Module - 3
Software Design & Development
Ch. 10 - Architectural Design

* Architectural Design:

Establishing the overall structure of a software system.
The design process for identifying the sub-systems making up a system & the framework for sub-systems control & communication is called architectural design.

* Software Architecture. The output of this design process is a description of the software architecture.

* Advantages of Explicit Architecture

- Stakeholder communication
- System analysis
- Large-scale reuse

* Architectural Design Process

- System Structuring
- Control Modelling
- Modular Decomposition

* Sub-systems: A sub-system is a system in its own right whose operation is independent of the services provided by other sub-systems.

* Modules: A module is a system component that provides services to other components but would not normally be considered as a separate system.

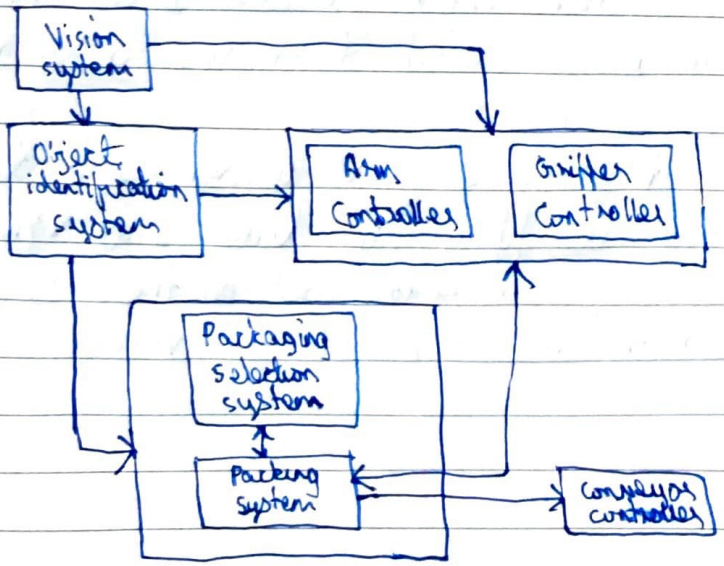
* Architectural Models

- Static Structural Model that shows the major system components
- Dynamic Process Model that shows the process structure of the system.
- Interface Model that defines sub-system interfaces.
- Relationships Model such as a data-flow model
- Distribution Model which shows how sub-systems may be distributed across computers.

* Architecture Attributes

- Performance
- Security
- Safety
- Availability
- Maintainability

* Packeting Robot Control System

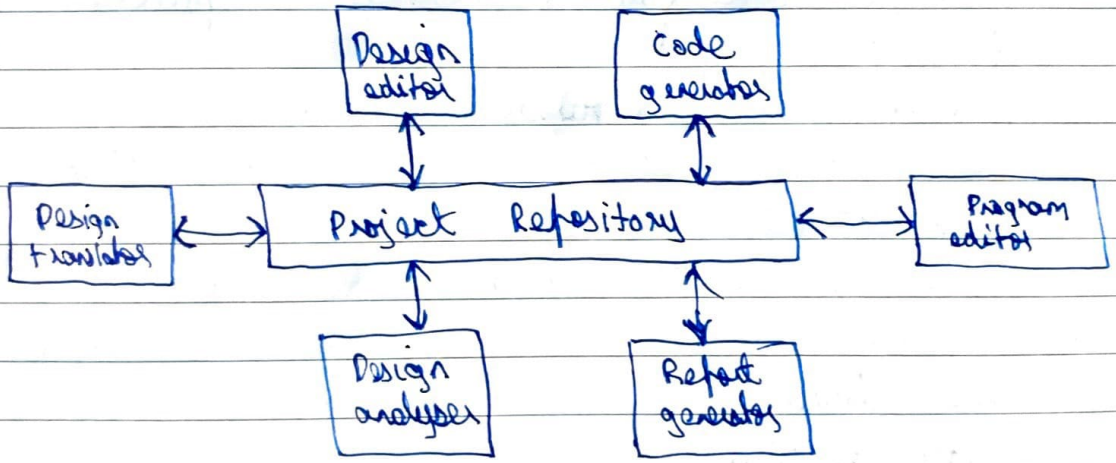


* The Repository Model

Sub-systems must exchange data. This may be done in two ways:
- Shared data is held in a central database or repository & may be accessed by all sub-systems.
- Each sub-system maintains its own database & passes data explicitly to other sub-systems.

