Agile Development

Another View on Software Process

- Because software, like all capital, is embodied knowledge, and because that knowledge is initially dispersed, tacit, latent, and incomplete in large measure, software development is a social learning process. The process is a dialogue in which the knowledge that must become software is brought together and embodied in the software.

-Howard Baetjer (economist)

Rapid software development

(as per Sommerville)

- 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 methods

(as per Sommerville)

- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
  - Focus on the code rather than the design
  - Are based on an iterative approach to software development
  - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.

- The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.

Sommerville, I., Software Engineering, Pearson Education, 9th Ed., 2010

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

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

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

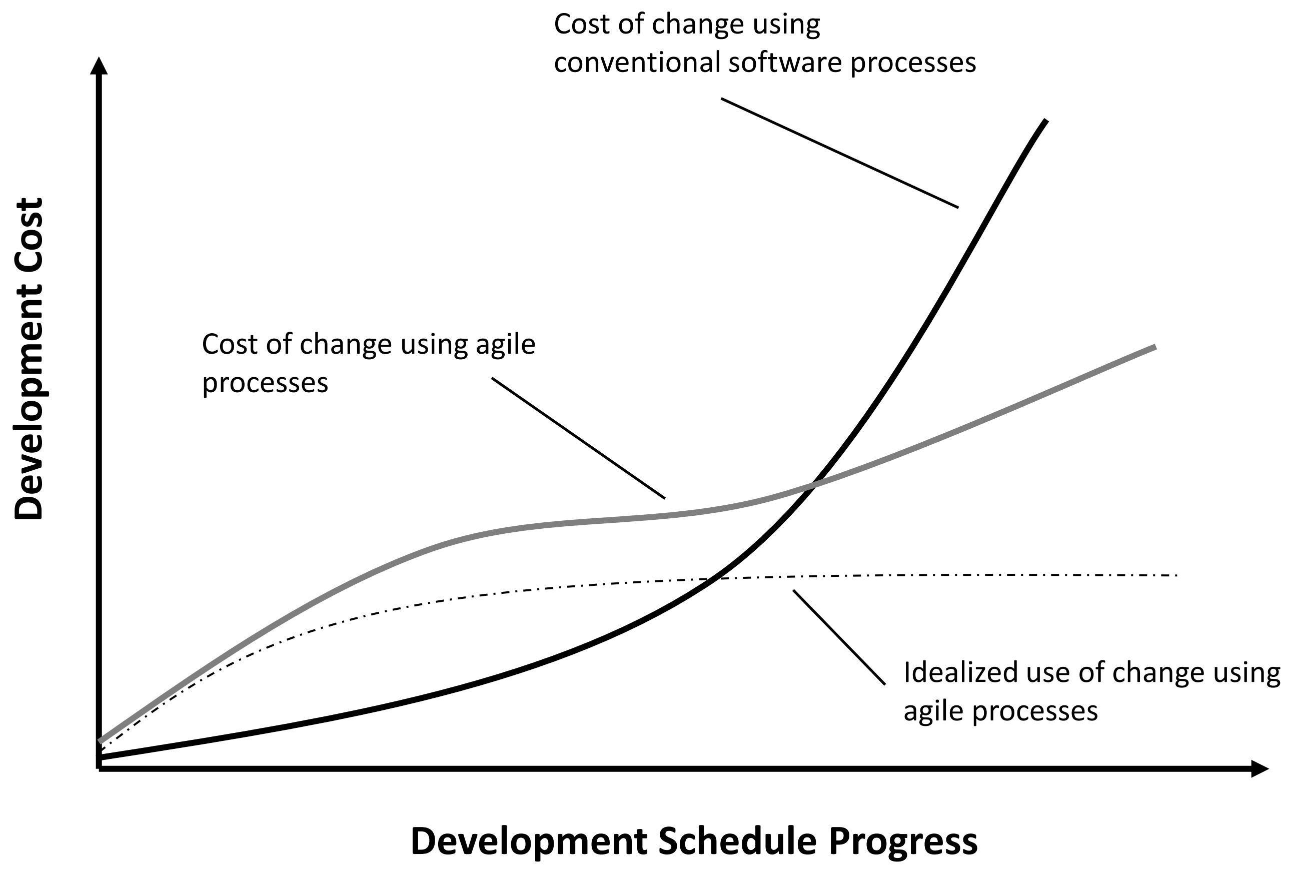
Agility Principles (contd)

- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity – the art of maximizing the amount of work not done – is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agility Human Factors

