Software Engineering

Presenting by:
B.Pranalini

Index

Unit 1 Part 2: Process Models

- Prescriptive Process Models
  - The Waterfall Model
  - Incremental Model
- Evolutionary Process Models
  - The Prototyping Model
  - The Spiral Model
- The Concurrent Development Model
- Specialized Process Models
- The Unified Process
- Personal and Team Process Models
- Agile Development.

Process Model

- Help in the software development
- Guide the software team through a set of framework activities
- Process Models may be linear, incremental or evolutionary

The Waterfall Model:

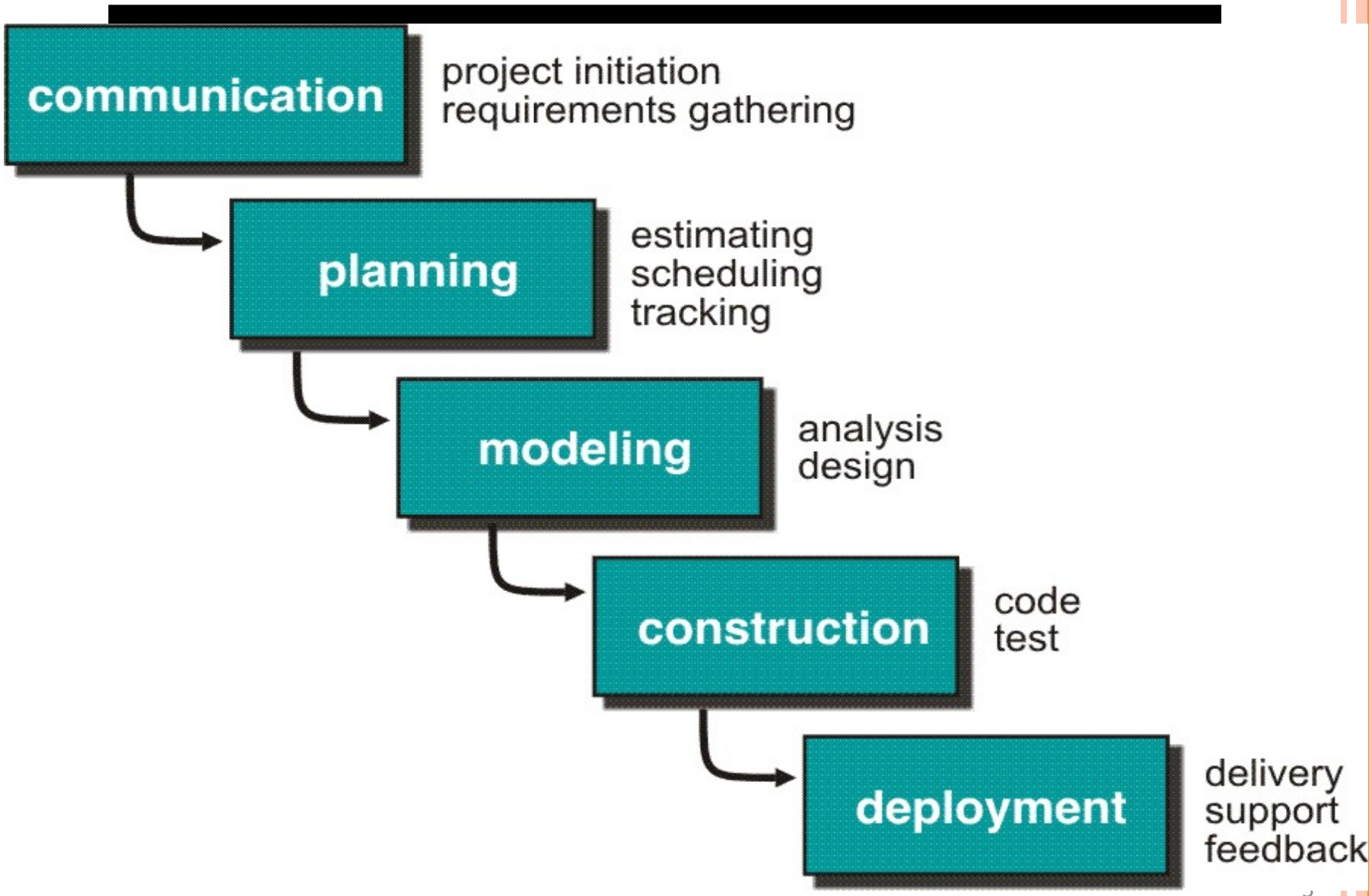
- Used when requirements are well understood in the beginning

- Also called classic life cycle

- Begins with customer specification of Requirements and progresses through planning, modeling, construction and deployment

- This Model suggests a systematic, sequential approach to SW development that begins at the system level and progresses through analysis, design, code and testing

The Waterfall Model



Merits :

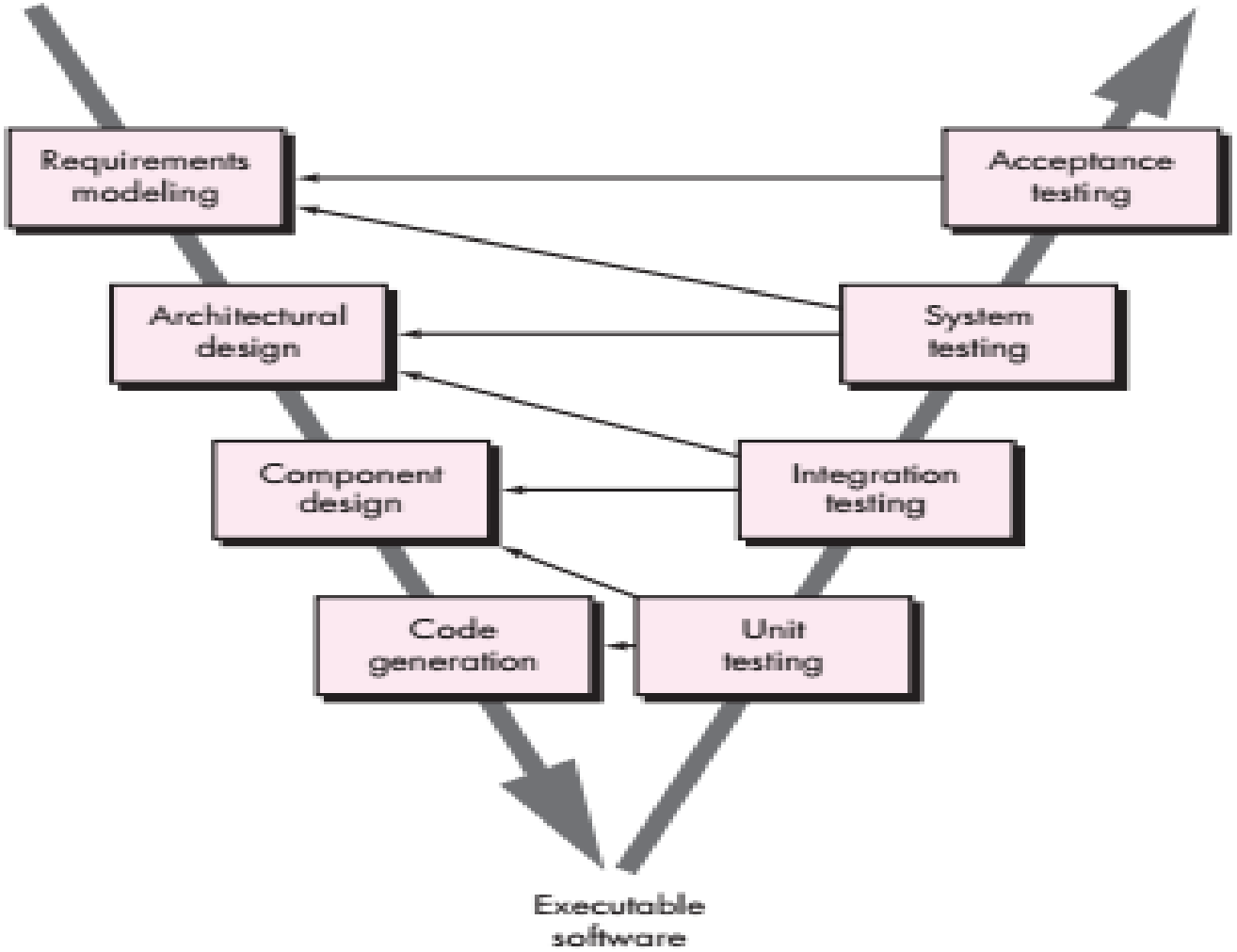
- It is systematic sequential approach for Software Development

Demerits:

- The model requires requirements to be explicitly spelled out in the beginning, which is often difficult
- All Customer Requirements at the start of project may be difficult
- Problems remain uncovered until testing phase
- Customer patience is needed, working version of the software is delivered too late

V-model

- A variation in the representation of the waterfall model is called the *V-model*.



- The V-model depicts the relationship of quality assurance actions to the actions associated with communication, modeling, and early construction activities.
- As a software team moves down the left side of the V, basic problem requirements are refined into progressively more detailed and technical representations of the problem and its solution.
- Once code has been generated, the team moves up the right side of the V, essentially performing a series of tests (quality assurance actions) that validate each of the models created as the team moved down the left side.
- In reality, there is no fundamental difference between the classic life cycle and the V-model. The V-model provides a way of visualizing how verification and validation actions are applied to earlier engineering work

Incremental Model:

- Each linear sequence produces deliverable "increments" of the software.
  (Ex: a Word Processor delivers basic file mgmt., editing, in the first increment; more sophisticated editing, document production capabilities in the 2 increment; spelling and grammar checking in the 3 increment).

