16 Aug

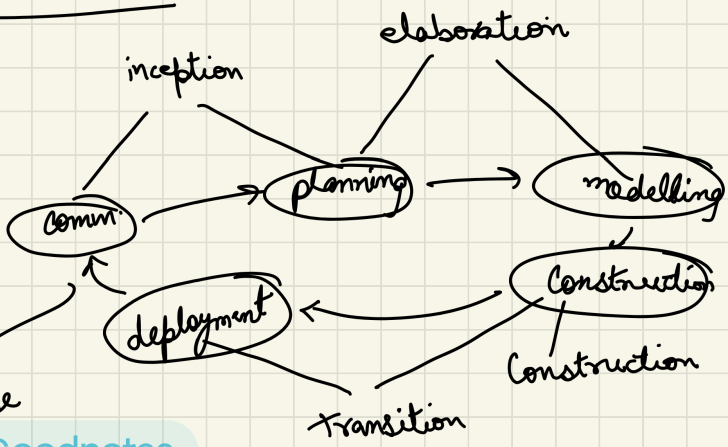
Unit - 2 (Process Models)

- Generic Process Model

- Spiral model : is similar to incremental development model for system with emphasis to risk management in each phase.
A software project repeatedly passes through these phases in iteration called spiral.
In this model, the radial dimension represents the cummulative cost incurred in doing the steps done so far. and the angular dimension represents the progress made in completing each cycle of the spiral.

(* situation based q)

- Unified Process Model :



Slide Set-1 (Intro to SE):

- Scope and Need of SE:

① Helps to reduce programming complexity by using two techniques: Abstraction and Decomposition

- Problems faced: (Evolving role of SE industry)

① takes too long
② finding errors before release
③ High cost
④ difficulty in measuring progress of software dev.

- factors contributing to software crisis:

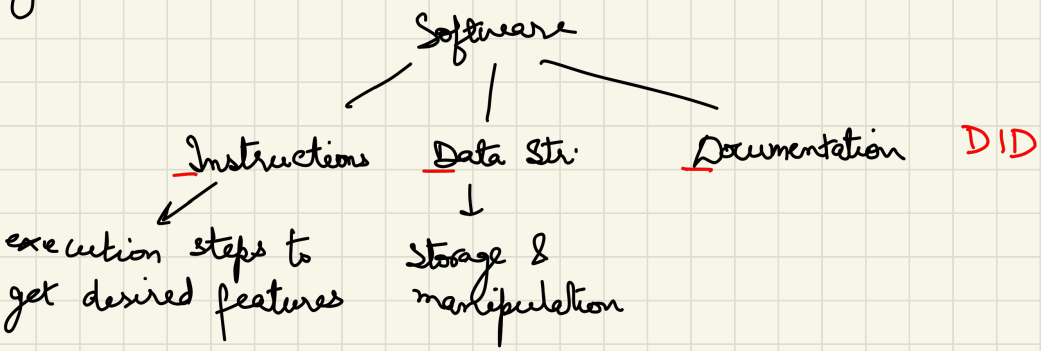
① Large problem
② Skill shortage
③ Lack of training
④ Low productivity improvement

- examples of software failures:

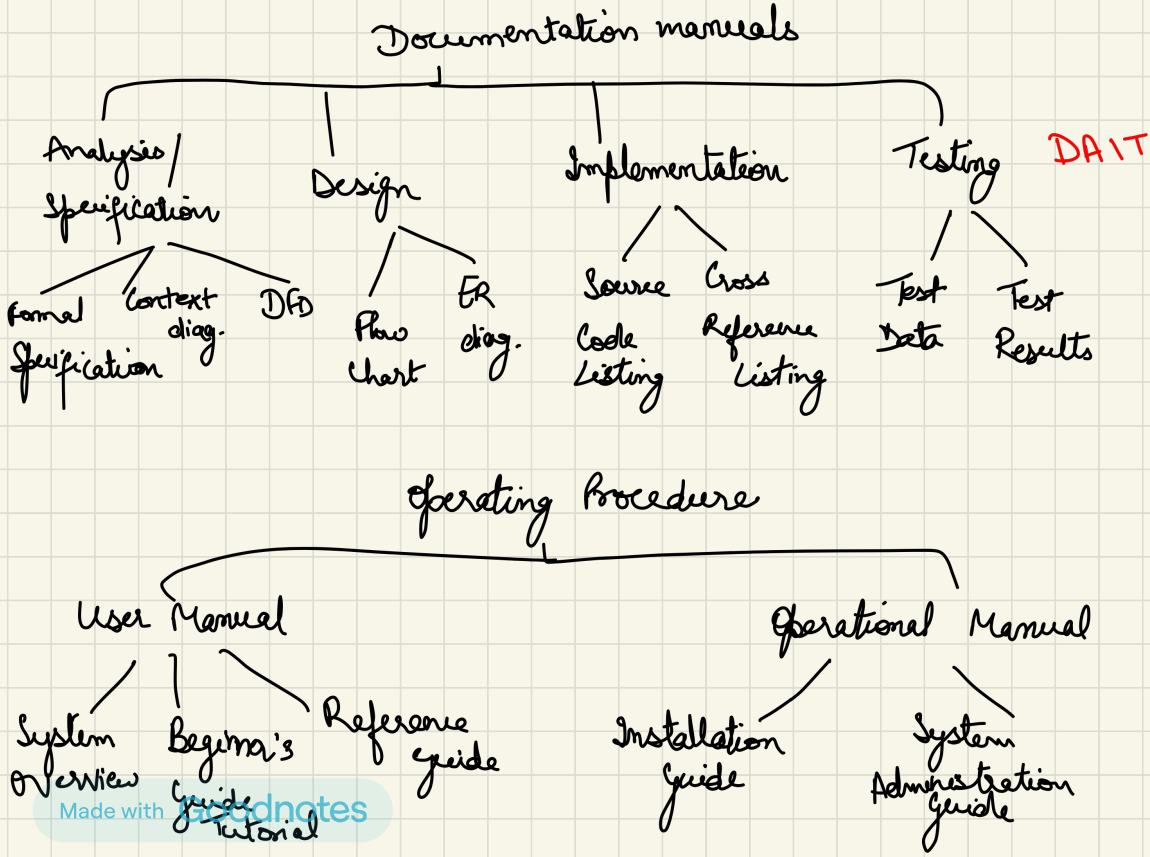
① Ariane 5 (1964 to 64)
② Patriot missile (timing error in system clock)
③ The space shuttle (Most scenario of shuttle, software crash)
④ Financial software (accounting errors)
(windows XP)

- What is Software :

Product that Software professionals build and than support over long term.



- Type of documentation manuals :



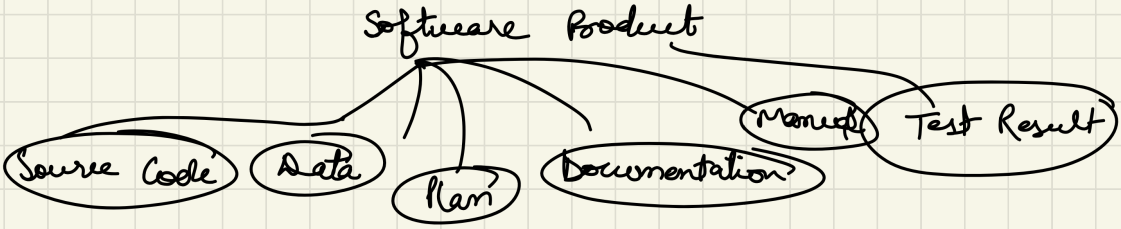
- **Software Products** :

Computer Program + Associated Documentation

- **Types of software products** :

① Generic (eg. word, excel)

② Tailor-made / Custom / Bespoke (eg. Commercial off the shelf)



- **Software Process** : Set of activities whose goal is development or evolution of software

SaVED

└ Specification, development, validation, evolution

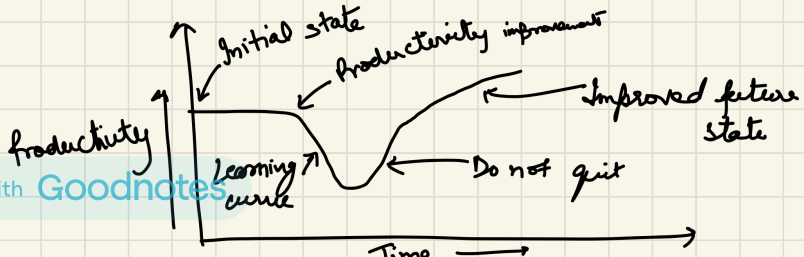
↳ (acc. to changing demands)

↓

(acc. to customer wants)

- **Why is it hard to improve software process** : KTMC

Lack of knowledge, not enough time, wrong motivation, insufficient commitments



- Software characteristics :

① It is developed
② doesn't wear out
③ is highly malleable

- Software Myths :

① Management Perspectives

- Confident about standards
- Infra: can help.
- Software Specialist can help.
- Software is easy to change

② Customer Perspectives

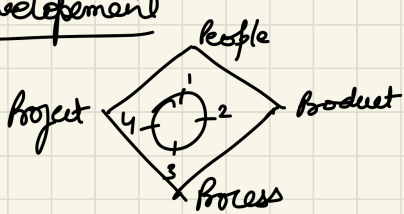
- vague requirements are enough
- Software demonstration
- Software with more features is better.
- Software can work right the first time.

