The Software Process

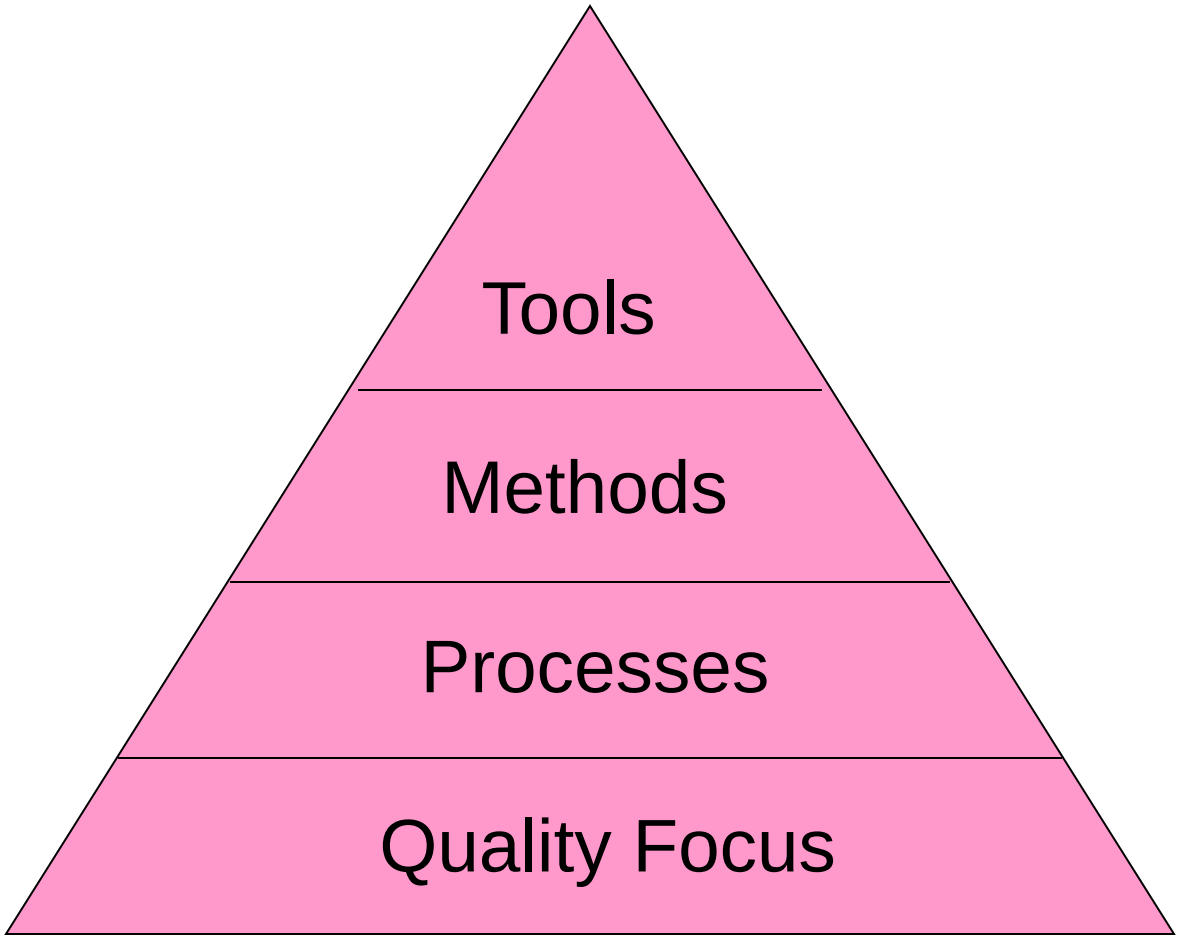
Software Life Cycle

- Requirement analysis and definition
- System and software requirements
- Implementation and unit testing
- Integration and system testing
- Operation and Maintenance

Software Process

- A set of activities and associated results which produce a software product.
- Activities include:
  - Software specification
  - Software development
  - Software validation
  - Software evolution

Software Engineering is a Layered Technology



Process, Methods, and Tools

- Process
  - Provides the glue that holds the layers together; enables rational and timely development; provides a framework for effective delivery of technology; forms the basis for management; provides the context for technical methods, work products, milestones, quality measures, and change management
- Methods
  - Provide the technical "how to" for building software; rely on a set of basic principles; encompass a broad array of tasks; include modeling activities
- Tools
  - Provide automated or semi-automated support for the process and methods (i.e., CASE tools)

Umbrella Activities

- Software requirements management
- Software project planning
- Software project tracking and oversight
- Software quality assurance
- Software configuration management
- Formal technical reviews
- Risk management
- Measurement – process, project, product
- Reusability management (component reuse)
- Work product preparation and production

What is a Process?

