CS435: Introduction to Software Engineering

Chapter 3

■■ Agile Development

Slide Set to accompany
Software Engineering: A Practitioner's Approach, 7/e
by Roger S. 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 (team members, new technology, requirements).
- Effective communication in structure and attitude among all team members, technological and business people, software engineers and the manager.
- Drawing the customer onto the team. Eliminate "us and them" attitude. Planning in an uncertain world has its limits and plan must be flexible.
- Organizing a team so that it is in control of the work performed.

What is "Agility"?

- Eliminate all but the most essential work products and keep them lean.
- Emphasize an incremental delivery strategy as opposed to intermediate products that gets working software to the customer as rapidly as feasible.

What is “Agility”?

Yielding …

- Rapid, incremental delivery of software
- The development guidelines stress delivery over analysis and design (although these activities are not discouraged), and active and continuous communication between developers and customers.

What and Why of “Agility”

What?

- May be termed as “software engineering lite”
- The basic activities- communication, planning, modeling, construction and deployment remain.
- These activities morph into a minimal task set that push the team toward construction and delivery sooner.

What and Why of "Agility"

Why?

- The modern business environment is fast-paced and ever-changing
- Represents a reasonable alternative to conventional software engineering for certain classes of software projects
- Has been demonstrated to deliver successful systems quickly

What and Why of “Agility”

- The only really important work product is an operational “software increment” that is delivered.

Agility and the Cost of Change

Conventional wisdom

- The cost of change increases nonlinearly as a project progresses.
- It is relatively easy to accommodate a change when a team is gathering requirements early in a project.
- If there are any changes, the costs of doing this work are minimal.

Agility and the Cost of Change

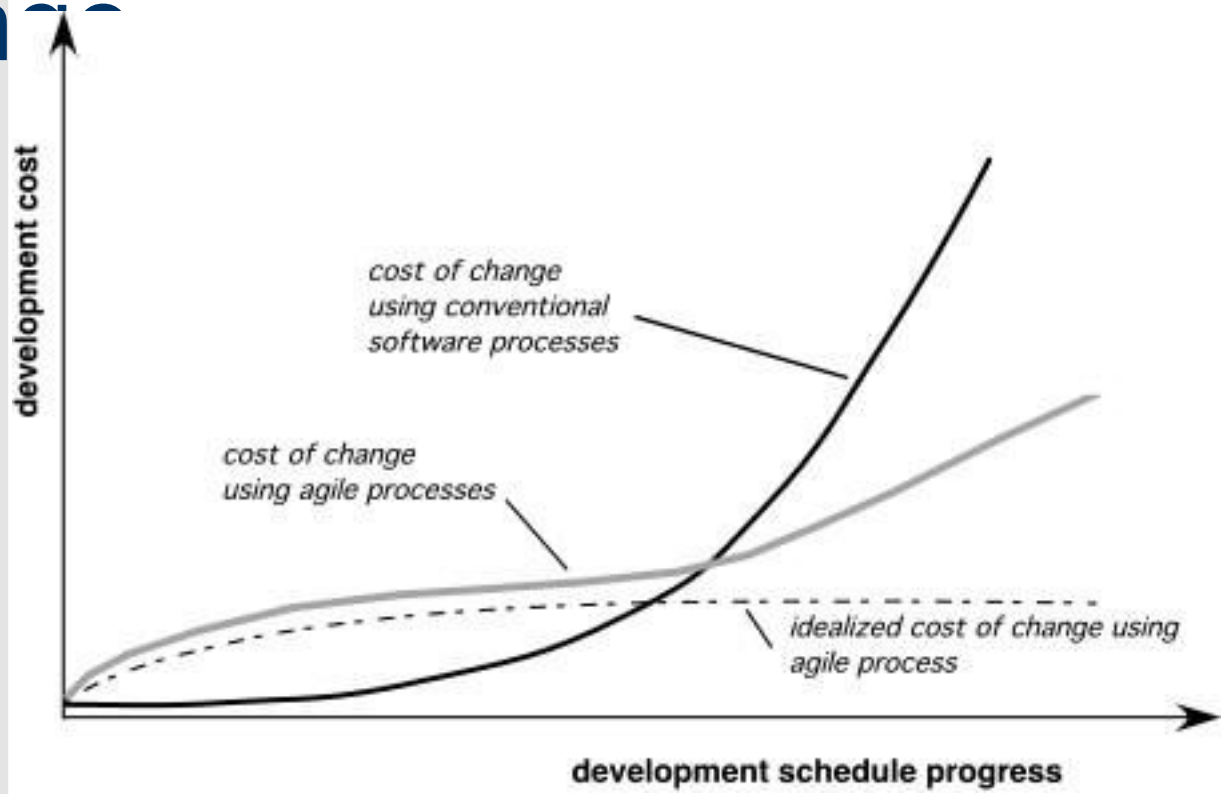
Conventional wisdom

- However, in the middle of validation testing, if a stakeholder requests a major functional change, then the change requires a modification to the architectural design, construction of new components, changes in other existing components, new testing and so on.
- Costs escalate quickly.

