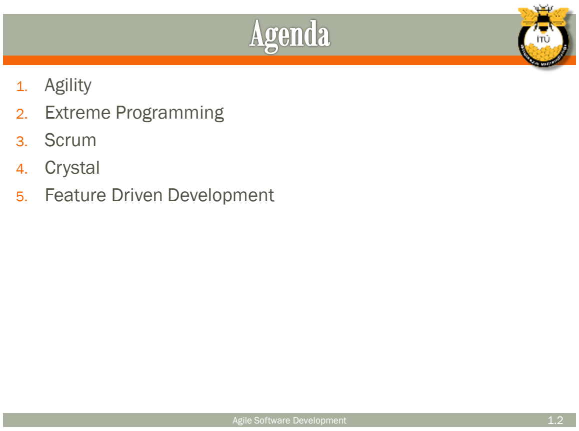
SOFTWARE ENGINEERING

Week 3

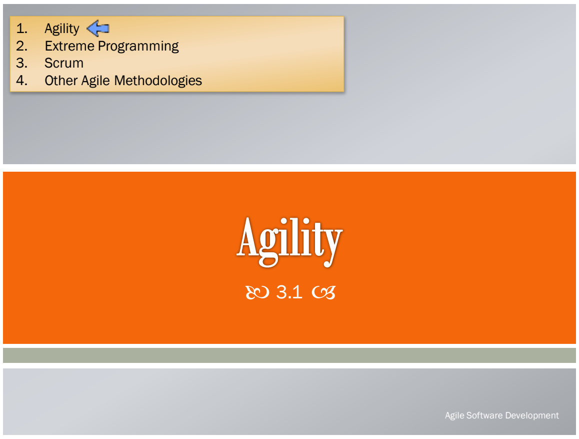
Agile Software Development

Prof. Dr. Muhittin GÖKMEN   Yard. Doç. Dr. A. Cüneyd TANTUĞ   Araş. Gör. Dr. Tolga OVATMAN
Istanbul Technical University
Computer Engineering Department



Agenda

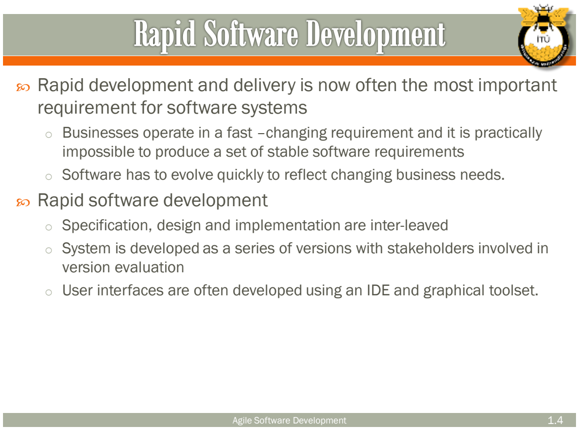
1. Agility
2. Extreme Programming
3. Scrum
4. Crystal
5. Feature Driven Development



