Software Engineering

Date / /

⇒ Analysis vs Design

①

Analysis ↓	Design ↓
• what is the problem	• how to build a Solution
• focuses on way human activities are conducted.	

⇒ Software Architecture

②

Styles { • pipe and filter
• Object oriented
• layered • repositories
$\underbrace{\text{layered • repositories}}_{\text{eventbased}}$ ↓ MVC arc

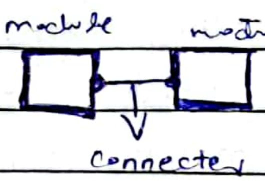
• components of the software
• how components use each other's functionality Eg Data
• how control is managed b/w components

⇒ Coupling and Cohension

③

• Coupling ↓
• Cohesion ↑

Cohension ↓ Apis me jo bond 1 module ka wo strong



Low coupling	High coupling
(Agr 1 ma change tou dsray pr impact normal)	(Agr 1 ma change ↑ tou dsray pr beh ↑)

Date / /

Pipe-and-filter

Example:-
- UNIX shell commands
- signal processing

(4)

⇒ • - Properties

- filters don't need to know anything about what they are connected to.
- filter can be implemented in parallel
- behaviour of system is composition of behaviour of the filters.

⇒ Advantages

- • - Easy to understand the overall input/output.
- • - They support reuse b/c two filters can be joined together but conditions/ data should be match for both filters
- • - System can be easily maintained and enhanced./ new filters can be added to system and old filters can be replaced by improved ones
- • - They permit analysis of many types e.g. deadlock
- • - Support concurrent execution

↓
Bhoot saari cheezian ek sath.

- • - Not good choice for interactive systems b/c of transformational character.

  $\underbrace{\text{interactive systems}}_{\text{User can interact and can change}}$

- • - Excessive parsing and unparsing leads to loss of performance and increase complexity.

  $\overbrace{\text{parsing}}^{\text{Zaida filters \& pipes}}$

Date / /

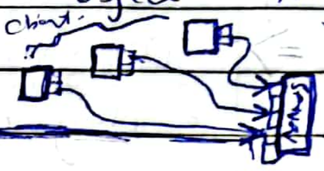
Object - Oriented e.g., abstract datatypes.

(5)

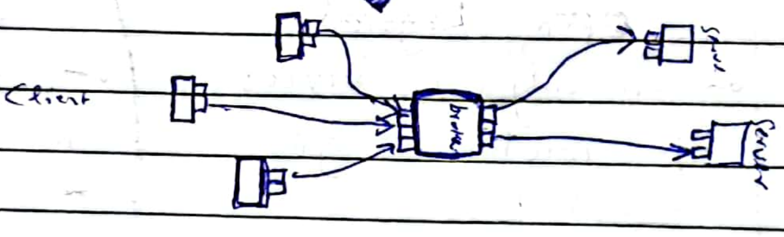
⇒ Properties:-
- data hiding.
- decompose problems into sets
- can be multi-threaded or single.

⇒ Disadvantage
- Object should know identity of object they wish to interact with

⇒ Varient 1: Client server



⇒ Varient 2: Object broker



(6)

$\underbrace{\text{Layered} \qquad \text{repositories}}_{}$ Event based

e.g., ① debugging system
② database management system
③ Graphical user interface

⇒ Properties:-
- supports reuse & evolution of system
- announcers of events don't need to know who will handle the event.

⇒ Disadvantage:
- Components have no control over ordering of computations.

↓

implement

Date / /

⑦

⇒ **Layered Systems:-** e.g., • operating system • communication protocol.

⇒ **Properties:-**

- Support increasing levels of abstraction during design
- support re-use and enhancement
- can define standard layer interfaces

⇒ **Disadvantages:**

- May not be able to identify (clean) layers.

Open Layered	Close Layered
•- can only use services jo neechay ho gai bilkul.	•- kahin sa bhi services ha sakta.
•- dependency kaam ho jati / impact ek ka dsray pr ~~[illegible]~~	•- More compact code, as services of lower layers can be accessed directly.
	•- Break encapsulation of layer, so dependencies b/w layers.

