Process Models

Perspective Models: The Waterfall

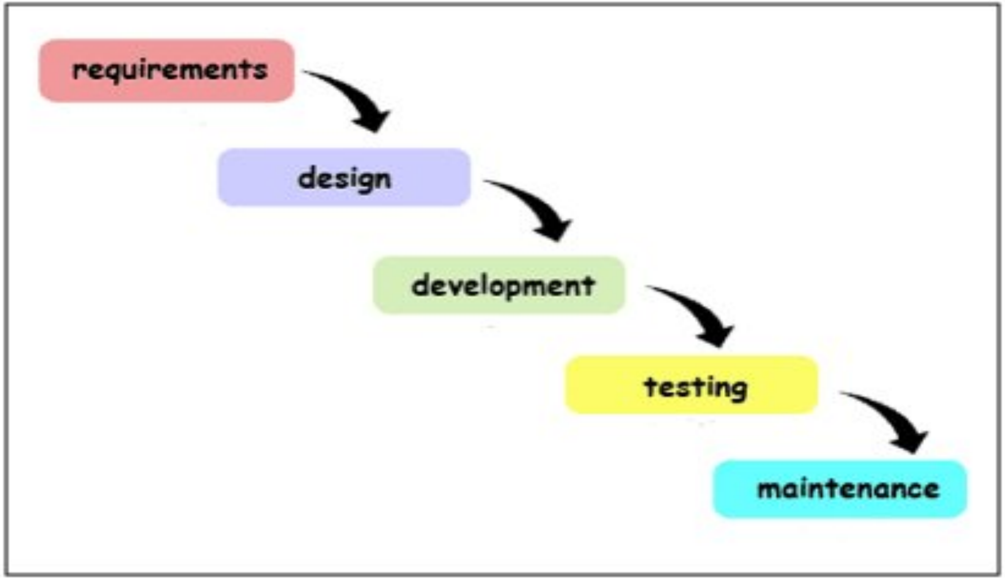
Incremental Models: Increment and RAD

Evolutionary Model: Prototype and Spiral

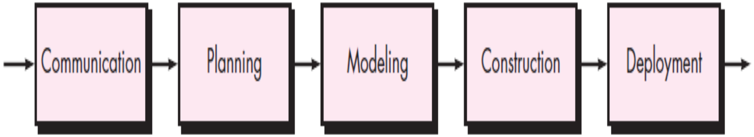
Specialized Process Models: Component, Formal Methods, AOSD

Unified Process Model

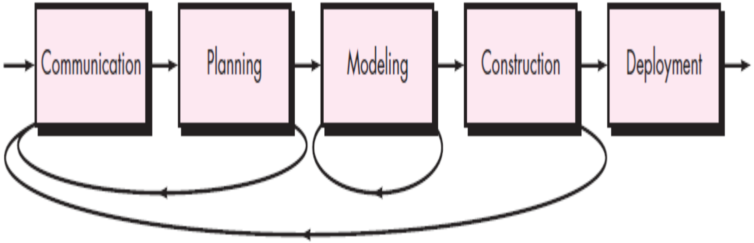
Software Development Life Cycle (SDLC) Phases