- (Webster) A system of operations in producing something; a series of actions, changes, or functions that achieve an end or a result
- (IEEE) A sequence of steps performed for a given purpose

What is a Software Process?

- (SEI) A set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products (e.g., project plans, design documents, code, test cases, and user manuals)
- As an organization matures, the software process becomes better defined and more consistently implemented throughout the organization
- Software process maturity is the extent to which a specific process is explicitly defined, managed, measured, controlled, and effective

Generic Process Framework

- Communication
  - Involves communication among the customer and other stake holders; encompasses requirements gathering
- Planning
  - Establishes a plan for software engineering work; addresses technical tasks, resources, work products, and work schedule
- Modeling (Analyze, Design)
  - Encompasses the creation of models to better understand the requirements and the design
- Construction (Code, Test)
  - Combines code generation and testing to uncover errors
- Deployment
  - Involves delivery of software to the customer for evaluation and feedback

Modeling: Software Requirements Analysis

- Helps software engineers to better understand the problem they will work to solve
- Encompasses the set of tasks that lead to an understanding of what the business impact of the software will be, what the customer wants, and how end-users will interact with the software
- Uses a combination of text and diagrams to depict requirements for data, function, and behavior
  - Provides a relatively easy way to understand and review requirements for correctness, completeness and consistency

Modeling: Software Design

- Brings together customer requirements, business needs, and technical considerations to form the “blueprint” for a product
- Creates a model that that provides detail about software data structures, software architecture, interfaces, and components that are necessary to implement the system
- Architectural design
  - Represents the structure of data and program components that are required to build the software
  - Considers the architectural style, the structure and properties of components that constitute the system, and interrelationships that occur among all architectural components
- User Interface Design
  - Creates an effective communication medium between a human and a computer
  - Identifies interface objects and actions and then creates a screen layout that forms the basis for a user interface prototype
- Component-level Design
  - Defines the data structures, algorithms, interface characteristics, and communication mechanisms allocated to each software component

Software Process Model

- A simplified description of a software process which is presented from a particular perspective.
- Process Model includes:
  - Activities of the software process
  - Software products
  - Role of people

Process Model Types

- “Prescriptive”
  - Model includes a specific set of tasks, along with a workflow for these tasks and definite milestones and outcomes for each task; end result is the desired product
- "Agile“
  - Model tends to be simpler than prescriptive models; emphasis is on incremental development, customer satisfaction, and minimal process overhead
- "Mathematical“
  - Formal Method Model stresses mathematical rigor and formal proofs that product is meeting carefully-defined goals

