CS435: Introduction to Software Engineering

Dr. M. Zhu

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 attitudes among all team members, technological and business people, software engineers and managers。
- Drawing the customer into 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
- 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 activates are not discouraged, and active and continuous communication between developers and customers.

Why and What Steps are "Agility" important?

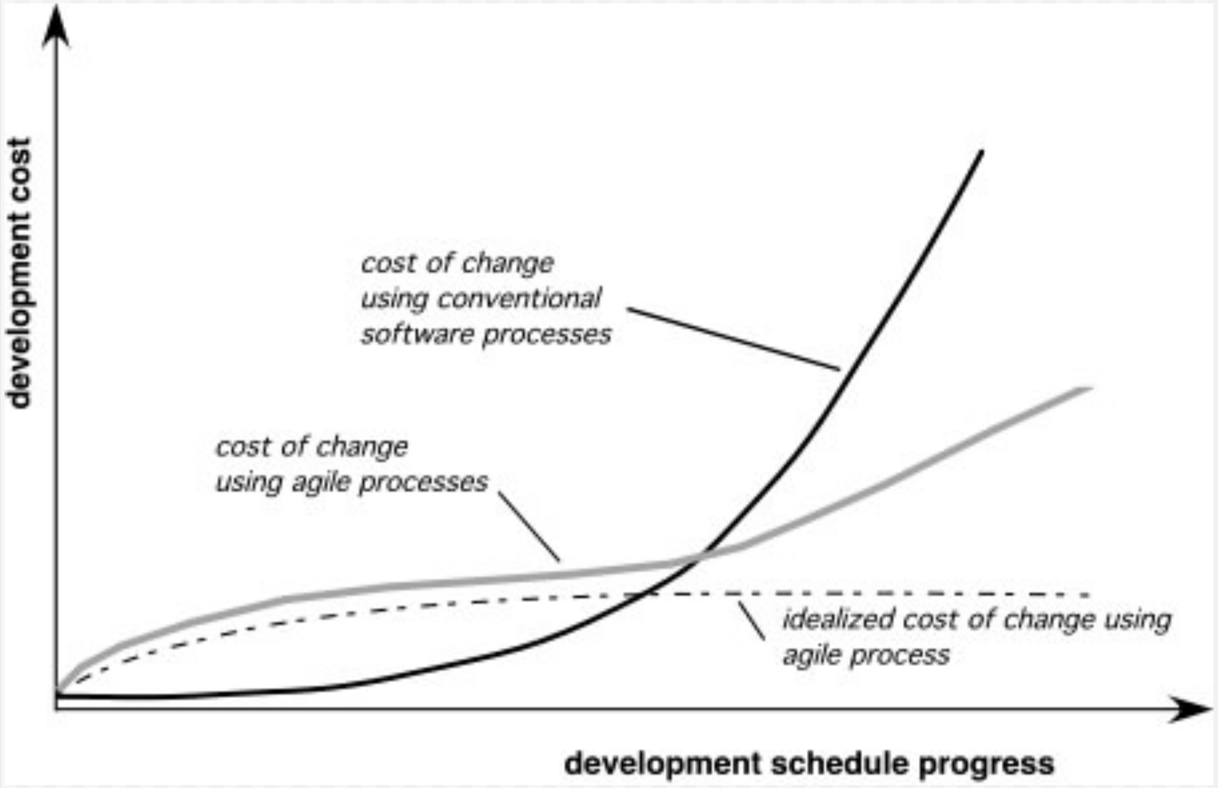
- Why? The modern business environment is fast-paced and ever-changing. It represents a reasonable alternative to conventional software engineering for certain classes of software projects. It has been demonstrated to deliver successful systems quickly.
- What? May be termed as "software engineering lite" The basic activities- communication, planning, modeling, construction and deployment remain. But they morph into a minimal task set that push the team toward construction and delivery sooner.
- The only really important work product is an operational "software increment" that is delivered.

Agility and the Cost of Change

- Conventional wisdom is that 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. But if the middle of validation testing, a stakeholder is requesting a major functional change. Then the change requires a modification to the architectural design, construction of new components, changes to other existing components, new testing and so on. Costs escalate quickly.