When large amounts of data are to be shared, the repository model of sharing is most commonly used

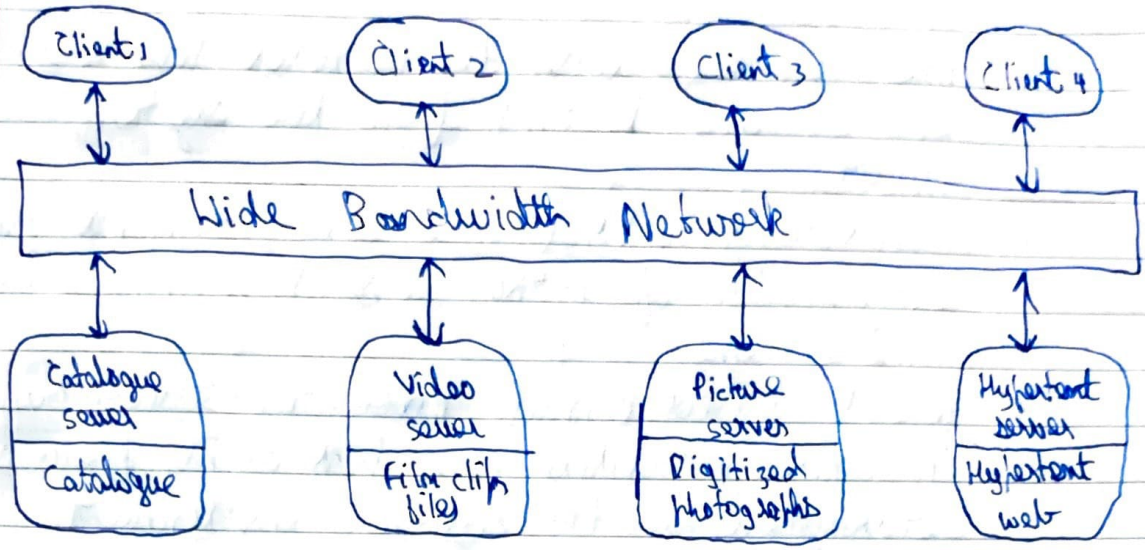
* CASE Toolset Architecture



* Client-Server Architecture

Distributed system model which shows how data & processing is distributed across a range of components.
Set of stand-alone servers which provide specific services such as printing, data management etc.
Set of clients which call on these services.
Network which allows clients to access servers.

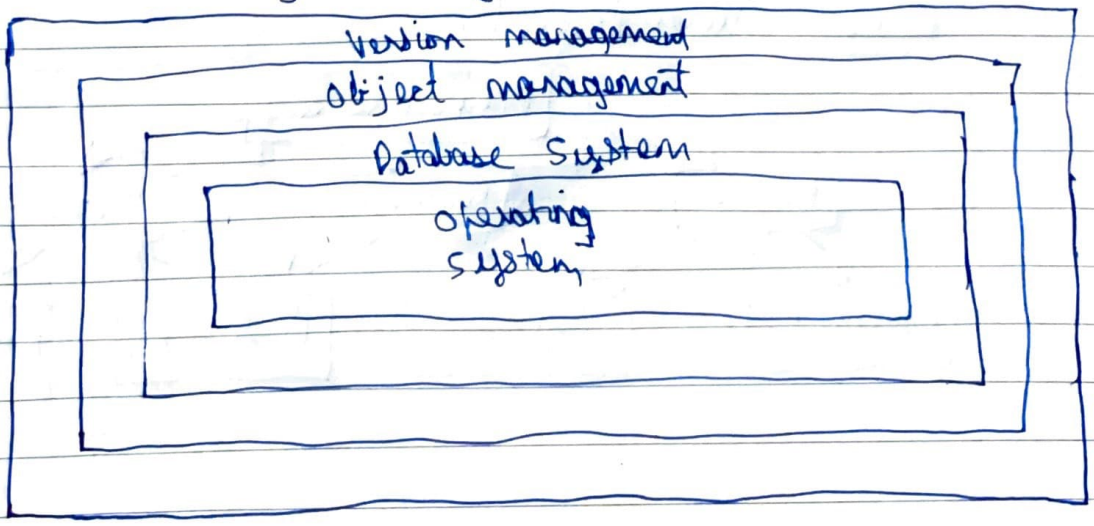
* Film & Picture Library



* Abstract / Layered Machine Model

Used to model the interfacing of sub-systems. Organises the system into set of layers (or abstract machines) each of which provide a set of services.
Supports the incremental development of sub-system in different layers. When a layer interface changes, only the adjacent layer is affected. However, often difficult to structure systems in this way

* Version Management System



* Control Models

Are concerned with the control flow between sub-systems. Distinct from the system decomposition model.