1. Agility ⇦
2. Extreme Programming
3. Scrum
4. Other Agile Methodologies

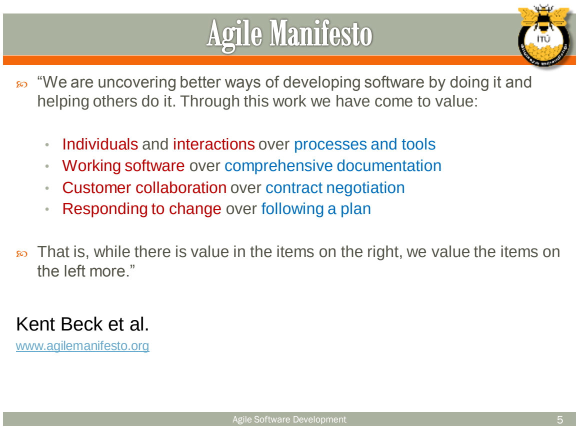
Agility

3.1



Rapid Software Development

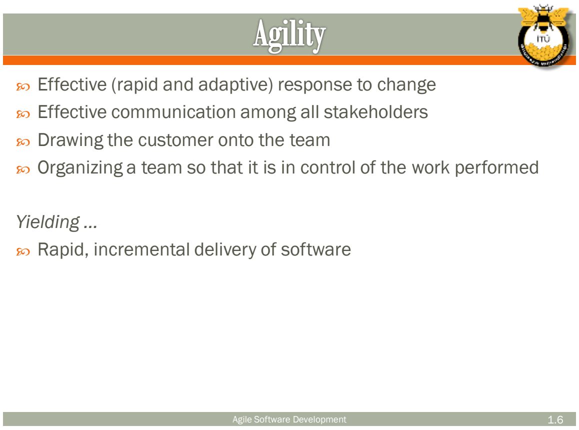
- Rapid development and delivery is now often the most important requirement for software systems
  - Businesses operate in a fast –changing requirement and it is practically impossible to produce a set of stable software requirements
  - Software has to evolve quickly to reflect changing business needs.
- Rapid software development
  - Specification, design and implementation are inter-leaved
  - System is developed as a series of versions with stakeholders involved in version evaluation
  - User interfaces are often developed using an IDE and graphical toolset.



Agile Manifesto

- “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.
www.agilemanifesto.org



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

Twelve Principles of Agile Software

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.
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.

Principles of Agile Methods

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

Problems with agile methods

- It can be difficult to keep the interest of customers who are involved in the process.
- Team members may be unsuited to the intense involvement that characterizes agile methods.
- Prioritizing changes can be difficult where there are multiple stakeholders.
- Maintaining simplicity requires extra work.
- Contracts may be a problem as with other approaches to iterative development.

Agile Methods Applicability

- Product development where a software company is developing a small or medium-sized product for sale.
- Custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are not a lot of external rules and regulations that affect the software.
- Because of their focus on small, tightly-integrated teams, there are problems in scaling agile methods to large systems.

Agile vs Plan-Driven Methods

Plan-driven development

- A plan-driven approach to software engineering is based around separate development stages with the outputs to be produced at each of these stages planned in advance.
- Not necessarily waterfall model – plan-driven, incremental development is possible
- Iteration occurs within activities.

Agile development

- Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process

Technical, human, organizational issues-I

- Most projects include elements of plan-driven and agile processes. Deciding on the balance depends on:
  - Is it important to have a very detailed specification and design before moving to implementation? If so, you probably need to use a plan-driven approach.
  - Is an incremental delivery strategy, where you deliver the software to customers and get rapid feedback from them, realistic? If so, consider using agile methods.
  - How large is the system that is being developed? Agile methods are most effective when the system can be developed with a small co-located team who can communicate informally. This may not be possible for large systems that require larger development teams so a plan-driven approach may have to be used.

Technical, human, organizational issues-II

- What type of system is being developed?
  - Plan-driven approaches may be required for systems that require a lot of analysis before implementation (e.g. real-time system with complex timing requirements).
- What is the expected system lifetime?
  - Long-lifetime systems may require more design documentation to communicate the original intentions of the system developers to the support team.
- What technologies are available to support system development?
  - Agile methods rely on good tools to keep track of an evolving design
- How is the development team organized?
  - If the development team is distributed or if part of the development is being outsourced, then you may need to develop design documents to communicate across the development teams.
- Are there cultural or organizational issues that may affect the system development?
  - Traditional engineering organizations have a culture of plan-based development, as this is the norm in engineering.
- How good are the designers and programmers in the development team?
  - It is sometimes argued that agile methods require higher skill levels than plan-based approaches in which programmers simply translate a detailed design into code
- Is the system subject to external regulation?
  - If a system has to be approved by an external regulator (e.g. the FAA approve software that is critical to the operation of an aircraft) then you will probably be required to produce detailed documentation as part of the system safety case.