Agility and the Cost of Change

- A well-designed agile process may "flatten" the cost of change curve by coupling incremental delivery with agile practices such as continuous unit testing and pair programming. Thus team can accommodate changes late in the software project without dramatic cost and time impact.

Agility and the Cost of Change



An Agile Process

- Driven by customer descriptions of what is required (scenarios). Some assumptions:
  - Recognizes that plans are short-lived:
    - some requirements will persist
    - some requirements will change
    - customer priorities will change

An Agile Process

- Driven by customer descriptions of what is required (scenarios). Some assumptions:
  - Develops software iteratively with a heavy emphasis on construction activities
    - design and construction are interleaved
    - hard to predict how much design is necessary before construction
    - Design models are assessed as they are created
  - Analysis, design, construction and testing are not predictable.

An Agile Process

- Therefore, the process has to Adapt as changes occur due to unpredictability
- Delivers multiple 'software increments'
  - deliver an operational prototype or portion of an OS to collect customer feedback for adaptation.

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 for the shorter timescale.

Agility Principles - II

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 - III

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.

Agility Principles - IV

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 that must exist among the people in an agile team and the team itself:
  - **Competence.**
    - **Talent, skills, knowledge**
  - **Common focus.**
    - **Deliver a working software increment**
  - **Collaboration.**
    - **With peers and stakeholders**
  - **Decision-making ability.**
    - **Freedom to control its own destiny**

Human Factors

- key traits must exist among the people in an agile team and the team itself:
  - **Fuzzy problem-solving ability.**
    - **Ambiguity and constant changes, today's problem might not persist till tomorrow**
  - **Mutual trust and respect.**
  - **Self-organization.**
    - **themselves for the work done, process for its local environment, the work schedule**

Extreme Programming (XP)

- The most widely used agile process, originally proposed by Kent Beck in 2004.
- Uses an object-oriented approach.
- Steps:
  - XP Planning
  - XP Design
  - XP Coding
  - XP Testing

Extreme Programming (XP)

- XP Planning
  - Begins with listening to the requirements of the customer
  - leads to creation of "user stories"
  - XP **user story** is a short description (usually in two or three sentences) of a certain function of the future product.
  - These user **stories** do not provide enough information to turn them into project tasks.
  - The stories include their required output, features, and functionality.
  - Customer assigns a value(i.e., a priority) to each story.

Extreme Programming (XP)

- XP Planning
  - Agile team assesses each story and assigns a cost
    - Usually in development weeks.
    - If the cost is more than 3 weeks, customer is asked to split that particular story into smaller stories
  - Working together, stories are grouped for a deliverable increment next release.

Extreme Programming (XP)

- XP Planning
  - A commitment (stories to be included, delivery date and other project matters) is made. There can be three ways of doing so:
    - all stories will be implemented in a few weeks
    - higher priority stories will be implemented first
    - the riskiest stories will be implemented first

Extreme Programming (XP)

- XP Planning
  - After the first increment, "project velocity", i.e., the number of stories implemented during the first release is used to help define subsequent delivery dates for other increments.
  - Customers can add stories, delete existing stories, change values of an existing story, split stories as development work proceeds

Extreme Programming (XP)

- XP Design (occurs both before and after coding as refactoring is encouraged)
  - Follows the KIS principle (keep it simple)
    - Nothing more and nothing less than the story.
  - Encourage the use of CRC (class-responsibility-collaborator) cards in an object-oriented context.
    - The only design work product of XP.
    - They identify and organize the classes that are relevant to the current software increment.

Extreme Programming (XP)

- XP Design
  - For difficult design problems, suggests the creation of "spike solutions"
    - a design prototype for that portion is implemented and evaluated.
  - Encourages "refactoring"
    - an iterative refinement of the internal program design.
    - Does not alter the external behavior yet improve the internal structure.
    - Minimize chances of bugs.
    - More efficient, easy to read.