- Software releases in increments

- 1st increment constitutes Core product

- Basic requirements are addressed

- Core product undergoes detailed evaluation by the

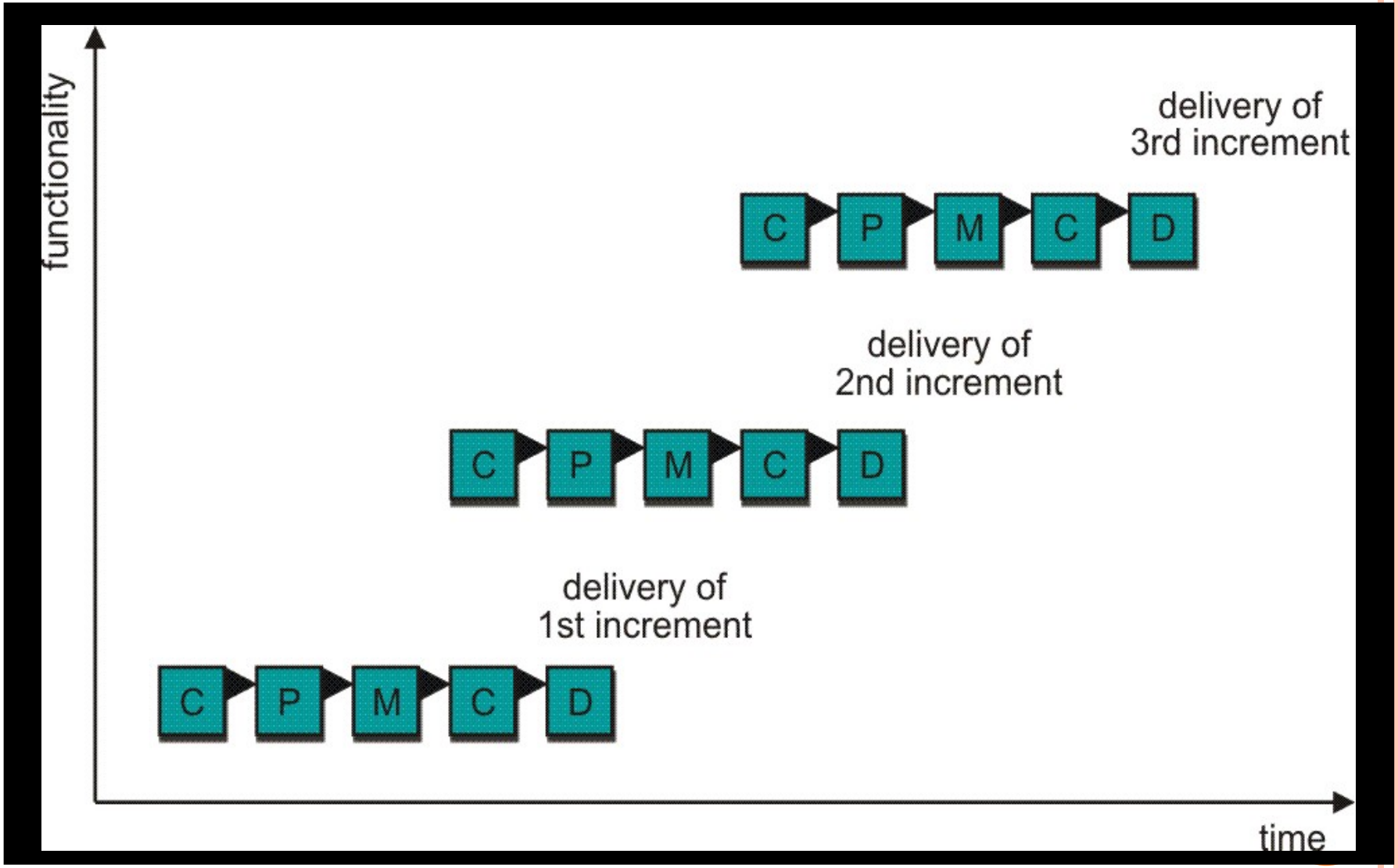
• As a result, of use and / or evaluation, plan is developed for the next increment. Plan addresses the modification of core product to better meet the needs of customer and the delivery of additional features and functionality

• Process is repeated until the complete product is produced

• Incremental model suits such projects which is used when initial requirements are reasonably well-defined and compelling need to provide limited functionality quickly

• Functionality expanded further in later releases

Incremental Models: Incremental



Merits :

- Less number of developers required
- All the requirements need not be known at the beginning of the project
- Technical risks can be managed

Demerits :

- Problems remain uncovered until testing phase
- Customer patience is needed, working version of the software is delivered too late.

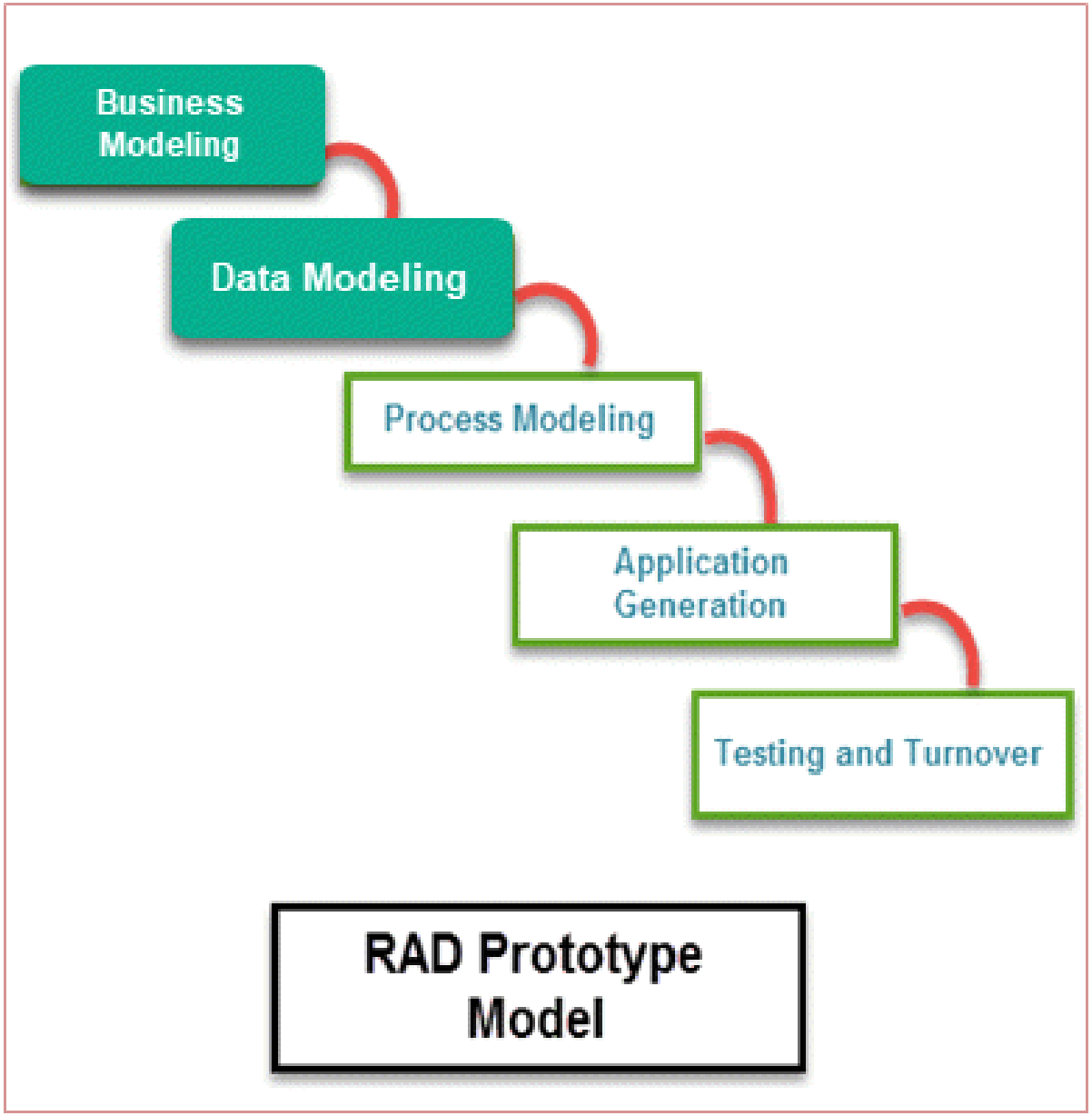
What is RAD Model?

RAD Model or Rapid Application Development model is a software development process based on prototyping without any specific planning. In RAD model, there is less attention paid to the planning and more priority is given to the development tasks. It targets at developing software in a short span of time.

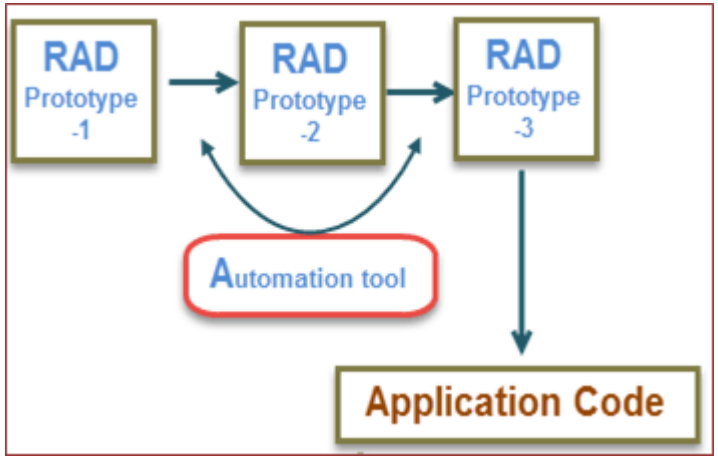
SDLC RAD modeling has following phases

- Business Modeling
- Data Modeling
- Process Modeling
- Application Generation
- Testing and Turnover

RAD Model Diagram


Business Modeling
Data Modeling
Process Modeling
Application Generation
Testing and Turnover
RAD Prototype Model

It focuses on input-output source and destination of the information. It emphasizes on delivering projects in small pieces; the larger projects are divided into a series of smaller projects. The main features of RAD modeling are that it focuses on the reuse of templates, tools, processes, and code.