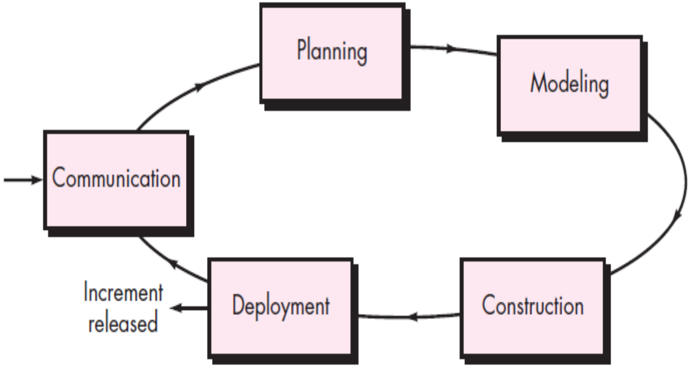
Process Flow



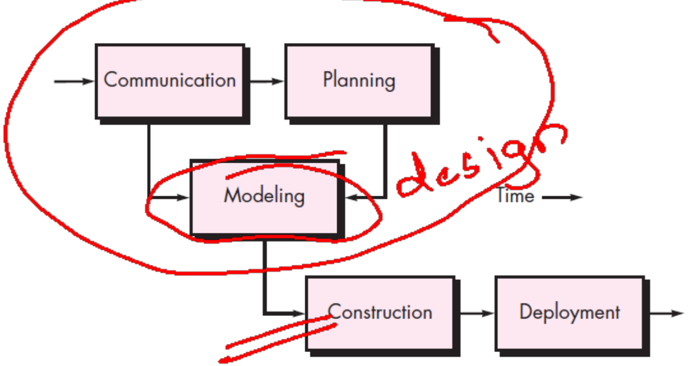
(a) Linear process flow



(b) Iterative process flow



(c) Evolutionary process flow



(d) Parallel process flow

Prescriptive Process Models

- Often referred to as "conventional" process models
- Prescribe a set of process elements
  - Framework activities ✓
  - Software engineering actions ✓
  - Tasks ✓
  - Work products ✓
  - Quality assurance and
  - Change control mechanisms for each project
- There are a number of prescriptive process models in operation
- Each process model also prescribes a *workflow* Sequential
- Various process models differ in their emphasis on different activities and workflow

The Waterfall Model

Sometimes called the *classic life cycle*

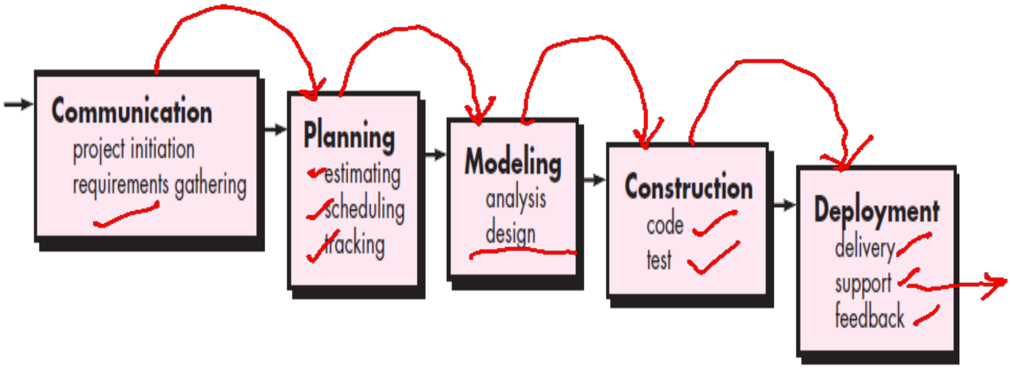
Suggests a systematic, sequential (or linear) approach to s/w development

The oldest paradigm for s/w engineering

Works best when –

- Requirements of a problem are reasonably well understood
- Well-defined adaptations or enhancements to an existing system must be made
- Requirements are well-defined and reasonably stable
- Technology is understood
- There are no ambiguous requirements
- Ample resources with required expertise are available freely
- The project is short