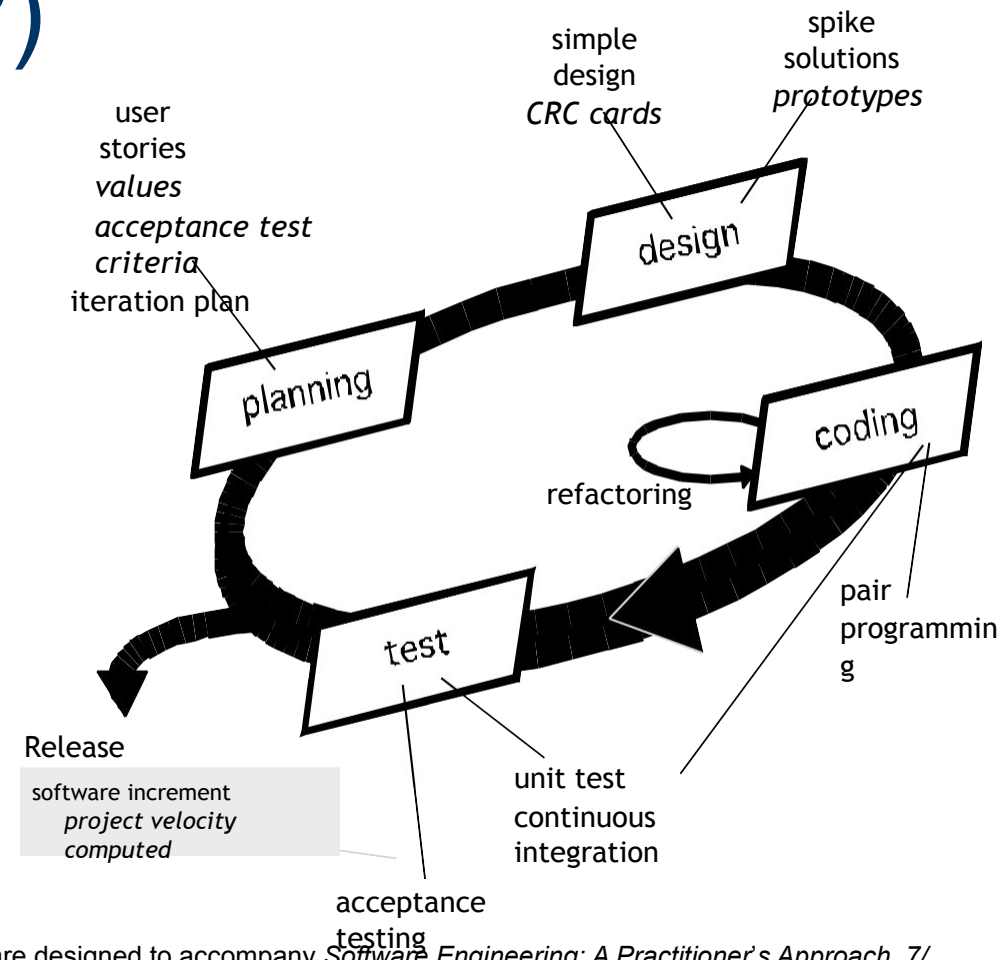
Extreme Programming (XP)

- XP Coding
  - Recommends the construction of a unit test for a story *before* coding commences. So, the implementer can focus on what must be implemented to pass the test.
  - Encourages "pair programming". Two people work together at one workstation.
    - real time problem solving
    - real time review for quality assurance
    - take slightly different roles.

Extreme Programming (XP)

- XP Testing
  - All unit tests are executed daily and ideally, they should be automated.
  - Regression tests are conducted to test the current and previous components.
  - "Acceptance tests" are defined by the customer and are executed to assess customer visible functionality

Extreme Programming (XP)



The XP Debate

- Requirements volatility:
  - Since the customer is an active member of XP team, changes in requirements are requested informally and frequently.
- Conflicting customer needs:
  - Different customers' needs need to be assimilated.
  - Different vision or beyond their authority.

The XP Debate

- Requirements are expressed informally:
  - User stories and acceptance tests are the only explicit manifestation of requirements.
  - Formal models may avoid inconsistencies and errors before the system is built.
  - Proponents argue that the changing nature of this approach makes such models obsolete as soon as they are developed.

The XP Debate

- Lack of formal design:
  - XP deemphasizes the need for architectural design.
  - Complex systems need overall structure to exhibit quality and maintainability.
  - Proponents of XP argue that its incremental nature limits complexity as simplicity is a core value.

