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

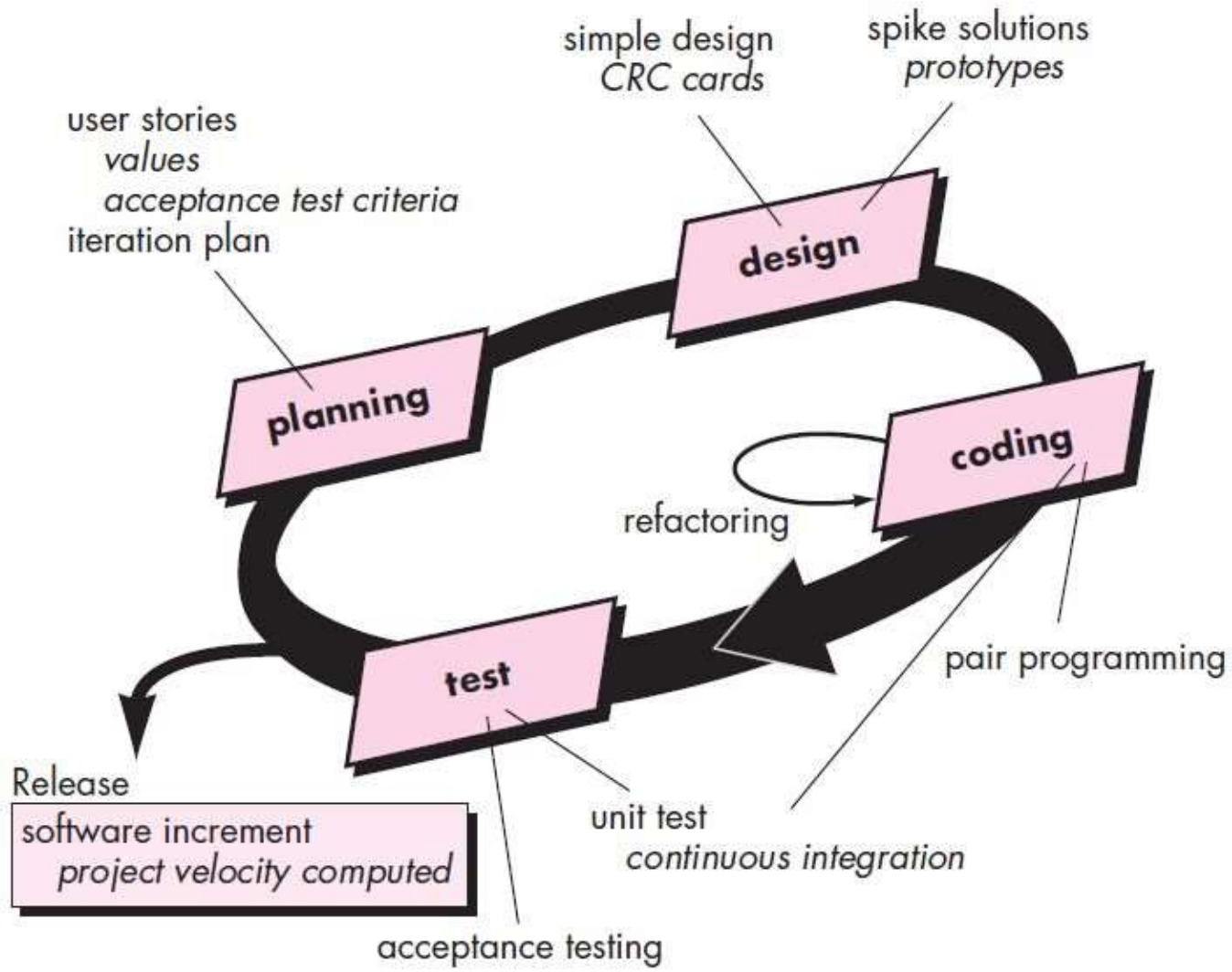
Extreme Programming (XP)

- 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 excuted 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 upgradedtechnical 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.*