- The process molds to the needs of the people and team, not the other way around.
- Some key traits must exist among the people on an agile team
  - Competence
  - Common focus
  - Collaboration
  - Decision-making ability
  - Fuzzy problem-solving ability
  - Mutual trust and respect
  - Self-organization

Cost of Change

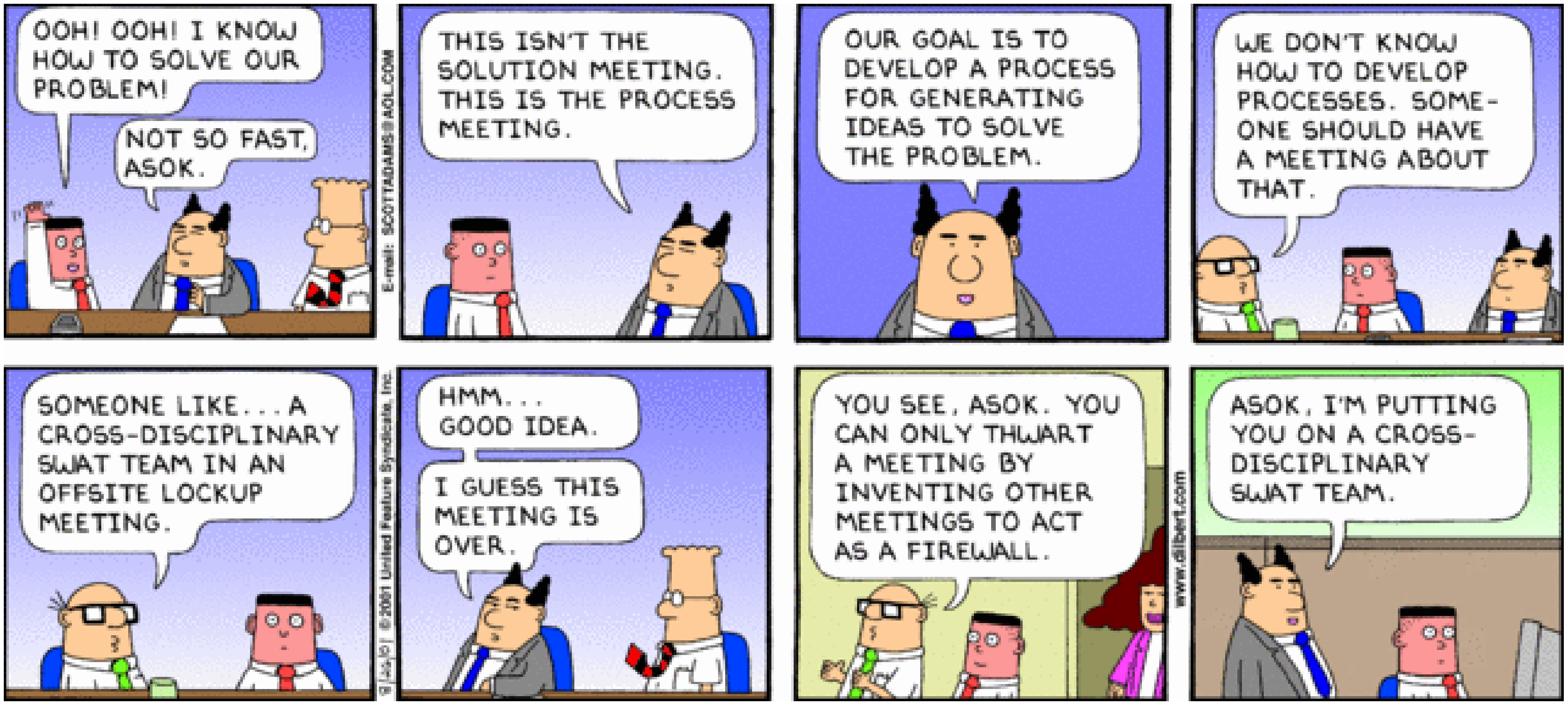


An Agile Concept - Time Boxing

... teams are managing the triple constraints that face any organisation - time, quality, scope. When using a fixed duration, we are telling everyone involved, 'time is urgent and we are going to include as much as we can within this time framework.' Since quality cannot be compromised, the only variable is scope. 'Time boxing' creates a sense of urgency and criticality for the entire organization