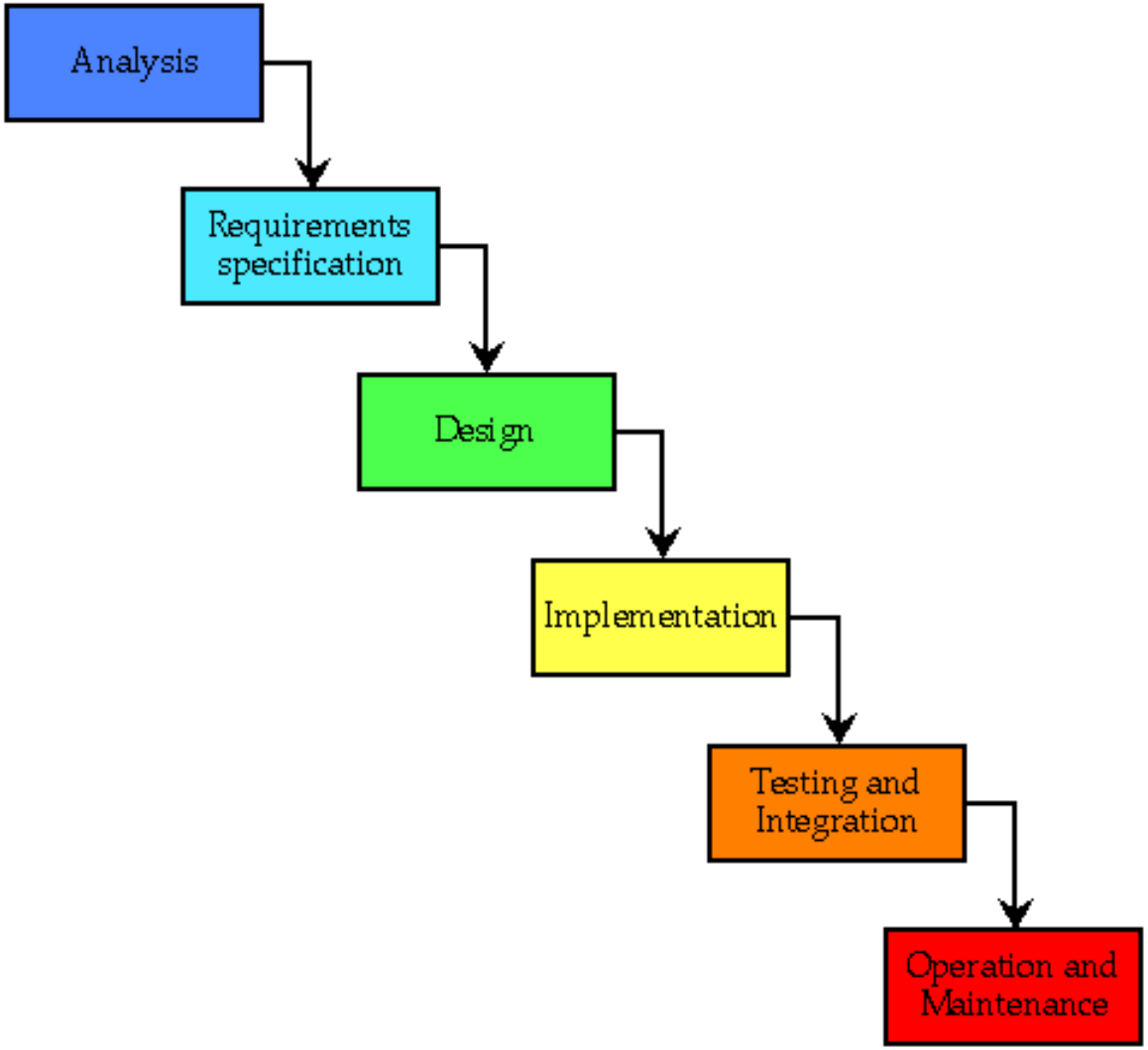
Prescriptive Process Models

- Also called “conventional process models”
- Have a specific set of:
  - framework activities
  - Software engineering tasks
  - Work products
  - QA
  - Change control mechanism

The Waterfall Model

- Proposed in 1970 by Royce
- Sequential Process Model
- Its name due to the cascading effect from one phase to the other
- Move to next phase when its preceding phase is completed and perfected
- No overlapping in the phases
- No jumping back and forth in the phases

The Waterfall Model (contd)



The Waterfall Model (contd)

- Documentation Driven Model
- Applicable when
  - The requirements are stable and well defined
  - Well defined enhancements to an existing system are to be made

The Waterfall Model (contd)

- Advantages
  - Testing is inherent to every phase
  - It is an enforced disciplined approach
  - Maintenance is easier
- Disadvantages
  - Not realistic approach
  - Complete system delivered at the end
  - Difficult to finalize requirements at early stage
  - Difficult cost and time estimations

Incremental Process Models

- Produce the software in increments
- Each increment delivers the customer a working product
- Iterative nature
- First increment-core product
- Evaluated by customer
- Feedback
- Next increment plan

Incremental Process Models (contd)

- Advantages
  - Helps when the project team is small
  - Feedback improves next increment
  - In time working product delivered to customer

- Example:
  - Incremental model
  - RAD (Rapid Application Development)

RAD

- Incremental software process model
- Emphasizes short development cycle
- Faster development
- Use of Computer Assisted Software Engineering (CASE) Tools
- Component based construction
- System delivered in short time (2 to 3 months)
- Useful where requirements are well understood and scope is limited