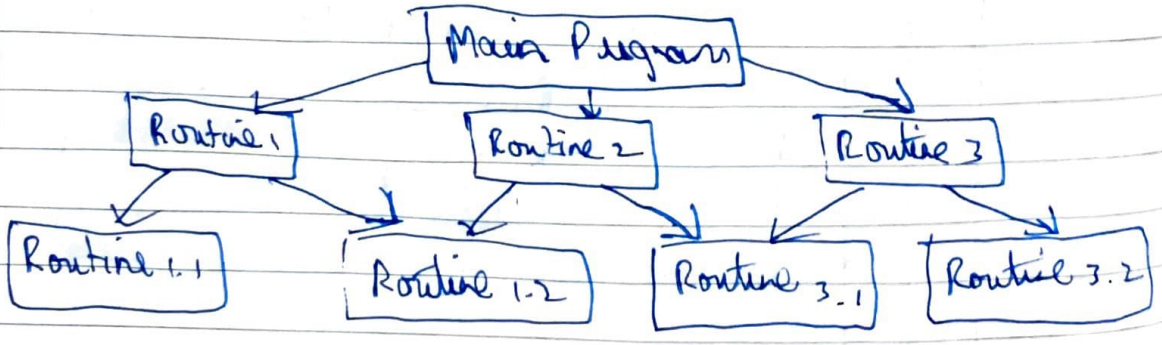
- Centralised Control - One subsystem has overall responsibility of control & starts & stops other sub-systems
- Event-based Control - Each sub-system can respond to ~~events~~ externally generated events from other subsystems or the system's environment.

* Centralised Control

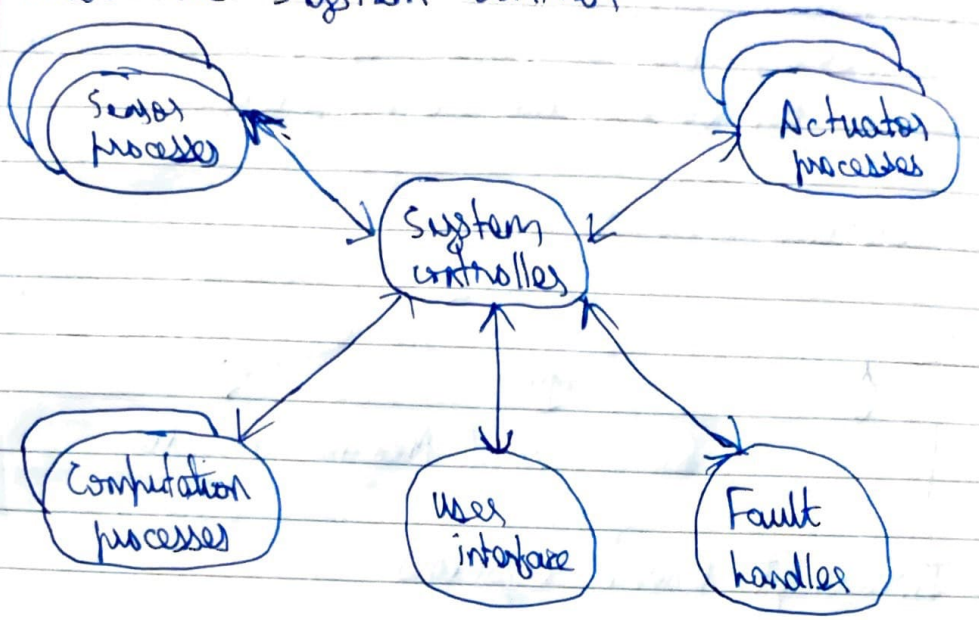
A control sub-system takes responsibility for managing the execution of other sub-systems.

- Call-return Model - Top-down subroutine model where control starts at the top of a subroutine hierarchy & moves downwards. Applicable to sequential systems.
- Manager Model - Applicable to concurrent systems. One system component controls the stopping, starting & coordination of other system processes. Can be implemented in sequential systems as a case statement

* Call-return Model



* Real-time System Control



* Event-Driven Systems

Driven by externally generated events where the timing of the event is outwith the control of the sub-systems which process the events.
Two principal event-driven models
- Broadcast models
- Interrupt-driven models

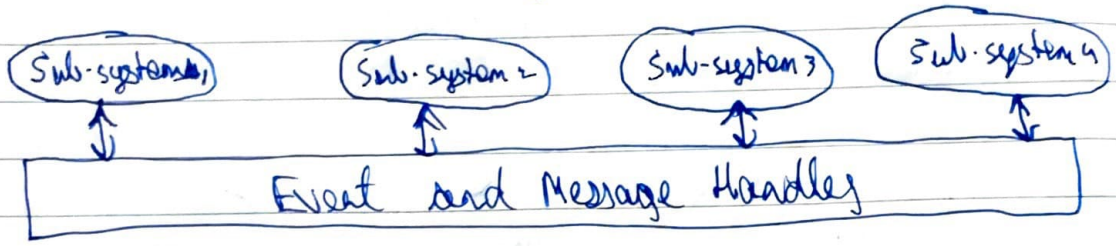
Other event driven models include spreadsheets & production systems.