The Waterfall Model



Communication
project initiation
requirements gathering

Planning
estimating
scheduling
tracking

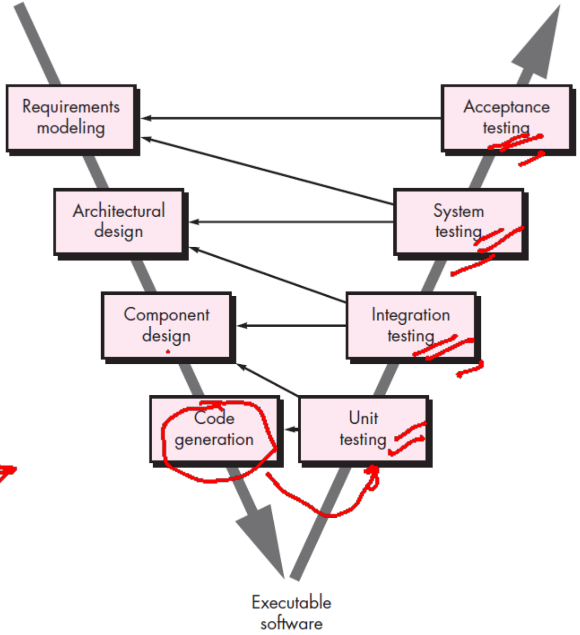
Modeling
analysis
design

Construction
code
test

Deployment
delivery
support
feedback

Basic Model

Variations in the Waterfall Model



Requirements modeling

Architectural design

Component design

Code generation

Unit testing

Integration testing

System testing

Acceptance testing

Executable software

The Waterfall Model - Problems

Real projects rarely follow the sequential flow

- Accommodates iteration indirectly
- Changes can cause confusion

It is often difficult for the customer to state all requirements explicitly

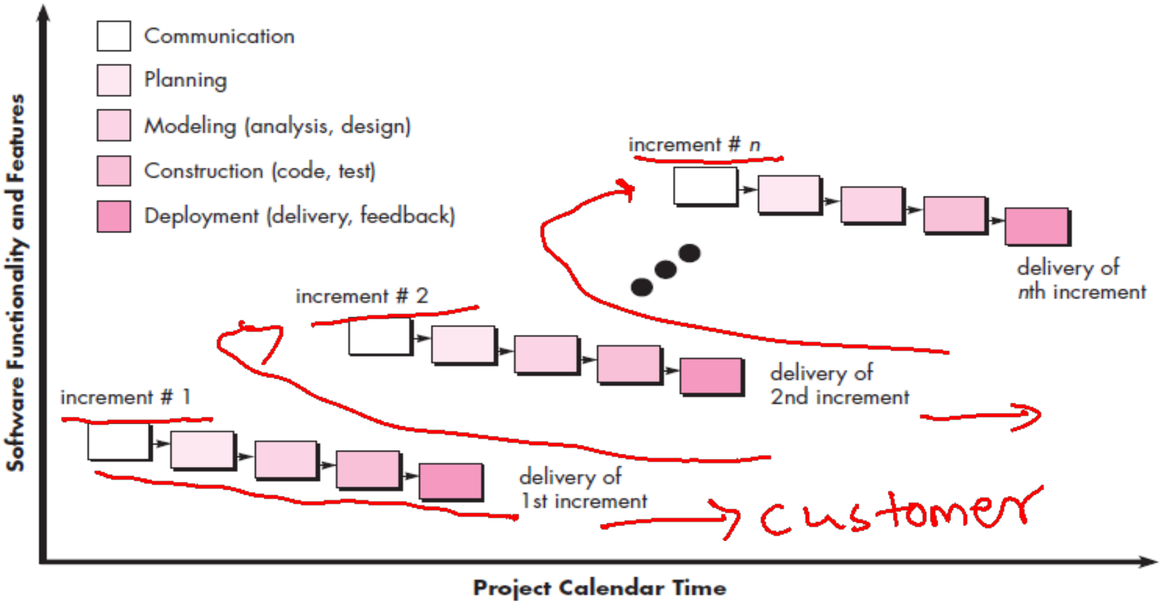
- Has difficulty accommodating the natural uncertainty that exists at the beginning of many projects

The customer must have patience

- A working version of the program(s) will not be available until late in the project time-span
- A major blunder, if undetected until the working program is reviewed, can be disastrous

Leads to “blocking states” for team members

Incremental Process Models


Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
increment # n
delivery of nth increment
increment # 2
delivery of 2nd increment
increment # 1
delivery of 1st increment
customer
Software Functionality and Features
Project Calendar Time

Incremental Process Models

Combines elements of the waterfall model applied in an iterative fashion

Each linear sequence produces deliverable “increments” of the software

The first increment is often a *core product*

The core product is used by the customer (or undergoes detailed evaluation)

Based on evaluation results, a plan is developed for the next increment

Incremental Process Models

→ Basic product

The incremental process model, like prototyping and other evolutionary approaches, is iterative in nature

But unlike prototyping, the incremental model focuses on the delivery of an operational product with each increment

Particularly useful when

- Staffing is unavailable
- This model can be used when the requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some details can evolve with time.
- There is a need to get a product to the market early.
- A new technology is being used
- Resources with needed skill set are not available → increment
- There are some high-risk features and goals.