RAD Model in Software Engineering

Different Phases of RAD Model

RAD Model Phases	Activities performed in RAD Modeling
Business Modeling	● On basis of the flow of information and distribution between various business channels, the product is designed
Data Modeling	● The information collected from business modeling is refined into a set of data objects that are significant for the business
Process Modeling	● The data object that is declared in the data modeling phase is transformed to achieve the information flow necessary to implement a business function
Application Generation	● Automated tools are used for the construction of the software, to convert process and data models into prototypes
Testing and Turnover	● As prototypes are individually tested during every iteration, the overall testing time is reduced in RAD.

Advantages of RAD Model	Disadvantages of RAD Model
• Flexible and adaptable to changes	• It can't be used for smaller projects
• It is useful when you have to reduce the overall project risk	• Not all application is compatible with RAD
• It is adaptable and flexible to changes	• When technical risk is high, it is not suitable
• It is easier to transfer deliverables as scripts, high-level abstractions and intermediate codes are used	• If developers are not committed to delivering software on time, RAD projects can fail
• Due to code generators and code reuse, there is a reduction of manual coding	• Reduced features due to time boxing, where features are pushed to a later version to finish a release in short period
• Due to prototyping in nature, there is a possibility of lesser defects	• Reduced scalability occurs because a RAD developed application begins as a prototype and evolves into a finished application
• Each phase in RAD delivers highest priority functionality to client	• Progress and problems accustomed are hard to track as such there is no documentation to demonstrate what has been done

Evolutionary Process Models:

- Software like all complex systems evolves over a period of time.
- Business and product requirements often change as development proceeds, making a straight-line path to an end product unrealistic
- Some of the core product or system requirements are well understood but the details of product or system extensions have yet to be defined.
- Solution is to adapt Evolutionary Model which is Iterative
- Types of evolutionary models
  – Prototyping
  – Spiral model

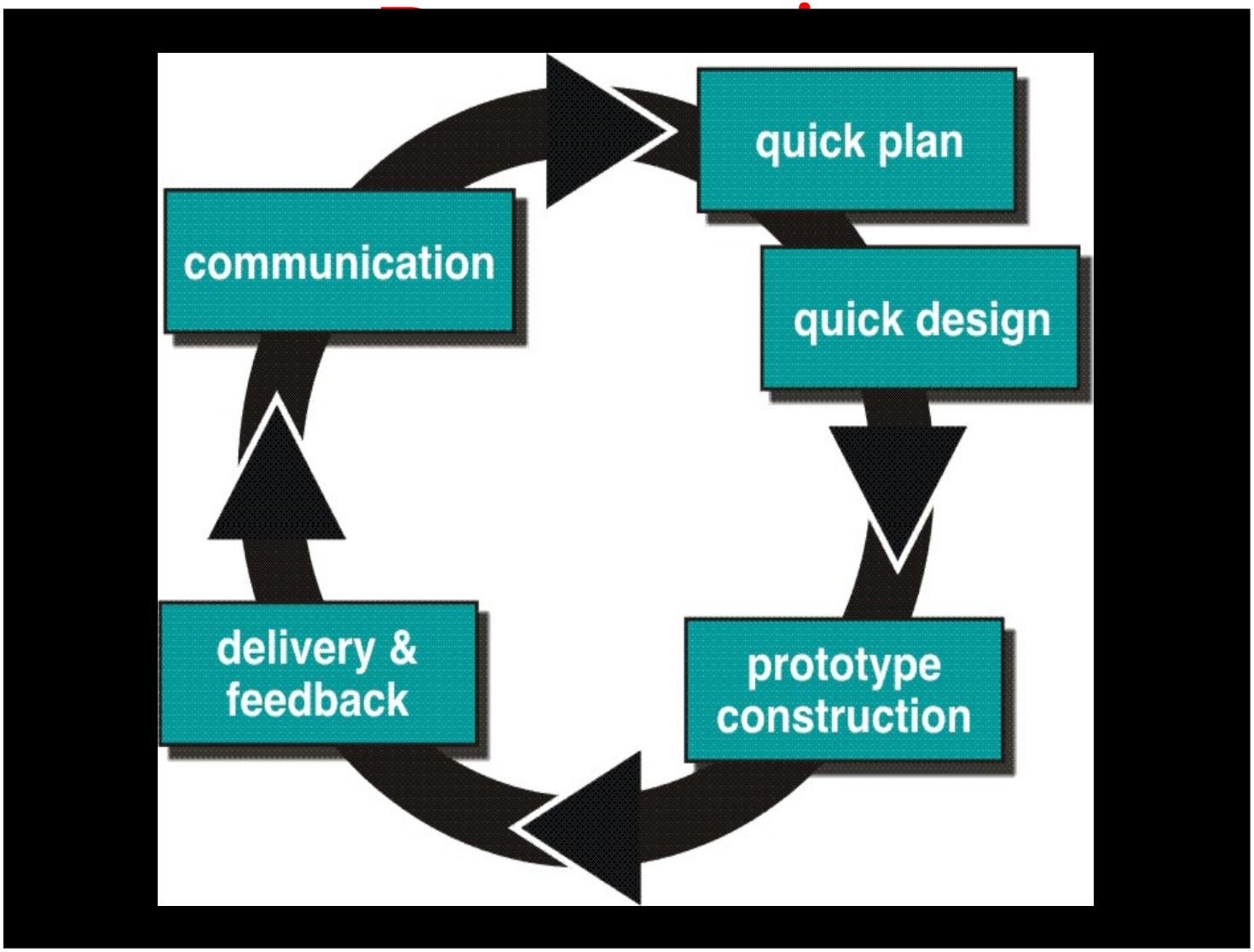
The Prototyping Model:

- Prototyping is used when customers requirements are fuzzy.

- OR the developer may not be sure of the efficiency of algorithm, the adaptability of an Operating System or the form that Human Computer interaction should take

- But we have to throw away the prototype once the customer requirements are clear & met for better quality. The product must be rebuilt using

Evolutionary Models:



STEPS IN PROTOTYPING

- Begins with requirement gathering
- Identify whatever requirements are known
- Outline areas where further definition is mandatory
- A quick design occur
- Quick design leads to the construction of prototype
- Prototype is evaluated by the customer
- Requirements are refined
- Prototype is turned to satisfy the needs of customer

The Prototyping Model:

Merits:

- Prototyping helps in requirement gathering & can be applied at any stage of the project.

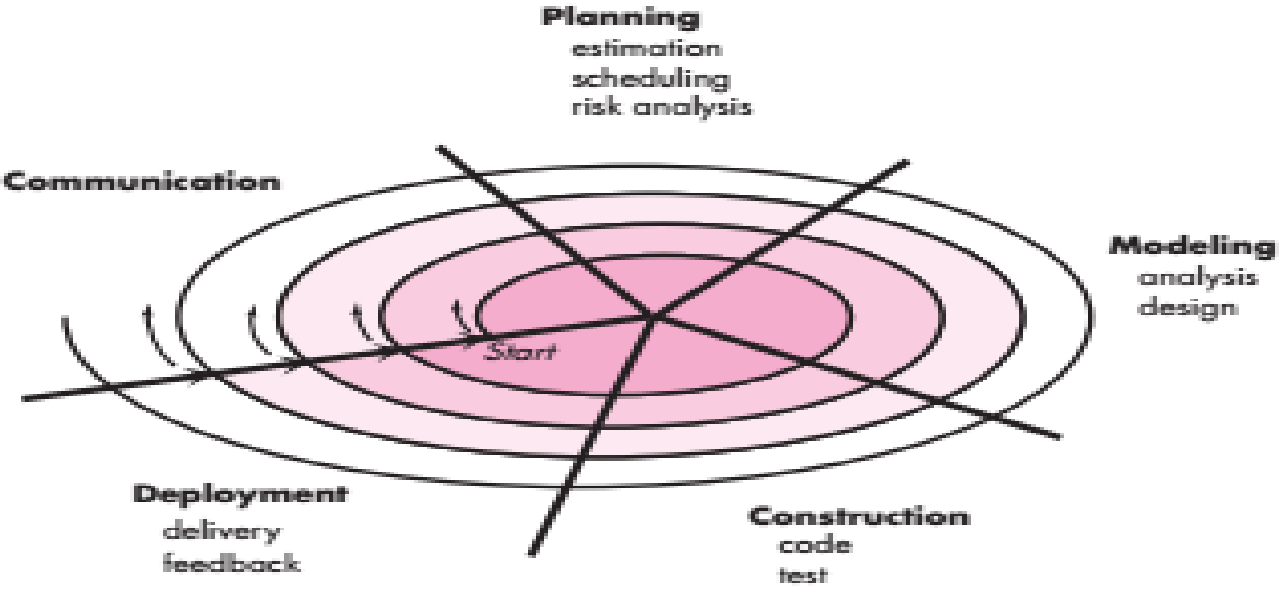
Demerits:

- Customer insists to convert prototype in working version by applying “few fixes”
- Developer may become comfortable with the compromises done. “The-less-than-ideal-choice” may become integral part of the system
- Use of inappropriate OS or PL
- Use of inefficient algorithm

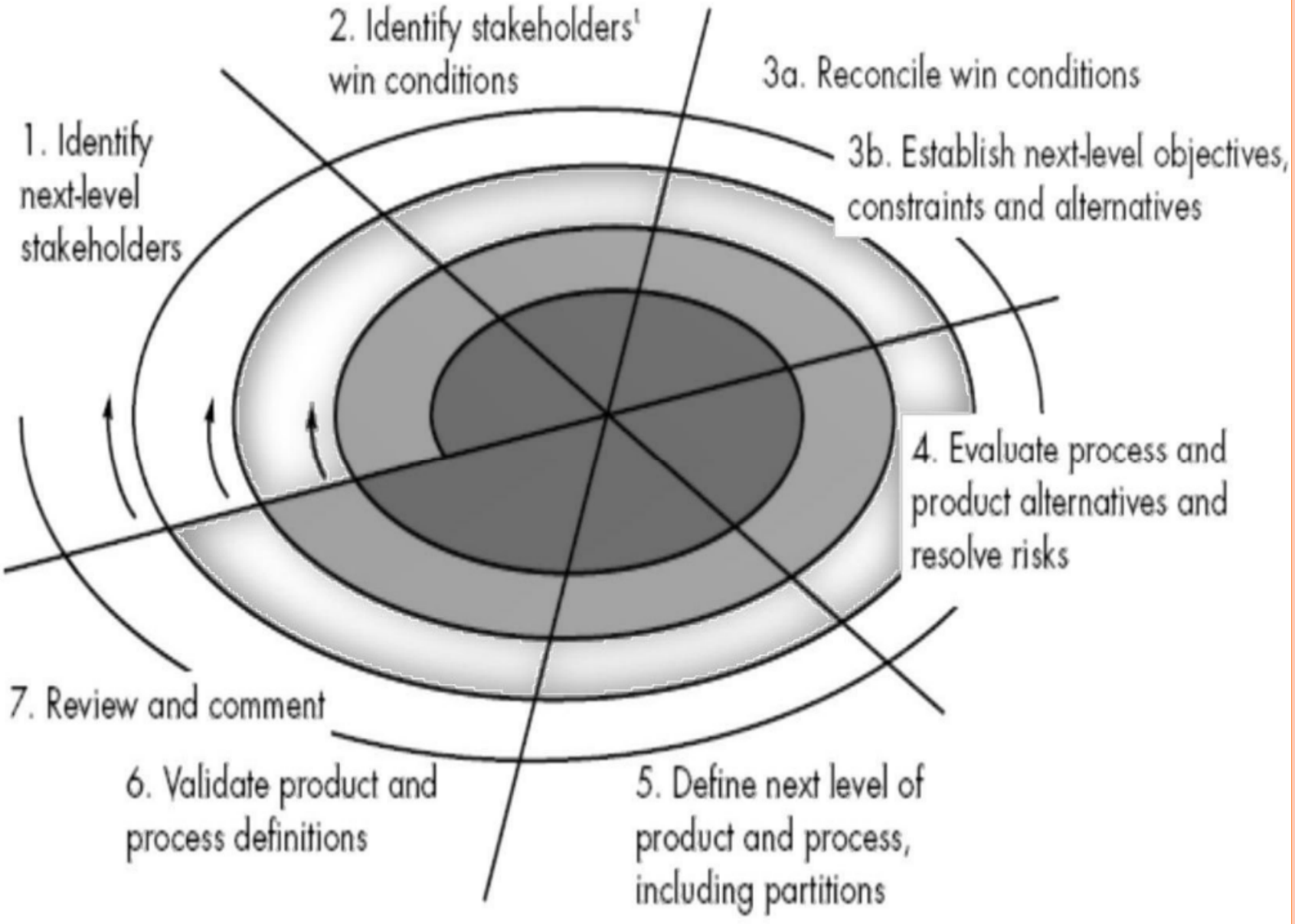
Evolutionary Models: Spiral

Spiral Model is an evolutionary software process model that couples the iterative nature of Prototyping with controlled & systematic aspects of the Waterfall Model

Include new element : Risk element. Starts in middle and continually visits the basic tasks of communication, plannir



- The first circuit around the spiral might result in the development of a product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software.
- Each pass through the planning region results in adjustments to the project plan. Cost and schedule are adjusted based on feedback derived from the customer after delivery.
- In addition, the project manager adjusts the planned number of iterations required to complete the software.


2. Identify stakeholders' win conditions
3a. Reconcile win conditions
1. Identify next-level stakeholders
3b. Establish next-level objectives, constraints and alternatives
4. Evaluate process and product alternatives and resolve risks
7. Review and comment
6. Validate product and process definitions
5. Define next level of product and process, including partitions

Merits:

- Risk is considered as each iteration is made
- Spiral Model can be applied throughout the life of the computer software.

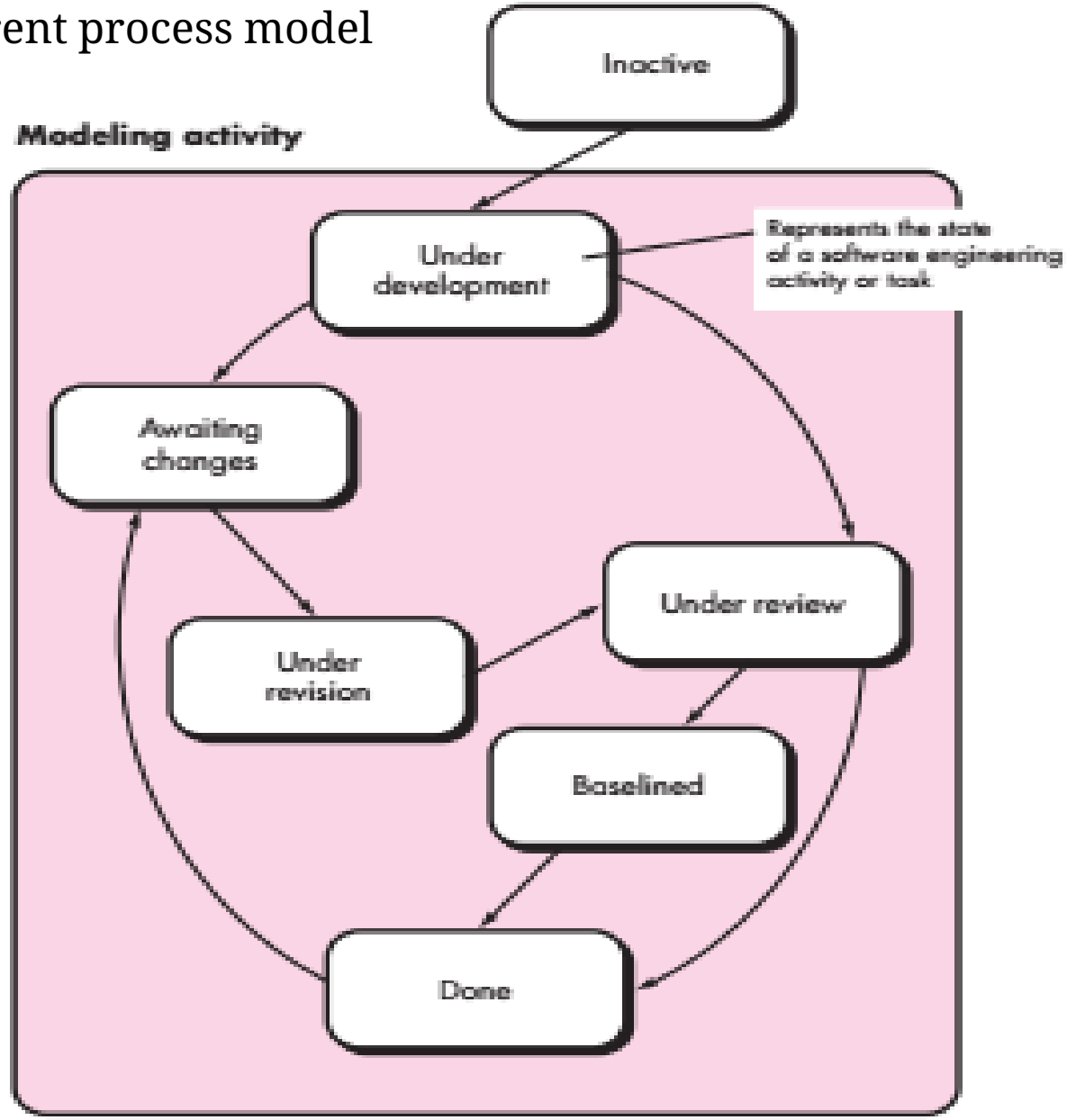
Demerits:

- It is difficult to convince customers that the evolutionary approach is controllable
- Considerable risk assessment expertise required
- If major risk is uncovered, problems will occur

Concurrent Development Model:

- Also called concurrent engineering
- Constitutes a series of framework activities, software engineering action, tasks and their associated states
- All activities exist concurrently but reside in different states
- Applicable to all types of software development
- Event generated at one point in the process trigger transitions among the states

Fig: One element of the concurrent process model



- Figure above provides a schematic representation of one Software engineering task within the modeling activity for the concurrent process model.
- The activity – modeling- may be in any one of the states noted at any given time. All activities exist concurrently but reside in different states
- **For example**, early in the project the communication activity has completed its first iteration and exists in the **awaiting changes state**. The modeling activity which existed in the **inactive state** while initial communication was completed now makes a transition into **underdevelopment state**. If, however, the customer indicates the changes in requirements must be made, the modeling activity moves from the **under development state** into the **awaiting changes state**.