—Mark P. Dangelo, Author: *Innovative relevance*

Scott Adams “supports” Agile


OOH! OOH! I KNOW HOW TO SOLVE OUR PROBLEM!
NOT SO FAST, ASOK.
E-mail: SCOTTADAMS@AOL.COM
THIS ISN'T THE SOLUTION MEETING. THIS IS THE PROCESS MEETING.
OUR GOAL IS TO DEVELOP A PROCESS FOR GENERATING IDEAS TO SOLVE THE PROBLEM.
WE DON'T KNOW HOW TO DEVELOP PROCESSES. SOMEONE SHOULD HAVE A MEETING ABOUT THAT.
SOMEONE LIKE... A CROSS-DISCIPLINARY SWAT TEAM IN AN OFFSITE LOCKUP MEETING.
© 2001 United Feature Syndicate, Inc.
HMM... GOOD IDEA.
I GUESS THIS MEETING IS OVER.
YOU SEE, ASOK. YOU CAN ONLY THWART A MEETING BY INVENTING OTHER MEETINGS TO ACT AS A FIREWALL.
www.dilbert.com
ASOK, I'M PUTTING YOU ON A CROSS-DISCIPLINARY SWAT TEAM.

Plan-driven versus Agile Development

(as per Sommerville)

- 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.

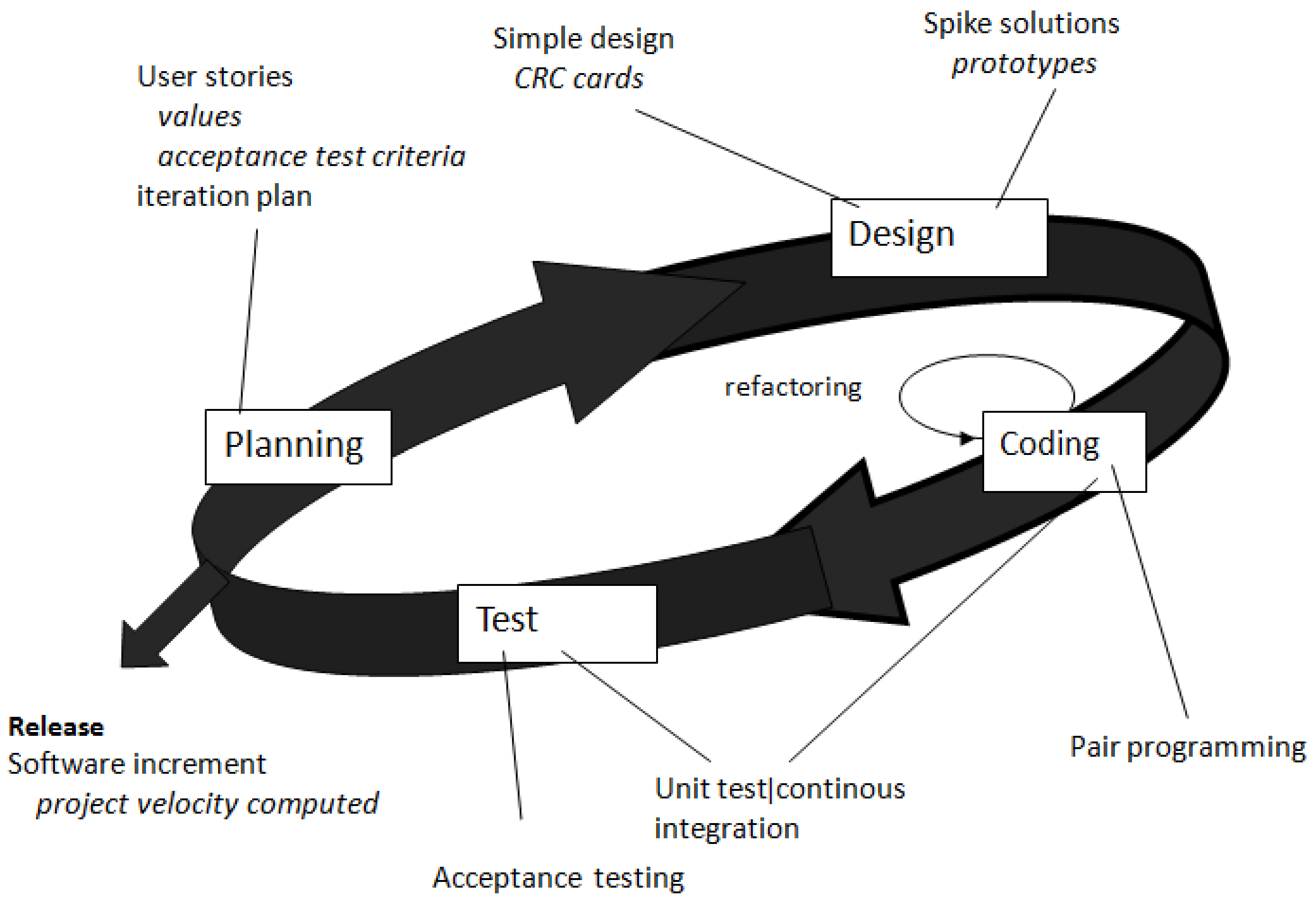
Extreme Programming (XP)

- The most widely used agile process, originally proposed by Kent Beck
- 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

Extreme Programming (XP)

- 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

Extreme Programming (XP)



Refactoring (as per Sommerville)

- 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.