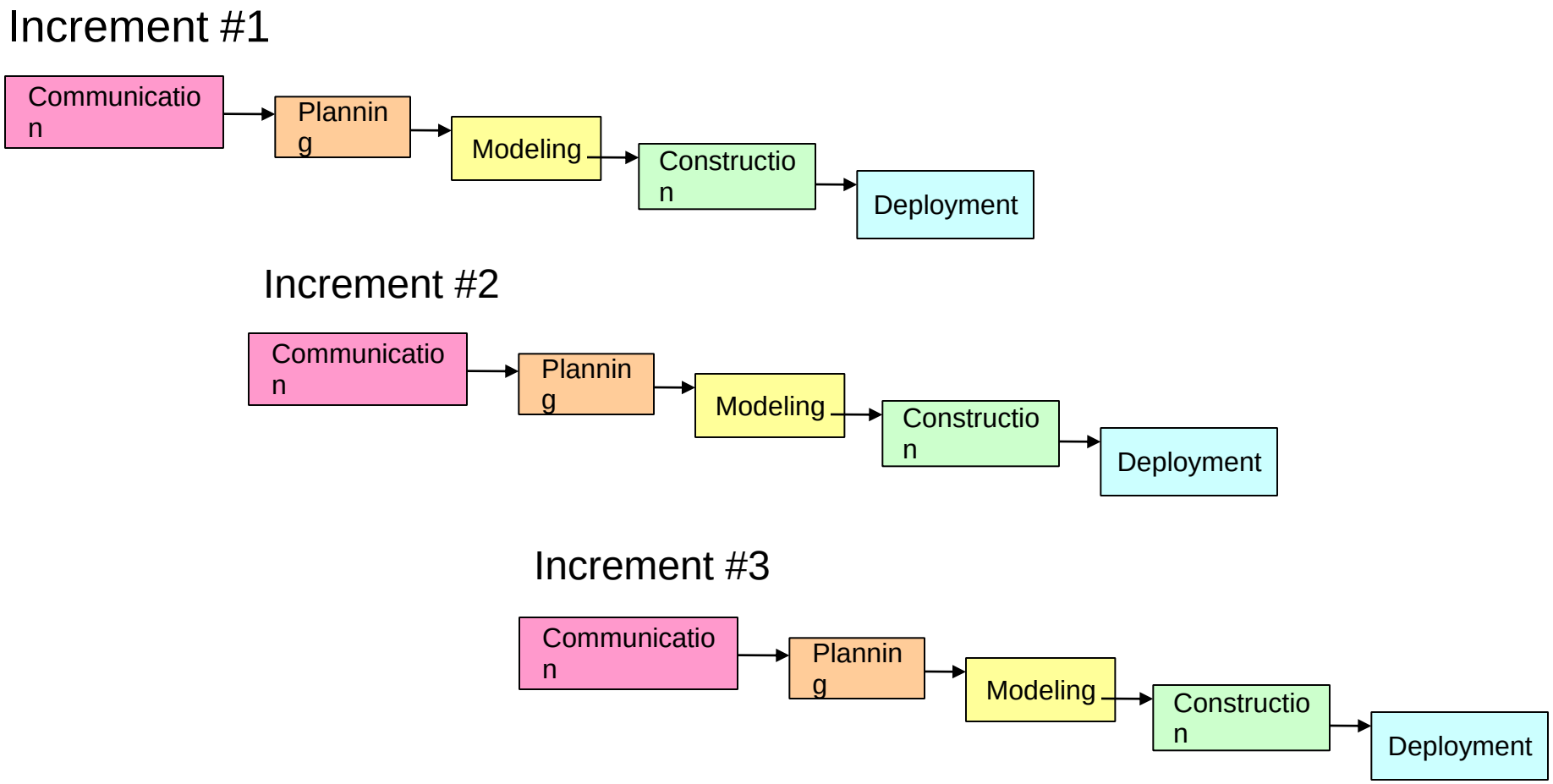
RAD (contd)

- Essential Aspects of RAD:
  - Methodology
    - Methods to be used in this model
  - People
    - Type of project team required
  - Management
    - Management's commitment in providing faster and timely increment
  - Tools
    - Computerized tools

RAD (contd)

- Difficult to manage for large projects
- Commitment of customer and the development team
- System cannot be modularized
- High technical risks

Incremental Model (Diagram)