1. Agility
2. Extreme Programming ⇦
3. Scrum
4. Other Agile Methodologies

Extreme Programming

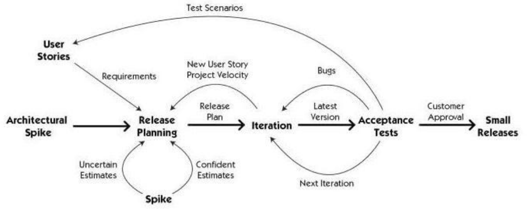
3.2

Extreme Programming

- The most widely used agile process, originally proposed by Kent Beck
- Extreme Programming (XP) takes an ‘extreme’ approach to iterative development.
  - New versions may be built several times per day;
  - Increments are delivered to customers every 2 weeks;
  - All tests must be run for every build and the build is only accepted if tests run successfully.

XP Principles

- Incremental development is supported through small, frequent system releases.
- Customer involvement means full-time customer engagement with the team.
- People not process through pair programming, collective ownership and a process that avoids long working hours.
- Change supported through regular system releases.
- Maintaining simplicity through constant refactoring of code.



A visual process model for XP

XP Phases

- 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
- XP Design
  - Follows the KIS principle
  - Encourage the use of CRC cards
  - 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

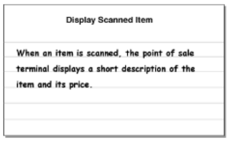
User Stories

- In XP, a customer or user is part of the XP team and is responsible for making decisions on requirements.
- User requirements are expressed as scenarios or user stories.
- These are written on cards and the development team break them down into implementation tasks. These tasks are the basis of schedule and cost estimates.
- The customer chooses the stories for inclusion in the next release based on their priorities and the schedule estimates.

EXTREME PROGRAMMING
I CAN'T GIVE YOU ALL OF THESE FEATURES IN THE FIRST VERSION.
AND EACH FEATURE NEEDS TO HAVE WHAT WE CALL A "USER STORY."
OKAY, HERE'S A STORY: YOU GIVE ME ALL OF MY FEATURES OR I'LL RUIN YOUR LIFE.
www.dilbert.com
scottadams@aol.com
© 2002 United Feature Syndicate, Inc.

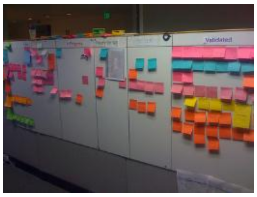
Copyright © 2003 United Feature Syndicate, Inc.

Story Boards



Display Scanned Item

When an item is scanned, the point of sale terminal displays a short description of the item and its price.



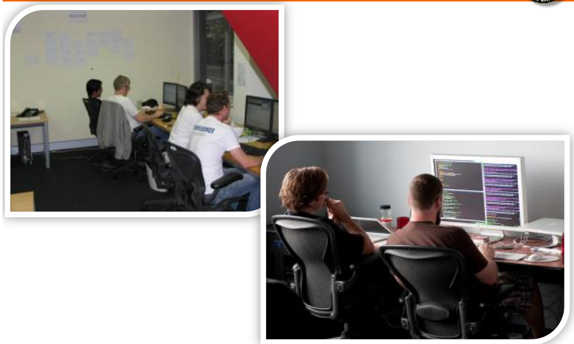
Refactoring

- Programming team look for possible software improvements and make these improvements even where there is no immediate need for them.
- This improves the understandability of the software and so reduces the need for documentation.
- Changes are easier to make because the code is well-structured and clear.
- However, some changes requires architecture refactoring and this is much more expensive.
- Examples of refactoring:
  - Re-organization of a class hierarchy to remove duplicate code.
  - Tidying up and renaming attributes and methods to make them easier to understand.
  - The replacement of inline code with calls to methods that have been included in a program library.