Sommerville, I., Software Engineering, Pearson Education, 9th Ed., 2010

Examples of refactoring

(as per Sommerville)

- 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.

Sommerville, I., Software Engineering, Pearson Education, 9th Ed., 2010

Pair programming (as per Sommerville)

- 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.
- It serves as an informal review process as each line of code is looked at by more than 1 person.
- Pair programming is not necessarily inefficient and there is evidence that a pair working together is more efficient than 2 programmers working separately.

Sommerville, I., Software Engineering, Pearson Education, 9th Ed., 2010

Advantages of pair programming

(as per Sommerville)

- 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.

Industrial XP

- Incorporates six new practices to ensure that XP works for significant projects in large organization.
- Incorporates six new practices
  - Readiness Assessment
    - Ascertain environment, team, culture
  - Project Community
    - Right people – team becomes community
  - Project Chartering
    - Appropriate business justification within org
  - Test-driven management
    - State of the project as per measurable destinations
  - Retrospectives
    - Specialized technical review after delivery of increment
  - Continuous learning

Criticism of XP

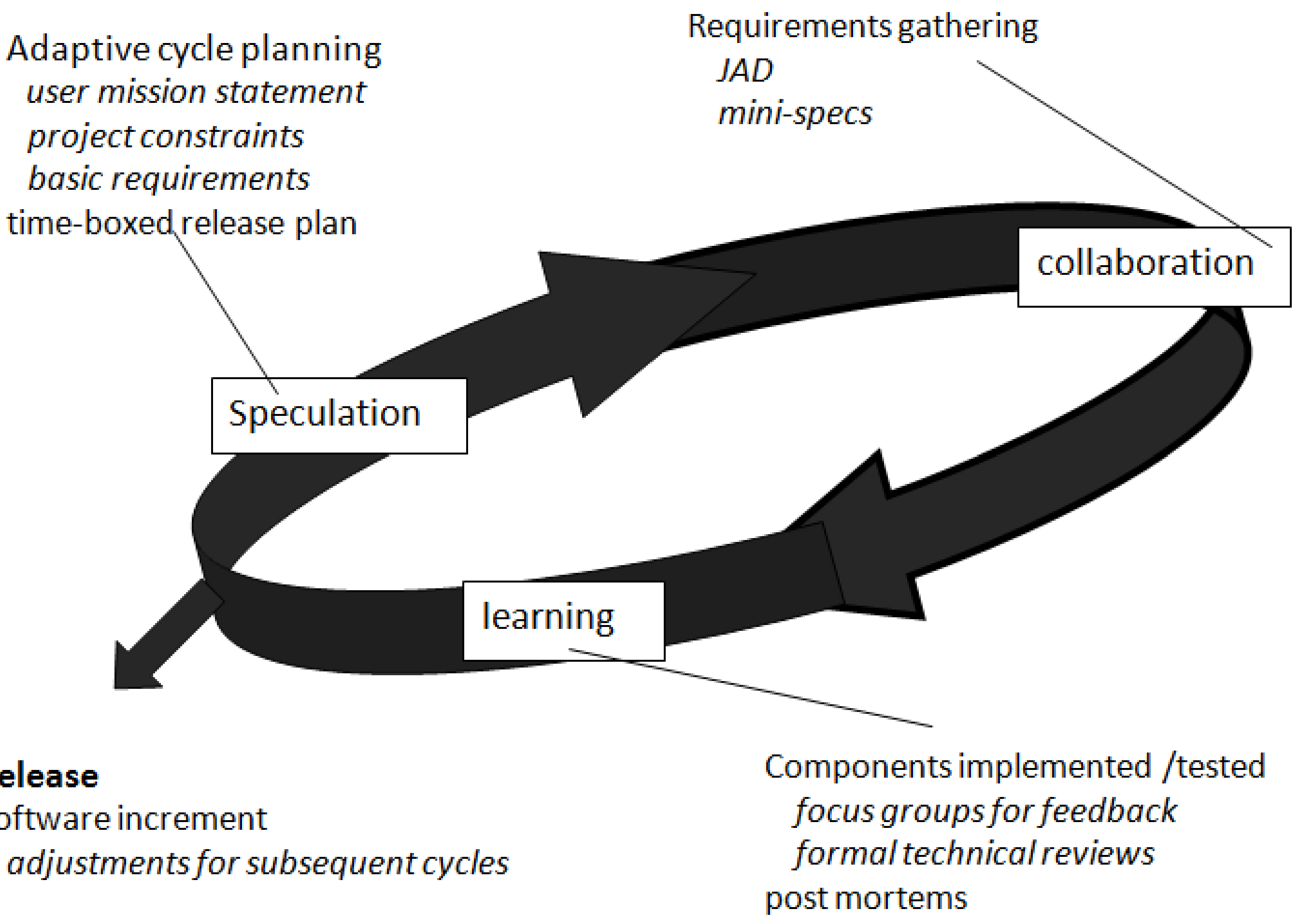
Concerns expressed by critics w.r.t. XP include