* Broadcast Model

An event is broadcast to all sub-systems. Any subsystem which can handle the event may do so.
Effective in integrating sub-systems on different computers in a network.
Sub-systems register an interest in specific events. When these occur, control is transferred to the subsystem which can handle the event.
Control policy is not embedded in the event & message handler. Sub systems decide on events of interest

to them.
However, sub-systems don't know if or when an event will be handled.

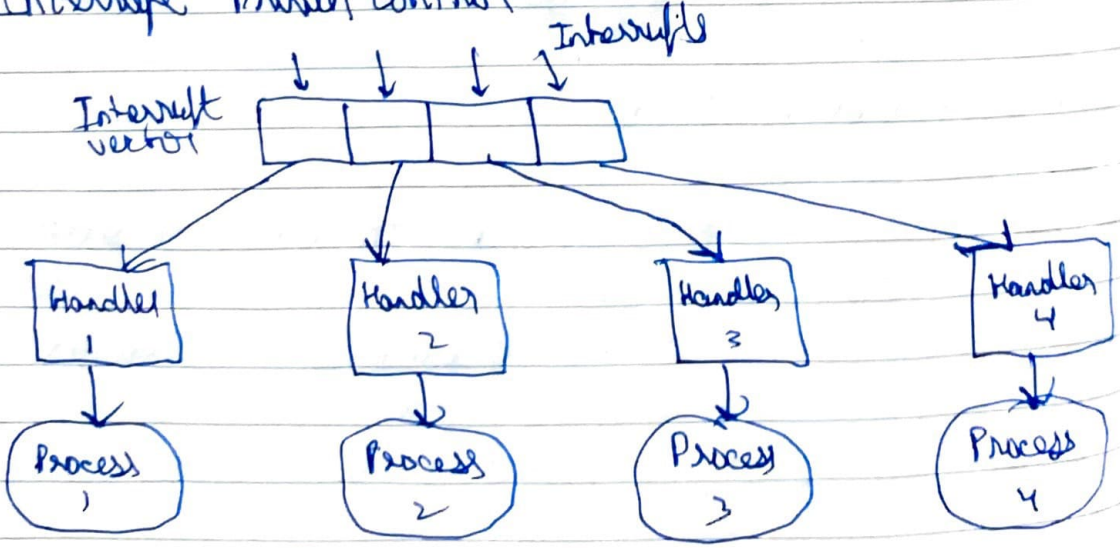
* Selective Broadcasting



* Interrupt-Driven Systems

Used in real-time systems where fast response to an event is essential.
There are known interrupt types with a handler defined for each type.
Each type is associated with a memory location.
A hardware switch causes transfer to its handler.
Allows fast response but complex to program & difficult to validate

* Interrupt-Driven Control



* Modular Decomposition

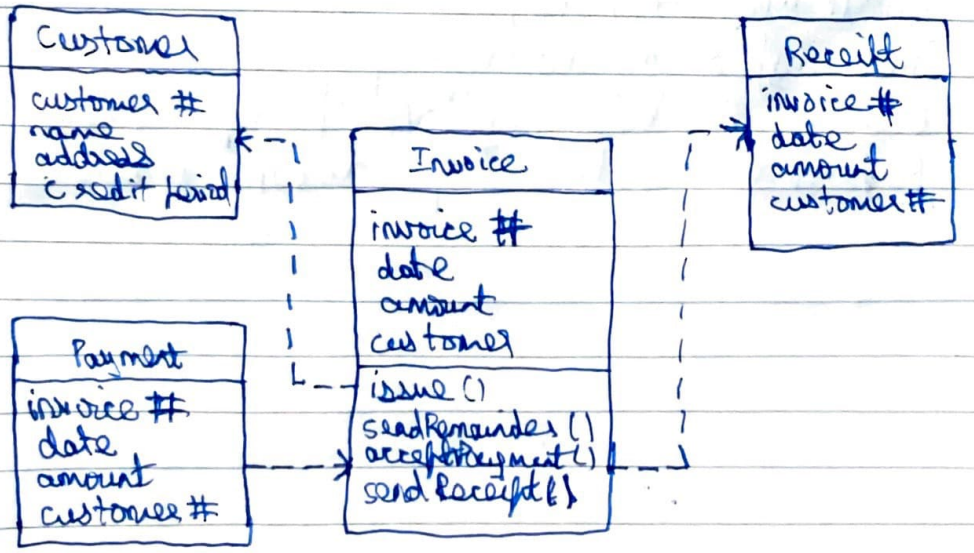
Another structural level where sub-systems are decomposed into modules.
Two Modular Decomposition models covered
- Object Model
- Data-Flow Model

If possible, decisions about concurrency should be delayed until modules are implemented.

