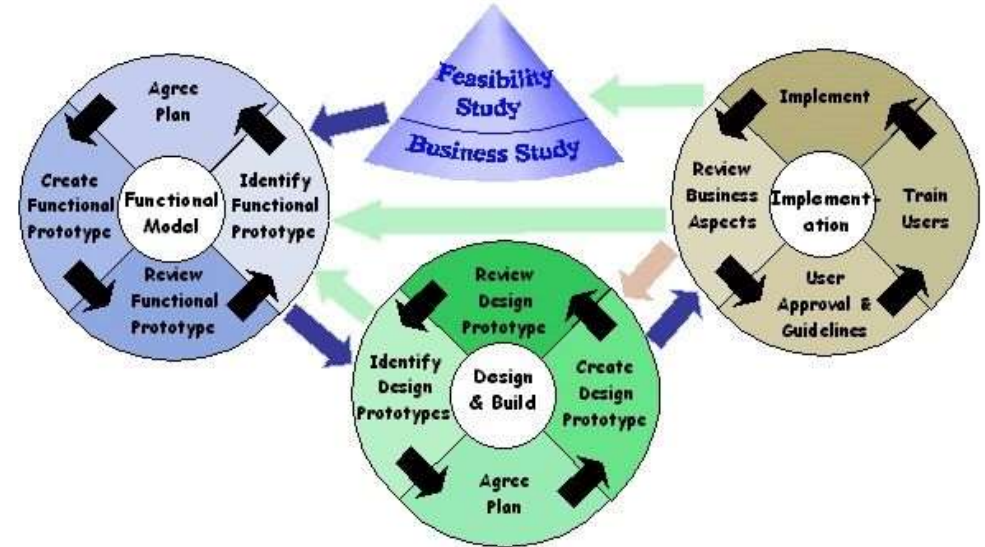
Dynamic Systems Development Method

- Promoted by the DSDM Consortium (www.dsdm.org)
- DSDM—distinguishing features
 - Similar in most respects to XP and/or ASD
 - Nine guiding principles
 - Active user involvement is imperative.
 - DSDM teams must be empowered to make decisions.
 - The focus is on frequent delivery of products.
 - Fitness for business purpose is the essential criterion for acceptance of deliverables.
 - Iterative and incremental development is necessary to converge on an accurate business solution.
 - All changes during development are reversible.
 - Requirements are baselined at a high level
 - Testing is integrated throughout the life-cycle.

Dynamic Systems Development Method

DSDM life cycle that defines three different iterative cycles, preceded by two additional life cycle activities:

*Feasibility
study
Business
study
Functional model
iteration
Design and build
iteration
Implementatio
n*



DSDM Life Cycle (with permission of the DSDM consortium)