- Requirements volatility
  - -rework may be unmanageable
- Conflicting customer needs
  - -team cannot reconcile conflicting demands
- Requirements are expressed informally
  - -omissions, inconsistencies, errors
- Lack of formal design
  - -Without architectural design, structure of the software lacks quality and maintainability

Adaptive Software Development

- Originally proposed by Jim Highsmith
- ASD — distinguishing features
  - *Mission-driven* planning
  - *Component-based focus*
  - Uses "*time-boxing*"
  - Explicit consideration of *risks*
  - Emphasizes *collaboration* for requirements gathering
  - Emphasizes "*learning*" throughout the process

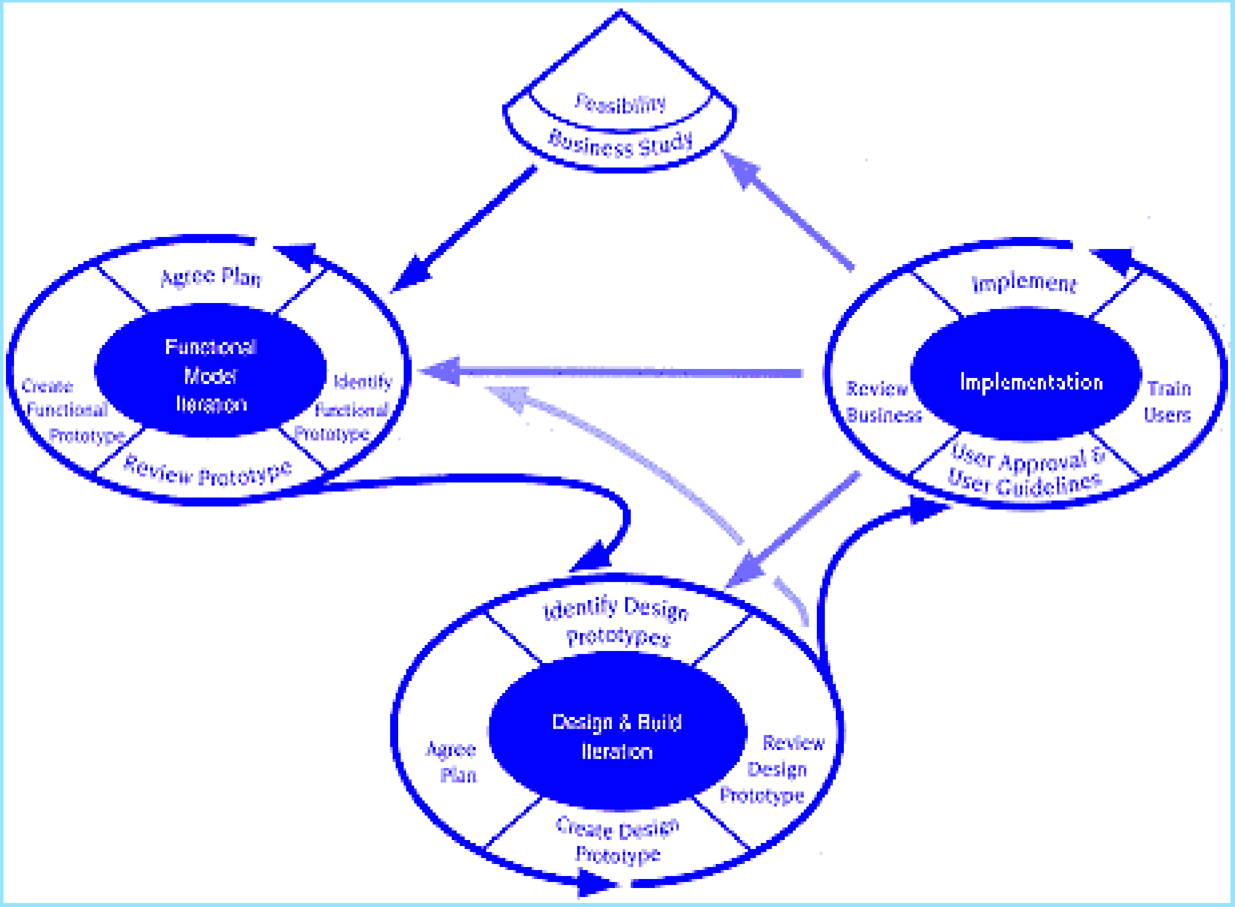
Adaptive Software Development



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 (with permission of the DSDM consortium)

Introduction to SCRUM

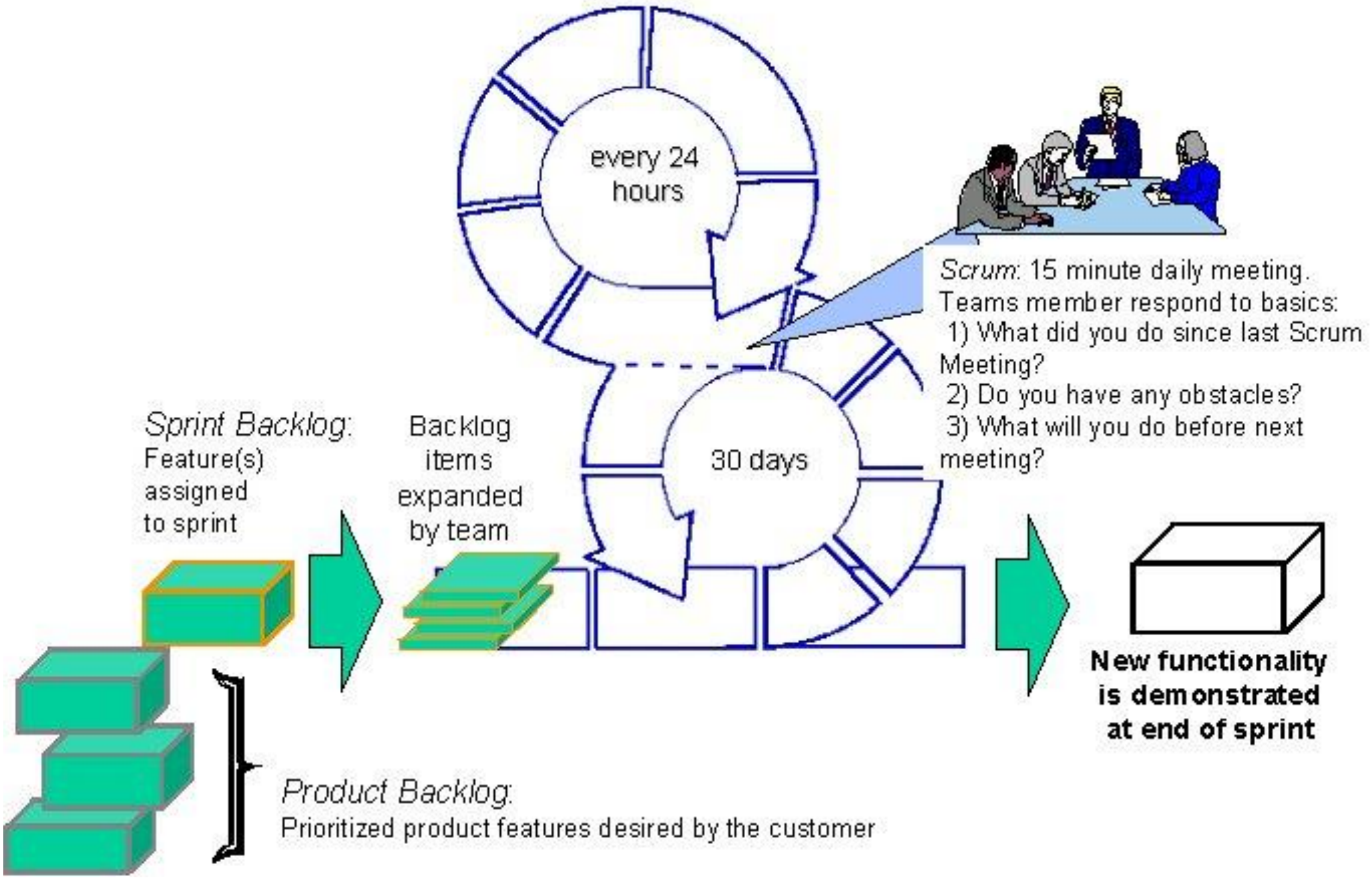
- Scrum is an Agile Software Development Process.
- Scrum is not an acronym
- name taken from the sport of Rugby, where everyone in the team pack acts together to move the ball down the field
- analogy to development is the team works together to successfully develop quality software