Concurrent Development Model:

Merits:

- Applicable to all types of Software development & provides an accurate picture of the current state of project.

Demerits:

- Problem to Project planning. How many number of iterations are to be planned? Uncertainty…
- Process may fall in chaos if the evolutions occurs too fast without a period of relaxation. On the other hand if the speed is too slow productivity could be affected.
- Software processes are focussed on flexibility & extendability, rather than on high quality.

Specialized Process Models:

Specialized process models use many of the characteristics of one or more of the conventional models presented so far, however they tend to be applied when a narrowly defined software engineering approach is chosen. They include,

- Components based development
- The Formal Methods Model
- Aspect oriented software development

Components Based Development :

- In this approach, Commercial Off-The-Shelf (COTS) Software components, developed by vendors who offer them as products are used in the development of software.
- Characteristics resemble to spiral model.
- The component-based development model constructs applications from prepackaged software components.
- Modeling and construction activities begin with the identification of candidate components.
- These components can be designed as either conventional software modules or object-oriented classes or packages of classes.

The component-based development model incorporates the following steps (implemented using an evolutionary approach):

1. Available component-based products are researched and evaluated for the application domain in question.

2. Component integration issues are considered.

3. A software architecture is designed to accommodate the components.

4. Components are integrated into the architecture.

5. Comprehensive testing is conducted to ensure proper functionality.

- Merits:
  - Leads to software reuse, which provides number of benefits
    - 70% reduction in development cycle time
    - 84 % reduction in project cost
    - Productivity index goes up to 26.2 (Norm : 16.9)
- Demerits:
  - Component Library must be robust.
  - Performance may degrade

The Formal Methods Model:

- The Formal methods model encompasses a set of activities that lead to formal mathematical specification of computer software.

- It consists of specifications, development & verification by applying rigorous mathematical notation. Example, Clean Room Software Engineering (CRSE) is used by some software development organizations.

- When formal methods are used during design, they serve as a basis for program verification and therefore enable to discover and correct errors that might otherwise go undetected.

- Merits:
  - Eliminate many of the problems that are difficult to overcome using other Software Engineering Paradigms.
  - Ambiguity, Incompleteness & Inconsistency can be discovered & corrected easily by using formal methods of mathematical analysis.
  - It offers the promise of defect-free software.
- Demerits:
  - Development is time consuming & expensive
  - Extensive training is required
  - Difficult to use with technically unsophisticated customers

Aspect Oriented Software Development (AOSD):

- A set of localized features, functions & information contents are used while building complex software.
- These localized s/w characteristics are modeled as components (e.g. OO classes) & then constructed within the context of a system architecture.
- Certain “concerns” (Customer required properties or areas of technical interest) span the entire architecture i.e. Cross cutting Concerns like system security, fault tolerance etc.
- AOSD is often referred to as *aspect-oriented programming* (AOP)
- AOP is a relatively new software engineering paradigm that provides a process and methodological approach for defining, specifying, designing, and constructing *aspects*—“mechanisms beyond subroutines and inheritance for localizing the expression of a

- Grundy [Gru02] provides further discussion of aspects in the context of what he calls *aspect-oriented component engineering* (AOCE):
  - AOCE uses a concept of horizontal slices through vertically-decomposed software components, called “aspects,” to characterize cross-cutting functional and non-functional properties of components.
  - Common, systemic aspects include user interfaces, collaborative work, distribution, persistency, memory management, transaction processing, security, integrity and so on.
- A distinct aspect-oriented process has not yet matured.
- However, it is likely that such a process will adopt characteristics of both evolutionary and concurrent process models.
- The evolutionary model is appropriate as aspects are identified and then constructed.

- Merits:
  - It is similar to component based development for aspects
- Demerits:
  - Component Library must be robust.
  - Performance may degrade

Unified Process Model

The *Unified Process*, Ivar Jacobson, Grady Booch, and James Rumbaugh discuss the need for a:

- use-case driven: The Unified Process recognizes the importance of customer communication and streamlined methods for describing the customer's view of a system (the use case).

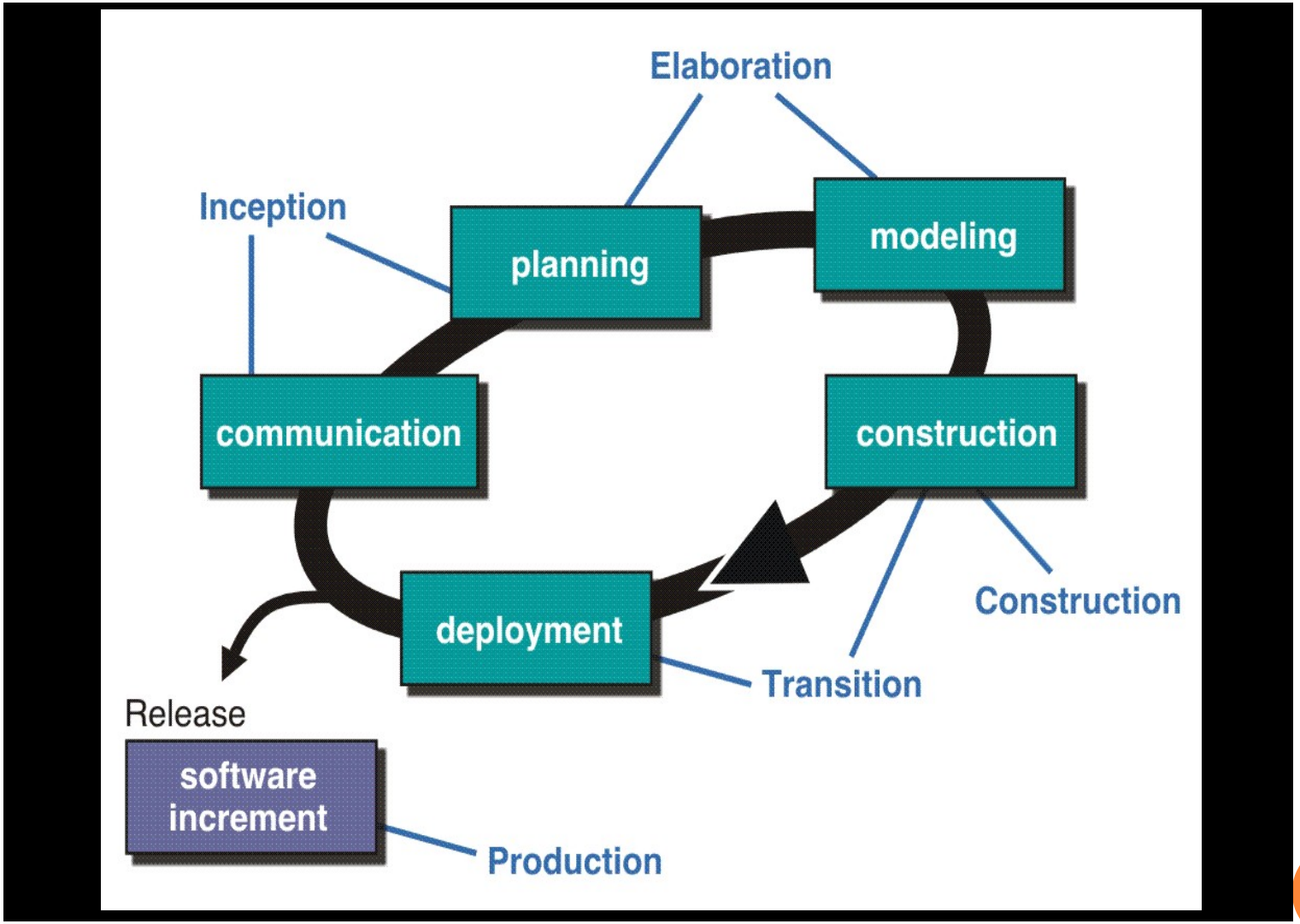
- architecture-centric:It emphasizes the important role of software architecture and "helps the architect focus on the right goals, such as understandability, reliance to future changes, and reuse".

- iterative and incremental: It suggests a process flow that is iterative and incremental, providing the evolutionary feel that is essential in modern

A Brief history

- A “unified method” that would combine the best features of each of their individual object-oriented analysis and design methods and adopt additional features proposed by other experts in object-oriented modeling.
- The result was UML—a *unified modeling language* that contains a robust notation for the modeling and development of object-oriented systems.

The Unified Process (UP)



Phases of Unified Process

- The figure above depicts the phases of the UP and relates them to the generic activities.
- The Inception phase of the UP encompasses both customer communication and planning activities.
  - By collaborating with the customer and end-users, business requirements for the software are identified, a rough architecture for the system is proposed, and a plan for the iterative, incremental nature of the ensuing project is developed.
- The elaboration phase encompasses the planning and modeling activities of the generic process model.
  - Elaboration refines and expands the preliminary use-cases that were developed as part of the inception phase and expands the architectural representation to include five different views of the software - the use-case model, the analysis model, the design model, the implementation model,

- The construction phase of the UP is identical to the construction activity defined for the generic software process.
  - Using the architectural model as input, the construction phase develops or acquires the software components that will make each use-case operational for end-users.

- The transition phase of the UP encompasses the latter stages of the generic construction activity and the first part of the generic deployment activity.
  - Software is given to end-users for beta testing, and user feedback reports both defects and necessary changes.
  - At the conclusion of the transition phase, the software increment becomes a usable software release “user manuals, trouble-shooting guides, and installation procedures”.