Increments can be planned to manage **technical risks**

The RAD Model

Rapid Application Development is Incremental Process Model

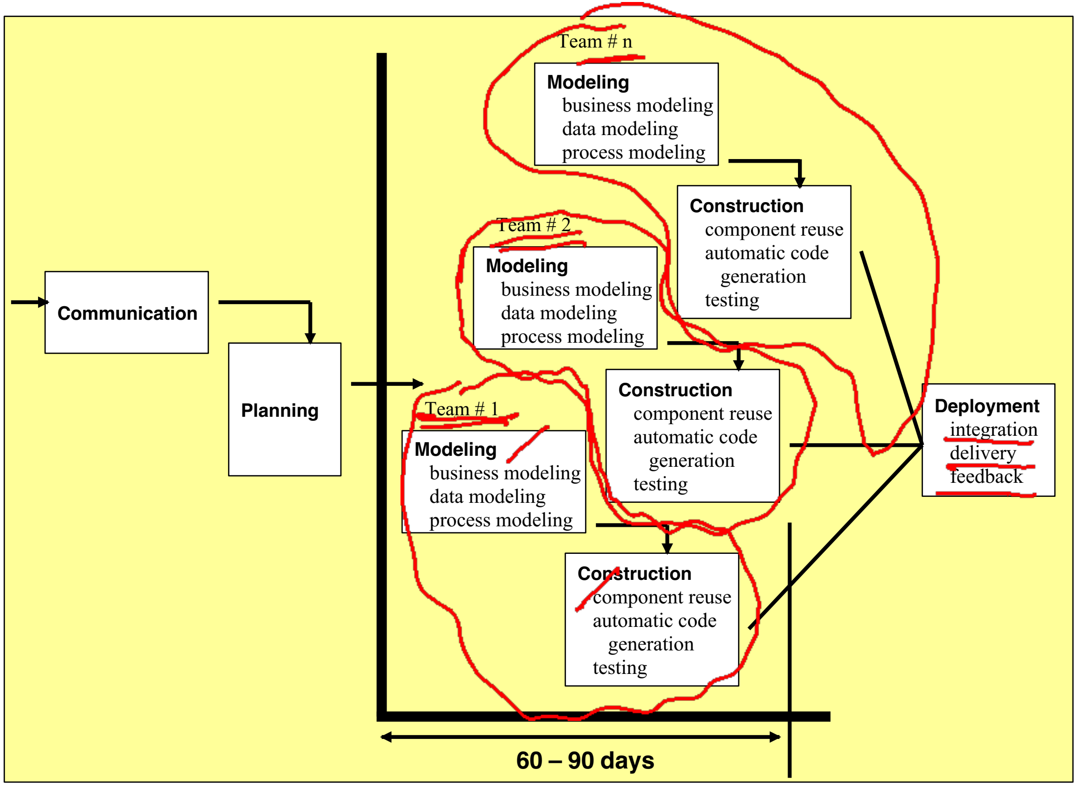
Emphasizes on short development cycle

A “high speed” adaptation of the waterfall model

Uses a component-based construction approach → Reusability

May deliver software within a very short time period (e.g. , 60 to 90 days) if requirements are well understood and project scope is constrained

The RAD Model



The RAD Model

The time constraints imposed on a RAD project demand "**scalable scope**"

The application should be modularized and addressed by separate RAD teams

Integration is required

Particularly useful when:

- RAD should be used when there is a need to create a system that can be modularized in 2-3 months of time.
- It should be used if there's high availability of designers for modeling and the budget is high enough to afford their cost along with the cost of automated code generating tools.
- RAD SDLC model should be chosen only if resources with high business knowledge are available and there is a need to produce the system in a short span of time (2-3 months).

The RAD Model - Drawbacks

For large, but scalable projects, RAD requires sufficient human resources

RAD projects will fail if developers and customers are not committed to the rapid-fire activities

If a system cannot be properly modularized, building the components necessary for RAD will be problematic

If high performance is an issue, and performance is to be achieved through tuning the interfaces to system components, the RAD approach may not work