③ Developer Perspectives

- quality assessment at end
- Only deliverable is tested code
- Aim is to develop working programs

- Role of Management in Software Development

① Effective delivery of 4 P's
People, Process, Product, Project



Slide Set-2 (Intro to SE):

- Soft. engineers should:

① adopt sys. & org. approach
② choose appropriate tools and techniques → Constraints, resources, problem

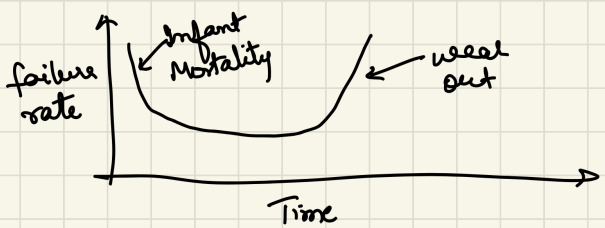
- Why SE:

① Software dev. is hard
② Distinguishing easy from hard system
③ Experience with easy system is misleading — Bridge building

- Attributes of good software: MUDE

Maintainability, Dependability, Efficiency, Usability

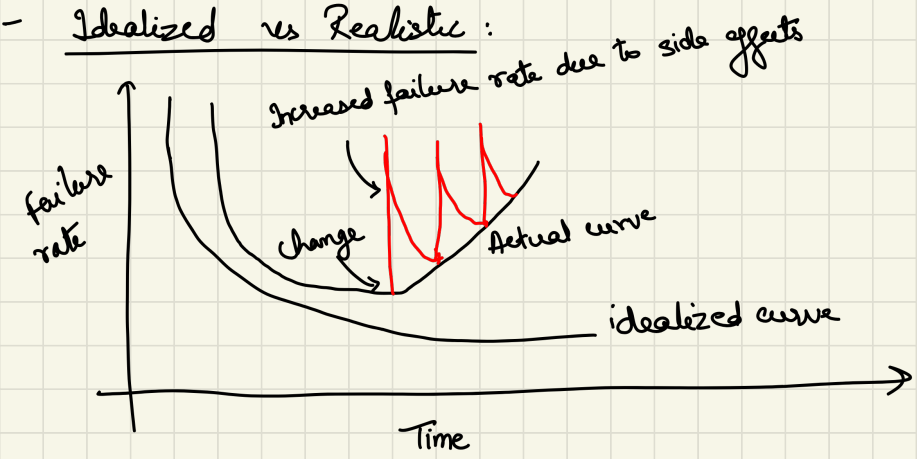
- failure curve for hardware:



- Software Applications: PEER SAW B

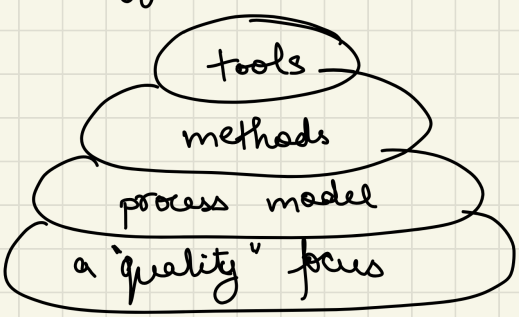
① System Software (compiler, editor)
② Real-time Software
③ Business software (MIS)
④ Engg. & scientific software (Numerical algo)
⑤ Embedded Software (Read only memory)
⑥ Personal Comp. Software
⑦ Web based Software
⑧ AI software (Non-Numeric Algo)

- Idealized vs Realistic :

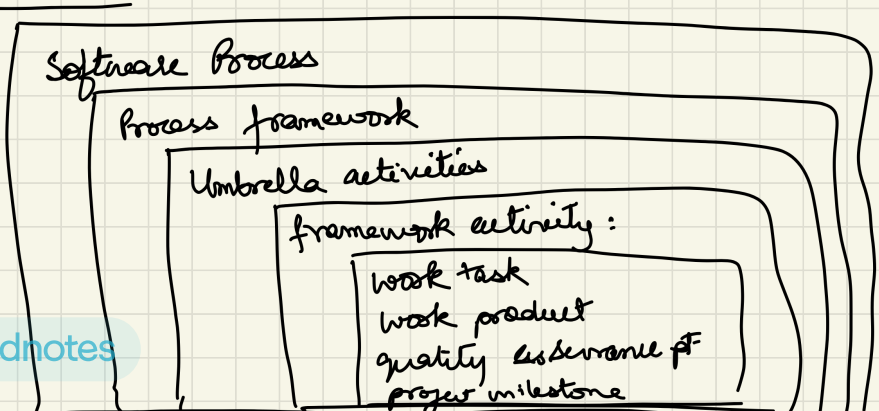


Slide Set-3 (Process Model):

- Layered technology (SE):



- Generic Process Model :

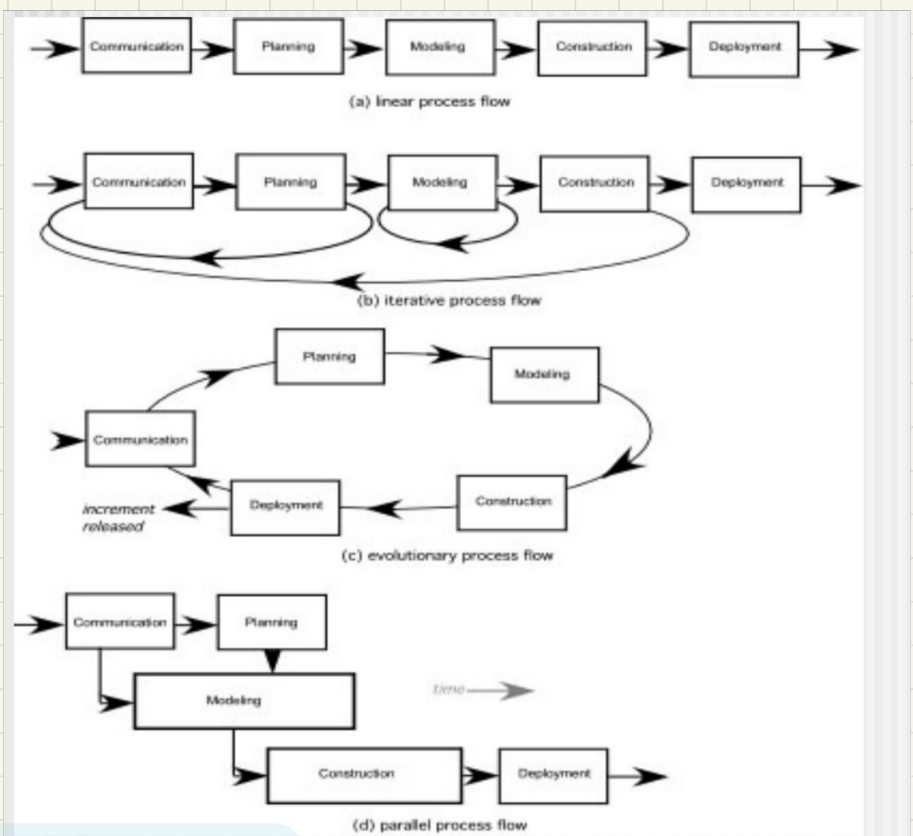


framework activity : Communication, Planning, Modeling, Const:, Deployment –

Analysis, Req. of Design, Code, Testing

Umbrella activity : software project management, formal technical reviews, Soft. quality assurance, Soft. config. management, work prod. prep. & production, reusability management, measurement, risk management

Process Flow : PILE



- Process Assessment and Improvement :

① SCAMPI : five phase : initiating, diagnosing, establishing, acting, learning IDEAL

(standard CMMI Assessment Method for Process Improvement)

② CBAIPI : provides diagnostic technique for assessing relative maturity of a software org., uses SEI CMM

(CMM Based Appraisal for Internal Process Improvement)

③ SPICE : standard that defines set of requirements for software process assessment

④ ISO : can be applied to any org. that wants to improve overall quality of products/services it provides