Pair Programming

- In XP, programmers work in pairs, sitting together to develop code.
- This helps develop common ownership of code and spreads knowledge across the team.
- It serves as an informal review process as each line of code is looked at by more than 1 person.
- It encourages refactoring as the whole team can benefit from this.
- Measurements suggest that development productivity with pair programming is similar to that of two people working independently.
- In pair programming, programmers sit together at the same workstation to develop the software.
- Pairs are created dynamically so that all team members work with each other during the development process.
- The sharing of knowledge that happens during pair programming is very important as it reduces the overall risks to a project when team members leave.
- Pair programming is not necessarily inefficient and there is evidence that a pair working together is more efficient than 2 programmers working separately.

Pair Programmin Environment



Advantages of Pair Programming

- It supports the idea of collective ownership and responsibility for the system.
  - Individuals are not held responsible for problems with the code. Instead, the team has collective responsibility for resolving these problems.
- It acts as an informal review process because each line of code is looked at by at least two people.
- It helps support refactoring, which is a process of software improvement.
  - Where pair programming and collective ownership are used, others benefit immediately from the refactoring so they are likely to support the process.

Testing in XP

- Testing is central to XP and XP has developed an approach where the program is tested after every change has been made.
- XP testing features:
  - Test-first development.
    Writing tests before code clarifies the requirements to be implemented.
  - Incremental test development from scenarios.
  - User involvement in test development and validation.
  - Automated test harnesses are used to run all component tests each time that a new release is built.
  - Tests are written as programs rather than data so that they can be executed automatically. The test includes a check that it has executed correctly.
  - Usually relies on a testing framework such as JUnit. An automated test framework (e.g. JUnit) is a system that makes it easy to write executable tests and submit a set of tests for execution.
  - All previous and new tests are run automatically when new functionality is added, thus checking that the new functionality has not introduced errors.

1. Agility
2. Extreme Programming
3. Scrum ⇦
4. Other Agile Methodologies

Scrum

3.3

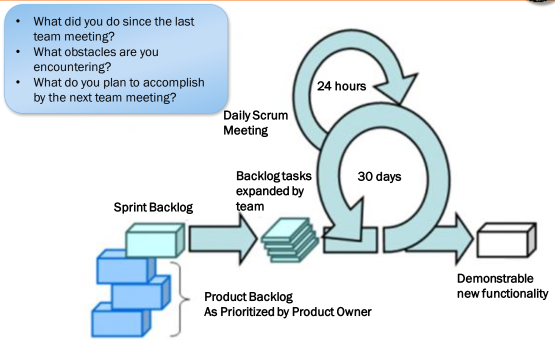
Scrum

- Scrum is a term borrowed from rugby.
  - the whole team "tries to go the distance as a unit, passing the ball back and forth
- In scrum based software development, the phases strongly overlap and the whole process is performed by one cross-functional team across the different phases.
- 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

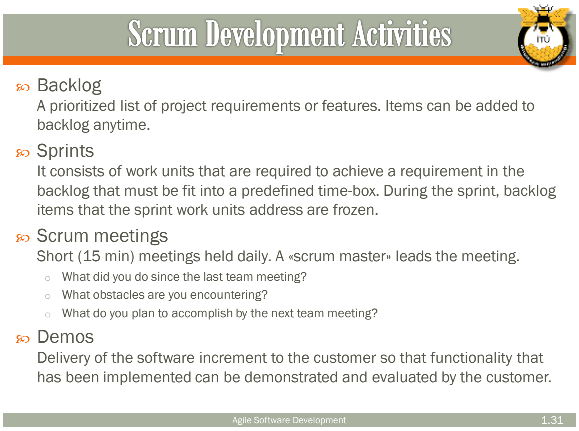
Scrum Lifecycle



- What did you do since the last team meeting?
- What obstacles are you encountering?
- What do you plan to accomplish by the next team meeting?