RAD may not be appropriate when **technical risks are high**

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 is unrealistic

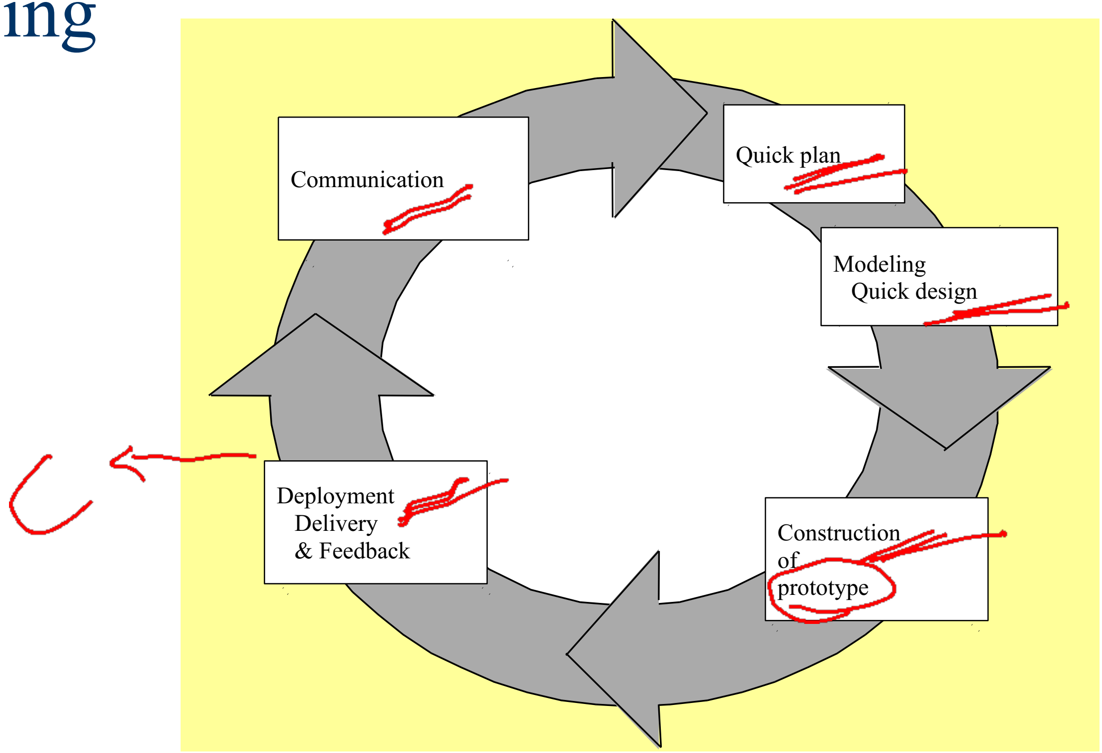
Evolutionary models are iterative

Prototyping

Customer defines general objectives but not sure with detailed input, processing and output.

This model assists the software engineer and the customer to better understand what to be built when requirements are fuzzy.

Prototyping



Prototyping - Problems

Customers may press for immediate delivery of working but inefficient products

The developer often makes implementation compromises in order to get a prototype working quickly

R&D

The Spiral Model

Couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model

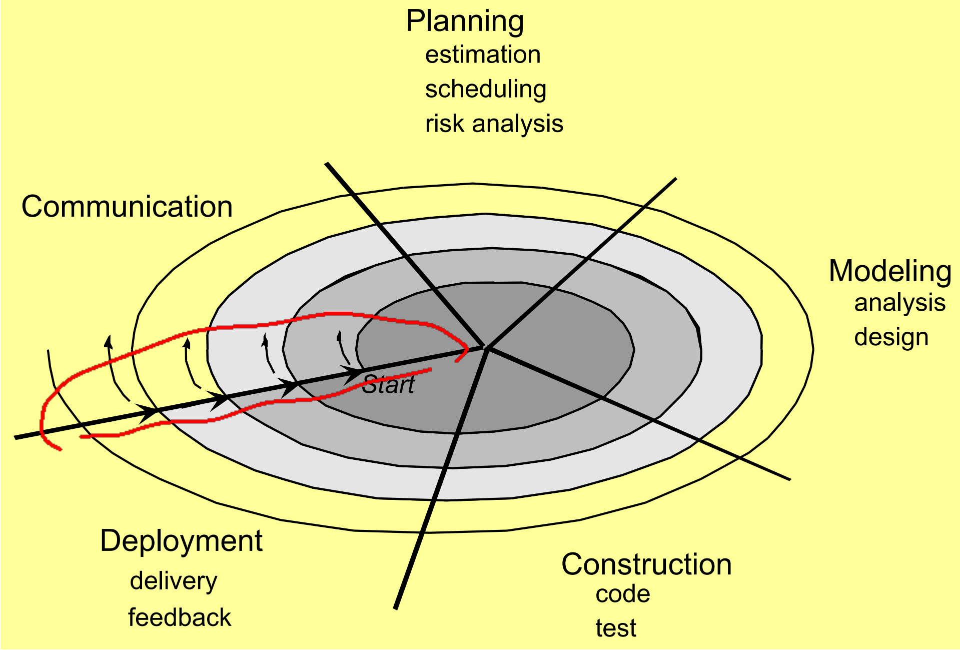
It provides the potential for rapid development of increasingly more complete versions of the software

It is a *risk-driven process model* generator

It has two main distinguishing features

- Cyclic approach
  - Incrementally growing a system's degree of definition and implementation while decreasing its degree of risk
- A set of *anchor point milestones*
  - For ~~ensuring stakeholder commitment~~ to feasible and mutually satisfactory system solution

The Spiral Model



The Spiral Model

Unlike other process models that end when software is delivered, the spiral model can be adapted to apply throughout the life of the computer s/w

The circuits around the spiral might represent

- Concept development project ✓
- New Product development project ✓
- Product enhancement project ✓

The spiral model demands a direct consideration of technical risks at all stages of the project

Particularly useful when

- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex
- New product line
- Significant changes are expected (research and exploration)

The Spiral Model - Drawbacks

It may be difficult to convince customers (particularly in contract situations) that the evolutionary approach is controllable.