* Object Models

Structure the system into a set of loosely coupled objects with well-defined interfaces.
Object-oriented decomposition is concerned with identifying object classes, their attributes & operations.
When implemented, objects are created from these classes & some control model used to coordinate object operations.

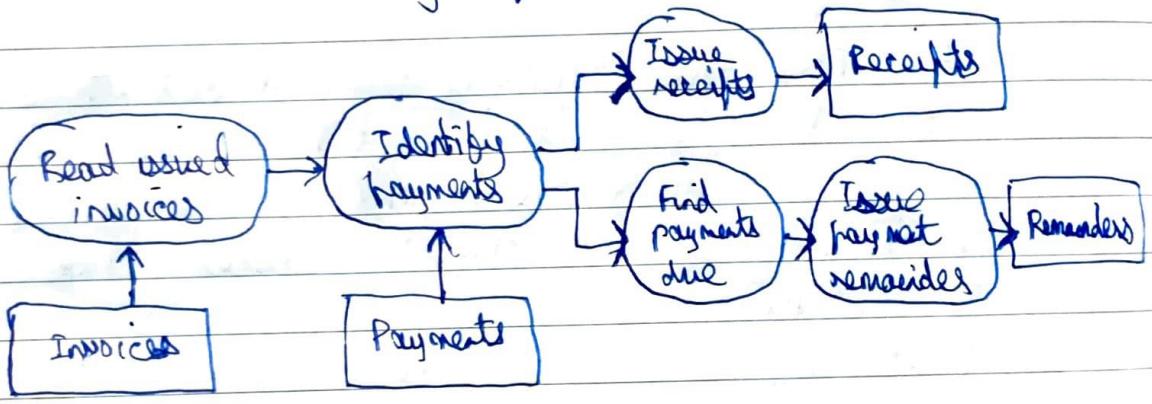
* Invoice Processing System



* Data-Flow Models

Functional transformations process their i/p to produce o/p
May be referred to as a pipe & filter model (as in UNIX shell).
Variants of this approach are very common.
When transformations are sequential, this is a batch sequential model which is extensively used in data processing systems.
Not really suitable for interactive systems

* Invoice Processing System



* Domain Specific Architectures

Architectural models which are specific to some application domain
Two types of domain-specific model
- Generic Models - Bottom-to-up models
- Reference Models - Top-down models.

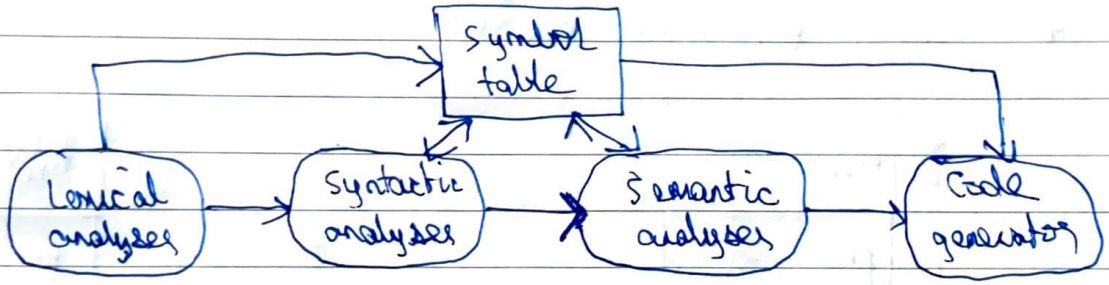
* Generic Models

Compilers model is a well-known example although other models exist more specialised application domains