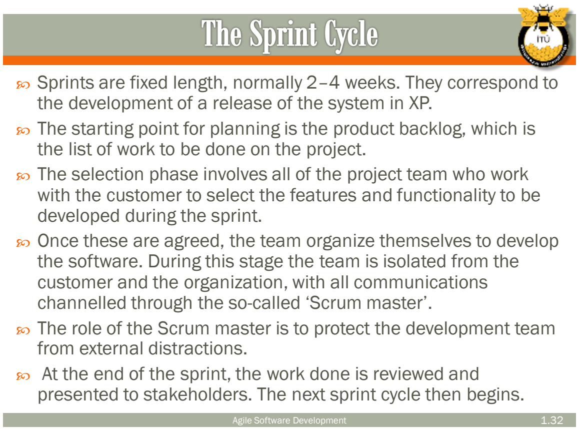
24 hours
Daily Scrum Meeting
30 days
Backlog tasks expanded by team
Sprint Backlog
Product Backlog
As Prioritized by Product Owner
Demonstrable new functionality

An Example Scrum Board



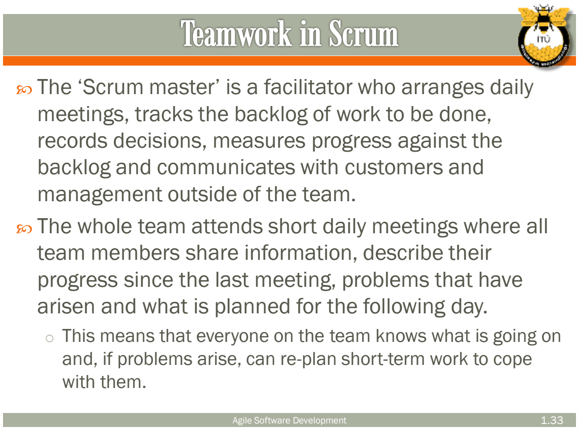
Scrum Development Activities

- Backlog
  A prioritized list of project requirements or features. Items can be added to backlog anytime.
- Sprints
  It consists of work units that are required to achieve a requirement in the backlog that must be fit into a predefined time-box. During the sprint, backlog items that the sprint work units address are frozen.
- Scrum meetings
  Short (15 min) meetings held daily. A «scrum master» leads the meeting.
  - What did you do since the last team meeting?
  - What obstacles are you encountering?
  - What do you plan to accomplish by the next team meeting?
- Demos
  Delivery of the software increment to the customer so that functionality that has been implemented can be demonstrated and evaluated by the customer.



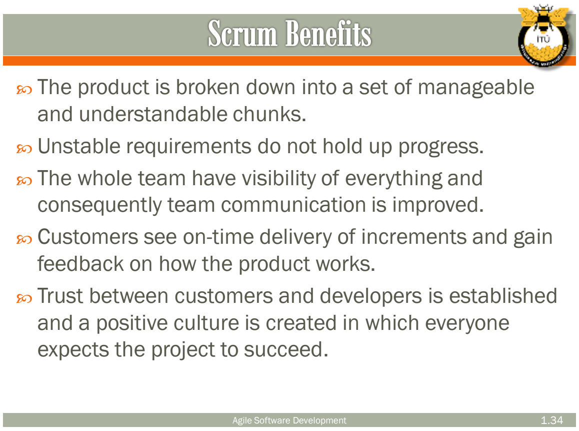
The Sprint Cycle

- Sprints are fixed length, normally 2–4 weeks. They correspond to the development of a release of the system in XP.
- The starting point for planning is the product backlog, which is the list of work to be done on the project.
- The selection phase involves all of the project team who work with the customer to select the features and functionality to be developed during the sprint.
- Once these are agreed, the team organize themselves to develop the software. During this stage the team is isolated from the customer and the organization, with all communications channelled through the so-called 'Scrum master'.
- The role of the Scrum master is to protect the development team from external distractions.
- At the end of the sprint, the work done is reviewed and presented to stakeholders. The next sprint cycle then begins.