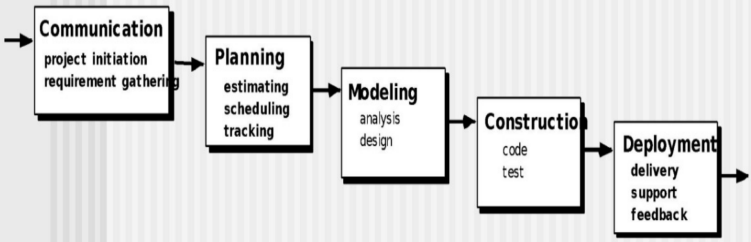
- Phases of software product :

① Feasibility Study
② Requirement analysis and Specification
③ Design
④ Coding
⑤ Testing
⑥ Maintanence

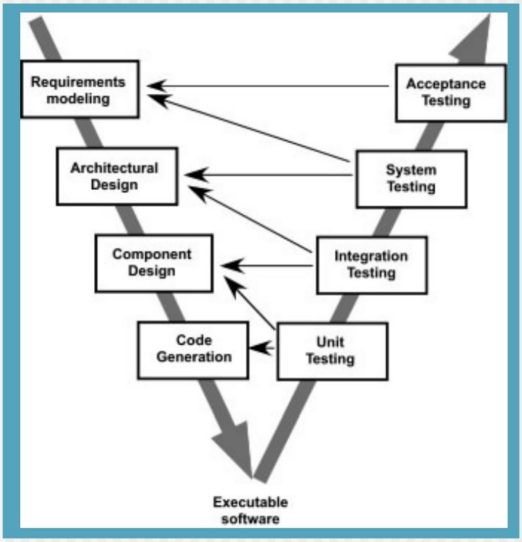
Life Cycle model helps the project managers to track the progress.

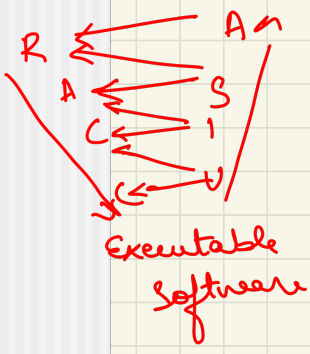
- Specific models :

The Waterfall Model

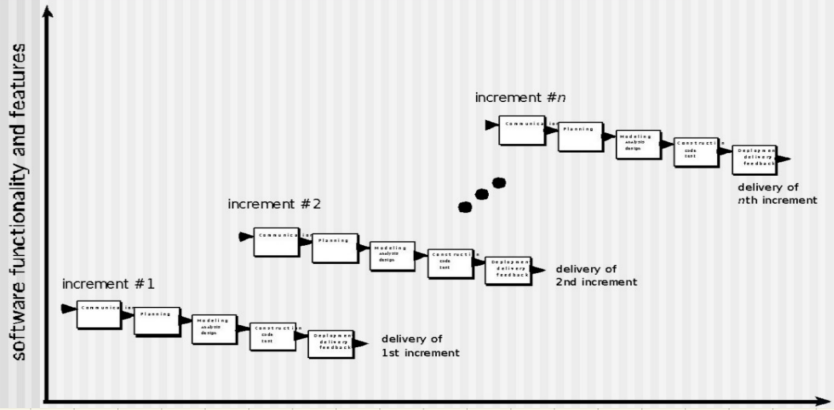


The V-Model

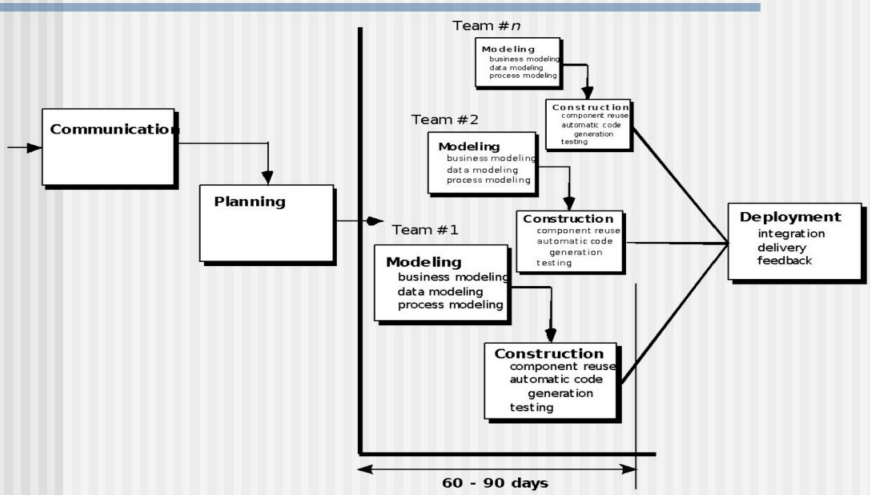




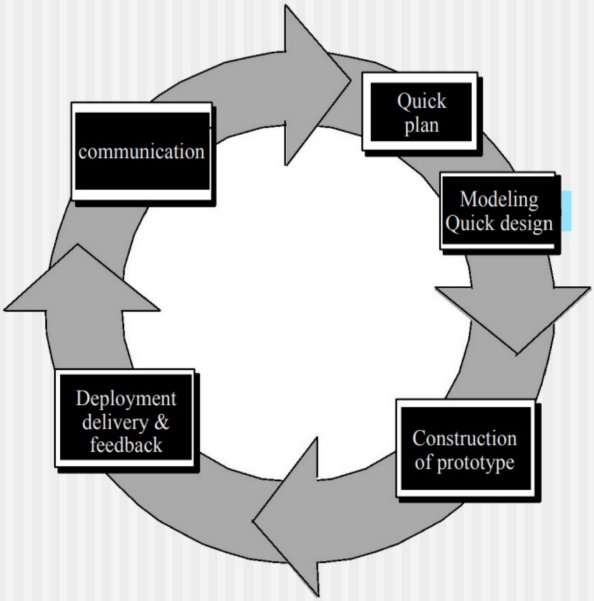
The Incremental Model



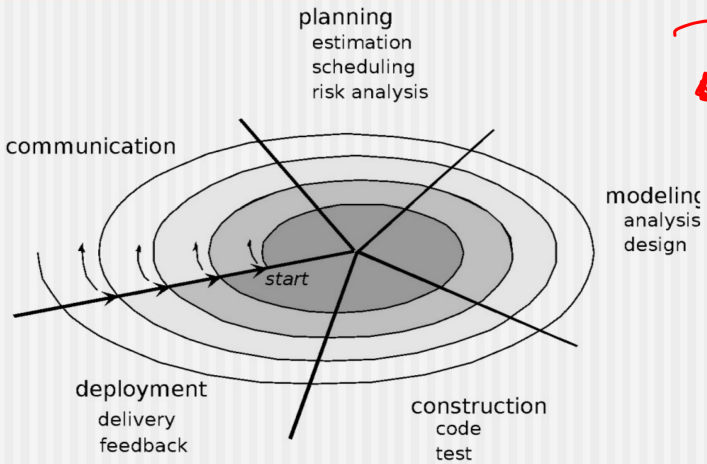
The RAD Model



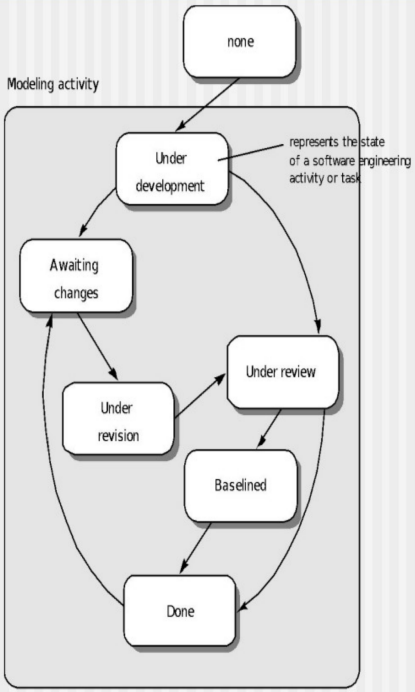
Evolutionary Models: Prototyping



Evolutionary Models: The Spiral



Evolutionary Models: Concurrent



- Iterative Vs Incremental Model :

Iterative :