It demands considerable risk assessment expertise and relies on this expertise for success.

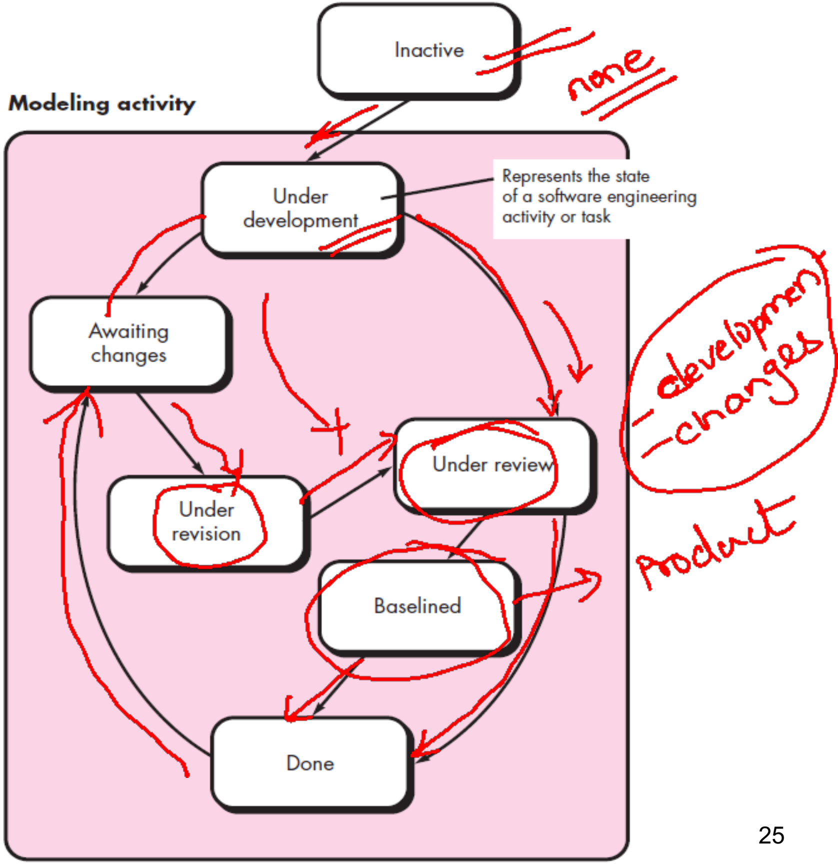
If a major risk is not uncovered and managed, problems will undoubtedly occur.

The Concurrent Development Model

- Sometimes called *concurrent engineering*
- Can be represented schematically as a series of framework activities, s/w engineering actions and tasks, and their associated states
- Defines a series of events that will trigger transitions from state to state for each of the s/w engineering activities, actions, or tasks
- Applicable to all types of s/w development
- Defines a network of activities
- Events generated at one point in the process network trigger transitions among the states

event, actions

The Concurrent Development Model



Weaknesses of Evolutionary Process Models

Uncertainty in the number of total cycles required

- Most project management and estimation techniques are based on linear layouts of activities

Do not establish the maximum speed of the evolution

Software processes should be focused on flexibility and extensibility rather than on high quality, which sounds scary

- However, we should prioritize the speed of the development over zero defects. Why?

Specialized Process Models

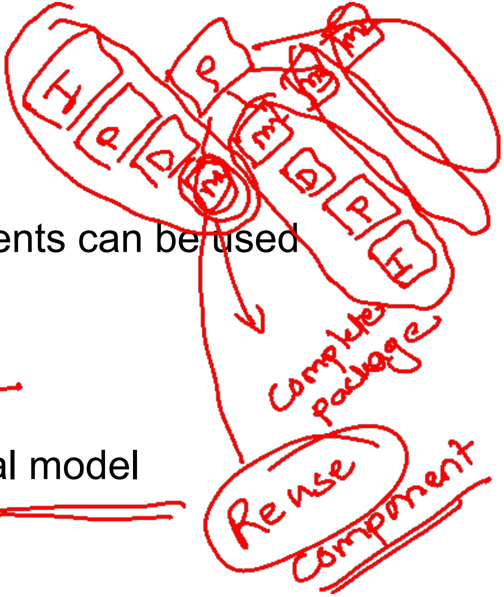
Take on many of the characteristics of one or more of the conventional models

