Chapter 3

- Agile Development

*Slide Set to accompany*

*Software Engineering: A Practitioner’s Approach, 7/e*

*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.

These slides are designed to accompany *Software Engineering: A Practitioner’s Approach, 7/e* (McGraw-Hill, 2009) Slides copyright 2009 by Roger Pressman.
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
- Uses an object-oriented approach as its preferred development paradigm

XP Planning

- Begins with the creation of “user stories”
- Agile team assesses each story and assigns a cost
- Stories are grouped to form 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. ‘Project velocity’ is the number of customer ‘stories’, i.e., requirements, implemented during first release
Extreme Programming (XP)

- **XP Design**
  - Follows the KIS principle
  - Encourage the use of CRC cards (see Chapter 8) – class responsibility collaborator – identify & organize the object oriented classes that are relevant to the current software increment
  - For difficult design problems, suggests the creation of “spike solutions”—a design prototype – intent is to lower risk when true implementation starts & to validate the original estimates for the ‘story’ containing the design problem
  - Encourages “refactoring”—an iterative refinement of the internal program design without altering the external behavior of the code

- **XP Coding**
  - Recommends the construction of a unit test for a store *before* coding commences
  - Encourages “pair programming” – two people working at one workstation – two heads better than one

- **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)

- User stories
- Values
- Acceptance test criteria
- Iteration plan
- Simple design
- CRC cards
- Spike solutions
- Prototypes
- Design
- Refactoring
- Coding
- Pair programming
- Test
- Unit test
- Continuous integration
- Acceptance testing
- Release
- Software increment
- Project velocity computed
- Iteration plan

These slides are designed to accompany Software Engineering: A Practitioner’s Approach, 7/e (McGraw-Hill, 2009) Slides copyright 2009 by Roger Pressman.
Adaptive Software Development – a technique for building complex software and systems. The philosophical underpinnings of ASD focus on human collaboration & team self organization.

- Originally proposed by Jim Highsmith
- ASD — distinguishing features
  - Mission-driven planning
  - Component-based focus
  - Uses “time-boxing” (See Chapter 24)
  
  **Timeboxing** is a planning technique common in planning projects (typically for software development), where the schedule is divided into a number of separate time periods (timeboxes, normally two to six weeks long), with each part having its own deliverables, deadline and budget.
  
  - Explicit consideration of risks
  - Emphasizes collaboration for requirements gathering
  - Emphasizes “learning” throughout the process
Adaptive Software Development

- adaptive cycle planning
  - uses mission statement
  - project constraints
  - basic requirements
  - time-boxed release plan

- Requirements gathering
  - JAD
  - mini-specs

- Collaboration
  - speculation
  - learning
  - components implemented/tested
  - focus groups for feedback
  - formal technical reviews
  - post mortems

- Time-boxing is a planning technology

Release
- software increment
- adjustments for subsequent cycles

Learning also refers to learning the technology, the process, & the project

Dynamic Systems Development Method

Another Agile software development approach. For systems which meet tight time constraints

- Promoted by the DSDM Consortium (www.dsdm.org)
- DSDM—distinguishing features
  - Similar in most respects to XP and/or ASD (Adaptive Software Development)
  - 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.
    - This method believes in the ‘Pareto Principle’ (80-20 rule) – 80% of an application can be delivered in 20% of the time it would take to deliver 100% of the application. Only enough work should be done in each increment to facilitate movement to the next increment.
Dynamic Systems Development Method

DSDM Life Cycle (with permission of the DSDM consortium)

These slides are designed to accompany *Software Engineering: A Practitioner’s Approach, 7/e* (McGraw-Hill, 2009) Slides copyright 2009 by Roger Pressman.
Scrum — another Agile method. Good for projects that have tight timelines, changing requirements, and business criticality

- 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

Scrum Process Flow (used with permission)

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
  - Is actually a set of example agile processes that have been proven effective for different types of projects. The intent is to allow agile teams to select the member of the Crystal family that is most appropriate for their project and environment.
Feature Driven Development - practical for object-oriented software engineering

- 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>
      Where an <object> is a person, place or thing. Examples of features for an e-commerce application might be:
        - Add the product to shopping cart
        - Display the technical specs of the product
        - Store the shipping info for the customer
  - A features list is created and “plan by feature” is conducted
  - Design and construction merge in FDD
Feature Driven Development

Reprinted with permission of Peter Coad

Agile Modeling

- Originally proposed by Scott Ambler
- Suggests a set of agile modeling principles
  - Model with a purpose – e.g., to communicate information to the customer, or to better understand some aspect of the software
  - Use multiple models
  - Travel light - as software engineering proceeds, keep only those models that will provide long term value and jettison the rest.
  - Content is more important than
  - Know the models and the tools you use to create them representation – understand the strengths & weaknesses of each model & the tools that are used to create it.
  - Adapt locally – the modeling approach should be adapted to the needs of the agile team.