Adaptive Software Development (ASD)

- Originally proposed by Jim Highsmith (2000)
- Focusing on human collaboration and team self-organization as a technique to build complex software and system.

Adaptive Software Development (ASD)

- ASD: distinguishing features
  - Mission-driven planning
  - Component-based focus
  - Uses "time-boxing"
    - **Timeboxing** refers to the act of putting strict **time** boundaries around an action or activity
    - For example, you may want to timebox a meeting to be 30 minutes long to help ensure that the meeting will begin and end on **time** with no exceptions.
    - **timeboxing** is a constraint used by teams to help focus on value.
  - Explicit consideration of risks
  - Emphasizes on collaboration for requirements gathering
  - Emphasizes on "learning" throughout the process

Three Phases of ASD

1. Speculation:
   - The project is initiated and adaptive cycle planning is conducted.
   - Adaptive cycle planning uses project initiation information- the customer's mission statement, project constraints (e.g. delivery date), and basic requirements to define the set of release cycles (increments) that will be required for the project.
   - Based on the information obtained at the completion of the first cycle, the plan is reviewed and adjusted so that planned work better fits the reality.

Three Phases of ASD

2. Collaboration:

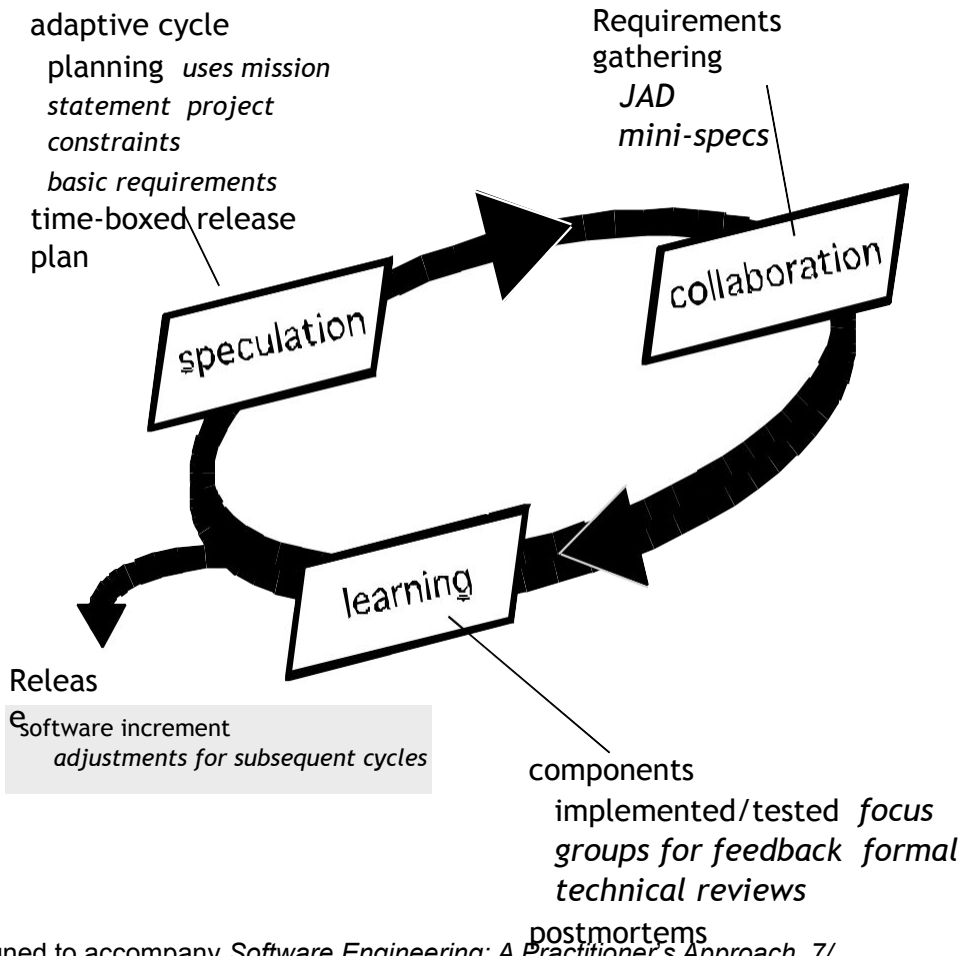
- Used to multiply talent and creative output beyond the absolute number of employees (1+1>2).
- Encompasses communication and teamwork, but also emphasizes on individualism, since individual creativity plays an important role in collaborative thinking.
- To facilitate collaboration:
  - criticize without animosity
  - assist without resentments
  - work as hard as or harder than the others do
  - have the skill set to contribute to the work at hand
  - communicate problems or concerns in a way that leads to effective action.