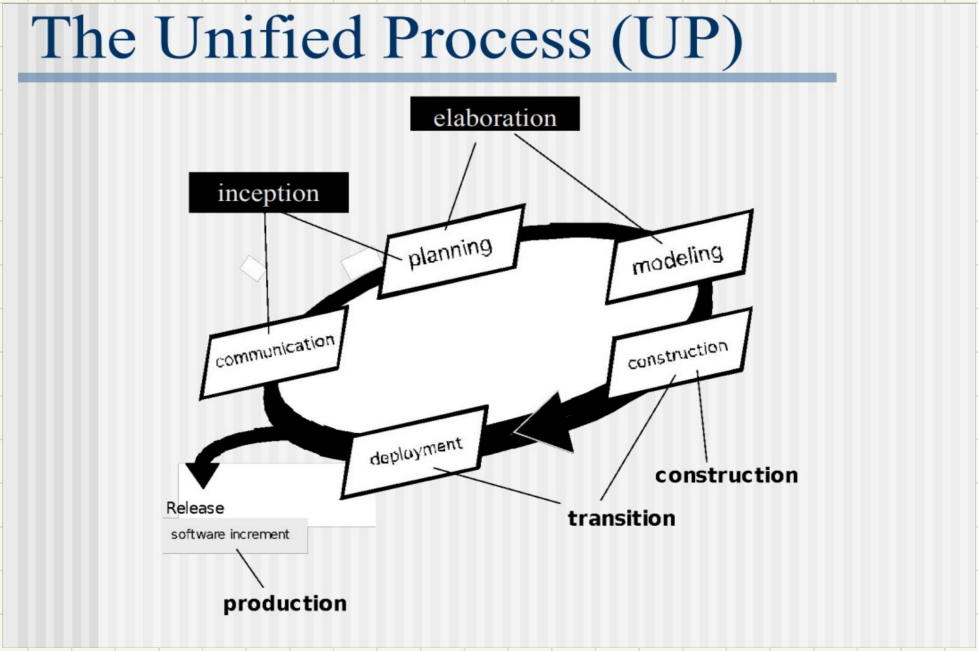
1. Develop through repeated cycles
2. Start simple, expecting to change
3. used to find right soln (fair early)
4. used to improve candidate soln

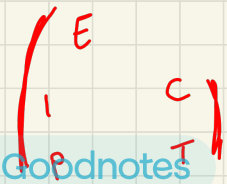
Incremental :

Develop smaller portions at a time
gradually build up functionality, allows value to be delivered early

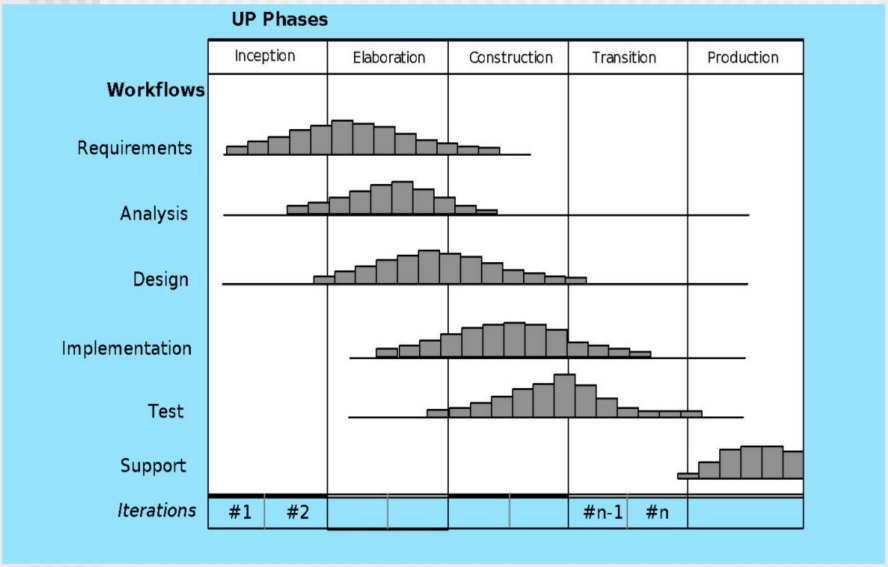
Other process models: CAUF

1. Component based development — for reuse
2. formal methods — mathematical specification of requirements
3. AOSD — Approach for defining, specifying, designing, constructing aspect
4. Unified process — "use case driven, architecture centric, iterative, incremental" with UML





UP Phases



UP work products:

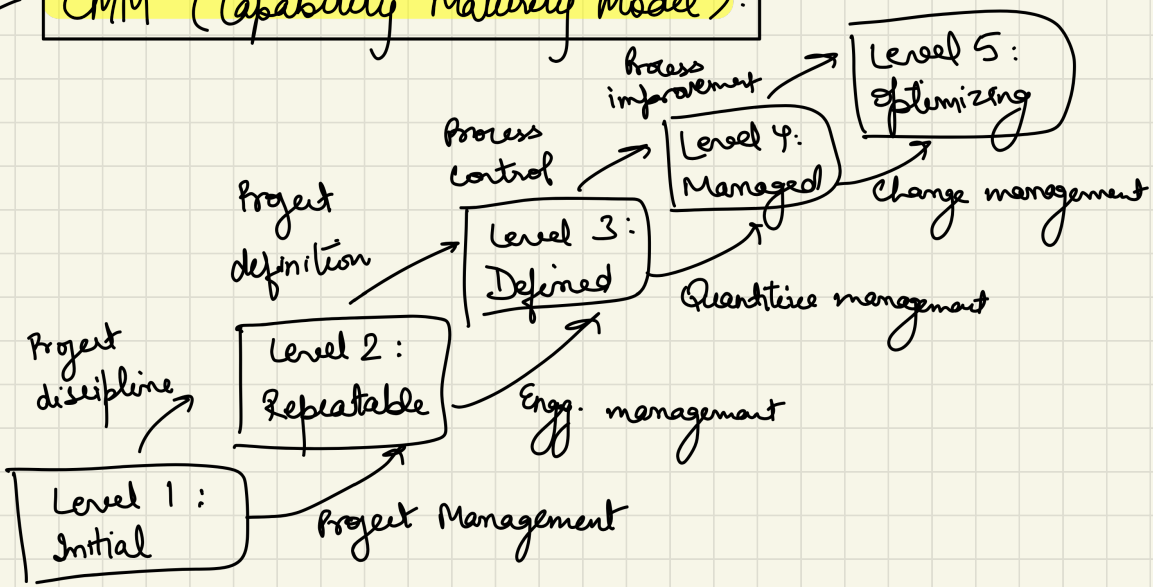
Inception phase: Vision document, initial use case model, initial project glossary, Initial business case, initial risk assessment, Project plan, Business model

Elaboration phase: Use case model, Supp. req. (non-f*), Analysis model, Soft. architecture, Executable architecture, Preliminary design model, Revised Risk list, Project plan (iteration plan, adapted workflow, milestone, technical product), Preliminary user manual

Construction phase : Design model, Soft. components, Integrated software, Test plan and procedure, Support doc, (user manual, installation guide) Test case

Transition phase : Delivered software increment
Beta test reports
General user feedback

CMM (Capability Maturity Model):

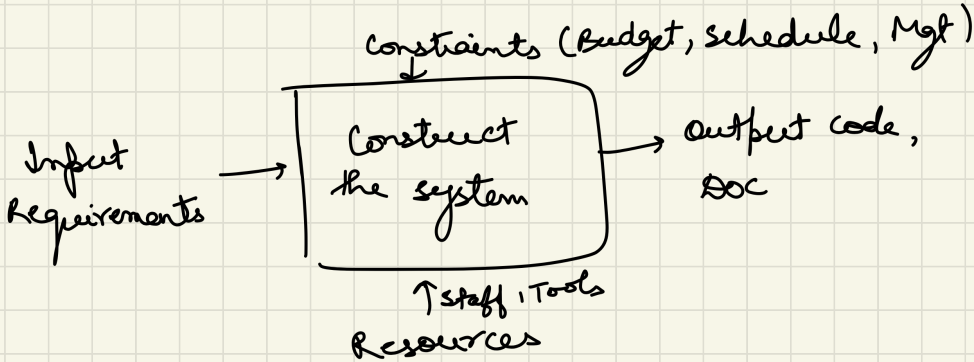


Level 1 : ad hoc

① inputs are ill-defined while outputs are defined
② transition from input to output is undefined & uncontrolled
③ visibility is nil and measurement difficult
④ compose more str.