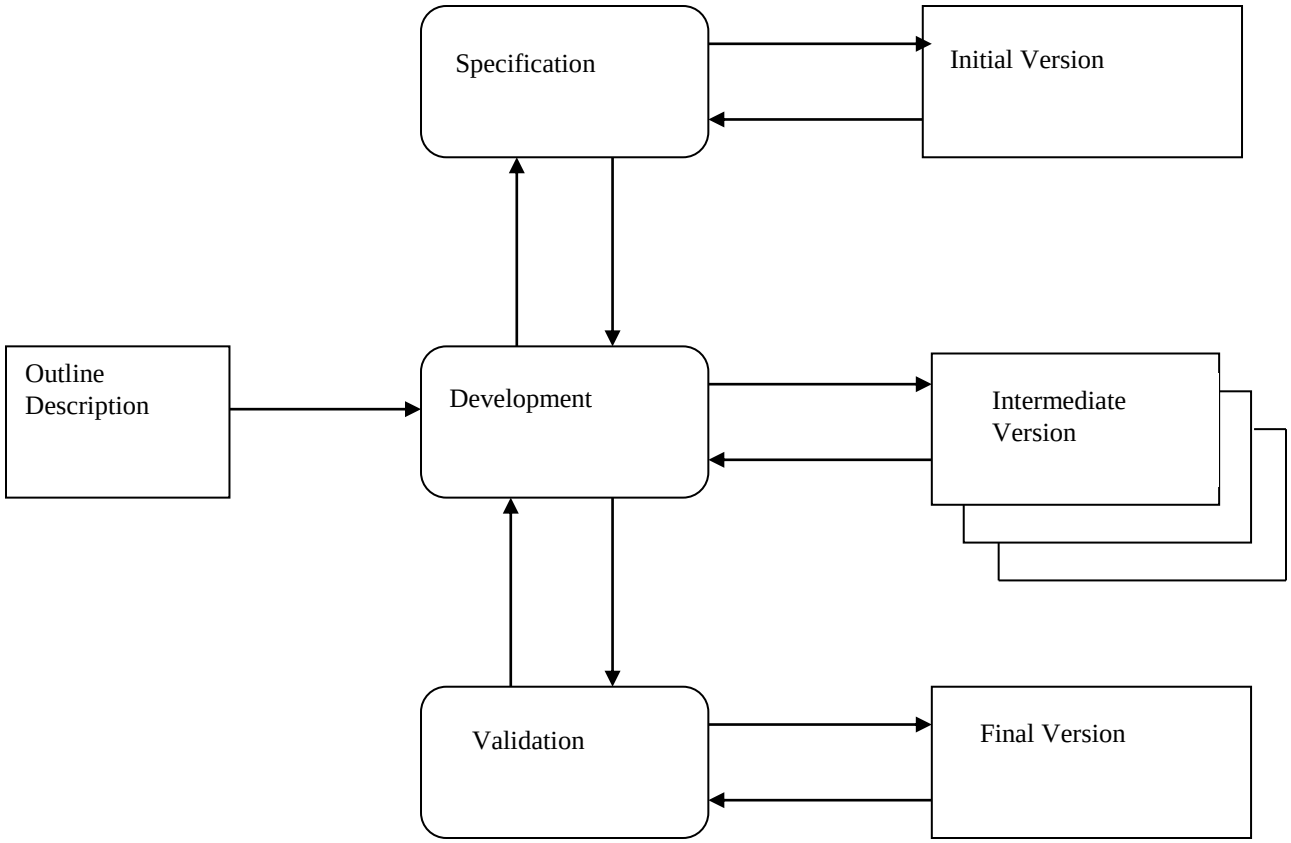
Incremental Model (Description)

- Used when requirements are well understood
- Multiple independent deliveries are identified
- Work flow is in a linear (i.e., sequential) fashion within an increment and is staggered between increments
- Iterative in nature; focuses on an operational product with each increment
- Provides a needed set of functionality sooner while delivering optional components later
- Useful also when staffing is too short for a full-scale development

Evolutionary Process Models

- They are also prescriptive process models
- Iterative in nature
- Based on the idea of
  - Developing an initial implementation
  - Exposing it to user comment
  - Refining
- Concurrent activities with feedback

Evolutionary Process Models (contd)



Evolutionary Process Models (contd)

- Types
  - Exploratory Development
    - Objective is to work with the customer to explore the requirements and deliver a final system.
  - Throw-away Prototyping
    - Objective is to understand the customer's requirements and develop better requirements definition.