Three Phases of ASD

3. Learning:

- As members of ASD team begin to develop the components, they emphasize on "learning".
- Highsmith argues that software developers often overestimate their own understanding of the technology, the process, and the project and that learning will help them improve their actual level of understanding.
- Three ways:
  - focus groups
  - technical reviews
  - project postmortems.

Adaptive Software Development



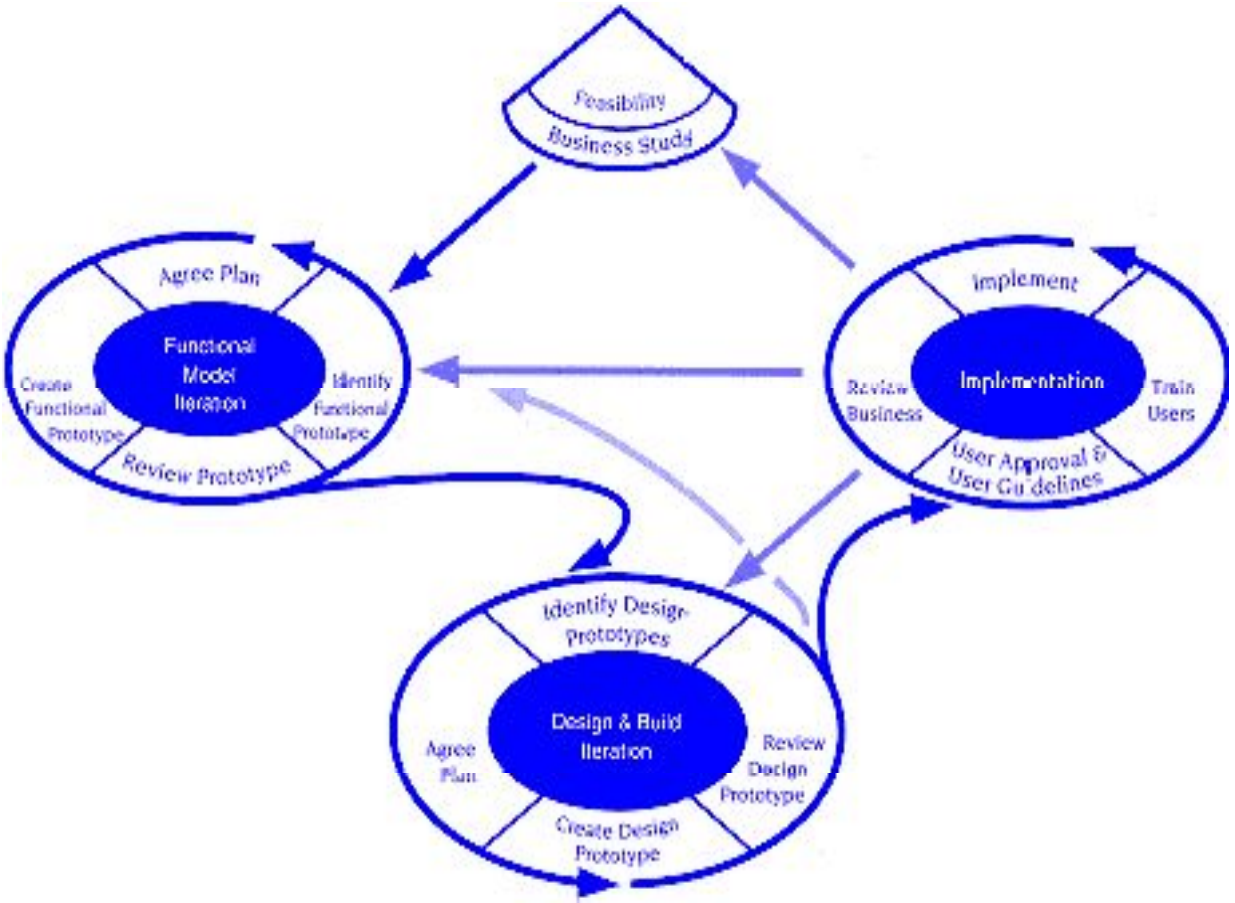
Dynamic Systems Development Method

- It is an agile software development approach that provides a framework for building and maintaining systems which meet tight time constraints through the use of incremental prototyping in a controlled project environment.
- Promoted by the DSDM Consortium (www.dsdm.org)