- The production phase of the UP coincides with the development activity of the generic process.
  - The on-going use of the software is monitored, support for the operating environment is provided and defect reports and requests for changes are submitted and evaluated.
- A Software Engineering workflow is distributed across all UP phases.

Common Fears for Developers

- The project will produce the wrong product.
- The project will produce a product of inferior quality.
- The project will be late.
- We'll have to work 80 hour weeks.
- We'll have to break commitments.
- We won't be having fun.

Personal Software Process (PSP) and Team Software Process (TSP)

- PSP is a high-maturity process framework for individuals
- TSP addresses high-maturity practices for teams of PSP-trained engineers
- The team itself can create its own process, and at the same time meet the narrower needs of individuals and the broader needs of the organization.
- Watts Humphrey argues that it is possible to create a “personal software process” and/or a “team software process.”
- Both require hard work, training, and coordination, but both are achievable

Personal Software Process (PSP)

- The *Personal Software Process* (PSP) emphasizes personal measurement of both the work product that is produced and the resultant quality of the work product.

- In addition PSP makes the practitioner responsible for project planning (e.g., estimating and scheduling) and empowers the practitioner to control the quality of all software work products that are developed.

PSP process model defines five framework activities: planning, high-level design, high-level design review, development, and postmortem. It stresses the need to identify errors early and to understand the types of errors.

Planning: it isolates requirements. And a project schedule is created.
High-level design: Prototypes are built when uncertainty exists.
High-level design review: Formal verification methods are applied to uncover errors in the design.
Development: Code is generated, reviewed, compiled, and tested.
Postmortem: using the measures and metrics collected, the effectiveness of the process is determined.

What does PSP Developer DO

- Tracks basic development process data
  – Size, time, defects, and task completion
  – Time & defects are tracked by phase, e.g., planning, design, code, personal reviews, test, postmortem
- Uses data derived from the basic data for process management and improvement
- Plans using historical data and tracks progress
  – “PROBE” (PROxy Based Estimating) estimating
  – “Earned Value” scheduling & tracking
  – Quality planning
- “Builds in” Quality
  – Produces verifiable designs
  – Conducts structured personal design and code reviews
- Improves development process using data

Team Software Process (TSP)

- The goal of TSP is to build a “self-directed” project team that organizes itself to produce high-quality software.
- Each project is “launched” using a “script” that defines the tasks to be accomplished.
  Teams are self-directed.
  Measurement is encouraged.
- Measures are analyzed with the intent of improving the team process.
- TSP defines the following framework activities: **project launch, high-level design, implementation, integration and test,** and **postmortem**

objectives for TSP:

- Build self-directed teams that plan and track their work, establish goals, and own their processes and plans. These can be pure software teams or integrated product teams (IPTs) of 3 to about 20 engineers.

- Show managers how to coach and motivate their teams and how to help them sustain peak performance.

- Accelerate software process improvement by making CMM23 Level 5 behavior normal and expected.

- Provide improvement guidance to high-maturity organizations

What does TSP Developer DO

- Developers use PSP practices for their personal work
- For each development phase (2-4 months), the team
  - Conducts a team "launch" to come to a common understanding of the project & to develop detailed plans
  - Tracks progress against schedule and quality weekly, adjusts plans, and takes immediate action if necessary to ensure commitments will be met
  - Uses team-level data the same way as developers use their individual data to assess schedule and quality
- Quality data, Software inspections, Time on Task, Earned value , etc

Agile Development

In 2001, Kent Beck and 16 other noted software developers, writers, and consultant signed the "Manifesto for Agile Software Development." It stated:

> "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 …

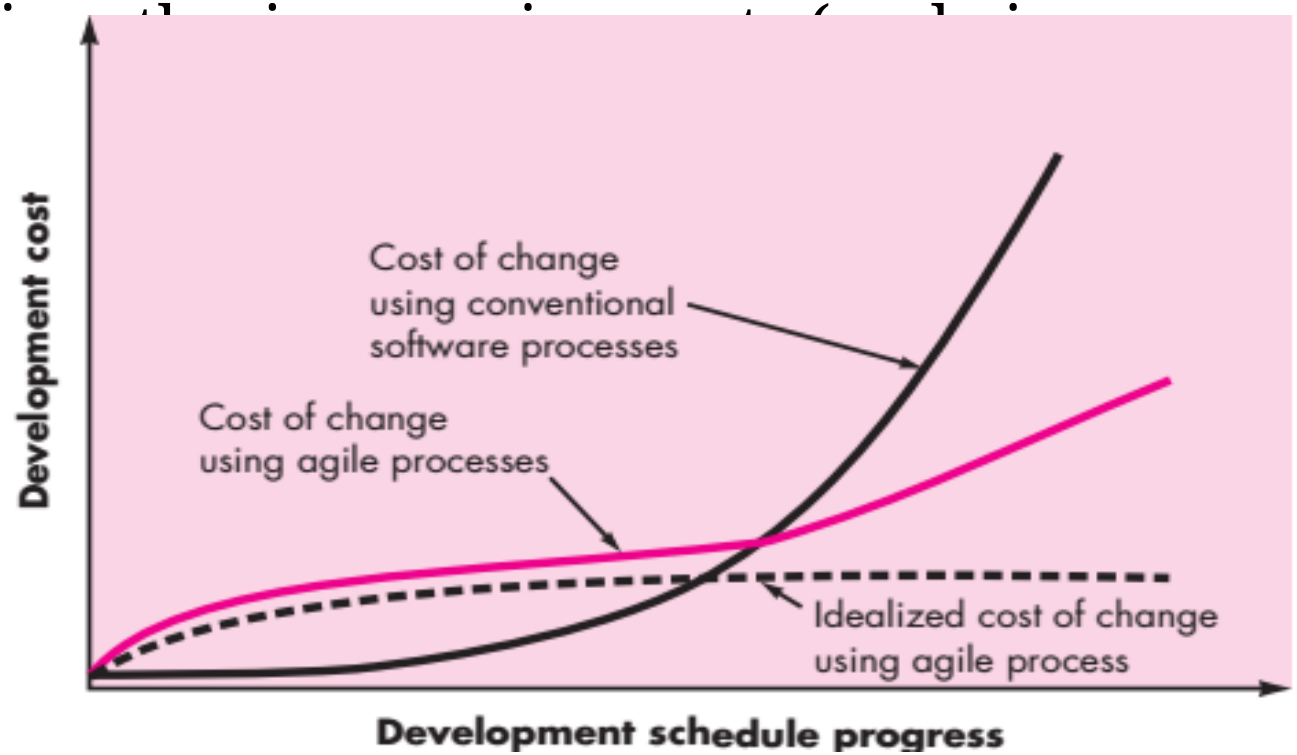
- Rapid, incremental delivery of software
- The development guidelines stress delivery over analysis and design although these activates are not
  discouraged, and active and continuous communication between developers and customers.

Agility and the Cost of Change

- The **conventional** wisdom in software development (supported by decades of experience) is that the cost of change increases nonlinearly as a project progresses (Figure, solid black curve).
- It is relatively easy to accommodate a change when a software team

project)

Figure: Change cc
as a function
of time in
development



- Proponents of agility argue that a well-designed **agile process** "flattens" the cost of change curve (Figure, shaded, solid curve), allowing a software team to accommodate changes late in a software project without dramatic cost and time impact.
- Agile process encompasses incremental delivery.
- When incremental delivery is coupled with other agile practices such as continuous unit testing and pair programming, the cost of making a change is attenuated.

What is an Agile Process ?

- Is driven by customer descriptions of what is required (scenarios)
  - 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(hard to say how much design is necessary before construction)
- Adapts as changes occur
- Delivers multiple 'software increments', deliver an operational prototype or portion of an OS to collect customer feedback for adaption.

Principles of Agility

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

1. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

1. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time scale.

1. Business people and developers must work together daily throughout the project.

Principles of Agility

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

5. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

5. Working software is the primary measure of progress.

5. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Principles of Agility

9. Continuous attention to technical excellence and good design enhances agility.

9. Simplicity - the art of maximizing the amount of work not done - is essential.

9. The best architectures, requirements, and designs emerge from self-organizing teams.

9. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

The Politics of Agile Development

- Jim Highsmith (facetiously) states the extremes when he characterizes the feeling of the pro-agility camp ("agilists"). "Traditional methodologists are a bunch of stick-in-the-muds who'd rather produce flawless documentation than a working system that meets business needs."
- As a counterpoint, he states (again, facetiously) the position of the traditional software engineering camp: "Lightweight, er, 'agile' methodologists are a bunch of glorified hackers who are going to be in for a heck of a surprise when they try to scale up their toys into enterprise-wide software."
- No one is against agility. The real question is: What is

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 :
  - 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,