- 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

- Is driven by customer descriptions of what is required (scenarios). Some assumptions:
  - Recognizes that plans are short-lived (some requirements will persist, some will change. Customer priorities will change)
  - Develops software iteratively with a heavy emphasis on construction activities (design and construction are interleaved, hard to say how much design is necessary before construction. Design models are proven as they are created.)
  - Analysis, design, construction and testing are not predictable.
- Thus 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 adaption.

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. (talent, skills, knowledge)**
  - **Common focus. (deliver a working software increment)**
  - **Collaboration. (peers and stakeholders)**
  - **Decision-making ability. (freedom to control its own destiny)**
  - **Fuzzy problem-solving ability.(ambiguity and constant changes, today problem may not be tomorrow' s problem)**
  - **Mutual trust and respect.**
  - **Self-organization. (themselves for the work done, process for its local environment, the work schedule)**

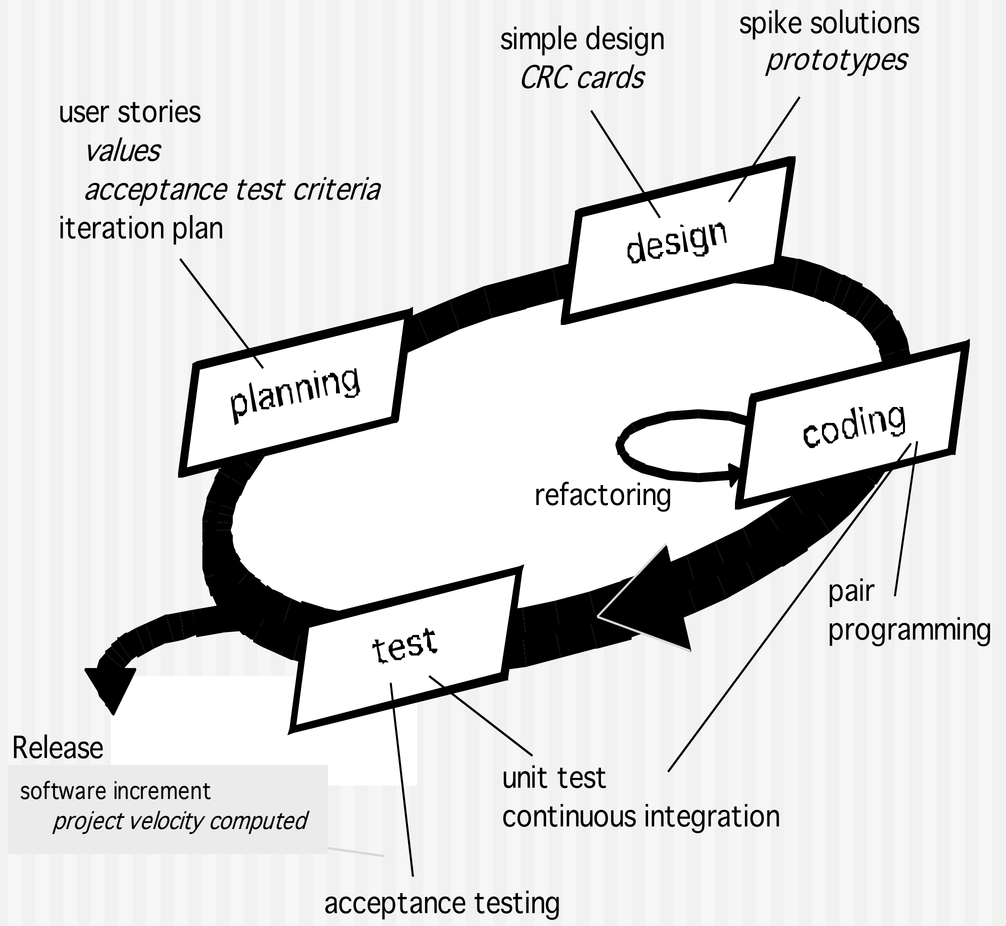
Extreme Programming (XP)