Dynamic Systems Development Method

- 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.
    - Focus is on frequent delivery of products.
    - Fitness for business purpose is an 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 (with permission of the DSDM consortium)

Scrum

- A software development method originally proposed by Schwaber and Beedle in early 1990.
- Scrum-distinguishing features
  - Development work is partitioned into "packets"
  - Testing and documentation are on-going as the product is constructed
  - Work units occurs in "sprints" and is derived from a "backlog" of existing changing prioritized requirements
  - Changes are not introduced in sprints (short term but stable) but in the backlog.

Scrum

- Scrum-distinguishing features
  - Meetings are very short (15 minutes daily) and sometimes conducted without chairs
    - what did you do since last meeting?
    - what obstacles are you encountering?
    - what do you plan to accomplish by next meeting?
  - "demos" are delivered to the customer with the time-box allocated.
    - might not contain all the functionalities.
    - customers can evaluate and give feedbacks.

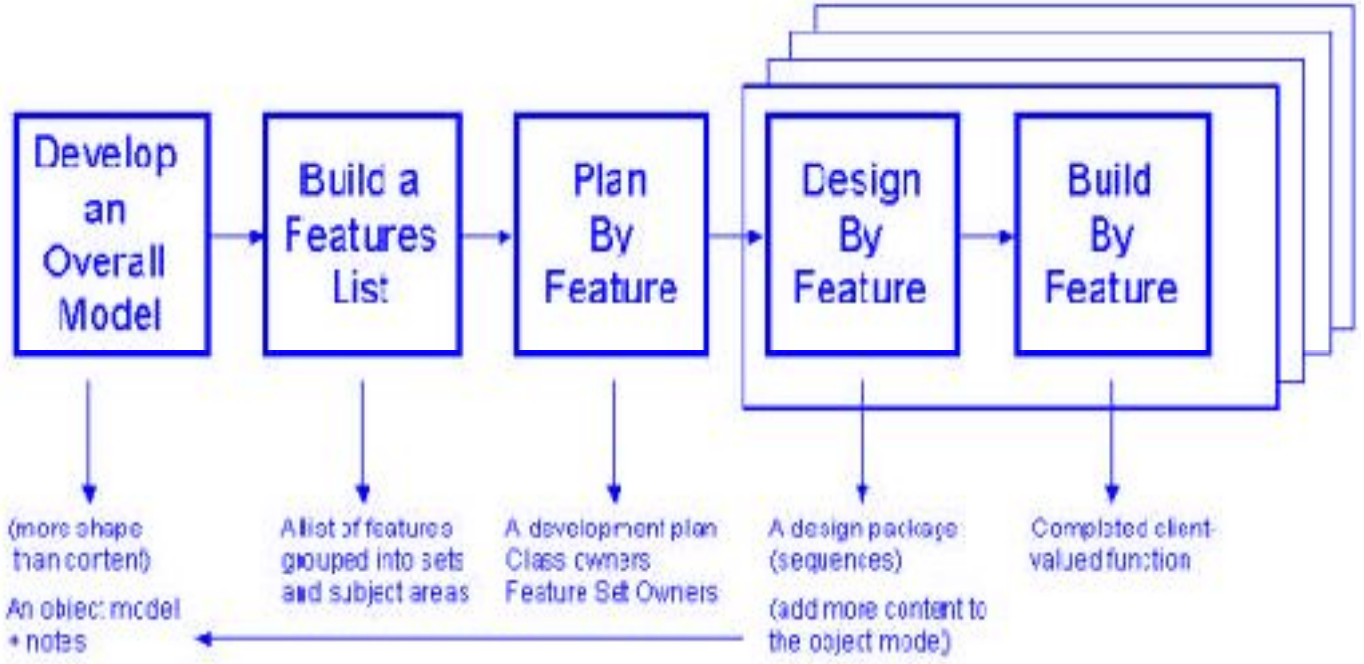
Crystal

- 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 as a object-oriented software engineering process model.
- FDD—distinguishing features
  - Emphasis is on defining "features" which can be organized hierarchically.
    - 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>
    - E.g. Add the product to shopping cart.
    - Display the technical-specifications of the product.
    - Store the shipping-information 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
  - Use multiple models
  - Travel light
  - Content is more important than representation
  - Know the models and the tools you use to create them
  - Adapt locally