Introduction to SCRUM (cont.)

- Jeff Sutherland (proponent) states:
  - “Scrum assumes that the systems development process is an unpredictable, complicated process that can only be roughly described as an overall progression.”
  - “Scrum is an enhancement of the commonly used iterative/incremental object-oriented development cycle”

Scrum

- Proposed by Schwaber and Beedle
- Scrum—distinguishing features
  - Development work partitioned into packets makes up "Backlog"
  - 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


every 24 hours
Scrum: 15 minute daily meeting.
Teams member respond to basics:
1) What did you do since last Scrum Meeting?
2) Do you have any obstacles?
3) What will you do before next meeting?
Sprint Backlog:
Feature(s) assigned to sprint
Backlog items expanded by team
30 days
New functionality is demonstrated at end of sprint
Product Backlog:
Prioritized product features desired by the customer

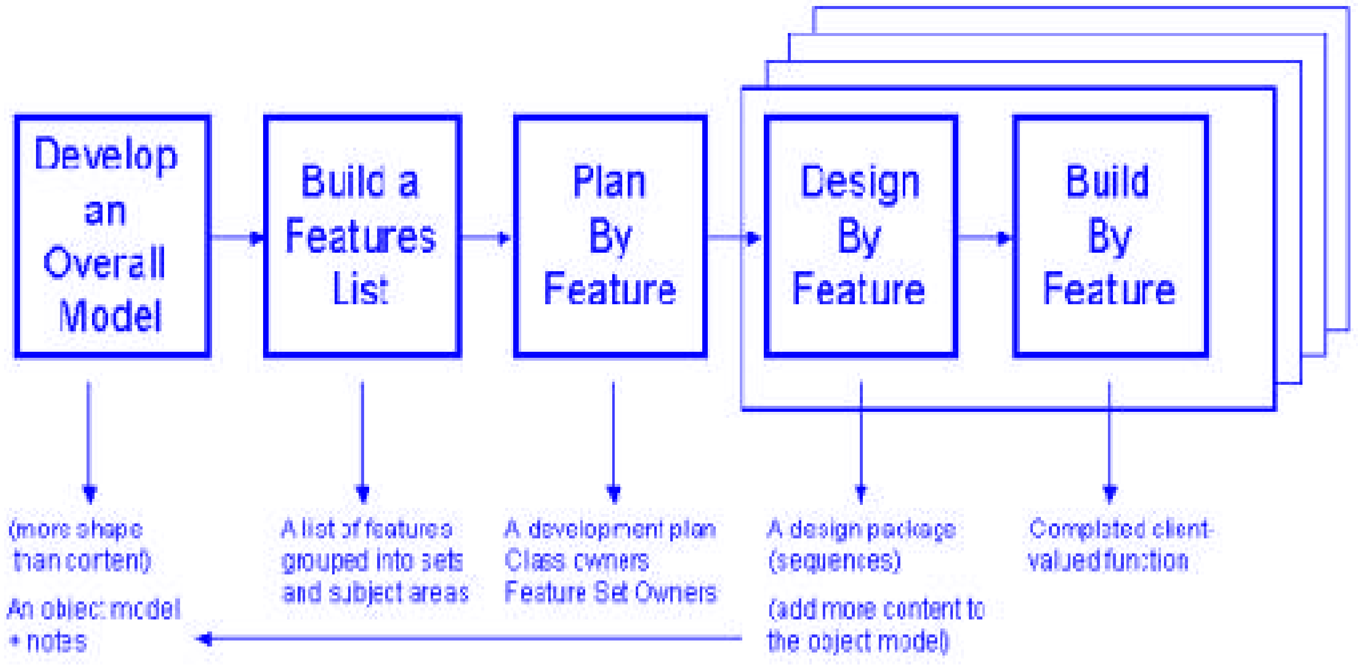
Crystal

- 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> for a feature e.g. Add the product to shopping cart.
    - <action><-ing> a(n) <object> for feature set group e.g. Making a product sale
  - 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