- The most widely used agile process, originally proposed by Kent Beck in 2004. It uses an object-oriented approach.
- XP Planning
  - Begins with the listening, leads to creation of "user stories" that describes required output, features, and functionality. Customer assigns a value(i.e., a priority) to each story.
  - Agile team assesses each story and assigns a cost (development weeks. If more than 3 weeks, customer asked to split into smaller stories)
  - Working together, stories are grouped for a deliverable increment next release.
  - A commitment (stories to be included, delivery date and other project matters) is made. Three ways: 1. Either all stories will be implemented in a few weeks, 2. high priority stories first, or 3. the riskiest stories will be implemented first.
  - After the first increment "project velocity", namely 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 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. (see Chapter 8)
  - 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.
- XP Coding
  - Recommends the construction of a unit test for a story *before* coding commences. So 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.
- XP Testing
  - All unit tests are executed daily and ideally should be automated. Regression tests are conducted to test current and previous components.
  - "Acceptance tests" are defined by the customer and executed to assess customer visible functionality

Extreme Programming (XP)



The XP Debate

- Requirements volatility: customer is an active member of XP team, changes to requirements are requested informally and frequently.
- Conflicting customer needs: different customers' needs need to be assimilated. Different vision or beyond their authority.
- Requirements are expressed informally: Use stories and acceptance tests are the only explicit manifestation of requirements. Formal models may avoid inconsistencies and errors before the system is built. Proponents said changing nature makes such models obsolete as soon as they are developed.
- Lack of formal design: XP deemphasizes the need for architectural design. Complex systems need overall structure to exhibit quality and maintainability. Proponents said 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.
- 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