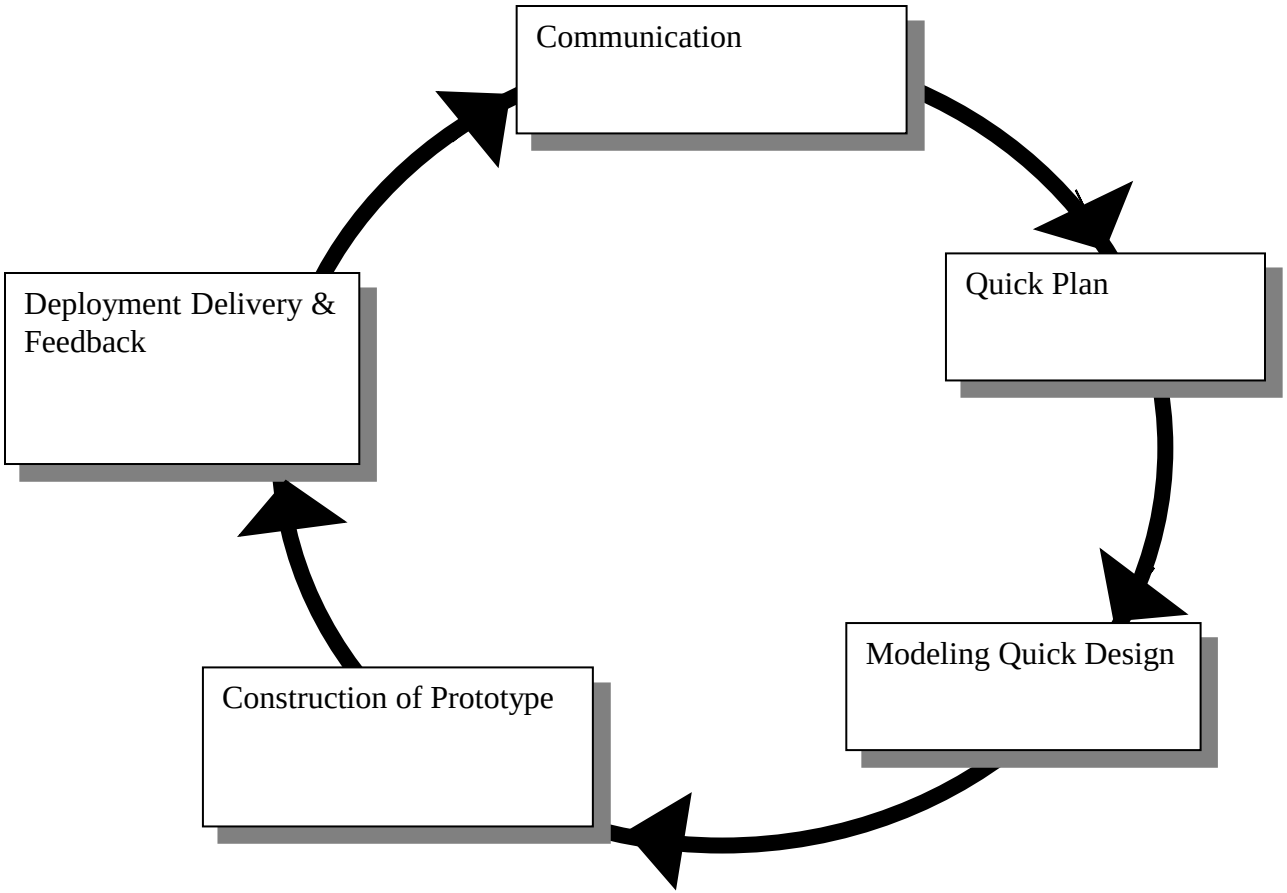
Evolutionary Process Models (contd)

- Examples:
  - Prototyping
  - Spiral Model
  - Concurrent Development Model

Prototyping

- When only general objectives are known
- Development issues are unknown
- Often used as a technique within other process models
- Better understand what is to be built when requirements are fuzzy

Prototyping (contd)