⑧

⇒ **Repositories:-** e.g., ① data base ② programming environment ③ blackboard expert system

⇒ **Properties:-**

•- can choose where is center of control.

•- reduce need to duplicate complex data

⇒ **Disadvantage:**

•- blackboard becomes a bottleneck.

information system ← Repository Styles Examples → Reverse Engineering System

↓ programming environment

↓ Graphical editors

↓ AI knowledge Bases

Date / /

Software Architecture

Coupling ↓ & Cohension ↑

⇒ Architecture styles
- Pipe & filter
- Object Oriented

⇒ **Pipe & Filter**

- Properties
- Advantages
- Examples →
  - UNIX Shell command
  - Signal Processing
- Easy to understand input/output
- Support reuse
- Maintained and enhanced
- Permit analysis (deadlock & through put)
- Support concurrent execuation
- Disadvantages
- not good choice for interactive system
- parsing & unparsing → loss of performance
  - complexity ↑

⇒ **Object Oriented** - Examples: Abstract datatypes.

- Properties
  - data hiding
  - decompose problems into sets
  - can be multi-threaded by single
- Disadvantage
  - object should know identity of object they wish to interact with

⇒ Event based
- Layered
- repositories (MVC)

e.g., debugging system, database management system, graphical user interface

- Properties
  - reuse
  - announcer of events doesn't know who will handle event
- Disadvantage

  components have no control over order of implementation

⇒ **Layered ~~repos~~**

e.g.,
- operating system
- communication protocol

- Properties
  - increasing level of abstraction.
  - reuse & enhancement
  - standard layer interfaces
- Disadvantage
  - not able to identify layers.

⇒ **Repositories**

e.g., database, programming environment, blackboard expert system

- Properties

  can choose center of control
- Disadvantage

  blackboard becomes bottleneck.

Software Testing

Date / /

→ **Program testing goals :-**

1 ① to demostrate to the developer and the customer that software meets its requirements.

② To discover situations in which behaviour of Software is incorrect.

→ **Verification vs Validation**

2

Verification ↓ Are we building Correct product

Validation ↓ Are we building the right product

→ **V & V confidence**

3

① AIM → establish confidence that system is fit for purpose

② Depends → System purpose, User expectation, marketing environment.

→ **Stages:**

4

•- Development testing → System tested during development

•- Release testing → Separate testing team. test whole system before it is released to users.

•- User testing → Users of the system test system in their own environment.

5

→ **Development testing** (carried out by team developing the system)

•- unit testing → where individual units are tested. It (defect testing process.) focuses on testing the functionality.

•- Component testing → individual units are combined to create components. It should focus on testing component interfaces.

•- System testing → components are integrated. Whole system is tested focused on testing component interactions. (tests emergent behavior of a system)

Date / /

•- Automated → unit testing should be automated. We make use of a test automation framework (JUnit) to write and run program tests.

•- Regression → • changes say code broke tou nai kr raha.

• Manual ma its expensive, automated ma straightforward. Tests are rerun every time a change is made to program.

⇒ ⑥ **Release testing (Black box)**

Goal ⇒ Convince supplier of the system that it is good enough for use.

Performance testing ⇒ • testing properties of system such as performance and reliability.

• involves planning a series where load is increased until performance unacceptable.

• Stress testing is form of performance testing where system is jaan pooch (deliberately) kr overloaded to test its failure behaviour.

⇒ ⑦ **User-testing**

•- Alpha testing → User of software work with development team at developer's site.

•- Beta testing → A release of software is made avalible to users and allow them to raise problem.

•- Acceptance testing ⇒ It is ready to be accepted from system developers. Primary for custom system.