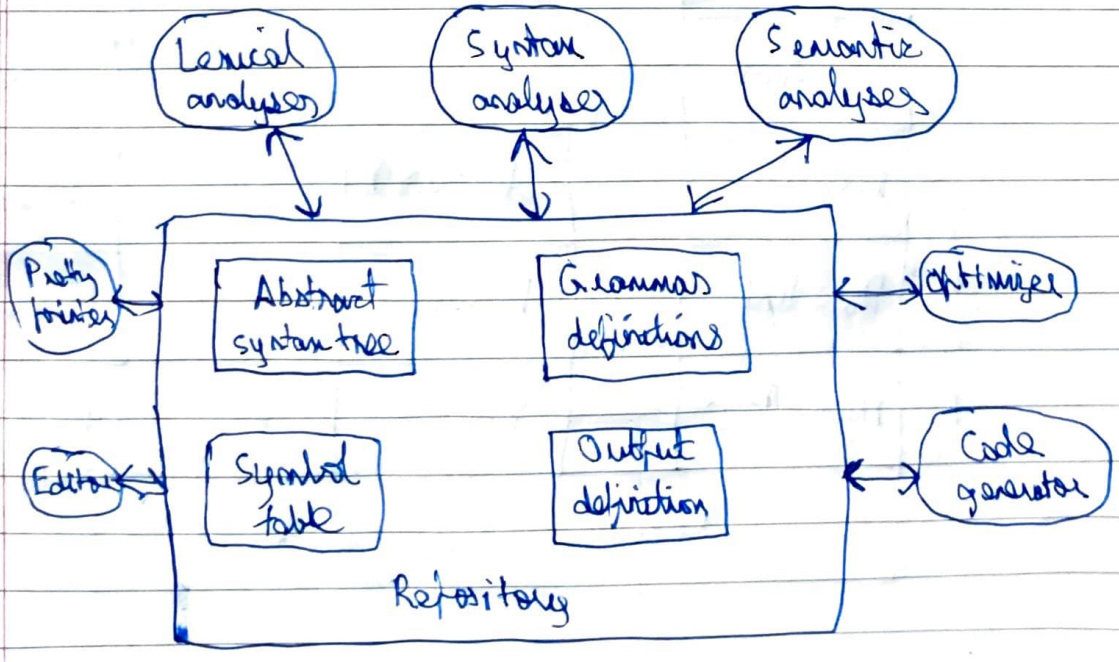
- Lexical analyser · Symbol table · Syntax analyser
- Syntax tree · semantic analyser · code generator

Generic compiler model may be organised according to different architectural models