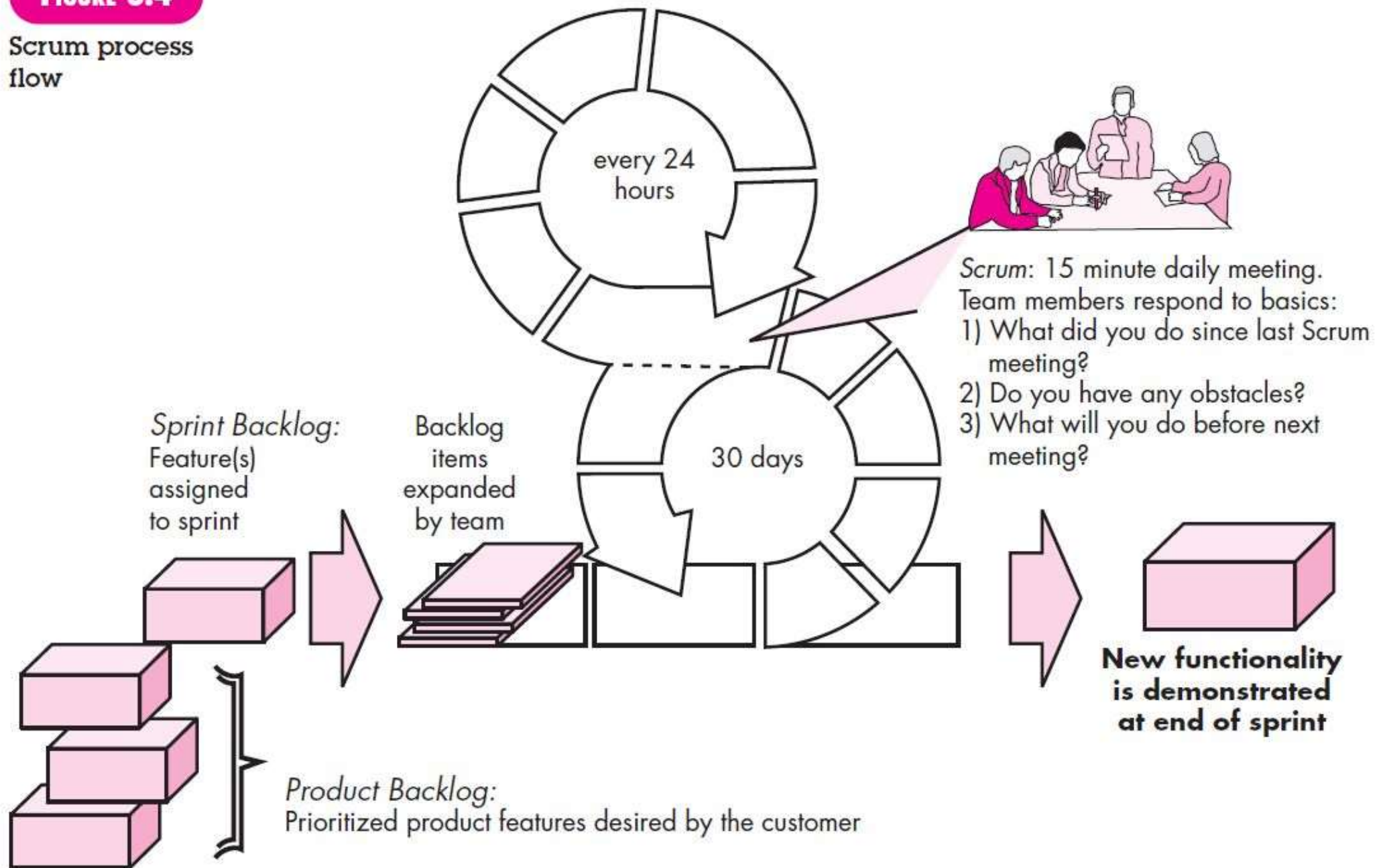
Scrum

- Originally proposed by Schwaber and Beedle
- Scrum—distinguishing features
 - Development work is partitioned into “**packets**”
 - **Testing and documentation are on-going** as the product is constructed
 - Work occurs in “**sprints**” and is derived from a “**backlog**” of existing requirements
 - **Meetings are very short** and sometimes conducted without chairs
 - “**demos**” are delivered to the customer with the time-box allocated

Scrum

FIGURE 3.4

Scrum process flow



Crystal

- Proposed by Cockburn and Highsmith
- Crystal—distinguishing features
 - Actually a **family of process models** that allow “**maneuverability**” based on problem characteristics
 - **Face-to-face communication** is emphasized
 - Suggests the use of “**reflection workshops**” to review the work habits of the team

Feature Driven Development

- Originally proposed by Peter Coad et al
- FDD—distinguishing features
 - Emphasis is on defining “features”
 - a *feature* “is a client-valued function that can be implemented in two weeks or less.”
 - Uses a **feature template**
 - <action> the <result> <by | for | of | to> a(n) <object>
 - A **features list** is created and “**plan by feature**” is conducted
 - Design and construction merge in FDD