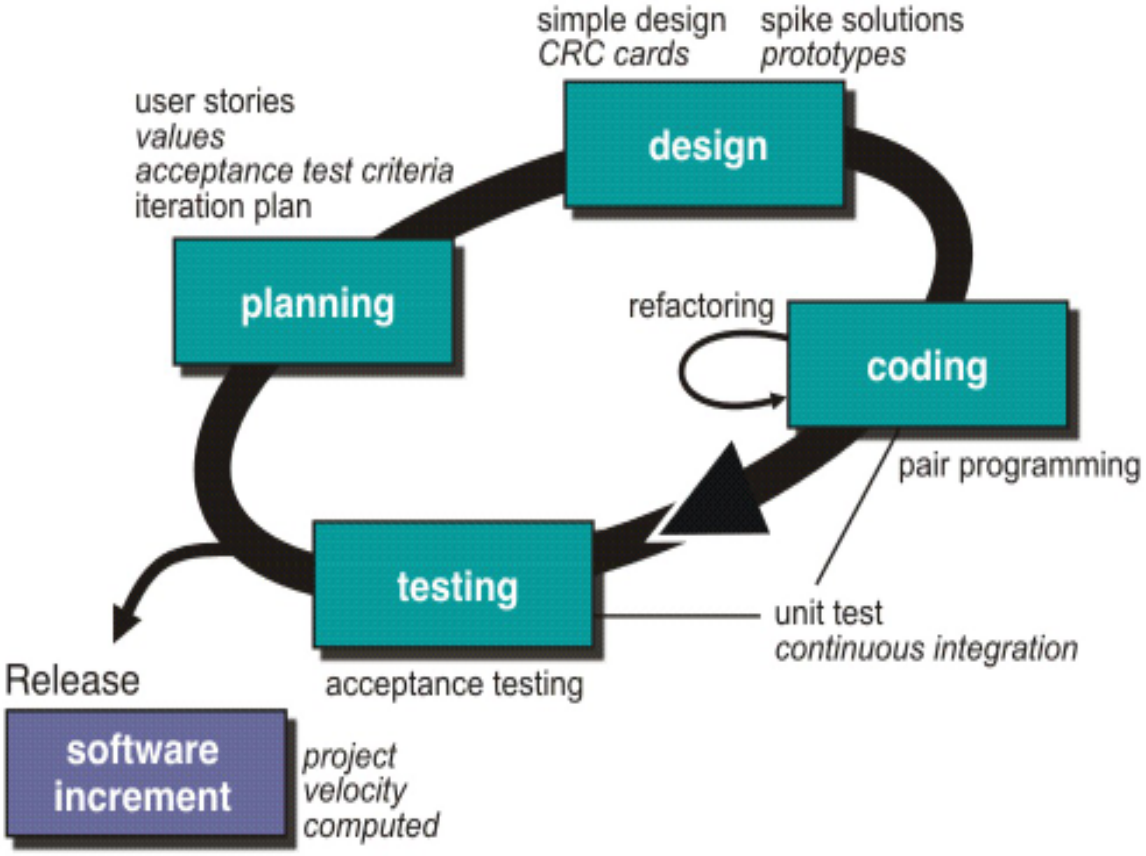
Extreme Programming (XP)

- 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
  - Working together, 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

Extreme Programming (XP)

- XP Design
  - Follows the KIS principle
  - Encourage the use of CRC cards (see Chapter 8)
  - 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)



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.

Other Agile Processes

- Adaptive Software Development (ASD)
- Dynamic Systems Development Method (DSDM)
- Scrum
- Crystal
- Feature Driven Development
- Agile Modeling (AM)

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- make more considered decisions
  - Component-based focus-decomposition of the design into individual functional or logical components
  - Uses "time-boxing" - time management technique(fixed time)
  - Explicit consideration of risks

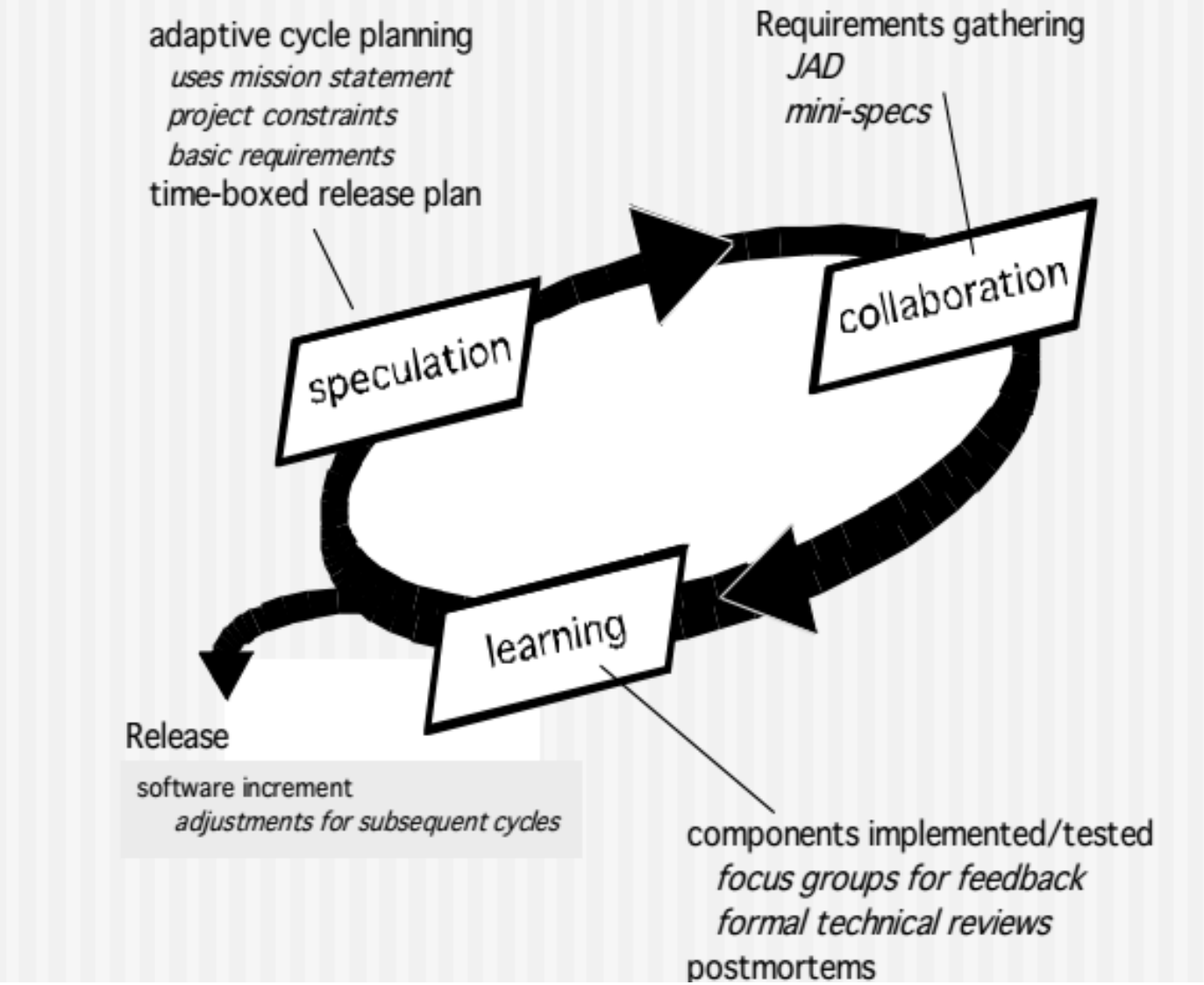
Three Phases of ASD

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

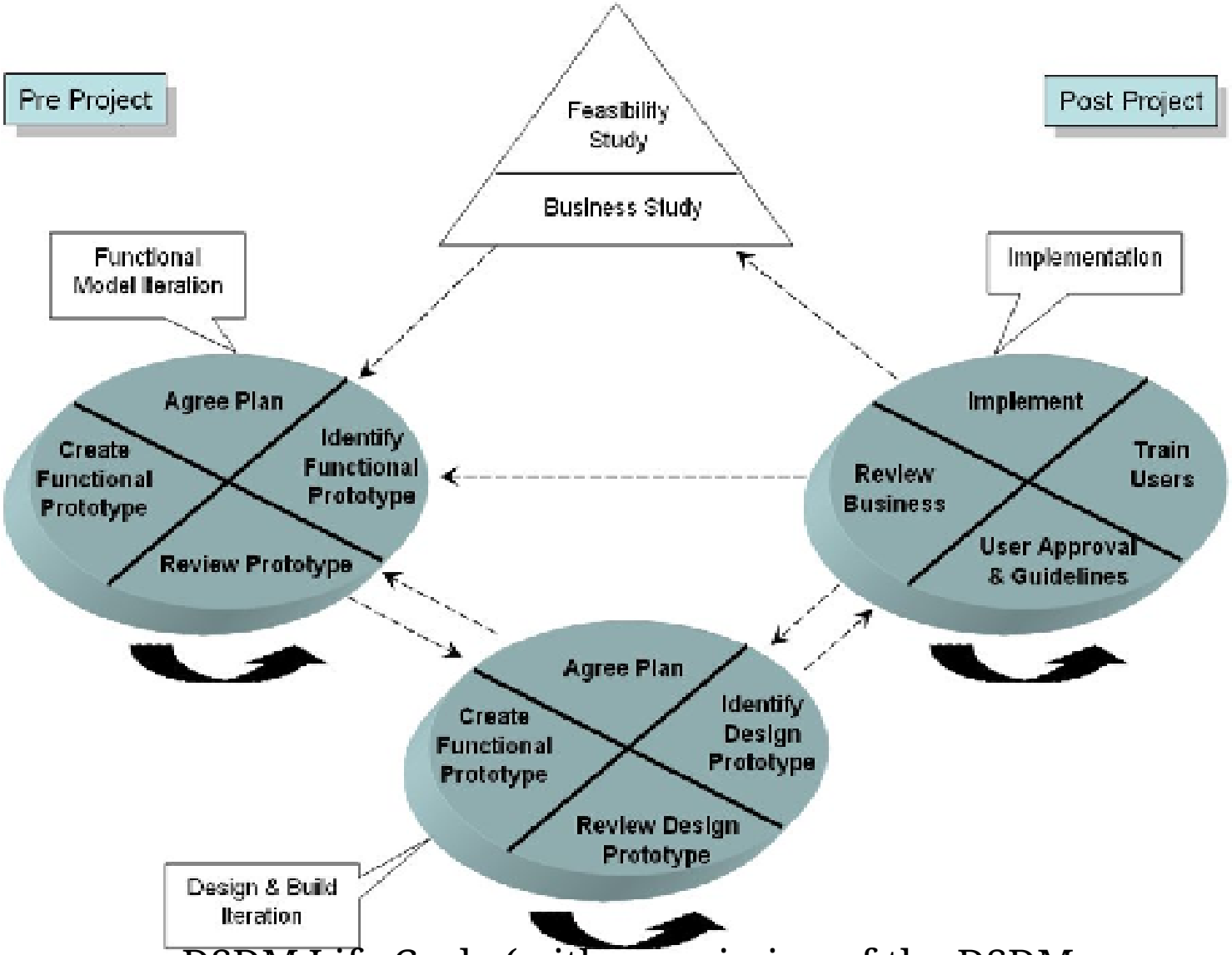
Adaptive Software Development (ASD)