* Compiler Model



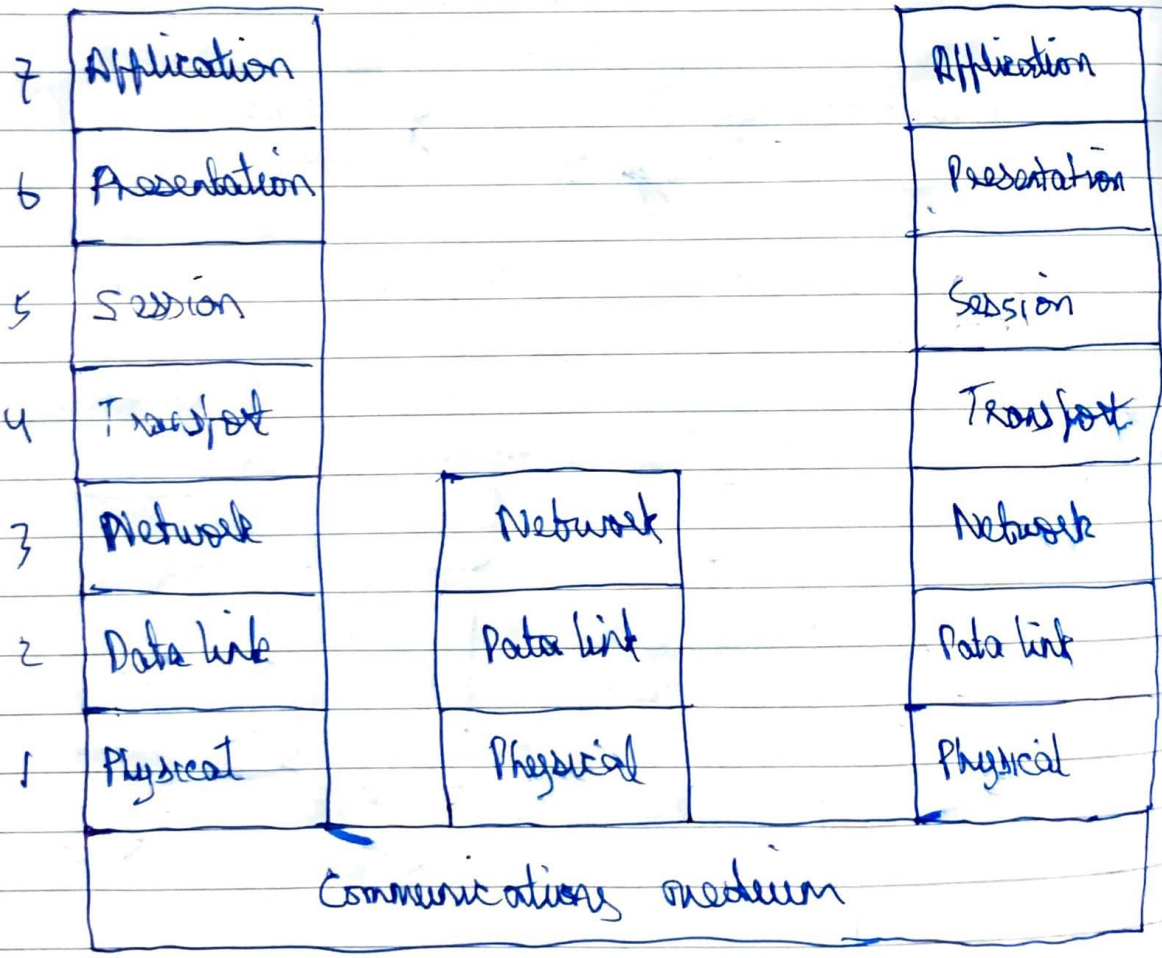
* Language Processing System



* Reference Architectures

Reference models are derived from a study of the application domain rather than from existing systems.
May be used as a basis for system implementation or to compare different systems. It acts as a standard against which systems can be evaluated.
OSI model is a layered model for communication systems.

* OSI Reference Model



Ch. 17 - Rapid Software Development

* Rapid Software Development:

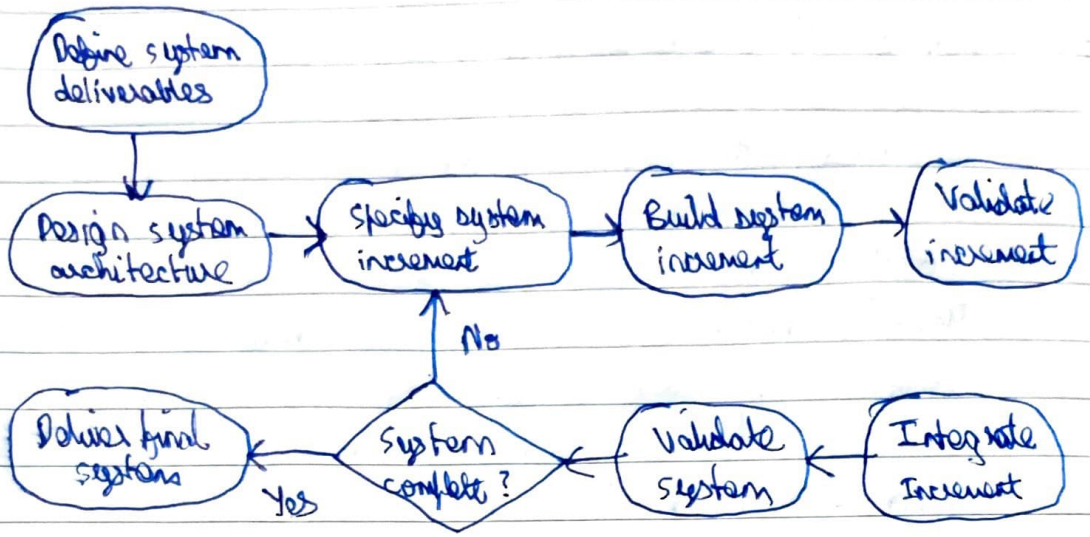
Because of rapid changing business environments, businesses have to respond to new opportunities & competition.
This requires software & rapid development & delivery is not often the most critical requirements of software..
Business may be willing to accept lower quality software if rapid delivery of essential functionality is possible.

* Requirements:

Because of the changing environment, it is often impractical & an approach to development based on iterative specification & delivery is the only .

Because of the changing environment, it is often impossible to arrive at a stable, consistent set of system requirements.
Therefore a waterfall model of development is impractical & an approach to development based on iterative specification & delivery is the only way to deliver software quickly

* An Iterative Development Process



* Advantages of Incremental Development

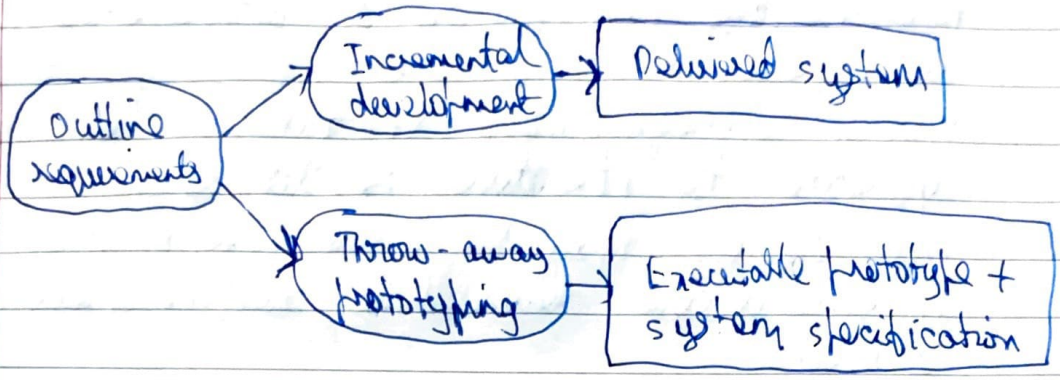
- Accelerated delivery of customer services.
- User engagement with the system.