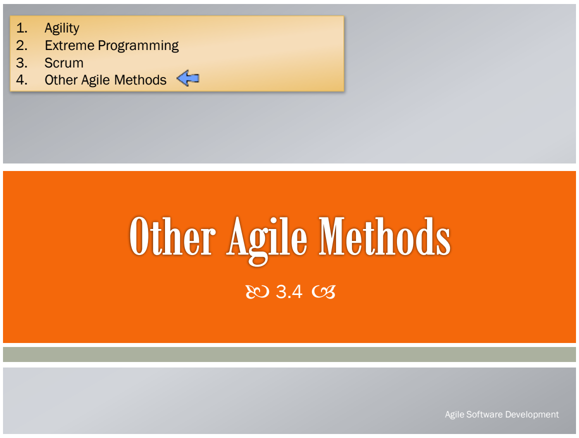
Teamwork in Scrum

- The 'Scrum master' is a facilitator who arranges daily meetings, tracks the backlog of work to be done, records decisions, measures progress against the backlog and communicates with customers and management outside of the team.
- The whole team attends short daily meetings where all team members share information, describe their progress since the last meeting, problems that have arisen and what is planned for the following day.
  - This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them.



Scrum Benefits

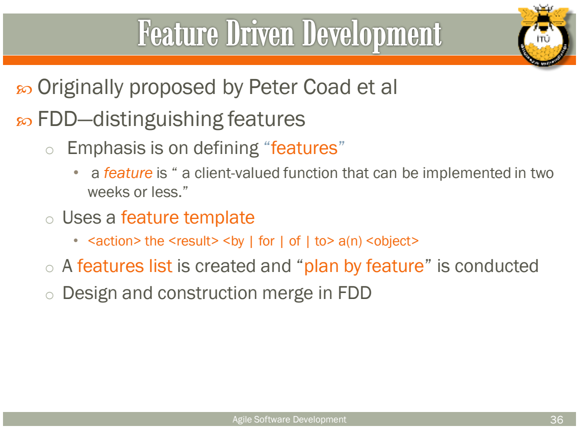
- The product is broken down into a set of manageable and understandable chunks.
- Unstable requirements do not hold up progress.
- The whole team have visibility of everything and consequently team communication is improved.
- Customers see on-time delivery of increments and gain feedback on how the product works.
- Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.



1. Agility
2. Extreme Programming
3. Scrum
4. Other Agile Methods

Other Agile Methods

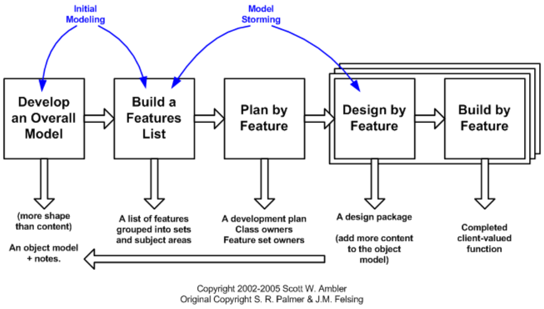
3.4



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

Feature Driven Development Phases



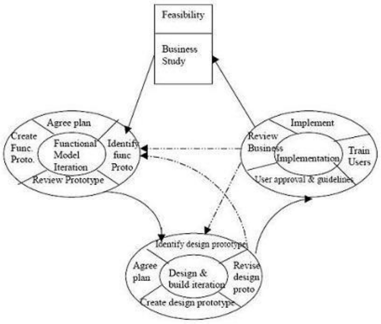
Copyright 2002-2005 Scott W. Ambler
Original Copyright S. R. Palmer & J.M. Felsing

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

Dynamic Systems Development Method

- Promoted by the DSDM Consortium (www.dsdm.org)
- DSDM—distinguishing features
  - Similar in most respects to XP
  - 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.


Feasibility
Business Study
Agree plan
Create Func. Proto.
Functional Model Iteration
Identify func Proto
Review Prototype
Implement
Review Business
Implementation
Train Users
User approval & guidelines
Identify design prototype
Agree plan
Design & build iteration
Revise design proto
Create design prototype