Add the product to shopping cart

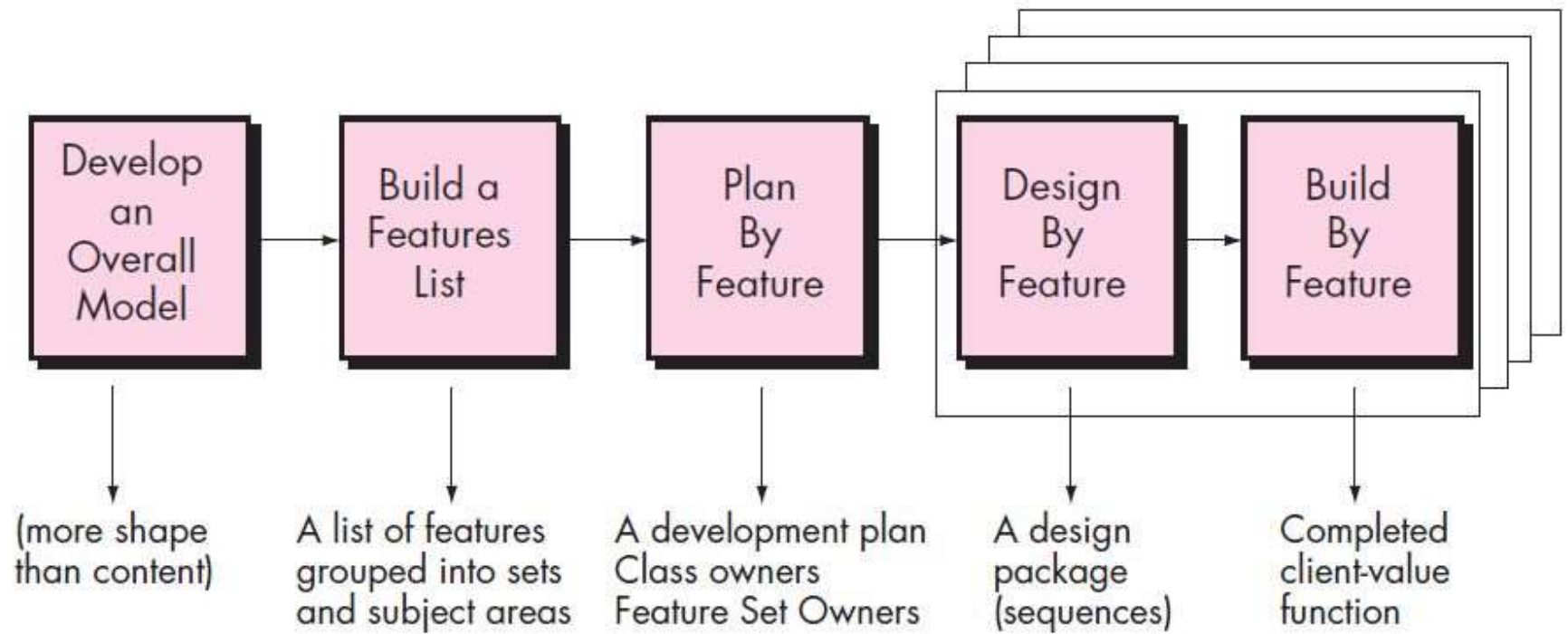
Display the technical-specifications of the product

Store the shipping-information for the customer

Feature Driven Development

FIGURE 3.5

Feature Driven Development [Coa99] (with permission)



<action><-ing> a(n) <object>

Like other agile approaches, FDD adopts a philosophy that

- (1) emphasizes collaboration among people on an FDD team;
- (2) manages problem and project complexity using feature-based decomposition followed by the integration of software increments, and
- (3) communication of technical detail using verbal, graphical, and text-based means.

The emphasis on the definition of features provides the following benefits:

- Because features are small blocks of deliverable functionality, users can describe them more easily; understand how they relate to one another more readily; and better review them for ambiguity, error, or omissions.
- Features can be organized into a hierarchical business-related grouping.
- Since a feature is the FDD deliverable software increment, the team develops operational features every two weeks.
- Because features are small, their design and code representations are easier to inspect effectively.
- Project planning, scheduling, and tracking are driven by the feature hierarchy, rather than an arbitrarily adopted software engineering task set.

Lean Software Development (LSD)

- *eliminate waste,*
- *build quality in,*
- *Create knowledge,*
- *defer commitment,*
- *deliver fast,*
- *respect people, and*
- *optimize the whole*

Agile Modeling

- Originally proposed by Scott Ambler
- Suggests a set of agile modeling principles
 - Model with a purpose
 - Use multiple models
 - Travel light
 - Content is more important than representation
 - Know the models and the tools you use to create them
 - Adapt locally