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

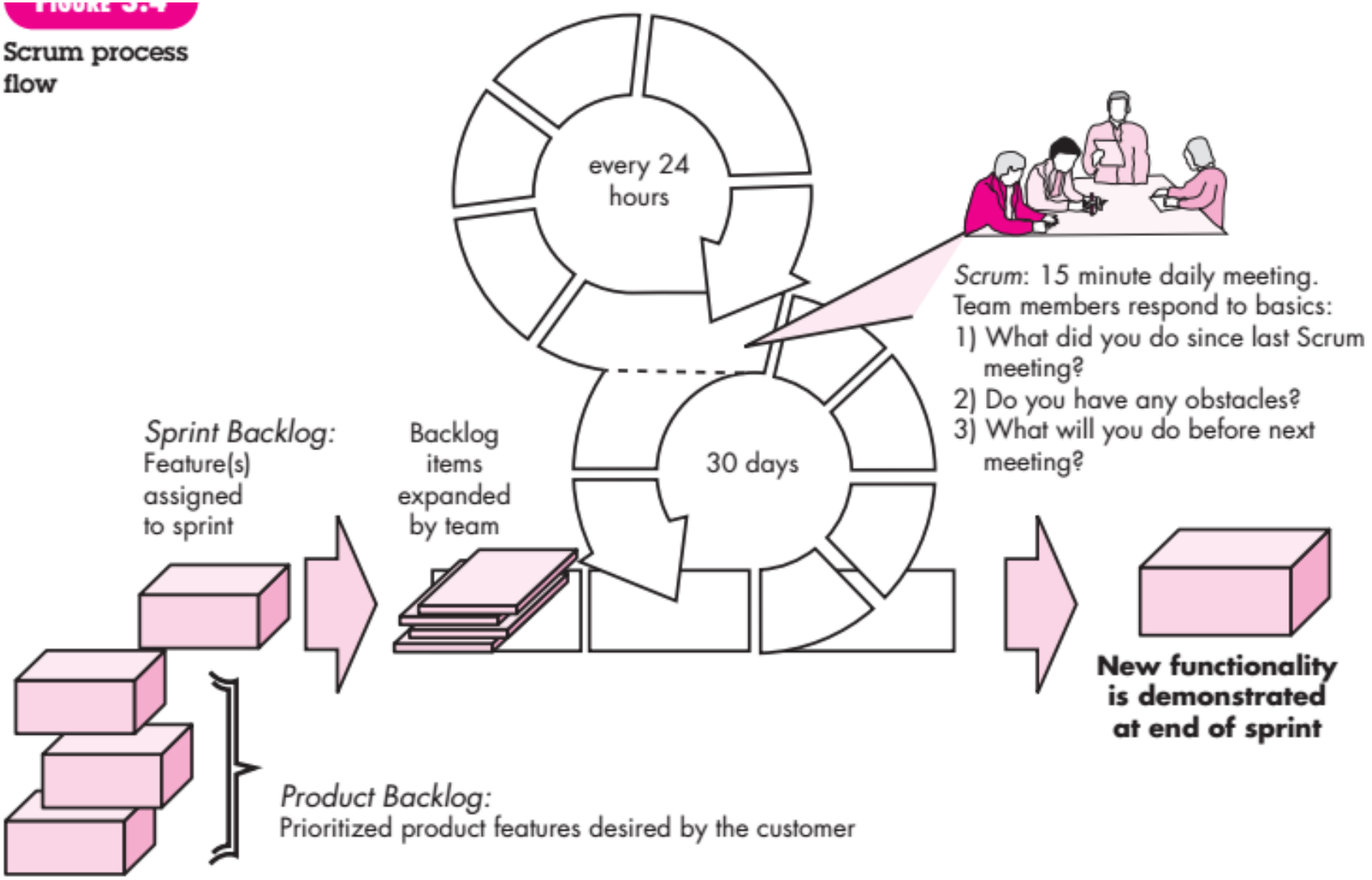


DSDM Life Cycle (with permission of the DSDM

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



Scrum process flow

Crystal

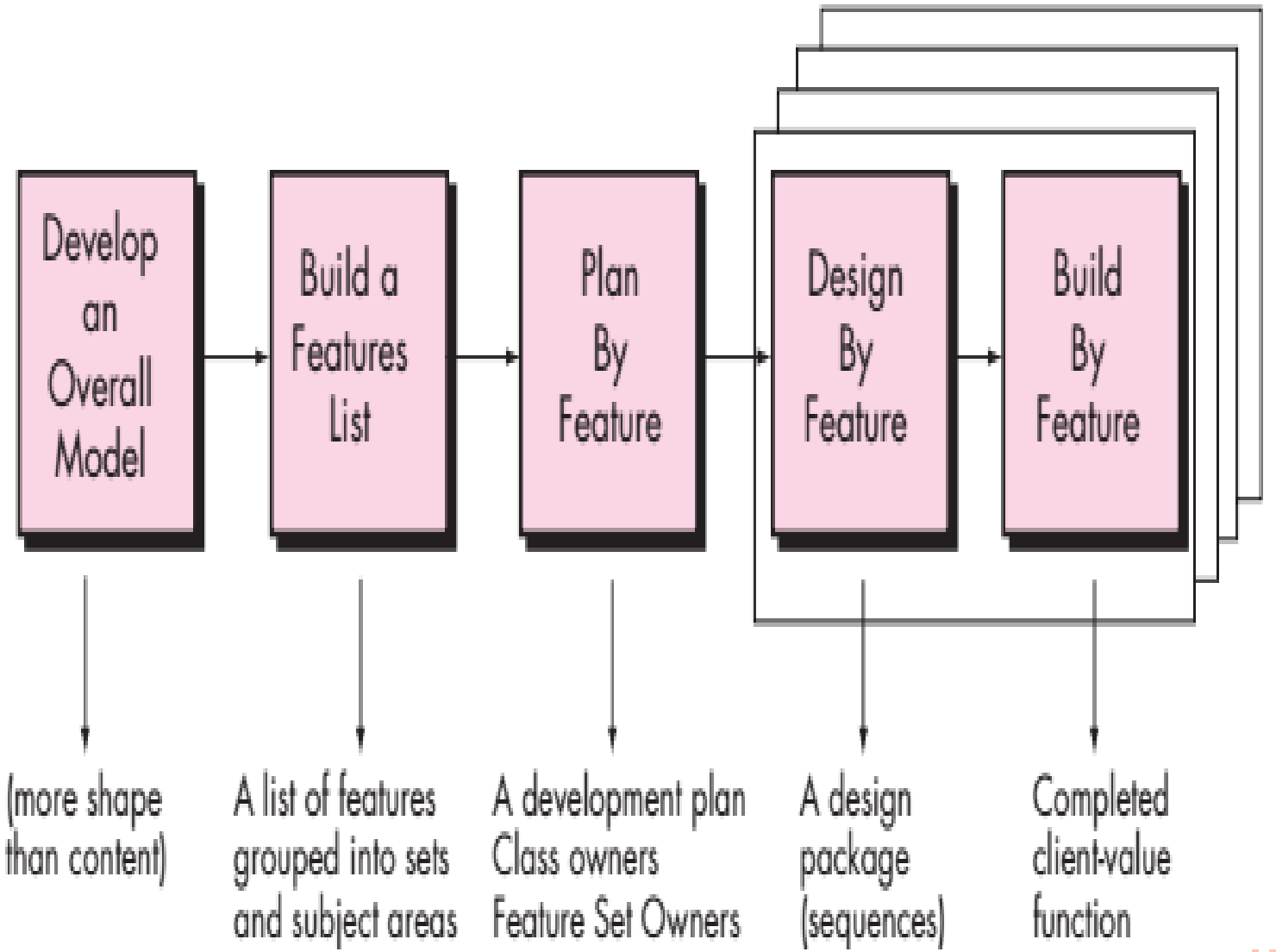
- Proposed by Cockburn and Highsmith
- Crystal—distinguishing features
  - Actually a family of process models that allow "maneuverability" based on problem characteristics
  - Face-to-face communication is emphasized
  - Suggests the use of "reflection workshops" to review the work habits of the team

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 feature list is created and "plan-by-feature" is conducted.

Feature Driven Development

Feature Driven Development [Coa99] (with permission)



Agile Modeling

- Originally proposed by Scott Ambler
- Suggests a set of agile modeling principles
  - Model with a purpose
  - Use multiple models
  - Travel light- enough documentation about the models that you are developing
  - Content is more important than representation
  - Know the models and the tools you use to create them
  - Adapt locally

Thank You