* Problems with Incremental Development

- Management problems
- Contractual problems
- Validation problems
- Maintenance problems

* Prototyping: Prototyping, where an experimental system is developed as a basis for formulating the requirements may be used. This system is thrown away when the system specification has been agreed.

* Incremental Development & Prototyping



* Conflicting Objectives

- The objective of incremental development is to deliver a working system to end users. The development starts with those requirements which are best understood.
- The objective of throw-away prototyping is to validate or derive the system requirements. The prototyping process starts with those requirements which are poorly understood.

* Agile Methods

Dissatisfaction with the overheads involved in design methods led to the creation of agile methods. These methods are:

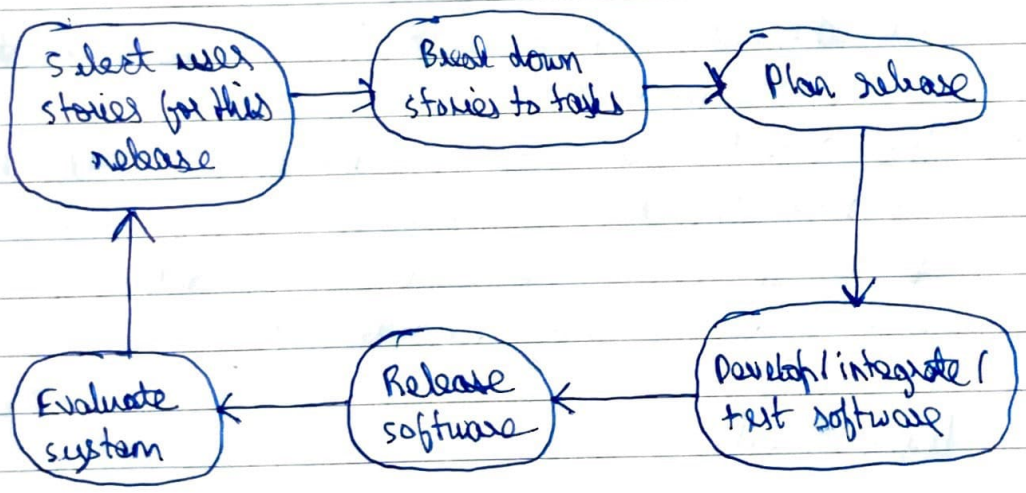
- Focus on the code rather than the design
- Are based on an iterative approach to software development.
- Are intended to deliver working software quickly & evolve this quickly to meet changing requirements. Agile methods are probably best suited to small/medium-sized business systems or PC products.

* Extreme Programming

Perhaps the best-known & most widely used agile method.
Extreme Programming (XP) takes an 'extreme' approach to iterative developments.

- New versions may be built several times per day.
- Increments are delivered to customers every 2 weeks
- All tests must be run for every build & the build is only accepted if tests run successfully.

* The XP Release Cycle



* Testing in XP

- Test-first development.
- Incremental test development from scenarios.
- User involvement in test development & validation.
- Automated test harnesses are used to run all component tests each time that a new release is built.

* Test-First Development

Writing tests before code clarifies the requirements to be implemented.
Tests are written as programs rather than data so that they can be executed automatically. The test includes a check that it has executed correctly.
All previous & new tests are automatically runs when new functionality is added. Thus, checking that the new functionality has not introduced errors

* Pair Programming

In XP, programmers work in pairs, sitting ~~and~~ together to develop code.
This helps develop common ownership of code & spreads knowledge across the team.
It serves as an informal review process as each line of code is looked by more than 1 person.
It encourages refactoring as the whole team can benefit from this.
Measurements suggest that development productivity with pair programming is similar to that of two people working independently.

* Rapid Application Development

Agile methods have received a lot of attention but other approaches to rapid application development have been used for many years. These are designed to develop data-intensive business applications & rely on programming & presenting information from a database.

* RAD Environment Tools

- Database Programming language
- Interface generators
- Links to office applications
- Report generators

* A RAD Environment