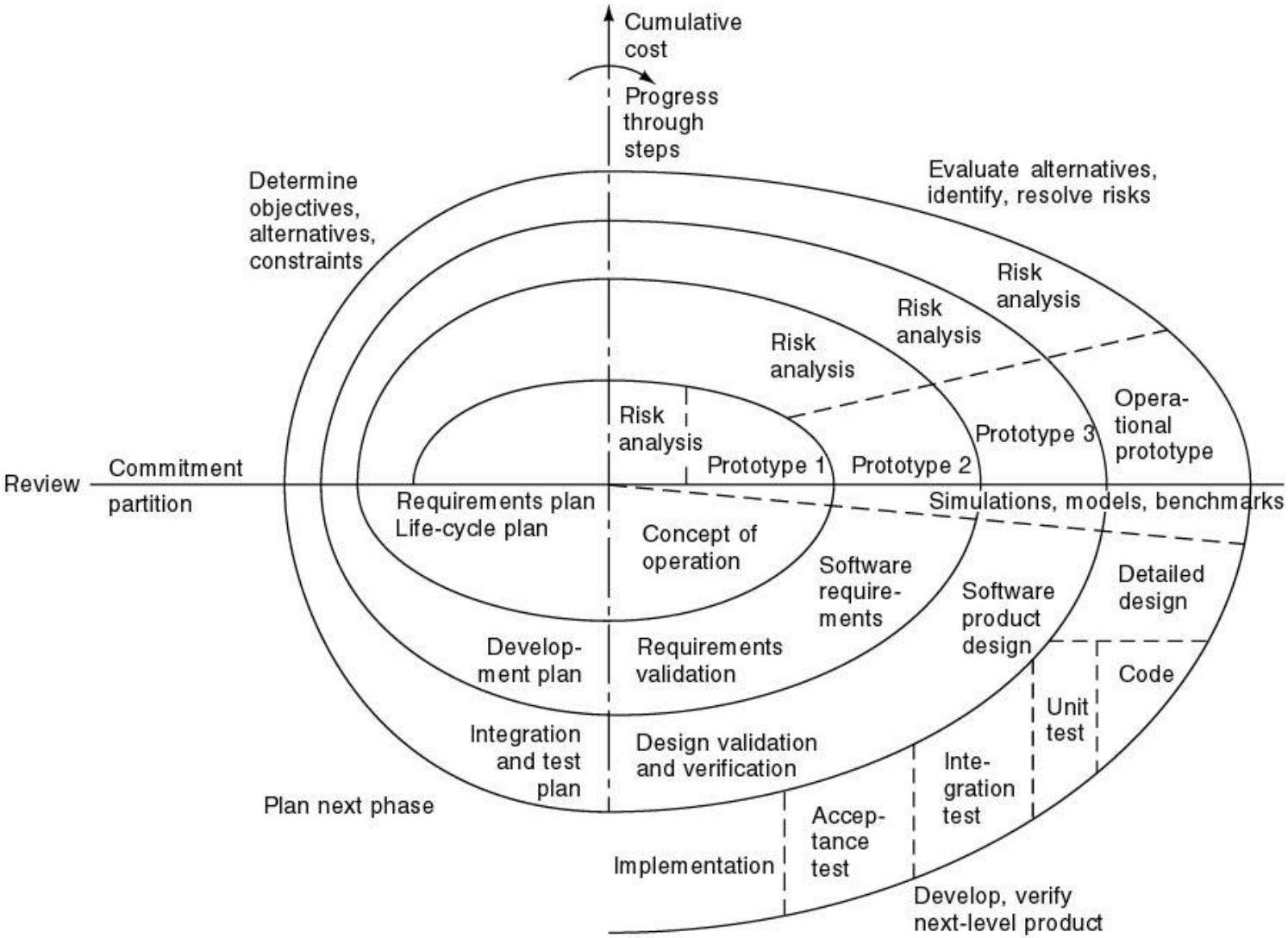
Prototyping (contd)

- Issues
  - Customer considers the prototype a working system
  - Less than ideal choice becomes an integral part of the system

Spiral Model

- Proposed by Boehm (1988)
- Process is represented as a spiral
- Each loop in the spiral represents a phase of the software process
- In early iterations the release might be a paper model or prototype

Spiral Model (contd)



Spiral Model (contd)

- An important aspect is the risk analysis preceding each phase
  - Risk- something which can go wrong
- If risk cannot be solved the project is terminated immediately
- Issues
  - Risk assessment needs expertise
  - Uncovered risks can result in later issues

Concurrent Development Model

- Also called Concurrent Engineering
- Tasks are done in parallel
- Each task or activity has its own state.
- States can be:
  - None
  - Under development
  - Under review
  - Done
  - Awaiting changes
  - Under revision
  - Under review
  - baselined

Concurrent Development Model (contd)

- All activities exist concurrently but reside in different states
- It defines a network of activities rather than a sequence of events
- Applicable to all types of software development

Conclusion

- Evolutionary process models focus on changing needs and customer satisfaction
- Certain issues are:
  - Project planning concerns in prototyping
  - Speed of evolution?
  - Balance between flexibility, extensibility, speed of development and quality

Difference between Evolutionary Models and Incremental Models

- Evolutionary Models:
  - All activities are in iterative fashion
- Incremental Models:
  - Requirements are established initially
  - development activities are iterative.