Tend to be applied when a narrowly defined software engineering approach is chosen

Examples:

- Component-Based Development ✓
- The Formal Methods Model ✓
- Aspect-Oriented Software Development ✓

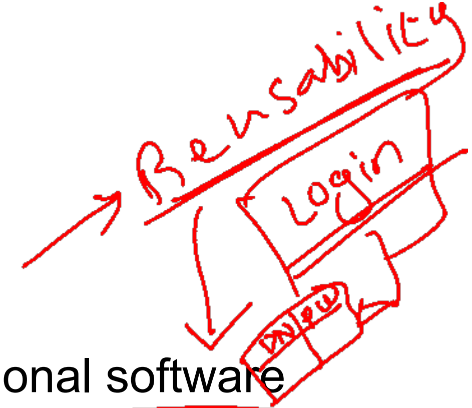
Component-Based Development



- Commercial off-the-shelf (COTS) software components can be used
- Components should have well-defined interfaces
- Incorporates many of the characteristics of the spiral model
- Evolutionary in nature

Component-Based Development

Candidate components should be identified first



Components can be designed as either conventional software modules or object-oriented classes or packages of classes

The Formal Methods Model

Encompasses a set of activities that leads to formal mathematical specification of computer software

Have provision to apply a rigorous, mathematical notation

Ambiguity, incompleteness, and inconsistency can be discovered and corrected more easily – not through *ad hoc* review, but through the application of mathematical analysis

Offers the promise of **defect-free** software

The Formal Methods Model – Critical Issues

The development of formal models is currently quite time-consuming and expensive

Extensive training is required

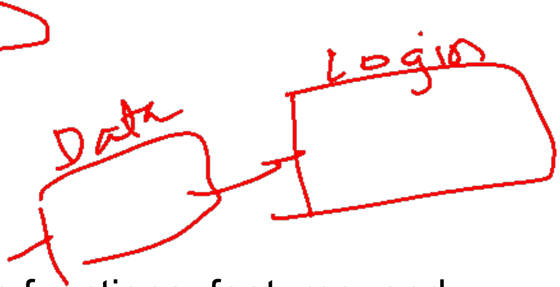
It is difficult to use the models as a communication mechanism for technically unsophisticated customers

Aspect-Oriented Software Development (AOSD)

Certain “concerns” – customer required properties or areas of technical interest – span the entire s/w architecture

Example “concerns”

- Security
- Fault Tolerance
- Task synchronization
- Memory Management



When concerns cut across multiple system functions, features, and information, they are often referred to as *crosscutting concerns*

Aspect-Oriented Software Development (AOSD)

Aspectual requirements define those crosscutting concerns that have impact across the s/w architecture

AOSD or AOP (Aspect-Oriented Programming) provides a process and methodological approach for defining, specifying, designing, and constructing aspects – “mechanisms beyond subroutines and inheritance for localizing the expression of a crosscutting concern”

A distinct aspect-oriented process has not yet matured

It is likely that AOSD will adopt characteristics of both the spiral and concurrent process models

The Unified Process (UP)

Rational Unified Process (RUP)

It is a use-case driven, architecture-centric, iterative and incremental software process

object-oriented

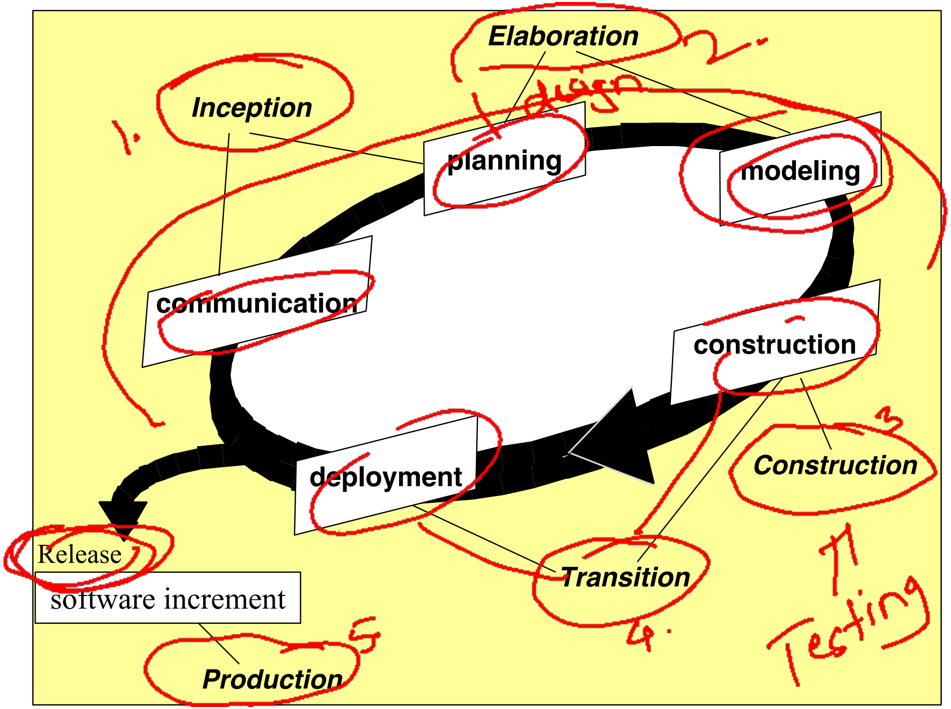
UP is an attempt to draw on the best features and characteristics of conventional s/w process models