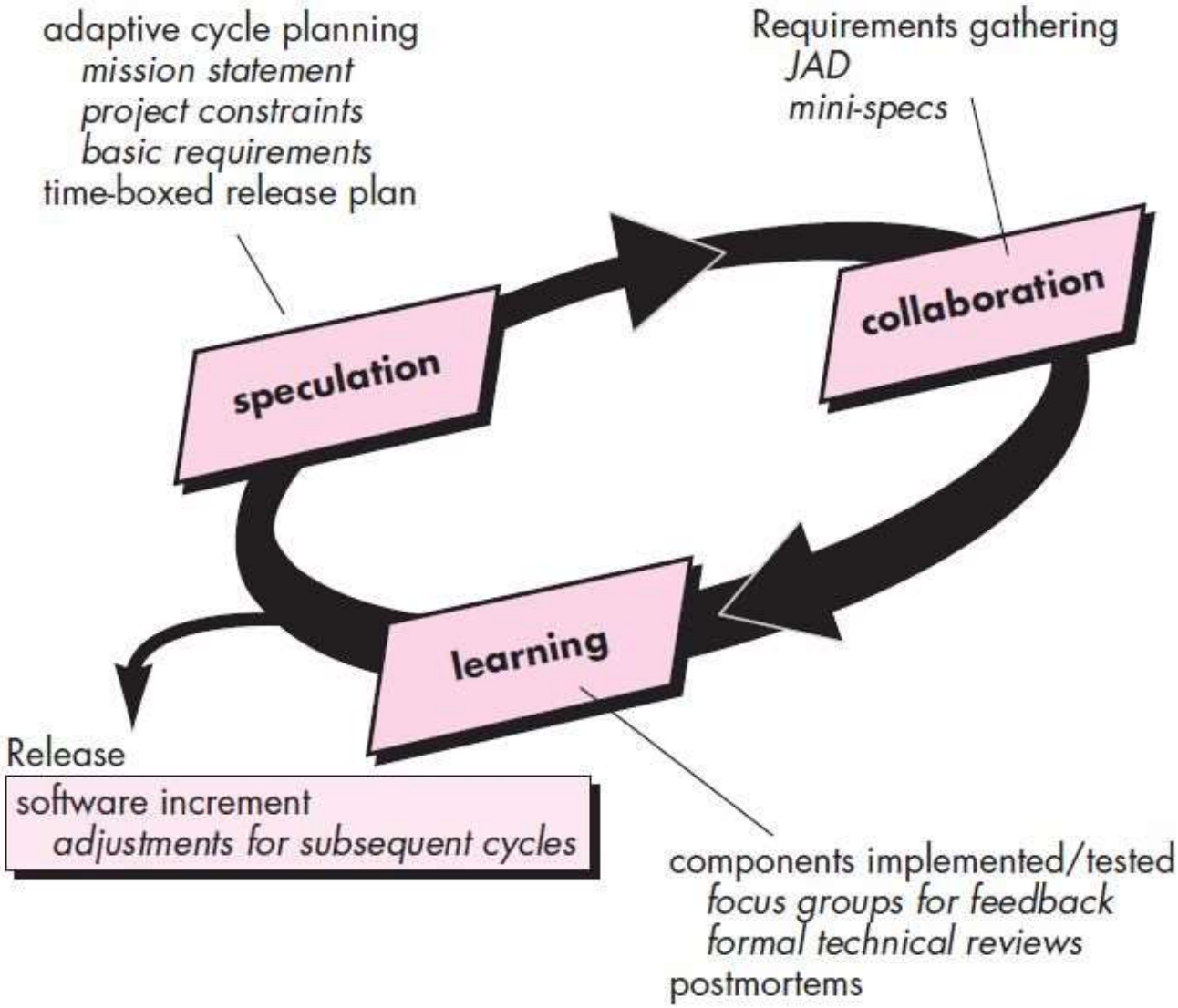
Adaptive Software Development

- 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



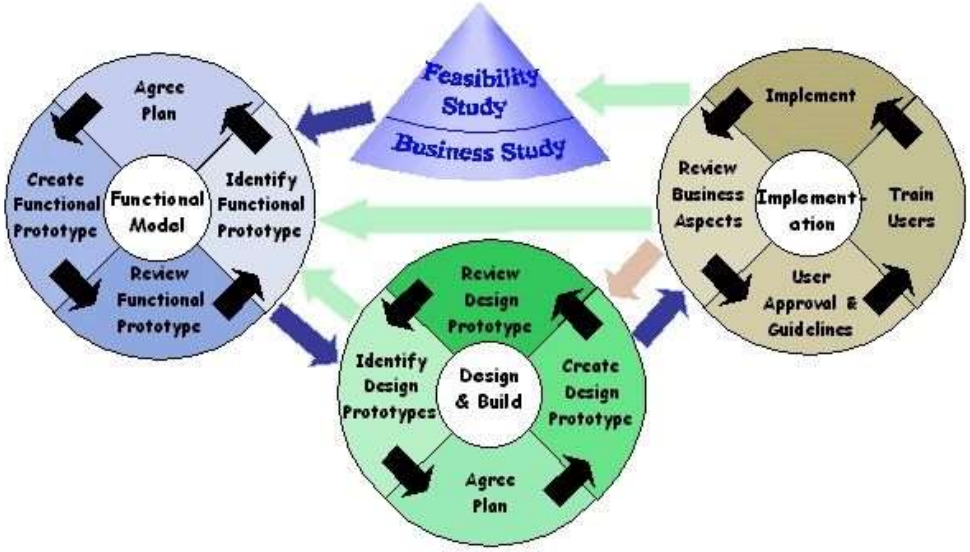
Dynamic Systems Development Method

- 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
Implementation



DSDM Life Cycle (with permission of the DSDM consortium)