Lean software development

Adapts the principles of lean manufacturing to software engineering

Lean Principles	Software engineering actions
•Eliminate Waste	•Add no extraneous features
•Build Quality in	•Ensure effective testing
•Create Knowledge	•Improve mechanisms to find information
•Defer Commitment	•Assess cost & schedule impact for changes
•Deliver Fast	•Remove superfluous process steps
•Respect People	•Streamline stakeholder communication
•Optimize the Whole	•Reduce time required to process a request

Agile Unified Process

- Adopts “serial in the large” and “iterative in the small” philosophy suggested by Scott Ambler
- Adopts UP phased activities – inception, elaboration, construction, and transition
- Within each of the activity, team iterates to achieve agility.
- UML representations are used, though modeling is kept to bare minimum

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

Agile Process and Documentation

There are two keys to successful documentation on agile projects.

- Finding the point of "just enough" documentation. This is difficult to determine and will vary by project. Fortunately, the iterative nature of agile development allows you to experiment until you get it right.
- Not get attached to it or have unrealistic hopes of keeping it updated.

Documentation must be created to serve a specific purpose, and after it has served that purpose you'll all probably have more important things to do than keep updating the documents. It may seem counterintuitive, but it's often better to produce fresh documentation the next time some is clearly required. A side benefit of starting over each time you need to document part of your project is that it's great incentive to keep your documentation efficient!

Agile Process with Offshore Development

Martin Fowler suggests following successful practices:

- Use Continuous Integration to Avoid Integration Headaches
- Have Each Site Send Ambassadors to the Other Sites
- Use Contact Visits to build trust
- Don't Underestimate the Culture Change
- Use wikis to contain common information
- Use Test Scripts to Help Understand the Requirements
- Use Regular Builds to Get Feedback on Functionality

Article by Martin Fowler, Author and Member, Agile Alliance

Agile Process with Offshore Development
(contd)

Martin Fowler suggests following successful practices:

- Use Regular Short Status Meetings
- Use Short Iterations
- Use an Iteration Planning Meeting that's Tailored for Remote Sites
- When Moving a Code Base, Bug Fixing Makes a Good Start
- Separate teams by functionality not activity
- Expect to need more documents.
- Get multiple communication modes working early

Article by Martin Fowler, Author and Member, Agile Alliance

Some challenges with agile methods

as per Sommerville

- 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 characterises agile methods.
- Prioritising 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.

Sommerville, I., Software Engineering, Pearson Education, 9th Ed., 2010

Agile methods and software maintenance

as per Sommerville

- Most organizations spend more on maintaining existing software than they do on new software development. So, if agile methods are to be successful, they have to support maintenance as well as original development.
- Two key issues:
  - Are systems that are developed using an agile approach maintainable, given the emphasis in the development process of minimizing formal documentation?
  - Can agile methods be used effectively for evolving a system in response to customer change requests?
- Problems may arise if original development team cannot be maintained.

Large systems development

(as per Sommerville)

- Large systems are usually collections of separate, communicating systems, where separate teams develop each system. Frequently, these teams are working in different places, sometimes in different time zones.
- Large systems are ‘brownfield systems’, that is they include and interact with a number of existing systems. Many of the system requirements are concerned with this interaction and so don’t really lend themselves to flexibility and incremental development.
- Where several systems are integrated to create a system, a significant fraction of the development is concerned with system configuration rather than original code development.

Large system development

(as per Sommerville)

- Large systems and their development processes are often constrained by external rules and regulations limiting the way that they can be developed.
- Large systems have a long procurement and development time. It is difficult to maintain coherent teams who know about the system over that period as, inevitably, people move on to other jobs and projects.
- Large systems usually have a diverse set of stakeholders.It is practically impossible to involve all of these different stakeholders in the development process.

Summary

The agile process models have come out of frustration with plan-driven software models. The agile alliance challenged established order in the software industry then.

Agile philosophy stressed four key issues
Self-organizing teams that have control over work they perform
Communication and collaboration among stakeholders
Respectful and responsible attitude towards change
Emphasis on rapid delivery of software

While agile alliance have given many new ideas, they could not come up with a single Solution or approach to the industry problems.

Agile proponents caused change of thinking among adherents of prescriptive process models.

Extreme Programming introduced concepts of Pair programming, Test-driven programming, Spike Solutions. Scrum introduced idea of brief and purposeful meetings.

Agile process models have brought in microplanning-oriented timeboxing in place of macroplanning.