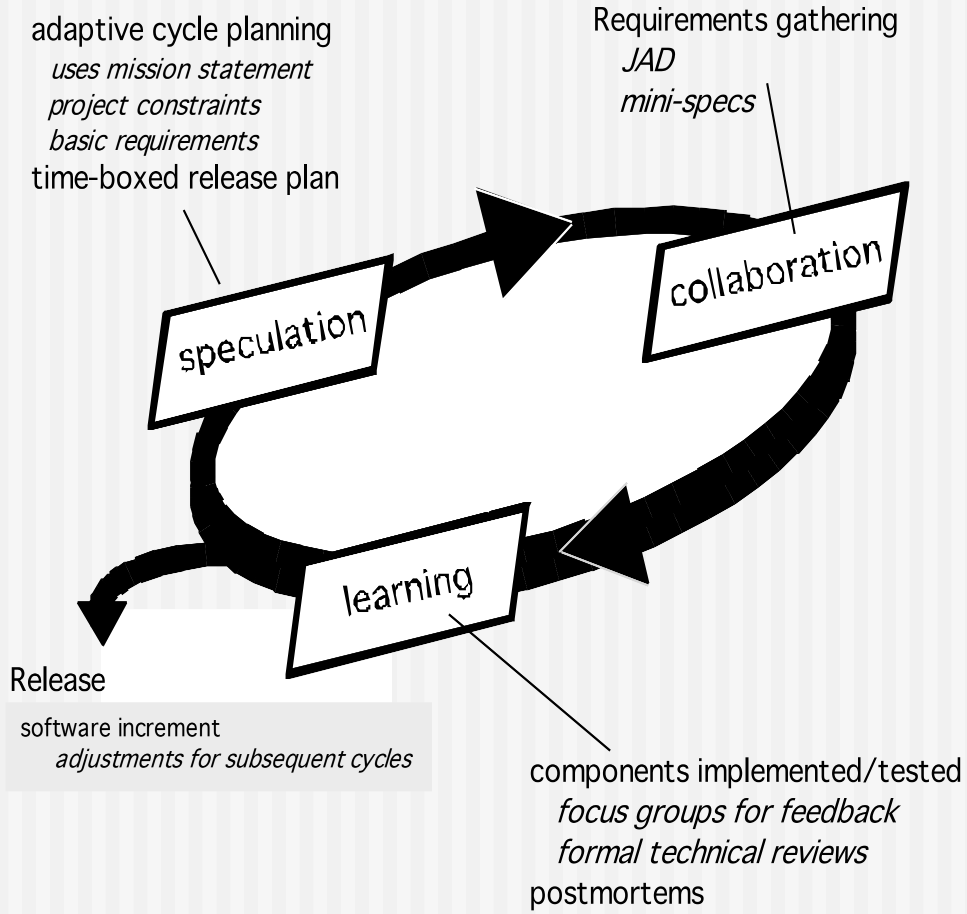
Three Phases of ASD

- 1. Speculation: 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. Collaborations are used to multiply their talent and creative output beyond absolute number (1+1>2). It encompasses communication and teamwork, but it also emphasizes individualism, because individual creativity plays an important role in collaborative thinking.
- It is a matter of trust. 1) criticize without animosity, 2) assist without resentments, 3) work as hard as or harder than they do. 4) have the skill set to contribute to the work at hand, 5) communicate problems or concerns in a way that leas to effective action.
- 3. Learning: As members of ASD team begin to develop the components, the emphasis is 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 to improve their level of real understanding. Three ways: focus groups, technical reviews and project postmortems.

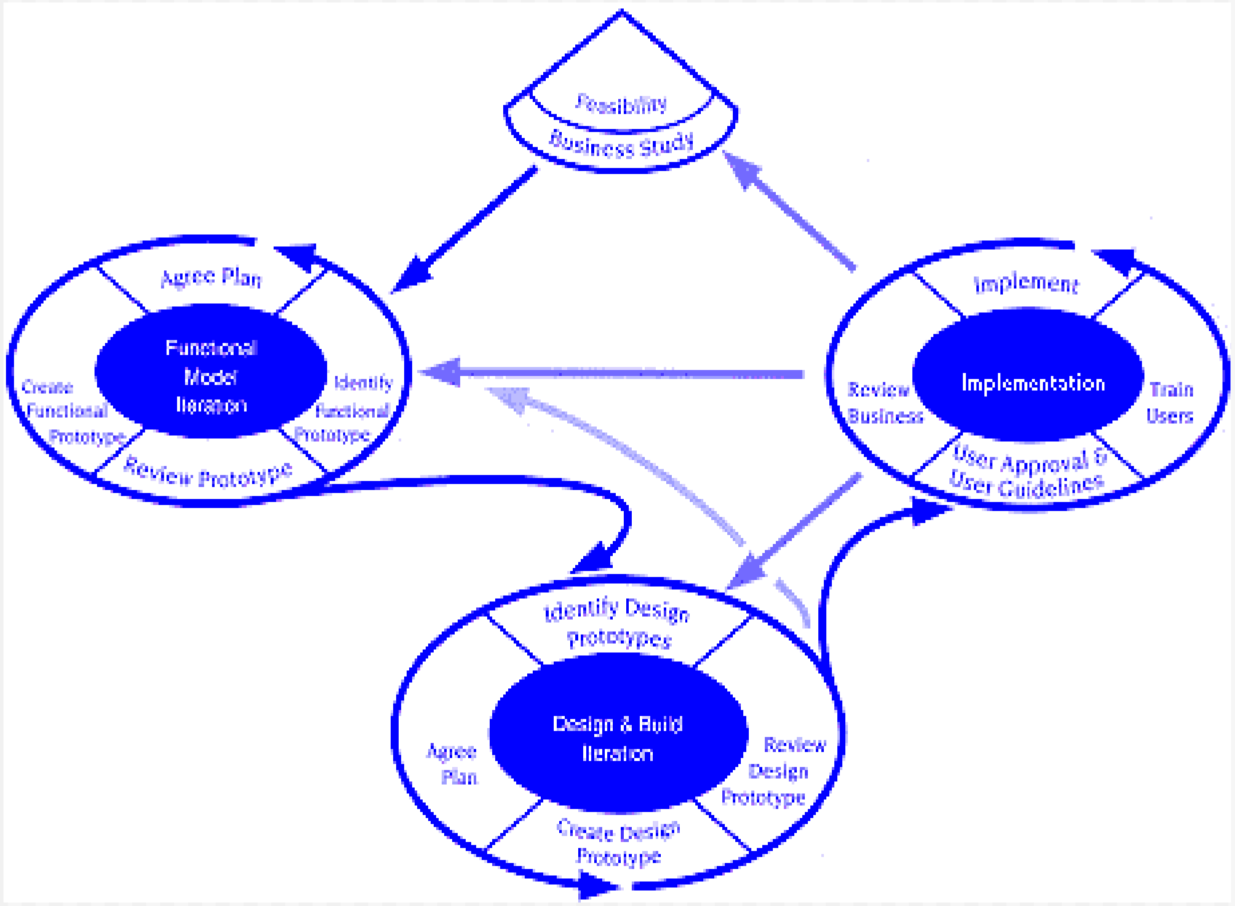
Adaptive Software Development



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)
- 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 (with permission of the DSDM consortium)

Scrum

- A software development method Originally proposed by Schwaber and Beedle (an activity occurs during a rugby match) 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 backlog.
  - 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. May not contain all functionalities. So customers can evaluate and give feedbacks.