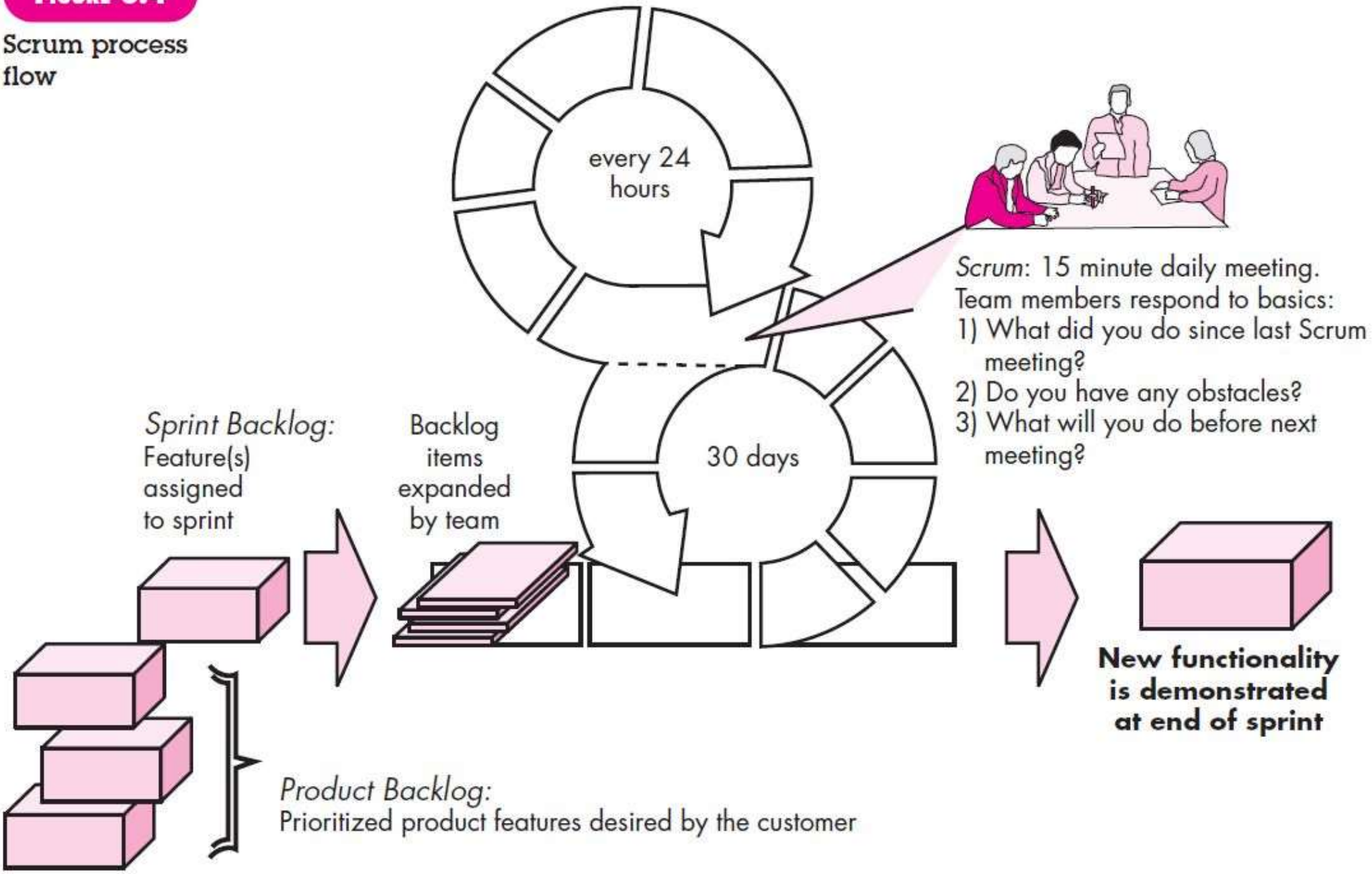
Scrum

- 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

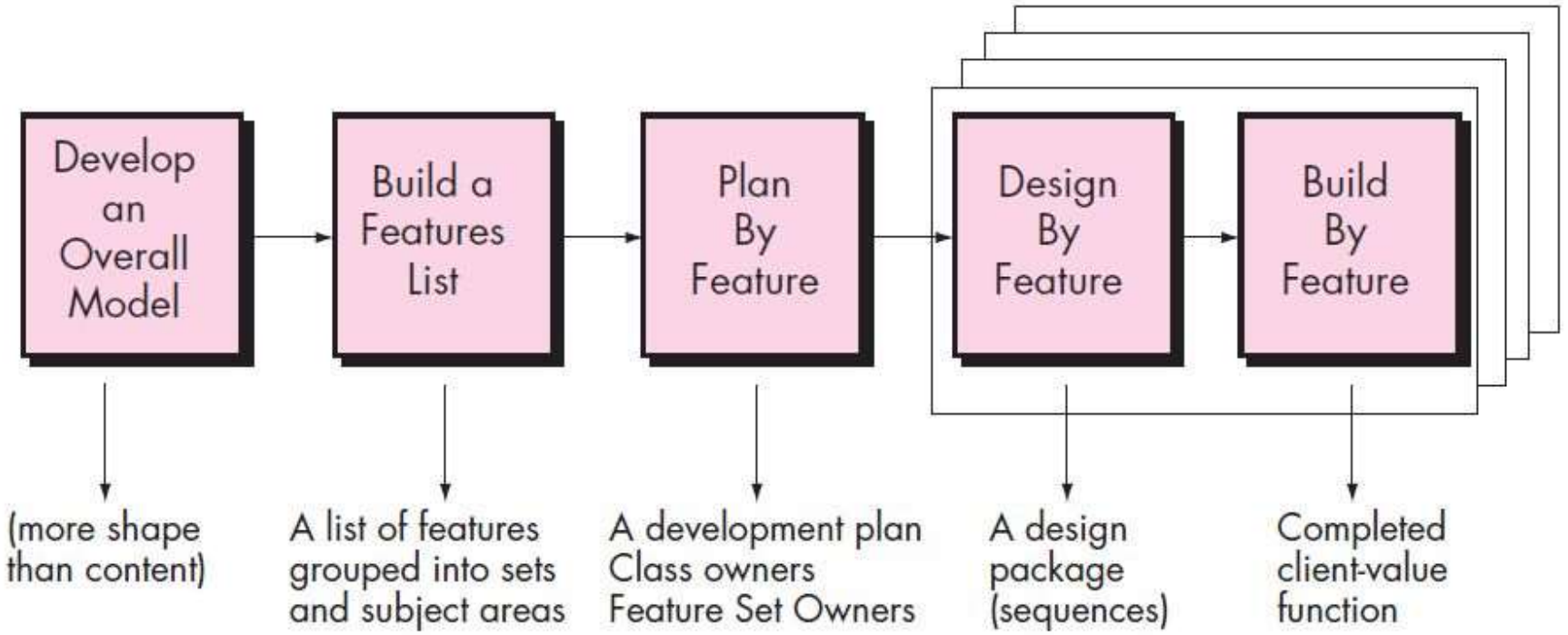
Display the technical-specifications of the product

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 collaborationamong 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 morereadily; 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