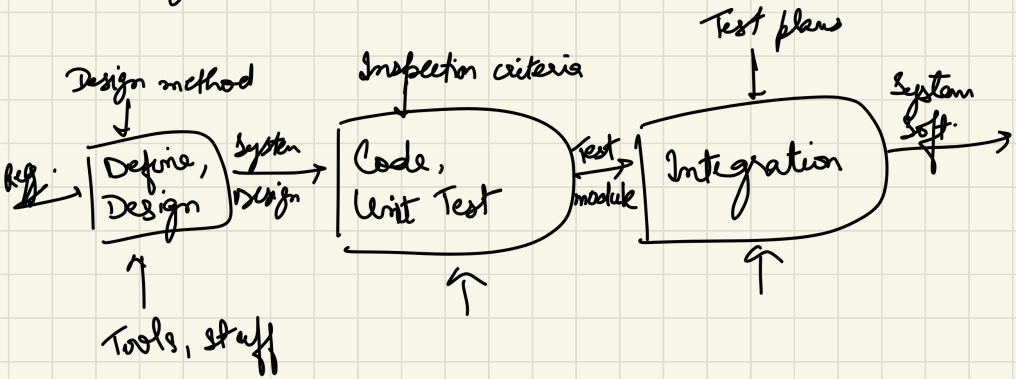
Level 2 : repeatable – process dependent on individuals

- diff. projects have diff. models
- basic processes used to track cost and schedule
- Repeatable only similar projects
- Clients interact at well defined pts



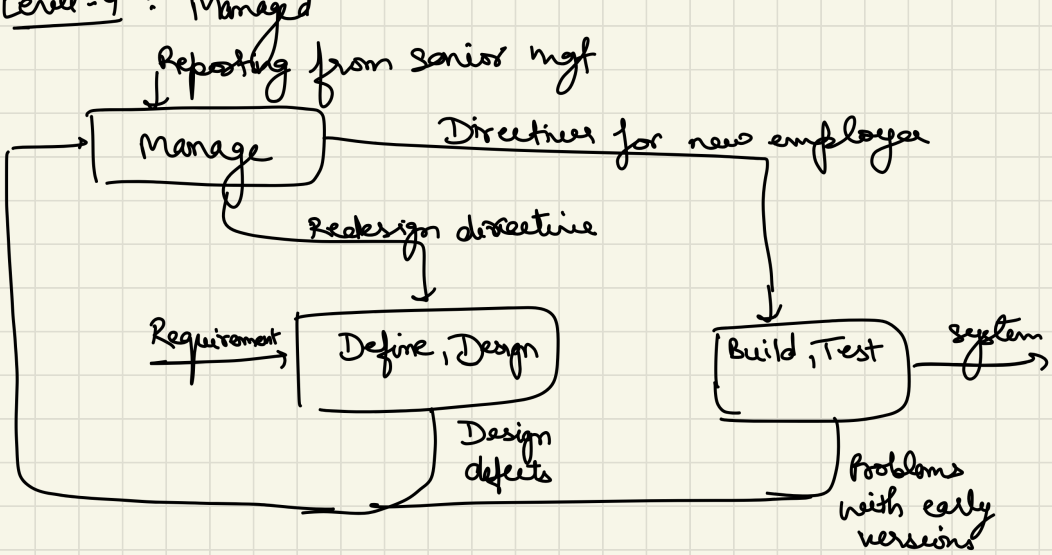
Level 3 : Defined

- Reduce excessive individual dependence
- Lack of predictable outcomes



Diff. b/w level 2 & 3 : providing process visibility

Level-4 : Managed



Diff. b/w Level 3 & 4 : measurement characteristic of organ process and of interaction among and across major activities (Agile method)

- Define metrics for activities and deliverables

detailed measurement

Control

process and product with qualified quality predictability

Level 5 : Optimizing

- Defect prevention
- Process change management
- Technology change management
- Improvement

Slide Set 4 (Requirements Engg):

% of tar. dep. ; No. of target systems

- Non-functional Req: Speed, Size, Robust, Portability, Reliability, Ease of use (PRRESS)
  - Reliability → failure/availability; (mean time to failure); (time to restart after failure)

- User Req.: are defined using natural language, tables and diagrams.
  - → should describe fn and non-fn req. so that they are understandable by system users who don't have detailed technical domain knowledge.

- Problems with natural language:
  - lack of clarity
  - Requirements confusion (fn & non fn)
  - Requirement amalgamation

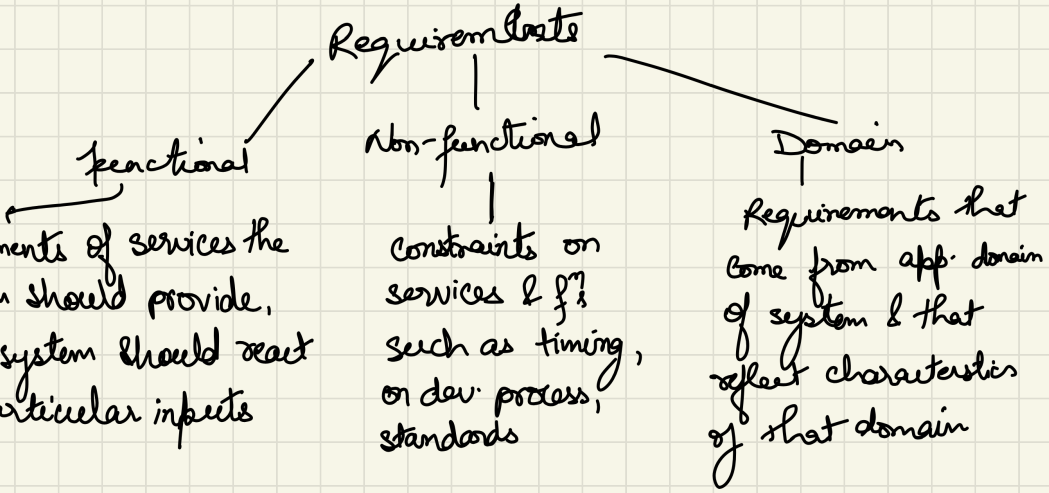
Req. should state what system should do and design should describe how it does this

guidelines for writing req.:
1. use standard format
2. use language in a consistent way: shall / should
3. text highlighting

- Problems with NL specifications:
  - Ambiguity, Over-flexibility, Lack of modularisation

- Requirements should be both complete and consistent.

- Requirement Engg : process of establishing
① services that customer requires from a system
② constraints under which it operates and is developed.



Slide Set 5 (Requirement Gathering) :

- Requirement Gathering (7 step Model) : IEENSVR

Inception – Elicitation – Elaboration – Negotiation – Specification – Validation
|
Req. Management

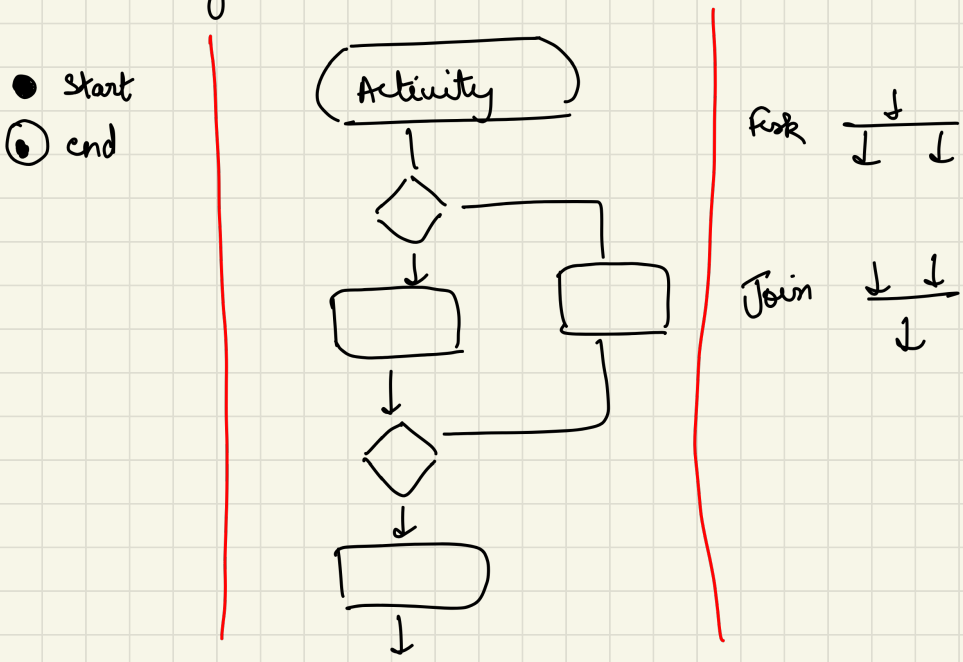
- Inception Task :

UML Diagram - 1 :

- Use case template : Precondition, Use case, Actor, Purpose, Overview, Type

Actor Action (AA)
System Response (SR)
System Action (SA)
Exceptional Flow of Events
Alternate flow
Post Condition

- Activity diagram : (use case at more detailed level)



- Swimlane / Partition : (flow of control b/w objects)

partition the activity diagram according to the objects

UML - Diagram II :

- Complexity and modelling :

Models → Visualization, Design & doc., Act as a guide for sys. const.

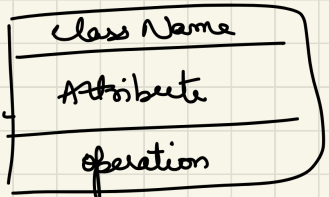
Models → High level, Low level

Model is an abstraction of a system, specifying the modeled system from a certain viewpoint and at a certain level of abstraction.