Scrum

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

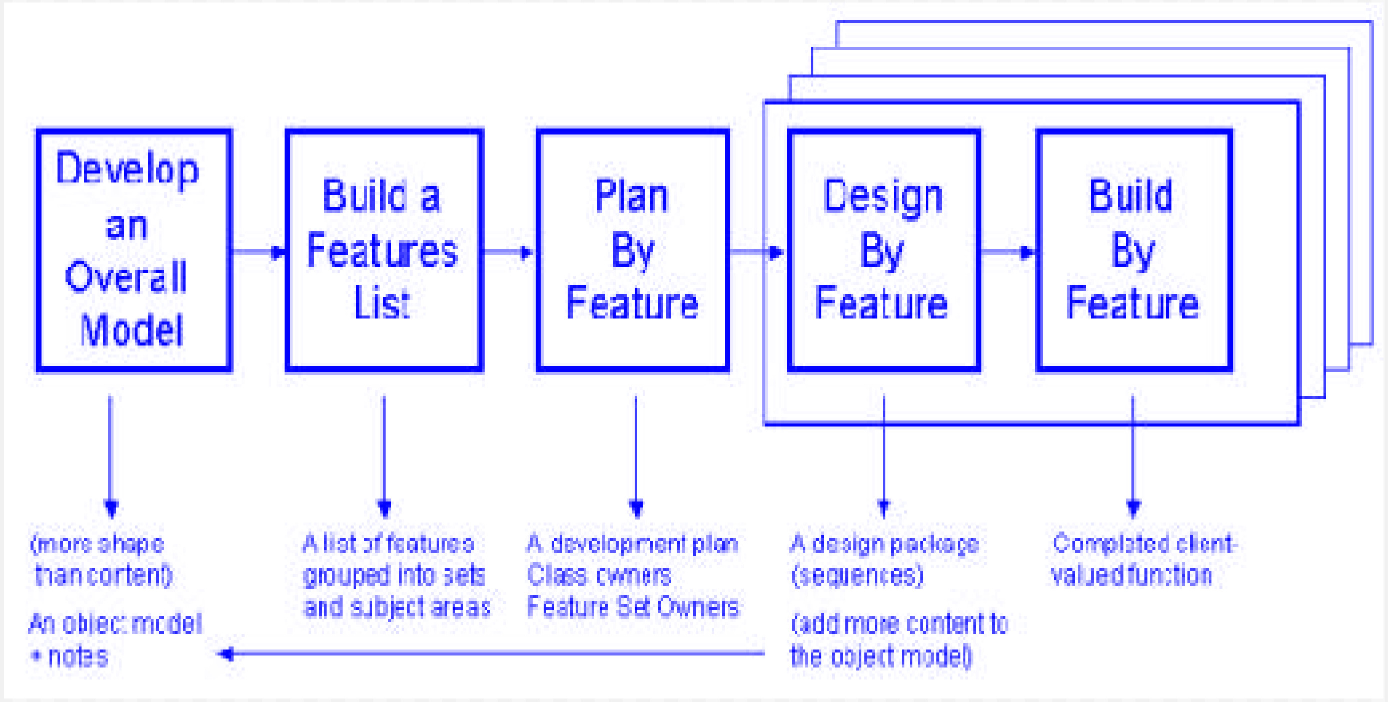
Crystal

- 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


Develop an Overall Model
Build a Features List
Plan By Feature
Design By Feature
Build By Feature
(more shape than content)
An object model + notes
A list of features grouped into sets and subject areas
A development plan
Class owners
Feature Set Owners
A design package (sequences)
(add more content to the object model)
Completed client-valued function

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