Stages:-

1. Software Testing

Development testing

- Unit testing (defect testing process)
  - Automated testing
- Component testing (testing component interfaces)
- System testing (testing component interactions)
- Regression testing (changes sa code broke to nai kr raha)

Release testing (Blackbox)

- Performance testing
  - Stress testing

User testing

- Alpha testing (User + developers)
- Beta testing
- Acceptance testing

→ **Stages:-**

1. Define acceptance criteria
2. Plan acceptance testing
3. Derive acceptance test
4. Run acceptance test
5. Negotiate test results
6. Reject / accept system

Date / /

bugs — system fails

- Fault & Failures

- Fault avoidance / Fault detection / Fault tolerance

Fault avoidance	Fault detection	Fault tolerance
↓	↓	⇓
bug-free	testing & verification	bugs hain pr phr beh chal rahe hy

- How to write good test cases
  - Identify purpose of testing
  - Define how to perform testing
  - Identify any non-functional requirements

- Structuring you test cases
  1. Testcase ID
  2. Testcase description
  3. Module to be tested
  4. Test data
  5. Test steps
  6. Expected resuelts
  7. Actual results
  8. A result
  9. Comments (screen short)

① Use-Case.

R-001:

Use Case:
Actors:
purpose:
overview:
Type: Primary, real.
Cross Refrences. R-001.

Actor Action	System Action
1	-
2	3
4	5

②
- Test Case ID: T-101
- Test Case description: Add any two numbers.
- ~~[illegible]~~ Test steps:

ENJOY YOUR WRITING WITH LUCKY PAPER PRODUCTS

LUCKY®

Date / /

⇒ Types of testing

- Software testing
  - Unit testing
  - Functional testing
- Integration testing
  - Compliance
  - Interoperability Testing
- System testing
  - Recovery
  - Security
  - environment

Acceptance testing & Release testing
Regression Testing
Stress, Security, Performance, Load testing

⇒ Three levels of correctness (obtaining correct output log)
- Possible correctness (single set input)
- Probable correctness (carefully selected input)
- Absolute correctness (every possible input)

⇒ **Black Box Testing**
- No functional requirements, no black-box testing.
- Uncovers different kind of errors than white box testing.
- Aisay test cases banao which reduce additional test cases
- Black box techniques can supplement the test cases generated by white box

How to design test cases?
- Class partitioning
- Boundary Value analysis
- cause/effect graphing
- error guessing

Date / /

⇒ Boundary Value Analyses (BVA)

Minimum

above minimum

nominal value

below max

max

⇒ Advantages of BVA :-

•– Reduces number of test cases that must be run, thus reduces cost.

•– Elimates the fuzzy criteria of test data selection that is inefficient

•– Helps identify different classes for which program is not working properly.

⇒ glass-box / Structural testing ← White-Box testing → our goal is to ensure that all statements and conditions executed at least once.

•– Typographical errors are random

•– Sometime path executes unexpectely.

•– Edge coverage, Condition coverage, Path coverage are defined mathematically

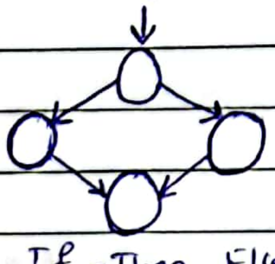
•– There are no algo for white box testing

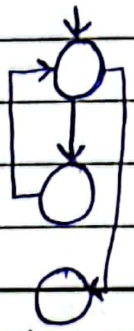
⇒ Control Flow graph (CFG) :

•– Directed graph

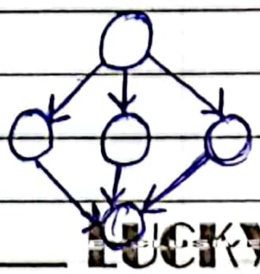
•– Nodes are blocks of sequential statements

•– Edges are transfer of control


If-Then-Else


While loop

Switch.



Date / /

=> Control Flow Coverage

a) Statement/Node Coverage
b) Edge Coverage
c) Condition Coverage
d) Path Coverage

=> Statement/Node Coverage = $\frac{\text{Number of executed Statement}}{\text{Total}} \times 100$