- UML diagrams
  - Str. → class, obj
  - Phy → Component, Deployment
  - Behavioural → Use, Comm., Seq., State, Activity
- way of using → Forward Engg, Reverse Engg, Sketching, Blueprints, Prog. Language, Conceptual Modeling, Soft. Modeling

- Class diagram :

- private
+ public
# protected

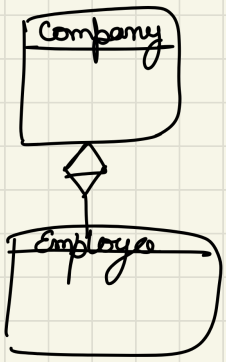


parameter → in, out, inout

Cardinality → * means any number

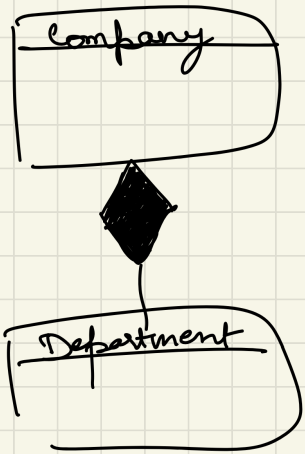
N..M = range

Aggregation



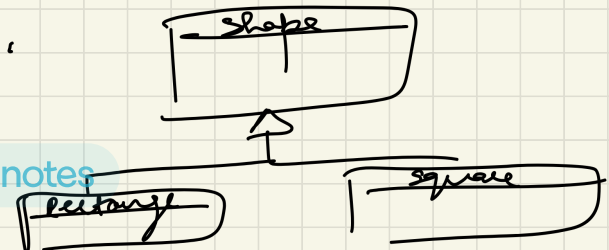
Employee is part of the company

Composition : used when the "part" doesn't exist independently from the whole.



Multiplicity of the whole end should always be 1.

Generalization :



----> shows dependence of one class on another

UML Diagram - III

- Sequence diagram: Shows set of objects and the messages sent & received by those objects

  └ we represent things in a chronological order and in swimlane diagrams, we partition on basis of object rather than activity eg:

- Collaboration diagram: emphasizes the structural organization of objects that send & receive messages

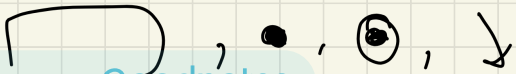
Type of arrows: → procedure call

→ asynchronous flow (caller returns immediately)

----> returns

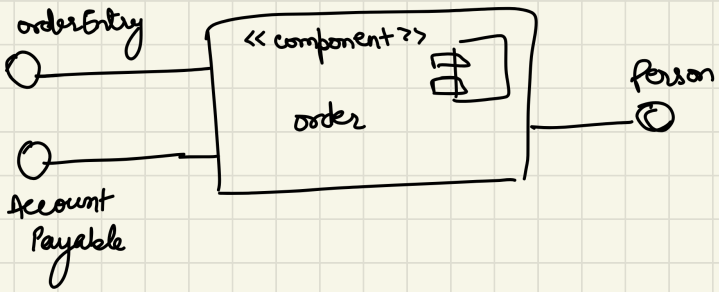
- CRC card ??? (Class Responsibility Collaborator)

- State chart: (doesn't has ◇)



- Purpose of component diagram: (captures physical str. of implementation)

1. Organize source code
2. construct an executable release
3. specify physical database



Lec: Data flow Diagram:

illustrates movement of data b/w external entity & processes & data stores within a system

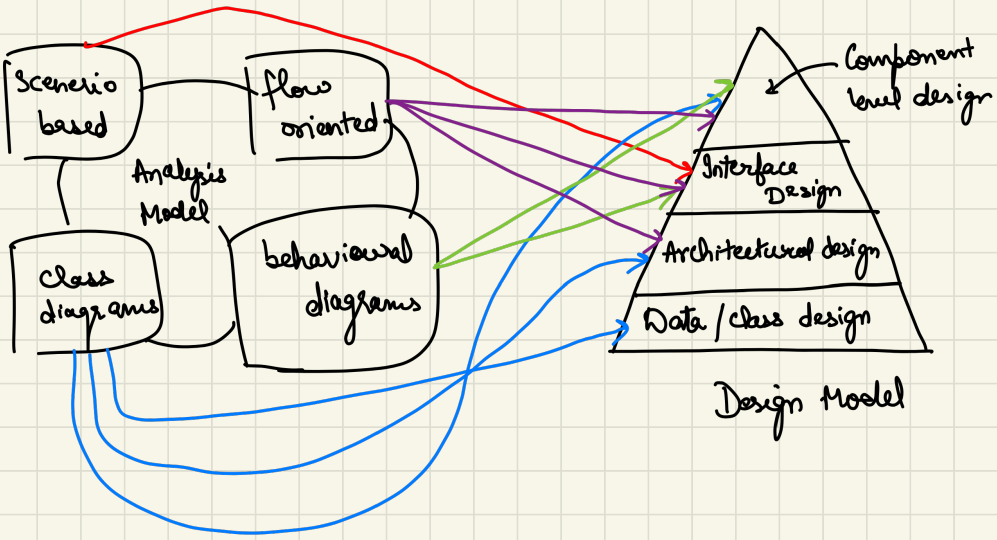
- Source / Sink
- Data flow →
- Process
- Datastore = Read ↓ / ↓ write

No. of inputs = outputs ⇒ balanced DFD

(external systems — source / sink)

Analysis to Design model :

DAIC



Task Set for Software design :

1. Examine info. design model and design appropriate data str.
2. select architectural style
3. partition analysis model into subsystem
4. Create set of design class
5. Design interface for external sys
6. " user "
7. Conduct component level design
8. Develop deployment model

AAP MI F RR D

Design Concept : Abstraction, Architecture, Pattern, Modularity, Information hiding, Functional independence, Refining, Refactoring, Design class

High cohesion
Low coupling

Lec: Architectural Design :

- Why architecture : ① analyze effectiveness of design,
② consider architectural alternatives at stages when making design changes is still easy
③ reduce risk associated with construction of software

- Software Architecture (Definition) :

① components — software components, ext. visible prop., relationship among components
② structure of data and program components
③ Architectural design is transferable

- Importance of Soft. Architecture :

① Enabler for comm.
② graspable model
③ early design decisions

= Data design : (Purpose)
① Translates data into { data structures at component level / application level
② focuses on representation of ds
③ store and retrieve
(Data warehouse & data garbage dumps)

- Data design principles:

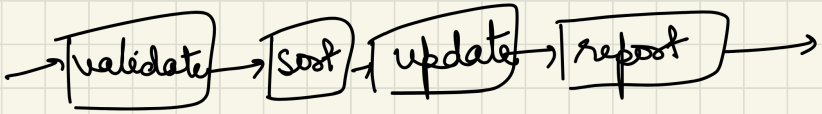
1. systematic analysis
2. ds & operations should be identified
3. Low level data decisions should be deferred
4. Library of useful ds
5. ds should be known only to those modules
6. abstract data types

- Software Architectural Style:

each style describes a system category that encompasses:

1. component types
2. connectors (enable comm., collaboration)
3. semantic constraints (how components can be integrated to form the system)
4. topological layout (components indicating their runtime interrelationship)

- Data flow style:



Data flow style (unmodifiable)

- Batch seq. style
- pipe & filter style

Data flow style — transformation on successive piece of input data

- Batch — each filter is independent
  each runs to completion before the other one starts
- Pipe & filter — pipes contain no info, just movement
  Emphasizes incremental transformation of data
  filters use little contextual info. & retain no state

Adv of data flow style:

1. simplistic
2. simplifies reuse and maintainence
3. easily made into N or distributed system

Disadv. of flow style:

1. encourages batch mentality
2. ordering of filters can be difficult
3. poor performance

Use this system when it makes sense to view your system as one that produces a well defined easily identified output. (sequentially transforming)

- **Call and Return Style** : (Modifiability and Scalability)

1. decomposes into subroutines (hierarchical)
2. single thread of control
3. Remote procedure call (Ted performance by distributing computations, incurs finite comm. time b/w subroutine call & response)
4. Layered (inter-component interaction, immediate neighbour comm., improves runtime)
5. object oriented or abstract data type
   (emphasizes bundling of data, permits inheritance & polymorphism
   keeps internal data representation)

- **Data-Centred System** : (Integrating data)

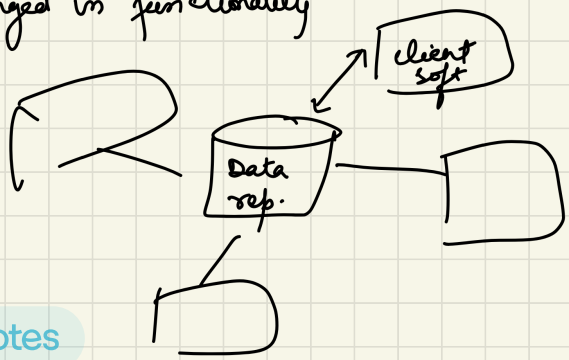
Refers to a system where access and update of a widely accessed data store occur

Client runs on independent thread of control

Shared data may be passive repository or an active blackboard

centralized data communicates with a no. of clients

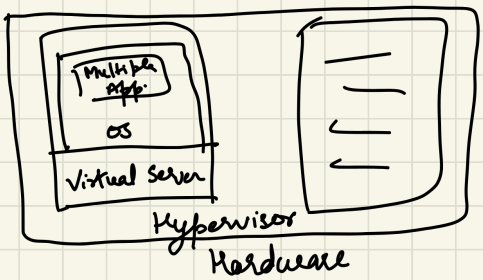
clients are relatively independent of each other so they can be added, removed or changed in functionality



Becomes client/server if clients are modeled as independent processes.