Also implements many of the best principles of **agile software development**

UP is a framework for object-oriented software engineering using UML (Unified Modeling Language)

design

Phases of the Unified Process



Phases of UP - Inception

→ Use-cases

requirem

Encompasses both customer communication and planning activities

Fundamental business requirements are described through a set of preliminary use-cases

- A **use-case** describes a sequence of actions that are performed by an actor (e.g., a person, a machine, another system) as the actor interacts with the software

A rough architecture for the system is also proposed

Phases of UP - Elaboration

- Encompasses customer communication and modeling activities
- Refines and expands the preliminary use-cases
- 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 deployment model

UML

- In some cases, elaboration creates an “executable architectural baseline” that represents a “first cut” executable system

Phases of UP - Construction

Unit

Makes each use-case operational for end-users

As components are being implemented, **unit tests** are designed and executed for each

Integration activities (component assembly and **integration testing**) are conducted

Use-cases are used to derive a suite of **acceptance tests** ✓

Phases of UP - Transition

Software is given to end-users for **beta testing**

The software team creates the necessary support information –

- User manuals
- Trouble-shooting guides
- Installation procedures

At the conclusion of the transition phase, the software increment becomes a usable software release

Phases of UP - Production

Coincides with the deployment activity of the generic process

The on-going use of the software is monitored

Support for the operating environment (infrastructure) is provided

Defect reports and requests for changes are submitted and evaluated

Unified Process Work Products

Inception
- Vision document
- Initial use-case model

Elaboration
- Analysis model, design model

Construction
- Implementation model, deployment model, test model

Transition
- Delivered software, beta test reports, general user feedback

Distribution of effort…

How programmers spend their time

– Typing programs	13%
– Searching and Reading programs	16%
– Job communication	32%
– Others	39%

Programmers spend more time in reading programs than in writing them.

Writing programs is a small part of their lives.

Defects

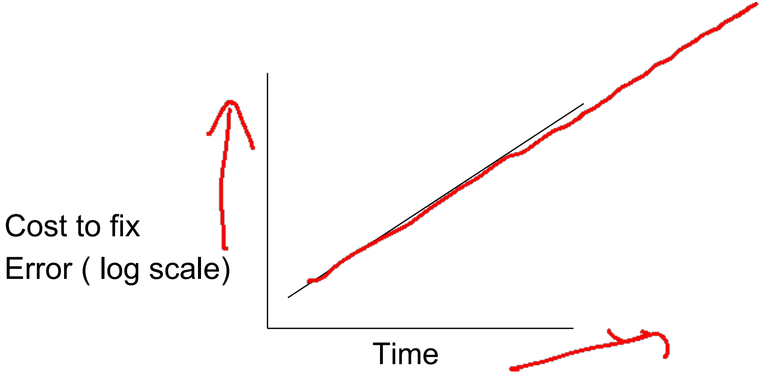
Distribution of error occurrences by phase is

- Req. - 20%
- Design - 30%
- Coding - 50%

Defects can be injected at any of the major phases.

Cost of latency: Cost of defect removal increases exponentially with latency time.

Defects…



Cheapest way to detect and remove defects close to where it is injected.

Hence must check for defects after every phase.

Factors to be considered for Software Development

User Satisfaction ✓

Time ✓

Quality factors ✓

Adaptable for changes ✓

Risk Management ✓

Evolution ✓