Component Based Development

- A software component is a system element offering a predefined service.
- Component-Based Development is a branch of the software engineering discipline, with emphasis on decomposition of the engineered systems into functional or logical components with well-defined interfaces used for communication across the components.

CBD (contd)

- Software applications are assembled from components from a variety of sources; the components themselves may be written in several different programming languages and run on several different platforms.

- COTS
  - Commercial off-the-shelf software components developed by vendors who offer them as products.

CBD (contd)

- Modeling
  - Identification of components.
- Available components are researched and evaluated for the domain.
- Component integration issues.
- Software architecture is designed.
- Components are integrated into architecture.
- Comprehensive testing.

CBD (contd)

- Conclusion
  - Software reusability
  - 70% reduction is software cycle
  - 84% reduction in software cost
  - Increased productivity

Formal Methods Model

- Mathematically based techniques for representing and analysis of software.
- Formal methods include
  - Formal specification
  - Specification analysis and proof
  - Transformational development
  - Program verification
- Hoare and Dijkstra

Formal Methods Model

- Reduces requirements errors as it forces a detailed analysis of the requirements
- Incompleteness and inconsistencies can be discovered and resolved
- Currently very time consuming and expensive
- Extensive training required
- Difficult to use this model to communicate with the customer.

Aspect Oriented Software Development

- Object-oriented programming (OOP) has become the mainstream programming paradigm where real world problems are decomposed into objects that abstract behavior and data in a single unit.

AOSD (contd)

- OOP encourages software re-use.

- Although OOP has met great success in modeling and implementing complex software systems, it has its problems.

AOSD (contd)

- Practical experience with large projects has shown that programmers may face some problems with maintaining their code because it becomes increasingly difficult to cleanly separate *concerns* into modules.

AOSD (contd)

- Aspect Oriented Programming (AOP) is a promising new technology for separating *concerns* specifically *crosscutting concerns* that are usually hard to do in object-oriented programming.

AOSD (contd)

- What is a concern?
- A concern is any piece of interest or focus in a program.
- separation of concerns (SoC) is the process of breaking a computer program into distinct features that overlap in functionality as little as possible.

AOSD (contd)

- Cross-cutting concerns are those parts of program that affect other concerns.

- Example:
  - Logging-logging affects all parts of the system that are being logged.
  - Logging thereby crosscuts all logged classes and methods.

AOSD (contd)

- AOP is a new technology for separating crosscutting concerns into single units called aspects.
- An aspect is a modular unit of crosscutting implementation. It encapsulates behaviors that affect multiple classes into reusable modules.

AOSD (contd)

- A distinct aspect-oriented process has not yet matured.
- May adopt characteristics of both spiral(evolutionary nature) and concurrent (parallel nature) process models.

Unified Process

- Came during the evolution period of Object Oriented software development methods (1990's)
- Contains features of OOA and OOD.
- UML- Unified Modeling Language
  - It was created to support the OO design and modeling.

UP (contd)

- Unified Process is an iterative and incremental software development process framework.
- Not only a process but an extensible framework that can be customized for specific organizations and projects.

UP (contd)

- Phases:
  - Inception
  - Elaboration
  - Construction
  - Transition
  - Production

UP (contd)

- Inception
  - Customer communication
  - Planning
    - Identify resources, assess risks, defines schedule
  - Business requirements are identified
    - In the form of use cases.
  - Rough architecture
    - A tentative outline of major sub-systems, functions and features that populate them.

UP (contd)

- Elaboration
  - Customer communication
  - Modeling activity
  - Expands the use cases.
  - Expands the architecture to:
    - Use case model, analysis model, design model, implementation model and deployment model.
  - Review plan and make modifications
    - Evaluate scope, risks, project delivery dates

UP (contd)

- Construction
  - Develop software components (that make the use cases operational).
  - Complete the analysis and design models.
  - Implement all functions and features for that increment.
  - Conduct unit testing for the components
  - Integrate components.

UP (contd)

- Transition
  - Create user manuals, guidelines, installation procedures.
  - Software is given to users for beta testing.
  - Get user feedback
  - The increment is now a useable software release.

UP (contd)

- Production
  - On going use of the software is monitored.
  - Provide support
  - Defect reports and request for changes are submitted and evaluated.

UP (contd)

- Phases do not occur in a sequence but are concurrent.
- One increment is in production and work on next increment is already started.

UP (contd)

- Conclusion
  - Recognizes the importance of customer communication.
  - Emphasizes architecture.
  - Iterative and incremental process flow.