- Virtual Machine Style: (goal of portability)

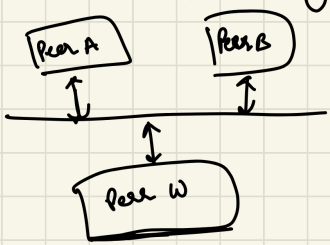
use this when you can make no ~~the~~ machine to directly run the program

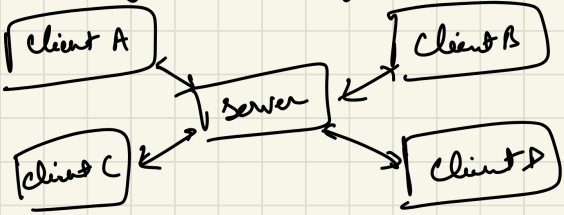


- for simulating functionality that is not native to hardware and or software
  - testing hardware
  - "disaster modes"

eg: interpreters (flexibility, performance cost), command language processors, rule based system

- Independent Component Style: (Modifiability by decoupling portions of computation)



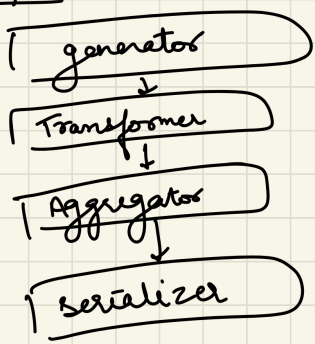


- consists of no. of independent processes that communicate through messages
- Event System Style (Individual components announce data that they wish to share)
- Communication process style (scalability)
  - client/server or peer to peer

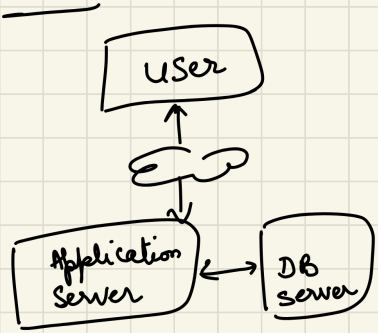
- use this system when your system has graphical user interface.
- your system runs on a multiprocessor system
- your system can be structured as a set of loosely coupled components

- Architectural design process : pipes and filters
  Tiered Architecture (2 tier, 3 tier)
  Model - view - controller

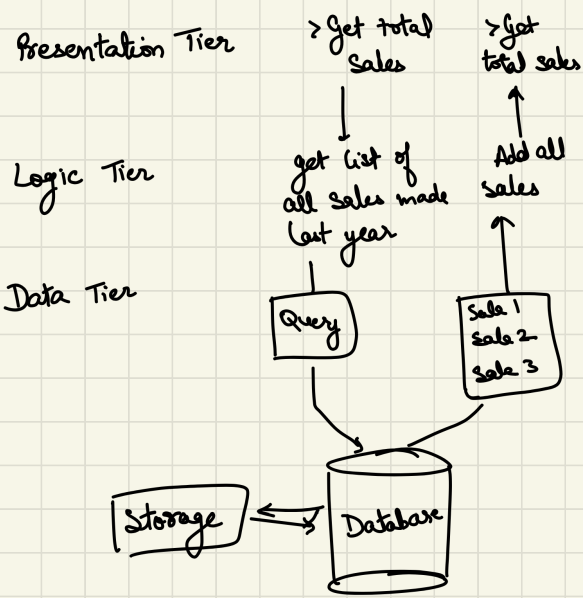
pipe & filter :



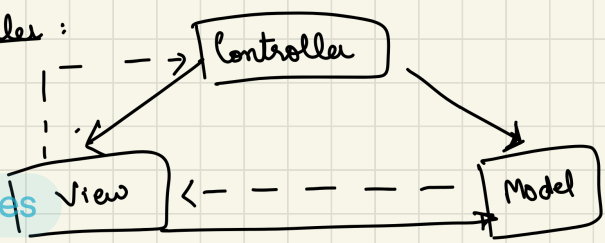
2 Tier : (Client / Server)



3 tier :



- Model - View - Controller :
  mapping traditional I/O into GUI realm



isolates business logic from input and presentation

Unit - Testing : Software Testing Strategies :

- Testing is process of excercising a program with specific intent of finding errors prior to delivery to the end user.
- what testing shows : errors, req. of conformance, performance, an indication of quality

- Who tests software:

Developer	Independent Tester
① understands the system	Must learn about the system
② will test gently	will attempt to break it
③ Driven by delivery	driven by quality

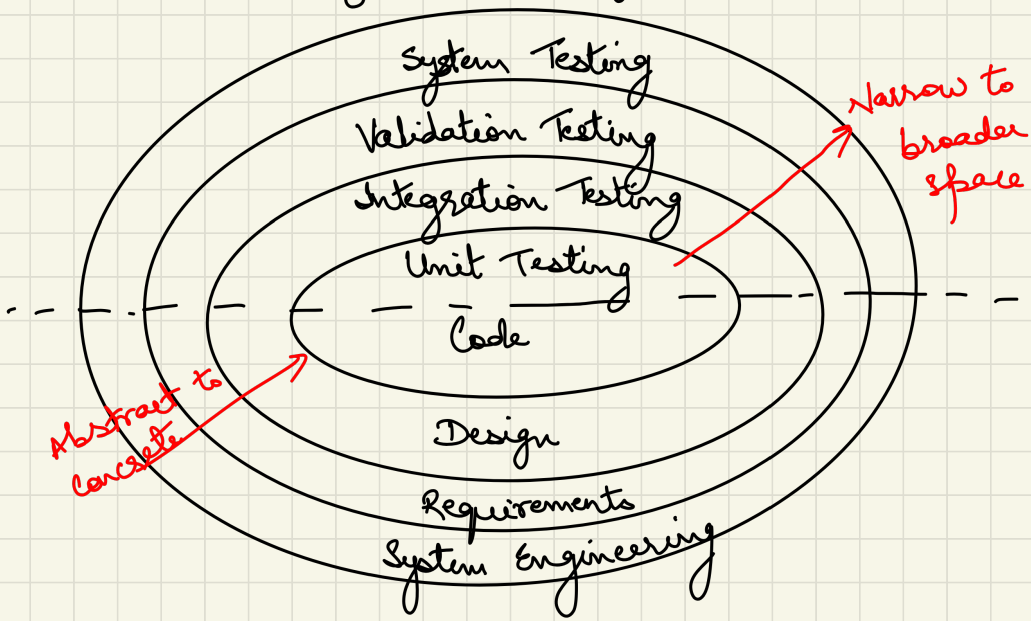
- Software testing is a part of broader activities like verification and validation

① Verification : (Are the algos coded correctly ?)
The set of activities that ensure that software correctly implements a specific f^n or algo.

② Validation : (Does it meet user requirements ?)
The set of activities that ensure that the software that has been built is traceable to customer requirements.

- Testing Strategy:

Unit Test, Integration Test, System Test, Validation Test



Unit testing : concentrates on each component of software as implemented in the source code.

Integration testing : focusses on the design and construction of the software architecture

Validation testing : Req. are validated against the constructed software

System Testing : Software and other system elements are tested as whole

Testing Strategy applied to Conventional Software

- Unit testing
  - Exercises specific paths in a component's control structure to ensure complete coverage and maximum error detection
  - Components are then assembled and integrated
- Integration testing
  - Focuses on inputs and outputs, and how well the components fit together and work together
- Validation testing
  - Provides final assurance that the software meets all functional, behavioral, and performance requirements
- System testing
  - Verifies that all system elements (software, hardware, people, databases) mesh properly and that overall system function and performance is achieved

- Strategic Issues :

specify product req. in a quantifiable manner long before testing

① State testing objectives explicitly

② Understand the users of software and develop a profile for each user category

③ Develop a testing plan that emphasizes "rapid cycle testing"

④ Build robust software that is designed to test itself

⑤ Use effective formal technical reviews as a filter prior to testing

⑥ Conduct formal technical reviews to assess the test strategy and test cases themselves.

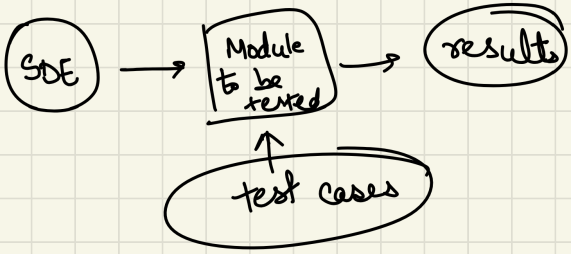
⑦ Develop a continuous improvement approach for testing process

When is testing Complete :

- Everytime a user executes the software, the program is being tested
- One approach is to divide test results into various severity levels, then considers testing to be complete when certain levels of errors no longer occur or have been repaired or eliminated

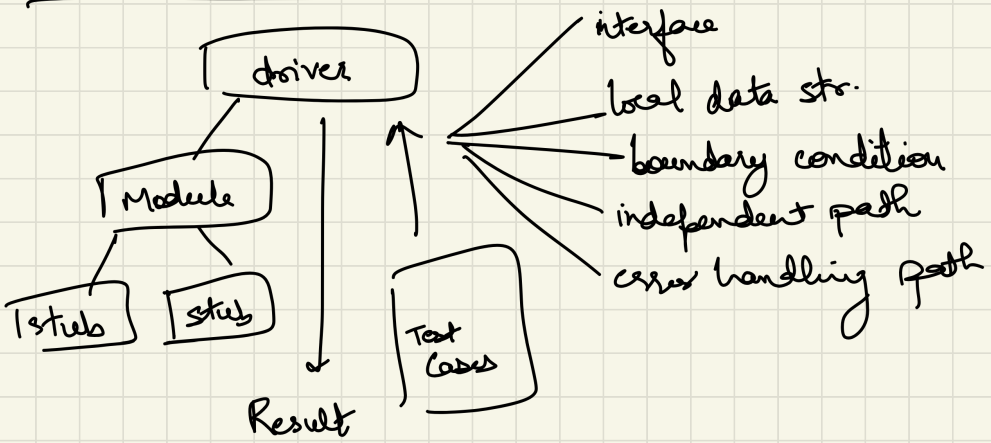
Ensuring a successful software test strategy : Strategic Issues !!

Test strategies for conventional software :



- focuses testing on f^n or software module
- Concentrates on the internal logic and data str.
- Is simplified when a module is designed with high cohesion
  - reduces no. of tc
  - allows errors to be more easily predicted & uncovered
- Concentrates on critical modules and those with high cyclomatic complexity when testing resources are limited

unit test environment :



Target for unit test cases :

(1) Module interface - Ensure that info flows properly into and out of module

(2) Local ds - Ensure that data stored temporarily maintains its integrity during all steps in algo execution

(3) Boundary conditions - Ensure that the module operates properly at boundary values established to limit or restrict processing

(4) Independent paths - Paths are exercised to ensure that all statements in a module have been executed atleast once.

(5) Error handling paths - Ensure that the algorithms respond correctly to specific error conditions

Common Computational errors in execution paths:

1. Misunderstood or incorrect arithmetic precedence
2. Mixed mode operations (eg int, float, char)
3. Incorrect initialization of values
4. Precision inaccuracy and round-off errors
5. Incorrect symbolic representation of an expression (int vs float)
6. comparison of different datatypes
7. Incorrect logical operators or precedence
8. incorrect comparison of variables
9. Improper or non-existent loop termination
10. failure to exit when divergent iteration is encountered
11. Improperly modified loop variable
12. Boundary value violations

Problems to uncover in error handling:

1. error description is unintelligible & ambiguous
2. error noted doesn't correspond to error encountered
3. error condition causes OS intervention prior to error handling
4. exception condition processing is correct
5. error description does not provide enough info to assist in the location of the cause of error.

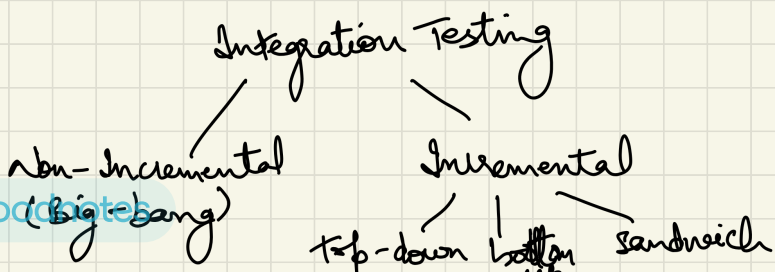
Driver and Stubs for unit testing:

- **Driver**: A simple main program that accepts test case data, passes such data to the component being tested and prints the returned results

- **Stubs**:
  1. serve to replace modules that are subordinate to the component to be tested.
  2. It uses the module's exact interface, may do minimal data manipulation, provides verification of entry and returns control to the module undergoing testing

- **Drivers and stubs both represent overhead**:
  Both must be written but don't constitute part of the installed software product

- **Integration Testing**:
- At the same time integration is occuring, conducts tests to uncover errors associated with interfaces.
- objective is to take unit tested modules and build a program structure based on the prescribed design



- Non-Incremental:
- All components are combined in advance
- The entire program is tested as a whole
- Chaos results
- Many seemingly unrelated errors are encountered
- Correct is difficult ∴ isolation of causes is complicated
- Once set of errors are corrected, more errors occur, and testing appears to enter an endless loop.

Incremental:

- The program is constructed and tested in small increments
- errors are easier to isolate and correct
- Interface are more likely to be tested completely
- A systematic test approach is applied

Top-down Integration

- Modules ar… control hierarchy, b…
- Subordinate… rst or breadth-first fashion…
  - DF: All r…
  - BF: All n… ntegrated
- Advantages…
  - This app… early in the test process…
- Disadvantag…
  - Stubs need to be created to substitute for modules that have not been built or tested yet; this code is later discarded
  - Because stubs are used to replace lower level modules, no significant data flow can occur until much later in the integration/testing process

Bottom-up Integration

- Integration and testing starts with the most atomic modules in the control hierarchy
- Advantages
  - This approach verifies low-level data processing early in the testing process
  - Need for stubs is eliminated
- Disadvantages
  - Driver modules need to be built to test the lower-level modules; this code is later discarded or expanded into a full-featured version
  - Drivers inherently do not contain the complete algorithms that will eventually use the services of the lower-level modules; consequently, testing may be incomplete or more testing may be needed later when the upper level modules are available

31

- object oriented testing :

- begins by evaluating the correctness and consistency of the OOA and OOD models
- testing strategy changes :

① concept of unit broadens due to encapsulation

② integration focuses on classes and their execution across a 'thread' or in context of a usage scenario

③ validation uses conventional black box methods

- OOT strategy :

① class testing is equivalent to unit testing
- operations within the class are tested
- the state behaviour of the class is examined

② integration applied three different strategies
- thread based testing - integrates the set of classes required to respond to one input or event
- use-based testing - integrates set of classes required to respond to one use case
- cluster testing - integrates set of classes required to demonstrate one collaboration

- Regression Testing:

: each new addition or change to baselined software may may cause problems with functions that previously worked flawlessly

- Regression reexecutes a small subset of tests that have already been conducted

① ensures changes have not propagated unintended side effects

② helps to ensure that changes do not introduce unintended behaviour or add^n errors

③ May be done manually or through the use of automated capture/playback tools

- Regression test suite contains three different classes of test cases

① Test that will exercise on all software f^n

② additional test cases that will work on likely to be affected f^n by the change

③ Tests that focus on the actual software components that have been changed.

- Smoke Testing:

- Taken from world of hardware

- designed as a pacing mechanism for time-critical projects

- Includes:
  - The software is compiled and linked into a build
  - A series of breadth tests is designed to expose errors that will keep the build from properly performing its f^n
  - The build is integrated with other builds and the entire product is smoke tested daily. show stopper error
  - After a smoke test is completed, detailed test scripts are executed.

- Smoke Testing Steps:

1. Software components that have been translated into code are integrated into a "build".
2. A series of tests is designed to expose errors that will keep the build from properly performing its function.
3. The build is integrated with other builds and the entire product is smoke tested daily.

- Benefits of smoke testing:

1. Integration risk is minimized
2. quality of end product is improved
3. Error diagnosis and correction are simplified.
4. Progress is easier to assess

- High order Testing :

① Validation Testing : focus is on software requirements
② System " : " " " system integration
③ α-β " : " " " customer usage
④ Recovery " : forces software to fail in a variety of ways and verifies that recovery is properly performed
⑤ Security " : verifies protection mechanism build into a system will, in fact, protect it from
⑥ Stress testing : executes a system in a manner that demands resources in abnormal quantity, freq. or vol.
⑦ Performance testing : test the run-time performance of software within the context of an integrated system.